UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

PG477

R_3D_3
Real Robo Rally
Dare-Devil Droids

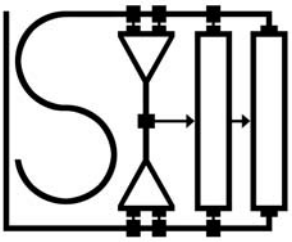
Endbericht

2. November 2006

INTERNE BERICHTE
INTERNAL REPORTS

LS XII (Embedded System Design)
Fachbereich Informatik
Universität Dortmund

Betreuer:
Heiko Falk
Robert Pyka



GERMANY · D-44221 DORTMUND

Endbericht

R3D3

REAL ROBO RALLY
DARE-DEVIL DROIDS

Judith Ackermann, Manuela Engels, Nina Grau, Harald Günther,
Daniel Höcker, Michael Kern, Robert Muhrbeck, Michael Poch,
Daniel Schulte, Jules Souna, Rene Thimel, Benjamin Titz

2. November 2006
Universität Dortmund

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Zu allererst möchten wir einen Überblick über die Projektgruppe 477: „R_3D_3 - Real Robo Rally: Dare-Devil Droids“ vermitteln. In diesem Projekt wird das Spielkonzept der „Robo Rally“ Spiele mit Hilfe interagierender Roboter umgesetzt.

Neben der Motivation (Abschnitt 1.1) zum Projekt wird insbesondere die Aufgabenstellung (Abschnitt 1.2) beleuchtet. Im Anschluss wird der Aufbau dieses Zwischenberichtes (Abschnitt 1.3) dargestellt.

1.1 Motivation

Unabhängig vom Alter empfinden viele Menschen Brettspiele als eine angenehme und unterhaltsame Möglichkeit zur Freizeitgestaltung. Die unzähligen Spielideen, die angeboten werden, sind ein starkes Indiz für die anhaltende Popularität der Spielart. Vertreter sind ganz klassische Spiele wie Schach, das schon seit Jahrhunderten Menschen fasziniert, oder das Gesellschaftsspiel Monopoly, bis hin zu den jährlich erscheinenden Variationen der „Mensch ärger dich nicht“ Spielidee.

Einer gelungenen Idee folgt auch die „Robo Rally“ Spielserie [AMI99, AMI00]. Die Grundidee besteht darin, mit Spielfiguren, die als Roboter gestaltet sind, auf einer Spielfläche eine Folge von Zielpunkten anzusteuern.

Die Spielfiguren können mit Anweisungskarten - von den Spielern zu „Programmen“ zusammengesetzt - gesteuert werden. Diverse Elemente der Spielfelder wirken auf die Roboter- Spielfiguren ein - so gibt es Transportbänder, Mauern und verschiedene schädigende Elemente. Die Roboter können sich auch gegenseitig beschädigen, wodurch jeweils die Fähigkeit zur Programmausführung beeinträchtigt wird.

Durch die Vielzahl der aktiven Spielfeldelemente und die unvorhersehbare Reihenfolge, in der Anweisungskarten gezogen werden, bietet das Spiel durch alle Runden hinweg viele Überraschungsmomente, wodurch den Spielern die Fähigkeit zur ständigen Anpassung ihrer Strategie und der fortwährenden Abschätzung der Aktionen anderer Spieler abverlangt wird. Diese begrenzte Unvorhersagbarkeit des Spielgeschehens und die Freude über eine richtig abgeschätzte Ausführung der Programme (aber auch Schadenfreude bei Missgeschicken anderer Mitspieler) sind sicherlich die Hauptgründe für eine heitere und angenehme Spielatmosphäre.

Gegenüber der klassischen Implementierung der Spielidee, bestehend aus einem Papierspielbrett und den üblichen Kunststofffiguren, gibt es die Möglichkeit, das Spiel als eine Softwarelösung umzusetzen. Obwohl dies die nahe liegendste Lösung wäre, erfüllt sie doch schlicht einige Anforderungen und Wünsche nicht. Das markanteste Defizit ist sicherlich die fehlende haptische Komponente. Ferner sollte bei der Wahl der Implementierungsart die Überlegung, dass Menschen seit jeher durch den Anschein des Spiels gegen eine Maschine fasziniert wurden, nicht außer Acht gelassen werden.

Stellvertretend sei hier Maelzels Schachspieler von 1783 [Poe36] - ein von einem Menschen gesteuerter Schach spielender „Automat“ - genannt, der das Publikum bereits zu Zeiten der Aufklärung fasziniert hat.

Die neuartige Umsetzung der Robo Rally Spielidee durch die Projektgruppe soll daher auf realen, selbst fahrenden Roboter-Spielfiguren basieren, die sich selbständig auf einer 12x12 Spielfelder großen Fläche bewegen.

Die Interaktion mit dem Benutzer kann durch Steuerkonsolen erfolgen, auf denen die Anweisungssequenzen mit virtuellen oder echten Spielkarten erstellt werden.

Mit einer modularen Spielfläche wird gegenüber dem Originalspiel ein weiterer Vorteil erzielt - nämlich, dass durch Variation der Spielfläche unterschiedliche Schwierigkeitsstufen vorgegeben werden können.

1.1.1 Organisation

1.2 Aufgabenstellung

Die Aufgabenstellung der Projektgruppe besteht aus zwei Teilen:

- Aus der Motivation heraus sind allgemeine Ziele zu entwickeln. Diese Ziele sollen durch die Projektgruppe erreicht werden und werden in Abschnitt 1.2.1 vorgestellt.
- Für die genaue Aufgabenstellung sind diese Ziele jedoch zu konkretisieren. Es werden daher aus den allgemeinen Zielen Minimalziele extrahiert, die das Projektteam verbindlich zu erfüllen hat. Diese werden in Abschnitt 1.2.2 dargestellt.

1.2.1 Ziele

Im Rahmen dieser Projektgruppe wurde ein Spiel erstellt, das die Spielidee der Robo Rally Spiele umsetzt. Die Kerneigenschaften dieser Implementierung sind eine modulare, aktive Spielfläche, auf der selbst fahrende Roboter-Spielfiguren das Spielgeschehen ausführen.

Die Aufgabenstellung ist dementsprechend in zwei Hauptaufgabenbereiche zu gliedern. Zum einen sind Spielfiguren zu entwickeln, zum anderen die Spielfläche.

1.2.1.1 Spielfiguren

Die Spielfiguren sollen höchstens $4.5cm$ Durchmesser und eine Höhe von etwa $6cm$ haben.

Die mechanische Realisierung der Spielfiguren besteht in der Entwicklung einer gemeinsamen Plattform für alle - mindestens vier - Spielfiguren. Insbesondere geht es dabei um die Entwicklung eines Antriebs. Der mechanische Aufbau muss ferner hinreichend Platz und entsprechende Haltevorrichtungen für Akkus und Steuerplatine vorsehen. Ebenfalls Bestandteil des mechanischen Aufbaus ist die Anordnung der Sensoren für die Kommunikation und Positionsfindung.

Der elektrotechnische Aufbau der Spielfiguren wird aus einer Mikrocontrollerschaltung bestehen, die für die Ansteuerung der Motoren und die Anbindung der Datenübertragungseinrichtung notwendig ist. Eine Schaltung zum Wiederaufladen der Akkus wird nicht in die Spielfiguren integriert, sondern in Form einer Ladestation realisiert.

Die Auswahl des Mikrocontrollers war ein kritischer Aspekt der Konzeptphase dieser Projektgruppe (PG). Es gab zahlreiche gegenläufige Anforderungen, zwischen denen ein sinnvoller Kompromiss gefunden werden musste. So sollte die Hardware ausreichend Rechenleistung und Speicher bieten, möglichst wenig Energie verbrauchen, leicht zu programmieren sein, eine gute Unterstützung durch Entwicklungstools haben und speziell in dieser PG auch so gewählt werden, dass gegebenenfalls der gleiche Mikrocontrollertyp für die Spielfeldkomponenten verwendet werden kann.

Die Implementierung der Software der Spielfiguren beinhaltet zeitkritische hardwarenahe Komponenten - beispielsweise die Ansteuerung von Motoren oder die Auswertung von Sensoren. Darüber hinaus muss ein Protokollstack für die Kommunikation mit einem Spielfeld und anderen Spielfiguren implementiert werden.

Schließlich werden die Spielfiguren eine gewisse abstrakte Steuerlogik besitzen, die sie auf Anforderungen reagieren lässt. Beispielsweise ist hier die Verarbeitung von Kommandos wie „Fahre ein Feld vor“ und ähnlichen notwendig.

Die Mikrocontroller werden ohne jegliches Betriebssystem ausgeliefert, so dass eine strukturierte Implementierung der Software Aspekte der Betriebssystementwicklung aufweisen wird. Für die Navigation wird auf Methoden aus der Robotik zurückgegriffen und für die Kommunikation muss ein Protokollstack entwickelt werden. Eine zuverlässige und flexible Implementierung wird durch den Gebrauch von Methoden aus dem Bereich des Protocol Engineering geprägt sein.

1.2.1.2 Spielfläche

Ähnlich gestalten sich die Aufgaben bei der Entwicklung der Spielfläche. Die Spielfläche soll aus Blöcken zu jeweils 3x3 Spielfeldern zusammengesetzt werden. Es werden mindestens 16 derartiger Blöcke benötigt, um eine dem Originalspiel entsprechende Spielfläche von 12x12 Feldern zu bieten. Ein Spielfeld soll eine Kantenlänge von 5 cm aufweisen.

Der mechanische Aufbau der Spielfläche ist jedoch wesentlich einfacher als der der Spielfiguren, da keine beweglichen Teile benötigt werden. Jedoch muss ein Block an vier Seiten Kontakte für Stromversorgung und Kommunikation bereitstellen.

Für die Kommunikation soll ein serielles Protokoll verwendet werden, so dass die Anzahl der Kontakte minimiert werden kann. Die Anordnung dieser ist so zu wählen, dass ein beliebiges Zusammenstecken der Blöcke möglich ist, und dabei die Oberseiten der Blöcke eine durchgehende Spielfläche ergeben.

Dabei besteht ein Spielfeldblock aus 9 Spielfeldern, die - für die Mitspieler erkennbar - entsprechend den Spielfeldeigenschaften gestaltet werden. Ferner sind in der Oberseite lichtdurchlässige Bereiche vorzusehen, um ggf. eine lichtbasierende Kommunikation mit den Robotern integrieren zu können.

Der elektrotechnische Aufbau der Spielfeld-Blöcke besteht im Wesentlichen aus dem Mikrocontroller, der nahezu ohne weitere Komponenten direkt die Ansteuerung der Verbindungen zu den anderen Blöcken und die der Oberflächenelemente übernimmt. Hierzu soll gegebenenfalls der gleiche Mikrocontrollertyp wie in den Spielfiguren verwendet werden.

Die Software der Spielfeld-Blöcke hat mehrere Teilaufgaben zu erledigen. In einer ersten Phase müssen die Blöcke eine Selbstorganisation durchführen. Da die Spieler die Möglichkeit haben sollen, die Blöcke beliebig anordnen zu können, muss das Kommunikationsprotokoll zwischen den Blöcken die Erstellung eines konsistenten Modells der Spielfläche ermöglichen.

Für den Spielbetrieb müssen Methoden zum Routing von Datenpaketen zwischen den Blöcken bereitgestellt werden. Ferner wird eine Softwarekomponente benötigt, die den Datenaustausch mit den Spielfiguren durchführen kann. Und schließlich muss ein Spielfeld entsprechend den Spielregeln Aktionen anstoßen können. Insbesondere muss es in der Lage sein, in Kooperation mit anderen Blöcken die Ausführung der Spiel-Programmschritte zu ermöglichen.

Hierbei werden im hohen Umfang Aufgaben aus dem Bereich der Verteilten Systeme bearbeitet werden. Eine besondere Eigenschaft dieses Verteilten Systems wird die reguläre Anordnung der Blöcke sein. Sie ergibt ein Mesh-Netzwerk, so dass beispielsweise auf Konzepte zu effizienten Routingverfahren in parallelen Computer-Architekturen zurückgegriffen werden kann.

1.2.2 Minimalziele

Konkret sind folgende Ziele von der Projektgruppe zu erreichen:

1. Kombinierter Entwurf der Roboter- und Spielflächen-Hardware inklusive Auswahl der zu verwendenden Bauelemente (Controller, Antrieb) für Roboter und Spielfläche.

   Der Hardware-Entwurf ist so vorzunehmen, dass sich ein Roboter unter Nutzung von Navigationshilfen der Spielfläche (optische oder induktive Markierungen) zielsicher von einem Spielfeld zu einem anderen bewegen und mit dem Spielfeld kommunizieren kann.

2. Definition und Entwurf der in der Spielfläche zu verwendenden Netzwerk-Protokolle.

   Die in der Spielfläche eingesetzte Software muss in der Lage sein, eine gezielte Kommunikation zwischen benachbarten Spielfeldern, zwischen Roboter und Spielfeld, Roboter und Roboter und zwischen Host-Computer und Roboter zu unterstützen. Die Software muss weiterhin fähig sein, sich selbständig auf Veränderungen der Spielfläche durch individuelles Zusammenstecken von Spielfeld-Blöcken vor Spielbeginn einzustellen.

3. Implementierung einer Steuerungssoftware zur Kommunikation mit Robotern und Spielfläche von einem Desktop-Computer aus (Host-Interface).

   Mit Hilfe dieser Software sollen die Roboter den Spielregen entsprechend gesteuert werden können.

4. Demonstration der Funktionsfähigkeit der erstellten R_3D_3 Hard- und Software im Spielbetrieb.

1.3 Gliederung des Dokumentes

Die Gliederung orientiert sich an dem Aufbau des Projektes, da dies am sinnvollsten erscheint. Einführend wird zuvor das Spiel Robo Rally in Kapitel 2 vorgestellt und die vom R_3D_3 Team angestrebte Umsetzung näher erläutert, denn der Realisierungsform entsprechend war eine sinnvolle Auswahl an Zusatzfeatures zu treffen.

Die Grundidee und das Konzept der Umsetzung werden in Kapitel 3 skizziert. In den darauf folgenden Kapiteln werden schließlich die einzelnen Hauptkomponenten behandelt.

Zuerst werden in Kapitel 4 die Spielfelder behandelt. Dies umfasst neben einem Überblick der Konstruktion des letzten Semesters die aus Testläufen entstandenen Konstruktionsänderungen und Softwareentwicklung, die die Hauptintelligenz des Spieles ist. Da die Felder auch in der Kommunikation eine entscheidende Rolle spielen, wird in diesem

Zusammenhang im Abschnitt 4.3 auf die Kommunikation zwischen allen Komponenten eingegangen.

In Kapitel 5 werden nach einer Analyse des Roboterentwurfes aus [ZWB06] der Aufbau des neuen Roboters und die entsprechenden Software-Änderungen beschrieben. Es schließt mit einer Zusammenfassung und Bewertung des Roboter-Ansatzes.

Anschließend widmet sich Kapitel 6 dem Host, der die Schnittstelle zwischen Spieler und dem Spiel darstellt. Dabei wird im besonderen auf seine Bedeutung zum Spielablauf eingegangen.

Dieser wird dann in Kapitel 7 exemplarisch an der grafischen Benutzerschnittstelle vorgeführt. Dabei werden die einzelnen Spielphasen erklärt und ihre Bedeutung für den internen Spielablauf dargestellt.

Abschließend finden sich in Kapitel 8 eine Zusammenfassung nebst Ausblick. Eine Skizze möglicher Erweiterungen beschließt diesen Endbericht.

2 Das Spiel

2.1 Das Spiel im Original

In diesem Kapitel soll auf das Spiel „Robo Rally“, seine Geschichte und seine Regeln eingegangen werden, das als Ausgangsbasis für diese Projektgruppe dient. Die Regeln der deutschen Erstausführung sind der Anleitung zu dem Spiel von 1999 [AMI99] und der Erweiterung „Crash and Burn“ [AMI00] entnommen.

2.1.1 Über das Spiel

Das Spiel „Robo Rally“, nach der Idee von Richard Garfield, kann bereits auf eine lange Geschichte zurückblicken. Die erste Veröffentlichung im Jahre 1994 und die letzte im Jahre 2005 zeugen davon, dass seine Idee bis heute lebendig geblieben ist.

In den Vereinigten Staaten von Amerika erschien „Robo Rally“ im Jahre 1994 im Verlag Wizards of the Coast, und 1999 kam es dann auch nach Deutschland, vertrieben von der Amigo Spiel und Freizeit GmbH. Bald gab es Erweiterungen für das Basisspiel. Die Erste, „Crash and Burn“ genannt, wurde in den USA 1997 und im Jahr 2000 dann auch in Deutschland veröffentlicht, allerdings in einer etwas veränderten Variante. Kurz darauf waren dann in englischer Sprache die Spielsätze „Armed and Dangerous“, „Grand Prix“ und „Radioactive“ erhältlich. Da das Spiel in Deutschland nicht den gewünschten Absatz hatte, gab es die Sätze ausschließlich für den amerikanischen und englischen Markt. Da die gesamte Produktion des Spiels irgendwann eingestellt wurde, ist es heute schwerer denn je, noch Spiele der alten Generation zu bekommen. Im Jahr 2005 brachte Wizards of the Coast eine Neuauflage des Spiels „Robo Rally“ heraus. Die Version 2005 hat ein neues Design und bringt ein paar Veränderungen mit sich. Allerdings ist bisher nur eine englische Version erhältlich.

Das Spiel, inklusive aller Erweiterungen, ist für bis zu 8 Spieler ab einem Alter von 12 Jahren geeignet. Die Spieldauer beträgt laut Angaben der Hersteller mindestens 60 Minuten. Erfahrungen zeigen, dass ein Spiel je nach Aufbau der Spielpläne einige Stunden dauern kann.

2.1.2 Idee des Spiels

Bei „Robo Rally“ machen Roboter auf einem Fabrikgelände ein Wettrennen. Jeder Spieler steuert einen Roboter und versucht, diesen als Ersten über die Rennstrecke zu bringen.

Diese wird durch Fabrikelemente, wie z.B. Wände, Laser und Förderbänder und auch durch die Roboter selbst, beeinflusst. Durch Optionskarten, die in den Erweiterungen enthalten sind, können Roboter besondere Eigenschaften erhalten. Da die Roboter sich nahezu zeitgleich bewegen, wird so manches lustiges Chaos verursacht. Wer als Erster alle Checkpoints in der korrekten Reihenfolge passiert hat und das Ziel erreicht, ist der Gewinner des Spiels.

2.1.3 Vorbereitung

Als Grundlage für die folgenden Beschreibungen dienen das deutsche Grundspiel von Amigo (1999) und die passende Erweiterung „Crash and Burn“. Auf die amerikanischen Varianten und insbesondere auf die Version 2005 wird in Abschnitt 2.1.8 eingegangen.

Spielplan

Zu Beginn wird der Spielplan ausgelegt, wobei die Spieler entscheiden können, wie viele und welche der vorhandenen Spielpläne sie benutzen möchten. Spielt man mit mehreren Plänen, so legt man diese zu einem großen Spielplan zusammen, so dass sie genau Seite an Seite aneinander passen. Die Spieler können frei wählen, wie sie diese Pläne zusammenlegen.

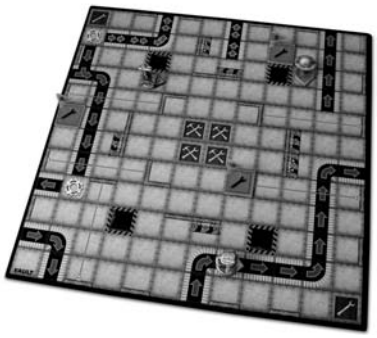


Abbildung 2.1: Ein Spielplan der Version 2005

Rennstrecke festlegen

Als nächstes legt man die Rennstrecke mit Hilfe von Checkpoints, das sind Teilziele, die von den Robotern erreicht werden müssen, fest. Sie sind mit Nummern versehen und müssen in aufsteigender Reihenfolge passiert werden. Beim Checkpoint 1 wird gestartet, Ziel ist der Checkpoint mit der höchsten Nummer. Wie viele Checkpoints verwendet werden und auf welche Felder diese auf dem Spielplan gelegt werden, entscheiden die Spieler. Mögliche Checkpoint-Felder sind jedoch nur leere Felder, Felder mit einer Mauer,

Felder mit zwei Mauern, Reparaturfelder mit zwei Schraubenschlüsseln ohne Mauer, Reparaturfelder mit zwei Schraubenschlüsseln und zwei Mauern, Reparaturfelder mit einem Schraubenschlüssel ohne Mauer, Reparaturfelder mit einem Schraubenschlüssel und einer Mauer.

Material für den Spieler

Sind Spielplan und Rennstrecke festgelegt, werden Programmierkarten, Schadenspunkte, Optionskarten, und für jeden Spieler ein Roboter (wie in Abb. 2.2), eine Programmierhilfe, eine Sicherheitskopie, Lebenspunkte und eine Übersichtstafel bereitgelegt. Programmierkarten sollen die Programme für die Roboter festlegen. Da Roboter während des Spiels beschädigt werden können, gibt es Schadenspunkte, die dies anzeigen. Der Roboter selbst ist eine Plastikfigur, bei der durch einen farbigen Pfeil die Vorder- und Rückseite kenntlich gemacht wird. (In der amerikanischen Urversion waren die Roboter Zinnfiguren, die selbst bemalt werden konnten.) Die Programmierhilfe ist ein Pappkärtchen in Form dieses farbigen Pfeils, auf dem der jeweilige Roboter abgebildet ist. Diese soll gerade Spielanfängern das Programmieren des Roboters erleichtern. Jeder Spieler bekommt außerdem eine Sicherheitskopie seines Roboters, ein quadratisches Pappkärtchen in der Farbe und mit dem Bild des jeweiligen Roboters, der von dem Spieler bewegt wird. Diese wird während des Spiels auf Checkpoints oder Reparaturfelder gelegt, so dass, wenn der Roboter zerstört wird, nach Abgabe eines Lebenspunktes, ein Duplikat des Roboters ab dem Standort der Sicherheitskopie das Spiel fortsetzen kann. Auf der Übersichtstafel werden alle Fabrikelemente und der Programmablauf erläutert. Die Lebenspunkte werden zu Beginn wie folgt verteilt: Wenn zwei bis vier Spieler auf ein bis vier Plänen spielen, erhält jeder Spieler drei Lebenspunkte. Jeder Spieler erhält einen weiteren Lebenspunkt, wenn mit fünf oder mehr Spielern gespielt wird, und einen weiteren Lebenspunkt, wenn mit fünf oder mehr Spielplänen gespielt wird. Es werden also bis zu fünf Lebenspunkte an alle Spieler ausgegeben. In der Erweiterung sind 10 Fabrikelemente als Plättchen mitgeliefert, die auch noch auf leeren Feldern des Spielfeldes verteilt werden können.

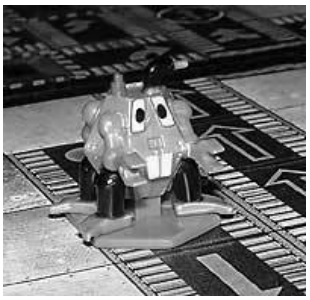


Abbildung 2.2: Ein Roboter der deutschen Spieleversion

2.1.4 Spielablauf

„RoboRally“ wird in Runden gespielt. Dabei besteht jede Runde aus vier Schritten: Dem Austeilen der Programmierkarten, dem Programmieren der Roboter, dem Programmablauf und dem Abschluss der Runde.

Austeilen der Programmierkarten

Alle 84 Karten werden gemischt, und jeder Spieler erhält verdeckt neun Karten auf die Hand. Die Karten zeigen unterschiedliche Befehle, durch die die Roboter gesteuert werden können: Die Vorwärtsbewegung um ein, zwei oder drei Felder, die Rückwärtsbewegung um ein Feld, die Vierteldrehung nach rechts oder links und die halbe Drehung. Erhält ein Spieler in der ersten Runde nur Drehbefehle, so erhält er neun neue Karten. Ab der zweiten Runde verteilt der Kartengeber nur so viele Karten an die Spieler, wie der Schaden ihres Roboters es zulässt.

Auf jeder Programmierkarte steht eine Programmnummer. Diese gibt die Reihenfolge an, in der die Spieler ihre Roboter bewegen dürfen. Ein Roboter, dessen Programmkarte die Nummer 840 hat, darf sich vor einem Roboter mit der Programmkartennummer 260 bewegen (vgl. Abb. 2.3).

Abbildung 2.3: Die Programmierkarten der Version 2005

Programmieren der Roboter

Jeder Spieler muss, nachdem die Karten alle verteilt worden sind, ein Programm für seinen Roboter zusammenstellen, das den Roboter steuern soll. Er wählt aus seinen neun Karten fünf aus und legt sie verdeckt so vor sich hin, dass die erste Karte des Programms links und die letzte rechts liegt. Die erste Karte stellt den ersten Befehl dar, und die letzte den fünften Befehl.

Nachdem ein Spieler seine Programmierung abgeschlossen hat, legt er die restlichen Karten auf den Stapel der nicht ausgeteilten Karten zurück und seine Programmierhilfe auf den ersten Befehl. Danach darf das Programm nicht mehr verändert werden. In der ersten Runde müssen die ersten Befehle aller Spieler eine Vorwärts- oder Rückwärtsbewegung sein.

Ablauf des Programms

Der Programmablauf sieht vor, dass alle Spieler ihre erste Karte aufdecken, derjenige Roboter, dessen Programmnummer die höchste ist, beginnen darf, und dann alle anderen in absteigender Reihenfolge ihren Befehl ausführen dürfen. Danach treten alle Fabrikelemente in Aktion. Erst dann drehen alle ihre zweite Programmkarte um, und verfahren wie im ersten Schritt. Nachdem alle Fabrikelemente nach Ausführung der letzten Befehle in Aktion getreten sind, werden alle Programmkarten neu gemischt und neu verteilt.

Befehle ausführen Beim Ausführen der Befehle können folgende Ereignisse eintreten: Ein Roboter schiebt einen anderen Roboter, stößt gegen eine Wand, fällt in eine Grube, bewegt sich vom Spielplan, bewegt sich auf einer Öllache, bewegt sich auf ein Portal, bewegt sich in einen Brenner hinein oder bewegt sich durch eine Schwingtür:

- Da immer nur ein Roboter auf einem Feld stehen darf, schiebt ein Roboter, der auf ein Feld ziehen will, das bereits von einem anderen Roboter besetzt ist, diesen vor sich her, unabhängig von der Richtung, in die der andere Roboter gerade zeigt.
- Stößt ein Roboter gegen eine Wand, so wird seine Bewegung blockiert und er kommt vor der Wand zum Stehen. Selbst wenn der Befehl, den der Roboter gerade ausführt, ihn ein Feld weiter ziehen lassen würde, wird sein Zug an der Stelle abgebrochen und er bleibt stehen. Das gilt ebenso für Rückwärtsbewegungen. Dies gilt auch beim Verschieben eines anderen Roboters.
- Fällt ein Roboter in eine Grube, so wird dieser vollständig zerstört. Ein Roboter kann nicht über eine Grube hinweg ziehen.
- Wenn sich ein Roboter über den Spielfeldrand hinaus bewegt, an dem kein anderes Spielfeldbrett anliegt, dann wird dieser ebenfalls vollständig zerstört.
- Eine Öllache verursacht keinen Schaden. Wenn ein Roboter seine Bewegung auf einer Öllache beendet, so rutscht er solange in seiner Bewegungsrichtung weiter, bis er auf einem Feld ohne Öl steht oder von einer Wand oder einem anderen Roboter blockiert wird. Wenn sich zwei Roboter auf derselben Öllache befinden, so kann ein Roboter einen anderen schieben. Sie rutschen dann so lange, bis der vordere Roboter auf einem Feld ohne Öl zum Stehen kommt. Wichtig ist, dass ein Roboter erst dann rutscht, wenn er seine Bewegung beendet, nicht solange er noch in Bewegung ist. Startet ein Roboter seine Bewegung auf einem Feld mit einer Öllache, so bewegt er sich ein Feld weniger weit als auf der Programmierkarte angegeben. Wenn der Befehl eine Vorwärtsbewegung um ein Feld vorsieht, so bleibt der Roboter auf der Stelle stehen. Drehbefehle werden von Öllachen nicht beeinflusst.
- Bewegt sich ein Roboter auf ein Portal, so wird der Roboter zu dem gleichfarbigen anderen Portal transportiert. Sollte dieses jedoch besetzt sein, so ist das Portalfeld ausgeschaltet und wird als ein Feld ohne Funktion behandelt. Der Roboter zieht dann darüber hinweg oder bleibt darauf stehen.

- Ein Brenner kann einem Roboter einen Schaden zufügen sowohl dann, wenn dieser sich auf ein Feld mit einem aktiven Brenner bewegt, als auch wenn er sich über ein solches Feld hinweg bewegt. Die Programmschritte, in denen der Brenner aktiv ist, sind auf dem Feld durch Zahlen angegeben.
- Auch Schwingtüren sind nur in bestimmten Programmschritten aktiv, die durch Zahlen angezeigt werden. Ist eine Schwingtür aktiv, so kann sich ein Roboter durch sie hindurch bewegen. Ist sie nicht aktiv, so verhält sie sich wie eine Wand, die Bewegung des Roboters wird dann an dieser Stelle unterbrochen.

Fabrikelemente treten in Aktion Es gibt folgende Fabrikelemente, die in dieser Phase in Aktion treten: Expressbänder, Förderbänder, Schieber, Zahnräder, Pressen, Laser, Brenner, Checkpoints und Reparaturfelder. Wenn die Fabrikelemente in Aktion treten, dann ist diese Reihenfolge einzuhalten:

- Expressbänder transportieren Roboter ein Feld weiter
- Expressbänder transportieren Roboter ein zweites Feld weiter; Förderbänder transportieren Roboter ein Feld weiter
- Schieber, die während des Programmschrittes aktiv sind, schieben Roboter auf ein benachbartes Feld
- Zahnräder drehen Roboter um 90° nach links oder rechts
- Pressen, die während des Programmschrittes aktiv sind, zerstören Roboter
- Laser fügen Roboter Schaden zu
- Brenner, die während eines Programmschrittes aktiv sind, fügen Robotern Schäden zu
- Checkpoints und Reparaturfelder erhalten die Sicherheitskopie eines Roboters, wenn dieser darüber hinweg gezogen ist oder darauf zum Stehen gekommen ist.

Expressbänder transportieren Roboter als erstes ein Feld in Pfeilrichtung, im zweiten Schritt dann erneut ein Feld weiter, wenn der Roboter noch immer auf einem Expressband steht. Gleichzeitig mit dem zweiten Schritt werden Roboter von Förderbändern ein Feld weiter transportiert. Auf den oben erwähnten Fabrikelementen können Ziffern abgebildet sein, die kenntlich machen, in welchem Programmschritt sie aktiv sind. Steht ein Roboter zum Beispiel im zweiten Programmschritt auf einer Presse, die nur im ersten, dritten und fünften Schritt aktiv ist, so passiert ihm nichts!

Abschluss der Runde

Wenn alle Programmschritte ausgeführt worden sind, dann findet der Abschluss der Runde statt. Roboter können ihre Schäden reparieren lassen, wenn sie auf einem Checkpoint

oder einem Reparaturfeld stehen. Auf Reparaturfeldern mit einem Schraubenschlüssel und auf einem Checkpoint kann ein Schadenspunkt abgegeben werden. Ein Spieler kann bis zu zwei Schadenspunkte abgeben, wenn er auf einem Reparaturfeld mit zwei Schraubenschlüsseln steht. Wenn zum Beispiel Karten durch Schadenspunkte blockiert sind, so kann er sich entscheiden, welche der blockierten Karten er wieder frei geben will. Steht der Roboter eines Spielers am Ende der Runde auf einem Reparaturfeld mit zwei Schraubenschlüsseln, so kann er wählen, ob er einen Schadenspunkt abgeben oder eine Optionskarte erhalten will. Dabei ist zu beachten, dass Spieler nur maximal drei Optionskarten besitzen dürfen. Erhalten sie eine vierte Karte, so müssen sie sich sofort entscheiden, welche der Karten sie zurückgeben. Wird ein Roboter zerstört, so muss der Besitzer eine Optionskarte abgeben.

Des Weiteren haben Roboter, die nun auf einem Checkpoint stehen oder während der Runde an einem vorüber gezogen sind, ein Teilziel der Robo Rally erreicht.

Ist ein Roboter während der Runde zerstört worden, so kann der Spieler durch Abgabe eines Lebenspunktes bewirken, dass er von dem Feld, auf dem seine Sicherheitskopie liegt, wieder mit einem Roboterduplikat ins Rennen gehen darf.

2.1.5 Ende des Spiels

Das Spiel „Robo Rally“ ist beendet, wenn ein Roboter, egal in welchem Programmschritt, den letzten Checkpoint überquert oder auf ihm zu stehen kommt. Falls alle Roboter zerstört werden, bevor der letzte Checkpoint erreicht werden konnte, so gewinnt der Spieler, der mit seinem Roboter die meisten Teilziele erreicht hat.

2.1.6 Varianten

Die Roboter besitzen Laser

Im Grundspiel sind noch einige Varianten vorgesehen. Zum Beispiel können Roboter an ihrer Vorderseite Laser besitzen. Diese schießen im Spiel zeitgleich mit den Lasern der Spielpläne. Auch hier können nur blockierende Wände und andere Roboter verhindern, dass man einen Schaden durch den Laser eines anderen Roboters erhalten kann.

Das Abschalten eines Roboters

Ein Spieler kann seinen Roboter abschalten und damit alle seine Schadenspunkte abgeben. Dies muss er allerdings eine Runde vorher ankündigen, und das Programm der aktuellen Runde wird auf jeden Fall vollständig durchgeführt. Der Roboter wird dann zu Beginn der neuen Runde abgeschaltet, das heißt er ist dann handlungsunfähig, kann aber weiterhin von Fabrikelementen und Robotern beeinflusst werden, sich also auch wieder

Schaden zuziehen. Bevor die Karten der nächsten Runde verteilt werden, kann der Spieler entscheiden, ob er seinen Roboter abgeschaltet lässt. Dann darf er wieder sämtliche Schadenspunkte abgeben.

Das Roboterduplikat ist beschädigt

Kommt der Roboter als Duplikat erneut ins Rennen, so erhält er zwei Schadenspunkte, weil ein Duplikat ja nie so gut sein kann wie das Original. Um mit der Erweiterung das Grundspiel sinnvoll ergänzen zu können, sollten alle diese Varianten mit ins Spiel einbezogen werden.

2.1.7 Optionskarten

Optionskarten nehmen eine Sonderrolle im Spiel ein und wurden erst mit der Erweiterung „Crash and Burn“ eingeführt. Verschiedene Optionskarten kann man in verschiedenen Situationen des Spiels einsetzen. Der Spieler kann seinem Roboter mit Hilfe dieser Karten weitere Eigenschaften geben. Beim Abschluss jeder Runde kann er sich, wenn sein Roboter auf einem Reparaturfeld mit zwei Schraubenschlüsseln steht, entscheiden, ob er zwei Schadenspunkte abgeben möchte, oder ob er eine Optionskarte vom verdeckten Stapel nehmen will. Diese muss er, wenn er sie vom Stapel aufnimmt, laut vorlesen und offen vor sich hinlegen. Der Spieler darf höchstens drei Karten besitzen, zieht er eine vierte, so muss er sich entscheiden, welche er ablegen will. Wird ein Roboter zerstört, so muss der Spieler eine Optionskarte abgeben.

Optionskartenbeeinflussung beim Austeilen der Programmierkarten

Hier können zwei Karten Einfluss ausüben:
„Zusätzlicher Speicherplatz“: Zu Beginn jeder Runde bekommt der Spieler eine zusätzliche Programmierkarte.
„Ersatz-Programm“: Einmal pro Runde darf der Spieler alle Karten, die ihm ausgeteilt wurden, zurückgeben, bevor er seinen Roboter programmiert hat, und erhält die gleiche Anzahl Karten neu vom Stapel. Dafür bekommt der Spieler dann einen Schadenspunkt.

Optionskartenbeeinflussung beim Programmieren der Roboter

„Befehl in Reserve“: Nachdem ein Spieler seine fünf Befehle programmiert hat, kann er eine sechste Karte verdeckt auf dieser Optionskarte ablegen. Diese kann er mit einer programmierten Karte austauschen, wenn diese noch nicht aufgedeckt wurde.

„Schwungrad“: Nachdem der Spieler fünf seiner Karten zur Programmierung verwendet hat, darf er eine sechste zur Seite legen und dann in der nächsten Runde zusätzlich verwenden.

Optionskartenbeeinflussung während des Programmablaufs - Befehle ausführen

„Notschalter“: Anstatt eine Programmierkarte aufzudecken, kann man diese mit dieser Optionskarte durch die oberste Karte des Stapels ersetzen. Danach müssen jedoch alle anderen Karten auch durch Karten vom Stapel ersetzt werden. Nach dem Abschluss der Runde wird die Funktion deaktiviert.

„Schutzschirm“: Der Schutzschirm kann den Roboter des Besitzers dieser Optionskarte vor Schaden aus einer bestimmten Richtung bewahren. Dieser Schutzschirm ist in jedem Programmschritt aktiv, und man muss sich vorher entscheiden, aus welcher Richtung man geschützt werden möchte. Dies macht man dadurch deutlich, dass man einen umgedrehten Schadenspunkt auf den Rand der Karte oben, unten, rechts oder links legt, je nachdem welche Seite geschützt werden soll. Falls die Richtung nicht auf diese Weise gekennzeichnet sein sollte, ist der Schutzschirm nicht aktiv.

„Drehscheibe“: Die Drehscheibe erlaubt einem Spieler, den Laser seines Roboters in die Richtung zu benutzen, die er vor einem Programmschritt festgelegt hat. Das gilt auch für die Benutzung des Turbo-Blasters, des Hochdruckstrahls, der Fernsteuerung, des Zufallsgenerators und des Traktorstrahls. Auch hier muss wieder mit Hilfe eines umgedrehten Schadenspunkts rechts, links oder unten angezeigt werden in welche Richtung sich die Drehscheibe bewegen soll. Liegt kein Punkt auf der Optionskarte, so schießt der Roboter nach vorn.

Auswirkungen erst nach dem Aufdecken der Programmierkarten

„Bremsen“: Wenn der Spieler den Befehl „Vorwärtsbewegung: 1 Feld“ programmiert hat, dann kann er mit dieser Optionskarte entscheiden, ob er sich ein Feld vorwärts bewegt, oder ob er stehen bleibt.

„Vierter Gang“: Bei dem Befehl „Vorwärtsbewegung: 3 Felder“ kann der Spieler wählen, ob er drei oder vier Felder vorwärts gehen möchte.

„Rückwärtsbeschleunigung“: Mit dieser Karte kann der Spieler wählen, ob er seinen Roboter ein oder zwei Felder rückwärts zieht, wenn er den Befehl „Rückwärtsbewegung“ programmiert hat.

„Mechanischer Arm“: Ist ein Programmschritt beendet, so kann der Roboter, wenn er auf einem der vier angrenzenden Feldern eines Checkpoints steht, den mechanischen Arm benutzen, um den Checkpoint zu berühren und damit ein Teilziel oder das Endziel der Robo Rally zu erreichen. Dabei sind diagonale Felder keine angrenzenden Felder. Der Arm kann durch eine Wand, aber nicht durch einen Roboter blockiert werden. Reparaturen oder die Aufnahme der Optionskarten können durch den mechanischen Arm nicht bewirkt werden.

„Rammbock“: Der Rammbock ist immer aktiv und fügt, wenn der Roboter einen anderen schiebt, diesem einen Schaden zu. Das tut er auch dann, wenn sich der Roboter nicht schieben lässt, weil er beispielsweise vor einer Wand steht.

Optionskartenbeeinflussung während des Programmablaufs - Fabrikelemente treten in Aktion

Mit den folgenden Karten werden die Auswirkungen, die bestimmte Fabrikelemente oder die Roboterlaser haben, verändert.

Expressbänder, Förderbänder und Zahnräder:

Im Folgenden die Optionskarten, die Auswirkungen auf Expressbänder, Förderbänder und Zahnräder haben.

„Kreiselstabilisator“: Wann immer der Besitzer dieser Optionskarte möchte, wird sein Roboter auf Express-, Förderbändern und auf Zahnrädern nicht gedreht (2.4).

Abbildung 2.4: Die englische Optionskarte Kreiselstabilisator

Veränderungen am Laser der Roboter:

Mit diesen Optionskarten kann man die Laser der Roboter verändern:

„Doppellaser“: Der Roboter kann mit dem Doppellaser einem anderen Roboter zwei Schäden zufügen. Kombinationsmöglichkeit besteht mit der Karte Zielkontrolle und/oder Hochleistungslaser.

„Zielkontrolle“: Der Spieler muss bestimmen, ob er dem Besitzer des Roboters, den er beschädigt, eine Programmierkarte blockiert, oder ob er eine seiner Optionskarten zerstört. Diese muss der Spieler dann sofort auf den Ablagestapel zurücklegen. Kombinationsmöglichkeit besteht mit der Karte Doppellaser und/oder Hochleistungslaser.

„Hochleistungslaser“: Mit Hilfe dieser Optionskarte kann der Roboter durch eine Wand und durch einen Roboter schießen, um sein endgültiges Ziel zu treffen. Der Roboter, durch den der Laser schießt, wird dabei auch beschädigt. Kombinationsmöglichkeit besteht mit der Karte Zielkontrolle und/oder Doppellaser.

„Hecklaser“: Mit diesem verfügt ein Roboter dann über zwei Laser, einem nach vorne und einem nach hinten gerichteten. Mit beiden kann er anderen Robotern Schaden zufügen. Die Laser schießen gleichzeitig.

Anstelle der Laser einzusetzen:

Die folgenden Karten dürfen nur anstatt des Lasers des jeweiligen Roboters verwendet werden. Diese Eigenschaften ersetzen die Laser.

„Turbo-Blaster“: Wenn ein Roboter diesen benutzt, dann wird dem getroffenen Roboter ein Schaden zugefügt, und er wird ein Feld in Schussrichtung weiter geschoben. Wer die Karte erhält, bekommt gleichzeitig fünf Schadenspunkte ausgehändigt, die er auf diese Optionskarte legt. Jedes Mal, wenn er einen Roboter beschädigt, händigt er seinem Besitzer einen Schadenspunkt aus. Die Karte muss nach fünfmaliger Benutzung zurückgegeben werden, wenn alle Punkte abgegeben sind.

„Hochdruckstrahl“: Dieser schiebt einen anderen Roboter ein Feld in Richtung des Strahls, richtet aber keinen Schaden an.

„Fernsteuerung“: Mit dieser Karte wird es einem Spieler möglich, das Programm, mit dem sein eigener Roboter programmiert ist, auch auf einen anderen zu übertragen. Steht ein Roboter maximal sechs Felder in gerader Richtung vor dem eigenen Roboter, so kann er diesen fernsteuern. Wände und Roboter behindern die Verbindung nicht. Ab diesem Zeitpunkt bis zum Ende der Runde führen die beiden Roboter die identischen Befehle aus, wobei der steuernde Roboter seine Bewegungen immer als Erster durchführt.

„Zufallsgenerator“: Der nächste Befehl des Roboters, der von einem Laserstrahl getroffen worden wäre, wird mit Hilfe dieser Karte durch den Befehl der obersten Karte auf dem Stapel ausgetauscht. Im fünften Programmschritt ist der Zufallsgenerator nicht einsetzbar.

„Traktorstrahl“: Mit dem Traktorstrahl kann ein Spieler einen Roboter in die Richtung seines Roboters ziehen. Der Roboter wird hierbei nicht beschädigt. Steht dieser Roboter jedoch auf einem der vier benachbarten Felder, so darf die Optionskarte nicht benutzt werden (2.5).

Abbildung 2.5: Die englische Optionskarte Traktorstrahl

Schaden:

Wenn ein Spieler für seinen Roboter einen Schadenspunkt erhalten soll, so kann er stattdessen einfach eine Optionskarte, die sich in seinem Besitz befindet, wieder zurückgeben. Mit den folgenden beiden Optionskarten kann der Schaden eines Roboters verhindert werden.

„Schutzhaut“: Diese kann verhindern, dass der Roboter Schadenspunkte erhält. Wird der Roboter des Besitzers dieser Karte beschädigt, so werden die Schadenspunkte, die er erhält, auf diese Optionskarte gelegt. Liegen auf dieser drei Schadenspunkte, so muss die Karte abgegeben werden.

„Schutzschilde“: Wenn ein Spieler seinen Roboter abgeschaltet hat, so ist er auf allen vier Seiten von je einem Schutzschild gesichert. Wird der Roboter wieder eingeschaltet, so verschwinden die Schutzschilde wieder.

Optionskartenbeeinflussung beim Abschluss der Runde:

Beim Einsetzen eines Roboterduplikats:
Setzt ein Spieler ein Duplikat seines Roboters ein, erhält er sofort zwei Schadenspunkte, es sein denn er besitzt die Optionskarte: **„Duplikat de Luxe“:** Der Spieler bekommt eine De-luxe-Ausführung seines Roboters, so dass er keine Schadenspunkte erhält.

Beim Abschalten eines Roboters:
„Stromkreisunterbrecher“: Besitzt ein Spieler diese Optionskarte, so muss er, wenn sein Roboter am Ende einer Runde drei oder mehr Schadenspunkte hat, seinen Roboter sofort für die folgende Runde abschalten. Die Ankündigung der Abschaltung entfällt hierbei.

2.1.8 Robo Rally 2005

In diesem Abschnitt sollen die Neuerungen im Vergleich zu den vorangegangenen deutschen Spieleversionen kurz erläutert werden.
Im Jahr 2005 erschien eine Neuauflage des Spiels Robo Rally bei Wizards of the Coast [Wiz05] erschienen. Allerdings ist diese Auflage nur in englischer Sprache erhältlich. Das Konzept ist jedoch dasselbe geblieben. Die Einzelheiten, die sich geändert haben, dienten uns zum Teil als Gedankenanstoß für die Realisierung des Spiels R_3D_3. Hier kurz die Neuerungen:

Checkpoints: Checkpoints sind nun durch Fähnchen gekennzeichnet. Ein eindeutiger Vorteil, denn man kann diese nun besser erkennen.

Roboter: Roboter haben jetzt alle Namen: Hammer Bot, Hulx X90, Spin Bot, Squash Bot, Trundle Bot, Twitch, Twonky, Zoom Bot (vgl. Abb. 2.6). Für jeden Roboter gibt es ein eigenes Program Sheet.

Abbildung 2.6: Die Roboter der Version 2005

Program Sheet: Hier können nun alle Programmierkarten, Lebens- und Schadenspunkte abgelegt werden. Wie oben erwähnt, erhält jeder Spieler für seinen eigenen Roboter einen eigenen Bogen, auf dem der Name des Roboters vermerkt ist.

Docking Bay Board: Dies ist ein zusätzlicher zweiseitiger Startspielplan, der es den Robotern ermöglicht auf gesonderten Feldern zu starten (vgl. Abb. 2.7 und 2.8). Jeder Spieler stellt seinen Roboter in ein Dock. Der erste Startplatz wird zufällig vergeben, alle anderen Docks werden dann im Uhrzeigersinn zugeteilt. Die Reihenfolge ist später wichtig, zum Beispiel wenn es darum geht, ob die Spieler ihren Roboter ausschalten möchten. Wenn die Entscheidung eines Spielers seinen Roboter in der nächsten Runde abzuschalten davon abhängt, ob andere Spieler dies tun, kann hierzu jeder Spieler in der Reihenfolge der Docking-Startnummer gefragt werden, ob er seinen Roboter in der nächsten Runde abschalten möchte. Das Docking Board bietet den Vorteil, dass alle Roboter auf einem eigenen Feld starten können. Somit kann das Chaos zu Beginn, das in der deutschen Spieleversion immer entsteht, ein wenig entschärft werden, weil nicht alle Roboter auf demselben Feld starten müssen.

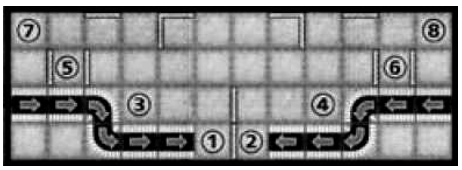


Abbildung 2.7: Docking Bay Board Seite 1

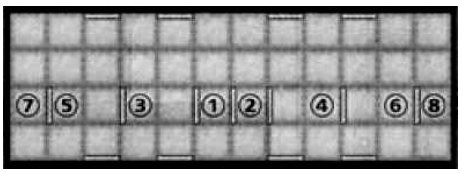


Abbildung 2.8: Docking Bay Board Seite 2

Timer: Wenn alle Spieler bis auf einen ihr Programm für die nächste Runde zusammengestellt haben, dann wird der Timer benutzt, der dann die Zeit angibt, die dem letzten Spieler noch verbleibt, um sein Programm fertig zu stellen. Gelingt ihm das nicht vor Ablauf der Zeit, so werden die restlichen Karten, im Extremfall sogar alle, verdeckt aus den Karten, die der Spieler erhalten hat, ausgewählt und die Register mit ihnen gefüllt.

Power Down Token: Jeder Spieler, der ankündigt seinen Roboter in der nächsten Runde ausschalten zu wollen, muss den Power Down Token auf sein Program Sheet legen.

Spielpläne und -varianten: Wizards of the Coast haben sich noch ein paar neue Spielmodi ausgedacht. So wird bei einer Version der Roboter „SuperBot“ gejagt, und von einem Spieler, der ihn endgültig zerstört hat, in Besitz genommen. Da er der einzige ist, der eine Checkpointfahne berühren darf, versuchen nun alle anderen, „SuperBot“ zu jagen und in ihren Besitz zu nehmen. Außerdem gibt es noch Teamvarianten, in denen Gruppen gegeneinander antreten können.

2.2 Unsere Realisierung des Spiels

In diesem Abschnitt werden die Teile der Spielregeln beschrieben, die in die Realisierung aufgenommen werden.

2.2.1 Verwendete Varianten

In Abschnitt 2.1.6 wurde erwähnt, dass es mehrere Varianten gibt, mit denen man „Robo Rally“ spielen kann. Von diesen wird die Variante „Roboter besitzen Laser“ in die Realisierung übernommen.

2.2.2 Fabrikelemente

Es werden alle Fabrikelemente aus dem originalen Spiel übernommen, außer dem Portal und der Schwingtür. Diese werden nur optional eingebunden, da deren Verwirklichung sehr komplex wäre.

2.2.3 Optionskarten

Bei einigen Optionskarten werden, da der Spielfluss nicht ständig unterbrochen werden soll, Entscheidungen über bestimmte Optionskarten bereits im Vorhinein der Runde von den Spielern festgelegt werden müssen. Dies wird bei der Optionskarte „Bremsen“, „Vierter Gang“ und „Rückwärtsbeschleunigung“ der Fall sein. Der Spieler legt vor Beginn der Runde fest, bei welcher Karte er gegebenenfalls bremsen, vier statt drei Schritte vorwärts gehen oder zwei statt einem Schritt rückwärts gehen möchte.

Um häufige Unterbrechungen des Spiels zu vermeiden, werden die Karten „Befehl in Reserve“, „Zielkontrolle“, „Notschalter“, „Schutzschirm“, „Drehscheibe“ von uns nicht übernommen. Die Spieler dürften bei den letzten drei Karten laut der „Robo Rally“ Regeln nach jedem Schritt entscheiden, ob sie diese im nächsten Schritt verwenden möchten, oder in welche Richtung sie die Karte verwenden möchten.

Optional werden die Karten „Turbo-Blaster“, „Krabbenbein“, „Dualer Prozessor“ eingebunden. Beim Turbo-Blaster wird dann die Benutzung auf 3 Runden beschränkt, da man diesen jeweils nur für eine ganze Runde einplanen kann.

2.2.4 Verwendete Neuheiten aus Robo Rally 2005

Die Version 2005 sieht ein Docking Bay Board als Startplatz für die Roboter vor. Diese Art Parkplatz wird auch in der Realisierung vorgesehen. Weitere Einzelheiten sind in Abschnitt 3.3.1 zu finden.

Des Weiteren wird die Idee der Fähnchen als Checkpoints übernommen. Diese sind auf einem solch großen Spielfeld, wie es konstruiert wird, viel besser zu erkennen. Detailliertere Ausführungen sind in Abschnitt 4.1.1.3 zu finden.

3 Konzept

Um den Anforderungen und Zielen aus Abschnitt 1.2 gerecht zu werden, wurde das folgend beschriebene Konzept erarbeitet.

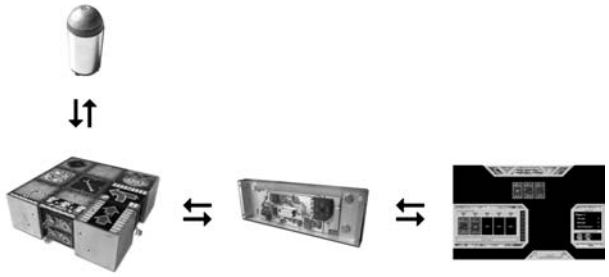


Abbildung 3.1: Das Konzept

Das Spiel besteht aus den folgenden Hauptkomponenten:

- Roboter
- Spielfeld
- Host-Schnittstelle
- Host

Diese werden nun - in umgekehrter Reihenfolge - kurz beschrieben, und angedeutet, wie sie die Kernanforderungen erfüllen.

3.1 Der Host

Als Benutzerschnittstelle dient eine Software auf einem PC oder einer anderen Art der Konsole. Jeder Spieler bedient einen Host, bzw. eine Konsole. Diese beiden Begriffe werden im folgenden synonym verwendet. Hier findet die Programmierung der Karten statt, jedoch hat der Host an sich keine weiteren Informationen über das Spielfeld oder die Position des Spielers. Einzig der Zustand seines Roboters ist ihm noch bekannt. Somit ist die Grundlage für ein dezentrales System gelegt.

Für einen korrekten Spielablauf ist jedoch eine zentrale Instanz nötig, die als eine Art Schiedsrichter fungiert. Ein Host wird dynamisch zu einem Master bestimmt. Dieser stößt die Befehlsverarbeitung an und bestimmt, wer welchen Befehl ausführen darf. Hierbei sind die Hauptaufgaben die Synchronisation und Konsistenzhaltung. Letzteres bedeutet, dass beispielsweise sichergestellt werden muss, dass alle Befehlskarten und Optionskarten im Spiel nur einmal existieren dürfen. Der Master teilt in jeder Runde die Karten aus und muss außerdem dafür sorgen, dass die Fabrikelemente parallel abgearbeitet werden. Die restlichen Host sind Clients und werden vom Master angestoßen, die Befehle auszuführen.

Die Aufgaben dieses Masters wurden auf ein Minimum beschränkt und die Zuteilung erfolgt dynamisch. Somit wird dieser Entwurf dem Begriff „dezentral“ immer noch gerecht.

Der Host wird über eine serielle Schnittstelle (RS-232) mit der Host-Schnittstelle verbunden.

3.2 Die Host-Schnittstelle

Die Host-Schnittstelle ist die physikalische Verbindung des Hosts zum eigentlichen Spiel. Sie ist so konzipiert, dass sie an jedem Spielfeld angedockt werden kann. In ihr werden die Befehle des Hosts umgewandelt. Technisch gesehen ist sie eine sehr stark vereinfachte Version eines Spielfelds und reicht die Befehle auf dieselbe Art und Weise weiter, wie es zwischen den anderen Feldern geschieht.

3.3 Die Spielfeldbox

Die Spielfeldboxen besitzen die Kernintelligenz des Spiels und führen das Spiel hauptsächlich aus. Jede Spielfeldbox ist ein in sich abgeschlossener Teilnehmer. Sie kennt die Besonderheiten seiner neun Felder und - nach einer anfänglichen Initialisierungsphase - seine Koordinaten in der Gesamtfläche und die seiner Nachbarn. Zudem weiß sie, auf welchen Feldern welcher Roboter steht. Jede Box hat ein Routing-Protokoll implementiert und kann somit Befehle für andere Felder weiterreichen oder initiieren. Da der Host durch die Host-Schnittstelle wie ein Spielfeld angeschlossen ist, funktioniert die Kommunikation auf dieselbe Weise.

Das jeweilige Spielfeld, auf dem der Roboter eines Hosts steht, sendet die Befehle an den Roboter weiter. Auf die selbe Weise teilt ein Spielfeld dem Host dann zum Beispiel neue Schäden mit. Wenn der Roboter eine Box verlässt, kommunizieren die beiden involvierten Boxen miteinander.

Dies geschieht über zwei Leitungen an der Box und ein eigenes Low-Level Protokoll.

3.3.1 Der Parkplatz

Da es eine Anforderung des Spieles ist, dass die Spielfeldblöcke frei zusammensteckbar sein sollen, ist es nötig eine Initialisierungsphase zu starten, in der die Koordinatenstruktur für das Spiel festgelegt wird. Die Phase ist notwendig, damit im Verlaufe des Spiels zwischen den Hosts und den Feldern Nachrichten gesendet werden können. Am Ende dieser Phase kennt dann jeder Host die Adressen der Felder und jedes Feld die Adressen der im Spiel befindlichen Hosts.

In der englischen Spielversion existiert ein gesonderter Startspielplan (vgl. [Wiz05]). Das bedeutet, dass am Anfang jedes Spiels jeder Roboter auf seinem ausgewiesenen Spielfeld stehen kann und so - anders gegenüber der deutschen Variante (vgl. [AMI99]) - von einem anderen Spielfeld startet. Ein Parkplatz soll diese Funktion übernehmen.

Unter den Spielfeldboxen nimmt der Parkplatz eine besondere Rolle ein. Er ist als zentrale Steuereinheit für die Initialisierung des Spielfeldes zuständig und hat weitere zusätzliche Aufgaben, die über die Funktionalität einer normalen Spielfeldbox hinausgehen.

Um allerdings Homogenität in Design von Hard- und Software bei allen Spielfeldboxen beizubehalten, wird der Parkplatz mit dem gleichen Controller ausgestattet werden, wie auch die übrigen Spielfeldblöcke, daher wird der Parkplatz im Folgenden nicht weiter einzeln betrachtet.

3.4 Der Roboter

Der Roboter ist zwar die Hauptfigur des Spiels, ist im Konzept jedoch nur ein Befehlsausführer. Er hat keine Kenntnis von seinem Zustand oder seiner Position. Dies ist nicht zuletzt deswegen günstig, da der Spielverlauf auch bei Ausfall eines Roboters nach Behebung ohne Informationsverlust weitergeführt werden kann.

Auch die Informationen, um welches Fabrikelement es sich jeweils bei dem Feld handelt, auf dem der Roboter steht, werden nur im Spielfeld gespeichert, da es zuviel Kommunikationsaufwand erfordern würde, wenn der Roboter die Daten erst beim Spielfeld anfordern müsste. Des weiteren spart man Kommunikationsaufwand und Energie im Roboter ein, so dass eine höhere Akkulaufzeit erzielt werden kann. Zudem hätte der Roboter keinen Nutzen von dieser Information, die nicht schon durch das Feld abgedeckt werden kann.

Die Kommunikation mit dem Roboter wird optisch realisiert und umfasst nicht nur den direkten Austausch an Informationen, sondern auch die Positionierung des Roboters durch Lichtmarkierungen, an denen sich dieser orientieren kann.

3.5 Spielablauf

Eine beliebige Zahl an Spielfeldboxen wird zusammengesteckt. Dies beinhaltet die entsprechende Anzahl an Host-Adaptern mit daran angeschlossenen Konsolen. Mit Einschalten des Stroms beginnt die durch den Parkplatz angeregte Initialisierung der Koordinaten. Dann wird auf dynamische Weise ein Host bestimmt, der Master wird. Nach weiteren Initialisierungen, die die Roboter einschließen, kann das Spiel beginnen. Dabei kommunizieren die Hosts die gewählten Befehle über die Adapter an die Spielfläche. Die Spielfeldboxen routen die Nachrichten zu den entsprechenden Boxen. Diese werten den Befehl aus und geben dem Roboter entsprechende Anweisungen. Als Teil dieses Prozesses werden auch die Fabrikelemente berücksichtigt und mit Unterstützung von Effekten der LEDs und akkustischer Untermalung abgearbeitet. Insofern Nachbarboxen an der Bewegung beteiligt sind, kommunizieren die Boxen mit diesen und übergeben schließlich die Kontrolle. Mit dem Wechsel eines Roboters auf eine andere Box wird insbesondere auch der Zustand des Roboters mitgereicht.

Eine detailiertere Darstellung des Spielverlaufs wird noch in Kapitel 7 gegeben werden. Doch schon aus dieser kurzen Beschreibung ist ersichtlich, dass die Anforderungen im Konzept erfüllt werden, insbesondere die folgenden drei:

- modulare Spielfelder (frei konfigurierbare Spielfeldboxen)
- aktive Spielfläche (Licht- und Soundeffekte)
- dezentrale Umsetzung (selbstorganisierendes Routing-Protokoll)

4 Spielfeld

In diesem Kapitel wird auf das Spielfeld von R3D3 eingegangen. Das gesamte Spielfeld besteht aus einzelnen Spielfeldboxen, die sich je aus mechanischen und elektronischen Komponenten zusammensetzen. Diese werden im Hardware Kapitel in den Abschnitten 4.1.1 und 4.1.2 näher beschrieben. Durchgeführte Tests haben gezeigt, dass in einigen Bereichen eine Überarbeitung der Hardware gemacht werden musste, welche in Abschnitt 4.2 erläutert werden.

Eine zentrale Aufgabe des Spielfeldes ist die Kommunikation mit dem Roboter. Da ein Großteil der nötigen Spielintelligenz in dem Spielfeld implementiert wurde, bekommt der Roboter nur kurze, einfache Nachrichten, wie z.B. Bewegungsbefehle, zugeschickt. In Abschnitt 4.3 wird dazu zunächst die Low-Level-Schicht des Kommunikations-Protokolls beschrieben.

Im letzten Abschnitt 4.4 wird auf die Spielfeldsoftware eingegangen, in der die Spielintelligenz implementiert wurde. Ihre Aufgabe ist zum Beispiel die Ausführung von Befehlen oder Fabrikelementen. Ähnlich wie in der Hardware, haben auch in der Software durchgeführte Tests Änderungen zur Folge gehabt, die in Abschnitt 4.5 näher erläutert werden.

4.1 Hardware

4.1.1 Mechanik

Dieser Abschnitt gibt einen Überblick über die Mechanik der Spielfeldboxen. Dabei wird zunächst der allgemeine Aufbau dieser beschrieben. Nachfolgend wird dann eine Übersicht über die einzelnen Elemente der Spielfeldboxen gegeben.

4.1.1.1 Allgemeiner Aufbau

Die Spielpläne des Brettspiels RoboRally setzen sich aus einzelnen Feldern zusammen. Diese einzelnen Felder gibt es auch in der R3D3 Realisierung. Hier werden 9 dieser Felder zu einer so genannten Spielfeldbox zusammengefasst. Eine Spielfeldbox besteht aus 3 x 3 Feldern. In diesem Teil des Endberichts werden hauptsächlich die Spielfeldboxen betrachtet und nicht einzelne Felder.

In dem PG-Antrag steht die Vorgabe, dass die einzelnen Spielfelder 5 *cm* x 5 *cm* groß sein sollen, dementsprechend wird jede Spielfeldbox 15 *cm* x 15 *cm* groß sein.

Nun zu den Anforderungen an die Spielfeldboxen. Diese Anforderungen sind bei der Planung und Erstellung der Spielfeldboxen berücksichtigt worden.

Eine wichtige Anforderung an die Boxen ist, dass alle Boxen auf beliebige Weise miteinander kombinierbar sein sollen. Das bedeutet, dass jede Box beliebig gedreht werden kann und trotzdem alle Boxen aneinanderpassen. Damit jede Box an jede andere Box gesteckt werden kann, müssen alle Seiten der Spielfeldboxen gleich aussehen, näheres hierzu in Abschnitt 4.1.1.2.

Über die Boxen müssen die Roboter fahren können. Deshalb haben die Spielfeldboxen eine Oberfläche, die dieses ermöglicht, näheres hierzu in Abschnitt 4.1.1.3.

Damit die Roboter "gerade" über die Boxen fahren können, werden Navigationshilfen benötigt. Dank dieser kann der Roboter exakt von einem zum nächsten Feld fahren. Als Navigationshilfen werden LEDs benutzt, näheres hierzu in Abschnitt 4.1.2 und Abschnitt 5.3.4.

Um alle diese Anforderungen erfüllen zu können, ist die Entscheidung getroffen worden, die Spielfeldboxen selbst herzustellen. Dazu ist eine Unterkonstruktion und eine Oberfläche hergestellt worden. Beides wird im nächsten Abschnitt beschrieben.

4.1.1.2 Unterkonstruktion

Die Unterkonstruktion besteht mechanisch gesehen aus einem Rahmen, in dem die benötigten elektronischen Bauteile untergebracht werden. In diesem Rahmen werden Magnete untergebracht, damit die Boxen später untereinander zusammenhalten. Außerdem werden für die Kommunikation und die Stromübertragung zwischen den Boxen Kontakte bereitgestellt.

Die Unterkonstruktion besteht aus Seitenflächen, die an 90 Grad Aluminiumwinkel geschraubt werden. Sowohl die Seitenflächen wie auch die Winkel werden nun genauer betrachtet:

Seitenflächen

Die Seitenflächen sind sowohl für die Stabilität als auch für die Befestigung der Kontakte zwischen den Boxen wichtig.

Diese Seitenflächen sind aus Epoxydharzplatinen hergestellt worden. Ein wesentlicher Vorteil von Platinen gegenüber anderen Materialien liegt darin, dass die Kontakte direkt auf die Platine gelötet werden können. Somit entfällt das Problem der Befestigung der Kontakte. Ein weiter Vorteil dieses Materials liegt darin, dass Platinen auch in einer sehr geringen Breite sehr stabil sind.

Damit die Platine sicher und bequem mit den Aluminiumwinkeln verschraubt werden kann, hat sie eine Größe von 4 *cm* x 10 *cm*.

Auf dieser Platine sind die Kontakte für die Kommunikation zwischen benachbarten Spielfeldboxen befestigt. Damit die Spielfeldbox beliebig gedreht werden kann und trotzdem mit jeder anderen Spielfeldbox zusammengesteckt werden kann, muss jede Seite der Spielfeldbox gleich aussehen. So können keine Stecker-Buchse Kontakte benutzt werden, sondern nur Batteriefederkontakte. Zur Realisierung der Daten- und Takt- Leitungen werden insgesamt 4 Kontakte benötigt. Die Entscheidung ist auf sehr stabile Batteriefedern gefallen. Diese erfüllen alle Anforderungen an die Kontakte. Es werden auf jede Seitenfläche 2 Federn angelötet. Eine Feder dient als Daten-Leitung und die andere als Takt-Leitung. Um den Kontakt zum anderen Spielfeld herzustellen, muss eine Unterlegscheibe als Gegenstück verwendet werden. Insgesamt sieht die Spielfeldbox wie in Abbildung 4.1 aus.

Winkel

Die Aluminiumwinkel befinden sich in den 4 Ecken der Spielfeldbox, wie in Abbildung 4.3 zu sehen. An den Winkeln werden die Seitenflächen, wie in Abbildung 4.2 zu sehen, befestigt. Dies geschieht durch jeweils 2 Gewinde pro Winkelfläche und sorgt so für die benötigte Stabilität der Unterkonstruktion.

In die Aluminiumwinkel werden Magnete von innen in das Aluminium versenkt. Es bleibt von außen gesehen eine 1mm dicke Schicht Aluminium. Wenn nun 2 Boxen zusammengesteckt werden, verhindert diese, dass zwei Magnete direkt aufeinander liegen. So wird sichergestellt, dass eine Gesamtkonstruktion aus mehreren Boxen zusammenhält, siehe Abbildung 4.4.

Die Größe der Winkel beträgt 5 *cm* x 5 *cm* bei einer Breite von 6 *mm*. Diese Höhe der Winkel ist dadurch begründet, dass in der Spielfeldbox die Elektronik untergebracht werden muss, siehe 4.1.2. Die Winkel sind 6 *mm* breit, damit genügend Platz für die versenkten Magnete vorhanden ist.

Die Winkel haben außer ihrer Funktion zur Stabilisierung der Spielfeldbox noch eine weitere wichtige Aufgabe: Sie müssen den Strom verteilen. Da alle Boxen von jeder

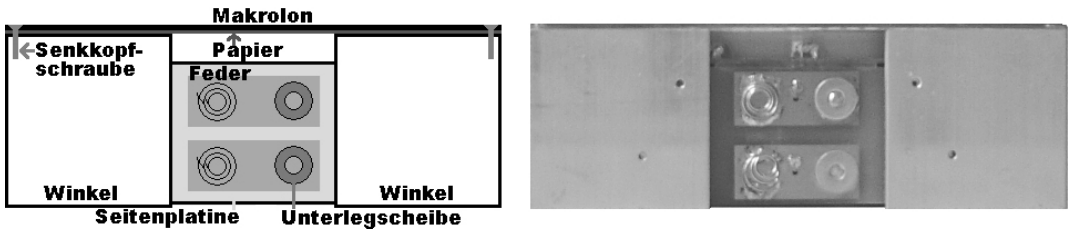


Abbildung 4.1: Seitenansicht einer Spielfeldbox

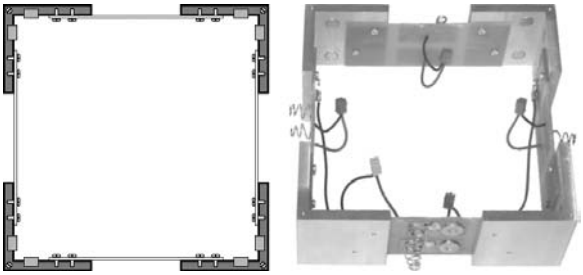


Abbildung 4.2: Aufsicht der Unterkonstruktion

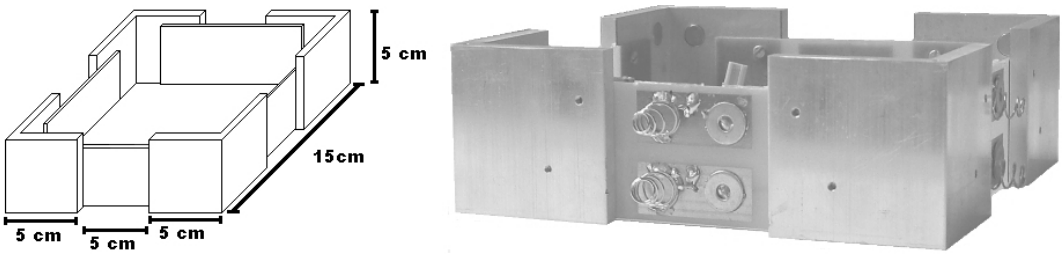


Abbildung 4.3: Ansicht der Unterkonstruktion

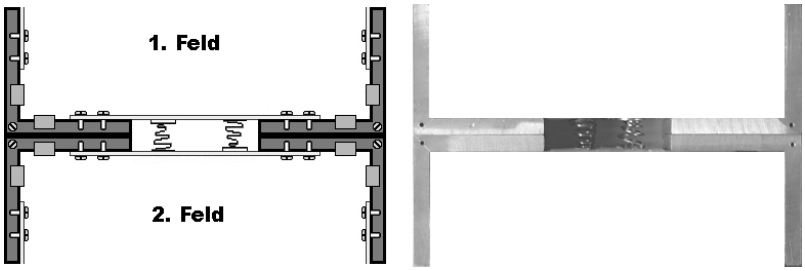


Abbildung 4.4: Zusammenhalten der Gesamtkonstruktion durch Magnete

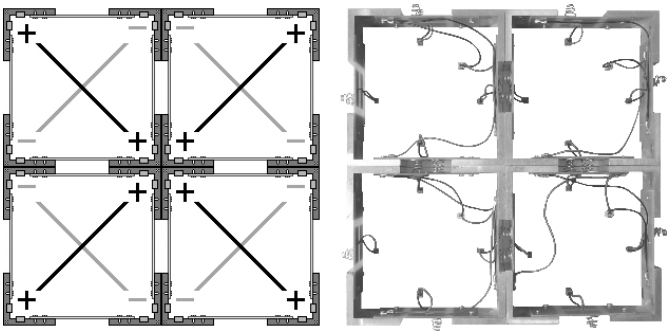


Abbildung 4.5: Stromversorgung über die Winkel

Seite gleich aussehen müssen, wird der Strom am Parkplatz (der nicht als Spielfeldbox gesehen wird und so nicht gleich aussehen muss) eingespeist und dann über die Winkel weiterverteilt. Damit jede Box beliebig gedreht werden kann und trotzdem aneinander gesteckt werden kann, müssen die Winkel mit gleichem Potential diagonal angeordnet werden. Die diagonal gegenüberliegen Winkel einer Box werden miteinander verbunden. So können dann mehrere Boxen miteinander kombiniert werden, siehe Abbildung 4.5.

4.1.1.3 Oberfläche

Die Oberfläche der Spielfeldbox dient vor allem dazu, dass der Roboter über diese fahren kann. Sie besteht aus einem grafischen Design, das dazu dient, dass der Spieler erkennen kann um welche Art von Feld es sich handelt, siehe Kapitel 2. Über dieses grafische Design muss dann der Roboter fahren können, somit muss die Oberfläche dafür stabil genug sein. Außerdem sollte es möglich sein, Checkpoints festzulegen.

Im Folgenden wird das grafische Design sowie die Möglichkeit zur Festlegung der Checkpoints genauer betrachtet.

Grafisches Design

Das grafische Design der Oberfläche dient dazu, dass der Benutzer erkennen kann, welches Feld welche Eigenschaften hat (z. B. wo sich eine Wand befindet). Da die Roboter über die Felder fahren müssen, gab es leider keinerlei Möglichkeiten, das Design 3-Dimensional zu gestalten. In Anlehnung an das englische Originalbrettspiel ist das Layout der einzelnen Felder der Spielbretter des Brettspiels verwendet worden und diese zu diesem Zweck eingescannt worden. Es sind einzelne Felder ausgesucht worden, die zu 3 x 3 Feldern zusammengesetzt worden sind. Diese 3 x 3 Felder sind auf 120g Papier gedruckt worden.

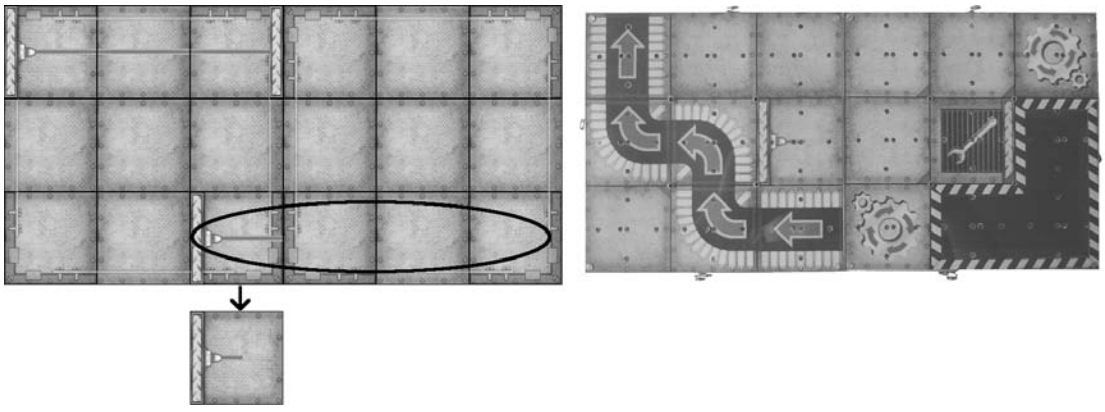


Abbildung 4.6: Spielfläche mit Laser

Allerdings gab es hierbei ein Problem mit der Darstellung der Laser. Wenn ein Laser auf einer Spielfeldbox beginnt, muss dieser nicht unbedingt auf derselben enden (siehe schwarzer Kreis in Abbildung 4.6). Leider kann dieses nicht auf dem Papier dargestellt werden. Deswegen wird nur der Anfangspunkt des Lasers grafisch auf dem Papier festgehalten. Damit der Spieler sich nicht vorstellen muss, wo sich ein Laserstrahl auf dem Spielfeld befindet, werden dazu die in Kapitel 4.3.2 beschriebenen LEDs benutzt. In der Programmierphase und in der Abarbeitungsphase werden an den Stellen, wo sich Laserstrahlen auf dem Feld befinden, die jeweiligen LEDs eingeschaltet. Auf diese Weise sind die Laser nicht auf eine Länge von 3 Feldern beschränkt.

Um die Papieroberfläche vor zu starker Abnutzung zu schützen, wurde über dem Papier eine $1,5\ mm$ dicke $15\ cm$ x $15\ cm$ große Makrolon-Platte befestigt. Da das Makrolon durchsichtig ist, müssen keine Löcher für die Kommunikation in das Makrolon gebohrt werden. Um das Makrolon mit der Unterkonstruktion zu verbinden, sind in die Ecken der Aluminiumwinkel Gewinde geschnitten worden, und die Makrolon-Platte ist mit Schrauben an den Winkeln befestigt worden. Damit der Roboter problemlos über diese Oberfläche fahren kann, sind zu diesem Zweck Senkkopfschrauben verwendet worden, siehe Abbildung 4.1.

Checkpoints

Die Oberfläche der Spielfeldbox muss eine Möglichkeit bieten, die schon in Kapitel 2.1.3 beschriebenen Checkpoints festlegen zu können, siehe [AMI99]. Dazu ist folgende Idee entstanden: In Anlehnung an das englische Spiel werden die Checkpoints als Fähnchen realisiert. Die Fähnchen werden in einer Ecke des Feldes platziert, und über Drucktaster auf der Platine wird festgestellt, wo ein Fähnchen steckt. Da die Roboter $4,5\ cm$ groß sind und nicht exakt in der Mitte der Felder fahren, müssen die Fähnchen möglichst weit vom Mittelpunkt des Feldes entfernt sein. D.h. die Drucktaster für die Fähnchen können leider nur in den Ecken der Felder platziert werden. Da auf der Platine der Platz für die Drucktaster vorhanden sein muss, bleiben so nur noch 4 Plätze für Drucktaster (siehe Abbildung 4.7).

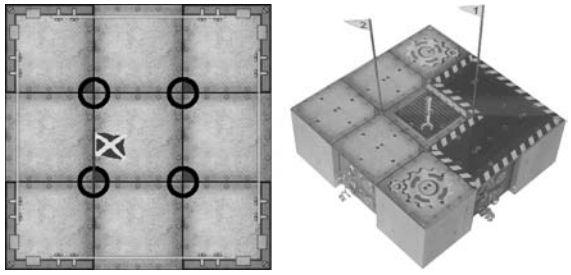


Abbildung 4.7: Plätze für die Drucktaster in der Spielfläche

Dementsprechend können sich auf jeder Spielfeldbox nur 4 Felder befinden, die ein Checkpoint sein können. Darauf musste beim grafischen Design natürlich geachtet werden. Dies

ist vor allem deshalb möglich, da laut Spielbeschreibung nur folgende Felder ein Checkpoint sein können (siehe [AMI99]):

- Feld ohne eine besondere Funktion
- Feld mit nur einer Mauer
- Feld mit 2 Mauern
- Reparaturfeld ohne Mauer
- Reparaturfeld mit einer Mauer
- Doppeltes Reparaturfeld ohne Mauer
- Doppeltes Reparaturfeld mit einer Mauer

Es war also ohne Probleme möglich alle Boxen so zu gestalten, dass höchstens 4 Felder auf jeder Box vorhanden sind, die ein Checkpoint sein können. Um eine eindeutige Zuordnung von Checkpointfahne zu dem jeweiligen Spielfeld zu garantieren, wird jedem der 4 Plätze für die Drucktaster ein Spielfeld zugeordnet. Dies geschieht dadurch, dass auf dem jeweiligen Spielfeld eine blaue Ecke dort gezeichnet wird, wo die zugehörige Fahne platziert werden könnte, siehe Abbildung 4.7.

Jeder Checkpoint hat eine eindeutige Nummer und diese muss festgestellt werden. Da durch die Drucktaster nur festgestellt werden kann, ob ein Feld ein Checkpoint ist oder nicht, muss die Nummer des Checkpoints durch die Software festgestellt werden. Der Host-PC fordert den Spieler dazu auf, den x-ten Checkpoint einzustecken. In dem Moment, wo ein Fähnchen in ein Feld eingesteckt wird, wird dies durch den Drucktaster festgestellt. Das Feld bekommt danach von dem Host die Nummer des Checkpoints, siehe Kap. 6.3.5.2.

4.1.2 Elektronik

Der elektronische Aufbau der Spielfeldboxen entspricht genau der Beschreibung des Zwischenberichts. Es wurden keine Veränderungen mehr vorgenommen. Jedes Spielfeld setzt sich demnach aus folgenden Komponenten zusammen:

- **Mikrocontroller:** Zentrales Bestandteil jeder Spielfeldbox ist der Mikrocontroller LPC2124 [Phi03] von Phillips. Über ihn werden die Kommunikations-LEDs und die Fototransistoren direkt und die Navigations-LEDs indirekt angesteuert. Außerdem ist er mit jeder Kommunikationsschnittstelle an jeder Spielfeldboxseite verbunden.
- **Kommunikations-LEDs:** Jede Spielfeldbox besitzt genau 9 Kommunikations-LEDs. Diese befinden sich in der Mitte jedes Feldes und werden zum Senden von Befehlen an den Roboter benötigt. Sie dienen aber ebenso zum Ausrichten des Roboters und zur Darstellung der Laserstrahlen.

- **Fototransistoren:** Neben den 9 Kommunikations-LEDs sind auch 9 Phototransistoren auf jeder Spielfeldbox vorhanden. Diese befinden sich ebenfalls in der Mitte jedes Feldes. Über die Fototransistoren werden Meldungen vom Roboter empfangen.
- **Navigations-LEDs:** Die 36 Navigations-LEDs auf jeder Spielfeldbox dienen zur Orientierung und zum Ausrichten des Roboters und zur Darstellung der Laserstrahlen. Sie werden nicht direkt über den Mikrocontroller angesteuert, sondern über einen CPLD.
- **CPLD:** Eingesetzt wird der Xilinx XC 9572XL TQ100 [Xil03]. Über ihn läuft die Ansteurung der Navigations-LEDs, sowie das Auslesen der Checkpoint-Taster.
- **Spannungsversorgung:** Im Spielfeld werden konstante Spannungen von $3,3\ V$ und $1,8\ V$ benötigt. Da an jeder Spielfeldbox eine höhere Spannung anliegt und diese auch durch Drehen der Spielfeldboxen ihre Polung ändern kann, ist eine extra Schaltung nötig. Durch den Einsatz von einem Brückengleichrichter und zwei Spannungsreglern werden die gewünschten Versorgungsspannungen erreicht.
- **Kommunikationsschnittstellen:** Die Kommunikation mit anderen Spielfeldern erfordert eine Schnittstelle an jeder Seite der Spielfeldbox. Eine Schnittstelle verwendet dabei 2 Leitungen, eine DATA und eine CLOCK-Leitung.
- **Sound:** Zur Erzeugung von Geräuschen wird der PWM-Ausgang des Mikrocontrollers benutzt. Über eine kleine Verstärkerschaltung mit einem Transistor wird direkt ein Lautsprecher angesteuert.

In Abbildung 4.8 ist die Kommunikation der verschiedenen Komponenten dargestellt. Der Mikrocontroller ist dabei die zentrale Steuereinheit.

Die Navigations-LEDs werden allerdings nicht direkt vom Mikrocontroller, sondern indirekt über den CPLD angesteuert. Diese Maßnahme wurde nötig, da der Mikrocontroller nicht über genügend Ausgänge zur seperaten Ansteuerung jeder LED verfügt. Auch die Checkpoint-Taster werden über den CPLD ausgelesen.

Die Kommunikations-LEDs und die Fototransistoren sind direkt mit dem Mikrocontroller verbunden. Der Grund hierfür liegt in den zu befürchtenden Geschwindigkeitseinbußen bei indirekter Ansteuerung über weitere Bausteine. Bei den Fototransistoren war der Einsatz einer kleinen Verstärkerschaltung, bestehend aus einem Transistor und drei Widerständen jedoch unumgänglich.

Die Kommunikationsschnittstellen sind ebenfalls direkt mit dem Mikrocontroller verbunden. Sie bestehen aus zwei Leitungen, die über einen Pull-up Widerstand auf logisch 1 gezogen werden. Weiterhin wurden an jeder Schnittstelle noch zwei Widerstände zur Kurzschlusssicherung und zwei Dioden zum Überspannungsschutz eingebaut.

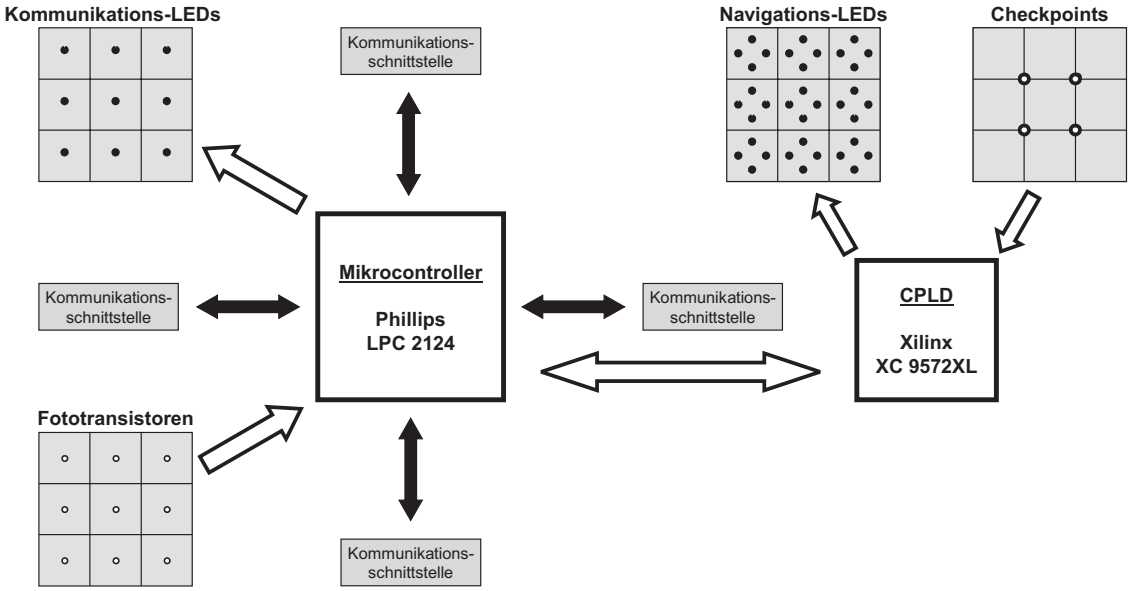


Abbildung 4.8: Schema des Aufbaus der Feldplatine

4.1.3 Host-Schnittstelle

Die Kommunikation des Host-PC mit dem Spielfeld sollte über die serielle Schnittstelle des PC realisiert werden. Das Spielfeld sollte aber aus den Spielfeldboxen variabel zusammen steckbar sein. Außerdem existierte kein Platz für einen seriellen Anschluss, der sowieso nur für vier Host-PCs vorhandensein musste. Um aber wieder die Variabilität zu wahren, hätte jede Box einen Pegelwandler für die RS232-Pegel und eine Buchse enthalten müssen. Aus diesen Gründen wurde ein Adapter konstruiert, der die Verbindung von Host-PC und Spielfeld realisiert.

4.1.3.1 Aufbau

Der Adapter enthält die Kontaktflächen (siehe Abschnitt 4.1), durch welche die Spielfeldboxen untereinander verbunden werden. Auf der anderen Seite besitzt der Adapter eine Buchse, um sich über ein serielles Kabel mit dem Host-PC zu verbinden.

Die Verbindung des Adapters mit den Spielfeldboxen über die Kontaktflächen impliziert auch die gleiche Spannungsversorgnung über die Aluwinkel (siehe Abschnitt 4.1.1.2).

Damit gleicht eine Seite des Adapters dem Aufbau einer Seite einer Spielfeldbox, zu sehen in der Abbildung 4.1.

Die doppelseitige Platine mit den Kontaktflächen auf der einen Seite enthält auf der anderen alle weiteren elekronischen Bauelemente des Adapters. Somit bestimmen die Bauelemente die mögliche Tiefe des Adapters, wodurch geplant wurde, jeweils eine Seite der zwei Winkel zu kürzen. Außerdem wurde die Verkleidung der noch offenen Seiten außer der Unterseite durch dünne Makrolonplatten, die auch im Feldaufbau Verwendung fanden, geplant.

4.1.3.2 Elektronik

Die Elektronik enthält als Hauptkomponente den Mikrocontroller LPC2124 [Phi03] von Philips, welcher auch in der Elektronik der Spielfeldboxen Anwendung fand. Für den Mikrocontroller wird ein Quarz mit $14,7456\,MHz$ eingesetzt.

Die Spannungsversorgung enthält den bedrahteten Brückengleichrichter FBU4A [Dio05], der auch in der Elektronik der Spielfeldboxen Verwendung fand. Dieser Brückengleichrichter wurde durch das Zurechtbiegen der Anschlussbeine und das flach auf die Platine Legen vom Prinzip her in ein SMD-Bauteil gewandelt. So konnte der selbe Gleichrichter wie im Spielfeld verwendet werden.

Der Mikrocontroller benötigt Spannungen von $1,8\,V$ und $3,3\,V$. Um diese bereit zu stellen, werden die SMD-Variante des LM1086 [Nat05] und der $1,8V$ Regler XC6201P162MP [Tor] gebraucht. Die in den Datenblättern vorgeschlagenen Kondensatoren für die Spannungsregler sind ebenfalls SMDs.

Für die Kommunikation mit dem Spielfeld wurden zwei Pins des Mikrocontrollers benutzt. Dabei besitzt einer Interruptfähigkeit für die Clockleitung der Kommunikation, und der andere ist ein einfacher IO-Pin für die Daten. Vor jeden der beiden Kommunikationspins ist ein Widerstand von $51\,\Omega$ geschaltet, und danach ein Pull-Up-Widerstand von $4,7\,k\Omega$. Außerdem ist zum Schutz der Pins jeweils eine Diode in Sperrrichtung zu Masse und eine in Durchlassrichtung zur Versorgungsspannung geschaltet.

In dem Schaltplan und im Layout im Anhang 9.1 und 9.2 sind mehrere $0\,\Omega$ Widerstände erkennbar. Diese dienen nur als Brücken im Layout der Schaltung.

Zur Kommunikation mit dem PC wird die UART0 Schnittstelle des Mikrocontrollers benutzt. Dabei wurden nur die Pins RxD0 (Seriell Input) und TxD0 (Seriell Output) benutzt. Die Pegel des Mikrocontrollers werden durch den RS232-Pegelwandler ICL3232CB [Int02] umgesetzt.

Die serielle Schnittstelle wird zusätzlich zur Programmierung des Mikrocontrollers benutzt. Dazu ist ein SMD-Jumper vorgesehen, mit dem zwischen ISP (In-System Programming) Modus und normalem Betrieb umgeschaltet werden kann. Weiterhin ist eine

Stiftleiste mit zwei Pins (Masse und RST) für die Beschaltung mit einem externen Resettaster vorgesehen. Ansonsten wird der Reset beim Anlegen der Versorgungsspannung durch einen Kondensator zwischen dem RST Pin und Masse generiert. Die Programmierung geschieht wie erwähnt über die serielle Schnittstelle. Im ersten Semester der PG wurde das Konzept des Adapters komplett entwickelt und eine Platine mit den beiden Winkeln produziert.

4.2 Änderungen Hardware

4.2.1 Mechanik

Nach einigen Tests sind vor allem in der Mechanik Schwachstellen aufgetreten, die es zu beseitigen galt:

- Stromverteilung über Winkel

  Die Winkel sind aus Aluminium gefertigt worden. Da dieses Material sehr schnell oxidiert, wird so die gute leitende Eigenschaft der Winkel beeinträchtigt. Dieses Problem ist durch Beseitigen der Oxidschicht gelöst worden.

- schwache Oberfläche

  Der zweite Roboterentwurf hatte leider ein wesentlich höheres Gewicht als zunächst geplant, somit bog sich die Oberfläche der Spielfeldboxen nach unten durch. Aus diesem Grund wurden auf der Platine unter der Oberfläche Abstützungen angebracht.

- Kommunikationsprobleme

  Bei den Tests hat sich herausgestellt, dass die Fototransistoren des Spielfeldes das Licht ihrer benachbarten LEDs emfangen und als Kommmunikationsversuch des Roboters interpretieren. Dieses Problem ist durch kurze Schrumpfschläuche über den Phototransistoren gelöst worden.

- nicht stabilisierte Checkpointfahnen

  Ein einfaches Loch in der Makrolonplatte bot zu wenig Halt für eine Checkpointfahne. Um eine längere “Führung“ für die Fähnchenstange zu erhalten, ist eine Aderendhülse zur Stabilisierung in das Loch in der Makrolonplatte eingearbeitet worden.

4.2.2 Host-Schnittstelle

An der Elektronik gab es keine Änderungen im zweiten Semester der PG. Es wurden lediglich drei weitere Platinen geätzt und bestückt. Außerdem wurden die Winkel entsprechend denen im Spielfeld verwendeten gefräst und auf einer Seite gekürzt. Hinzu

kam das Einkleben der Magnete. Den Abschluss beim Aufbau bildete die Verkleidung mit Makrolon. Somit hat die Schnittstelle wie eine Spielfeldbox eine Breite von 150 mm und eine Höhe von 51 mm. Die Tiefe konnte bei der Schnittstelle durch das Kürzen der Winkel auf 17 mm reduziert werden. Im Abschnitt 4.3.3 wird auf die Programmierung des Adapters eingegangen. Die Abbildung 4.9 zeigt die fertige Host-Schnittstelle mit der durchsichtigen Makrolonverkleidung.

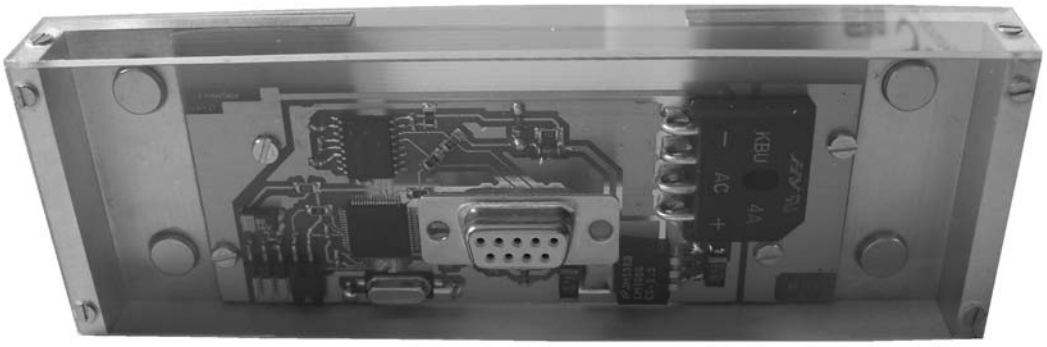


Abbildung 4.9: Host-Schnittstelle

4.3 Kommunikation

Im folgenden wird beschrieben wie die Kommunikation im gesamten R_3D_3-Projekt realisiert wird. Dabei geht es hier nur um die Low-Level Schicht der Kommunikation, sprich die einzelne Nachrichtenübertragung zwischen jeweils zwei Komponenten. Wie die Nachrichten durch das komplette System geleitet werden, wird im Abschnitt 4.4.3 beim Protokollstack beschrieben. Dabei ist hier vorab anzumerken, dass die Kommunikation in Schichten unterteilt ist. Dabei sind die hier beschriebenen Protokolle ganz unten anzusiedeln, der Low-Level Kommunikation. Alle Protokolle sind eine Neuentwicklung und verwenden keinerlei externe Quellen.

4.3.1 Spielfeld - Spielfeld

Die Kommunikation zwischen den Spielfeldern basiert auf ganz einfachen Prinzipien. Folgende Vorraussetzungen müssen dabei erfüllt werden:

- ROM- und insbesondere RAM-Bedarf müssen möglichst gering sein
- Die Kommunikation sollte nebenläufig und unabhängig zum eigentlichen Spielfeld-Programm sein
- Fehlertolerant
- Großer Datendurchsatz

- Geringe Anzahl von Leitungen

Um die oben erwähnten Voraussetzungen zu erfüllen, werden bei der Feld zu Feld Kommunikation folgende Eigenschaften genutzt:

- Einfaches, aber ausreichendes, Protokoll ohne überflüssige Erweiterungen (Details siehe unten)
- Nutzen der Controller-Interrupts für Senden sowie Empfangen

Bei der Feld zu Feld Kommunikation werden 2 Leitungen benötigt: Eine Takt- und eine Daten-Leitung. Die Leitungen können sich entweder im Zustand 0 oder im Zustand 1 befinden. Dabei können die teilnehmenden Kommunikationspartner den Zustand nur auf '0 setzen' oder 'nicht auf 0 setzten'. Falls die Leitung von keinem Teilnehmer auf 0 gesetzt wird, hat die Leitung automatisch den Zustand 1. Dies wird erreicht, indem der Pin des Controllers auf 'Eingang' geschaltet wird. Wenn mindestens einer der beiden Teilnehmer die Leitung auf 0 setzt, ist die gesamte Leitung auf 0.

Einzelne Bits werden gesendet, indem der Sender die Datenleitung auf 0 oder 1 setzt, je nachdem welchen Wert das zu übertragende Bit hat. Daraufhin wird die Taktleitung kurzzeitig als Impuls auf 0 gesetzt, um dem anderen Teilnehmer zu signalisieren, dass ein neues Datenbit anliegt.

Das Empfangen eines Bits geschieht beim Empfänger, indem ein Interrupt durch die Taktleitung ausgelöst wird. Sobald die Interruptroutine dann aufgerufen ist, wird der Datenpin des Empfängers auf 'Eingang' gesetzt und der daraufhin anliegende Zustand als Bit übernommen.

Um nun aus dem eingehenden Bitstrom ganze Datenpakete zu generieren, wird ein Protokoll benötigt, welches 'on-the-fly' beim Eintreffen der Datenbits das nächste Datenpaket aufbauen kann. Dabei ist der Ablauf des Protokolls aus Sicht des Senders in 3 Schritte unterteilt. Diese sind Synchronisationsdaten, Paketlänge, Paketdaten. Dabei werden diese Teile von jeweils einer 1 abgetrennt. Dadurch entsteht folgender detaillierter Ablauf:

1. Senden von 9x '0'
2. Senden einer '1'
3. Senden der Paketlänge in Bit (0..255)
4. Senden einer '1'
5. Senden der nächsten 8 Bits
6. Senden einer '1'
7. Falls Paket nicht vollständig übertragen, wiederhole bei Schritt 5

Das Senden von 9x '0' bezweckt, dass das Protokoll sich re-synchronisieren kann, falls es zu einer Fehlübertragung kommen sollte. Das funktioniert, da nur an dieser Stelle neun

Nullen hintereinander gesendet werden. Indem die Paketlänge am Anfang mit übertragen wird, ist die Paketlänge variable und reduziert damit die Übertragungsmenge im Durchschnitt deutlich. Abbildung 4.10 zeigt eine einzelne Paketübertragung im zeitlichen Verlauf. Die obere Linie zeigt die Taktleitung, diese ist die meiste Zeit im Zustand 1. Die kurzen Impulse sind leicht, aber deutlich zu erkennen. Die Datenleitung ist mit halber Skalierung auf dem Bild überlagert. Dort sind die einzelnen Bits deutlich zu erkennen. Die Paketlänge beträgt hierbei 56 Bits.

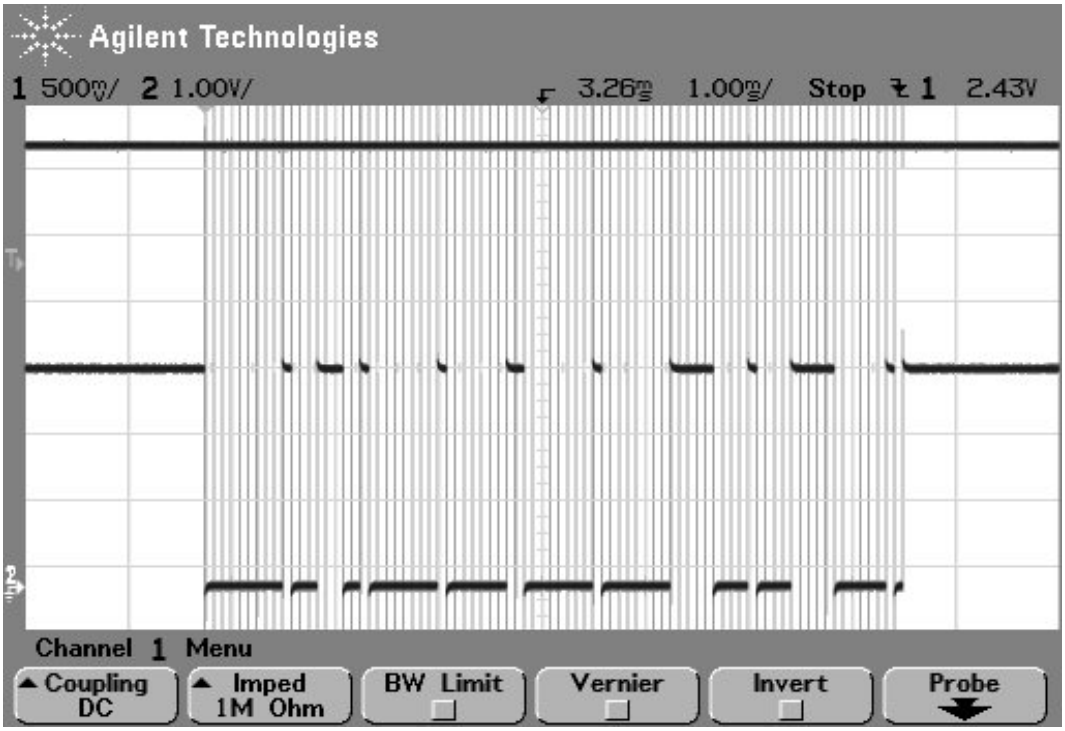


Abbildung 4.10: Bild einer Paketübertragung bei der Feld zu Feld Kommunikation

4.3.2 Spielfeld - Roboter

Die Kommunikation zwischen Spielfeld und Roboter wird primär genutzt, um dem Roboter Bewegungsbefehle mitzuteilen. Dies folgt aus der Entscheidung, die Spielintelligenz im Spielfeld zu implementieren.

4.3.2.1 Kommunikationsdesign

Eine Reihe drahtloser Kommunikationstechniken stehen für einen Einsatz zur Verfügung, von Infrarot bis zu Wireless LAN. Es wurde jedoch eine Licht-basierte Kommunikation gewählt. Diese Wahl ist wie folgt motiviert:

- Orientierung

  Bei einer Licht-basierten Kommunikation kann die Kommunikationshardware zusätzlich für eine genaue Ausrichtung des Roboters auf dem Spielfeld genutzt werden (vgl. Fehlerkorrektur durch den Roboter in Abschnitt 5.4).

- Komplexität

  Eine Licht-basierte Kommunikation bedarf eines geringen Hardwareaufwandes. Sie kann direkt vom Mikrocontroller angesteuert werden und erlaubt die einfache Entwicklung eines eigenen Kommunikationsprotokolls.

- Visualisierung

  Die LEDs im Spielfeld können gleichzeitig genutzt werden, um die Spielkomponente Laser zu visualisieren.

- Softwaredebugging

  Die LEDs können beim Debuggen der Software als Ausgabemedium genutzt werden (um z.B. interne Zustände darzustellen). Ferner ist bei Integrationstests beobachtbar, ob eine Kommunikation tatsächlich stattfindet. Somit sind mögliche Fehlerquellen in der Zusammenarbeit von Spielfeld und Roboter genauer und einfacher zu lokalisieren.

4.3.2.2 Anforderungen an das Kommunikationsprotokoll

Einige besondere Anforderungen sind bei der Entwicklung des Kommunikationsprotokolls zu beachten:

- Kommunikationsinitiierung

  Die Kommunikation zwischen Spielfeld und Roboter findet sporadisch statt und es ist nicht garantiert, dass der Kommunikationspartner gerade zum Empfang bereit ist. Folglich ist eine Initiierung so zu konstruieren, dass dies sicherstellt wird. Ggf. ist auf die Empfangsbereitschaft des Kommunikationspartners zu warten.

- Fehlertoleranz

  Eine genaue Kommunikationsgeschwindigkeit kann nicht garantiert werden, da z.B. die Kommunikation des Spielfeldes mit seinen Nachbarn über Interrupts gesteuert wird. Leichte Schwankungen in der Übertragungsgeschwindigkeit sind also möglich und müssen vom Protokoll toleriert werden.

- Fehlerdetektion

  Einfache Kommunikationsfehler sind zu erkennen.

- Tageslichterkennung

  Die Kommunikation seitens des Roboters ist so anzustoßen, dass das Spielfeld sie von einem Lichteinfall - z.B. durch Sonnenlicht auf ein nicht besetztes Feld - unterscheiden kann.

- Orientierung

  Das Kommunikationsprotokoll ist so zu entwickeln, dass dem Roboter eine Ausrichtung an einer LED ermöglicht wird, an der er sich unmittelbar darauf als am Ziel angekommen anmelden muss.

Es sind dabei folgende Nachrichten zu spezifizieren:

- Nachrichten zur Roboteridentifizierung:
  - MESSAGE_IDENTIFY
    Spielfeld fordert Roboter zur Identifizierung auf.
  - MESSAGE_MY_ID
    Roboter identifiziert sich mit eindeutiger ID (4 Roboter $\Rightarrow$ 4 Nachrichten).
- Nachrichten zur Kommunikationsunterstützung
  - MESSAGE_ACK
    Roboter bestätigt erfolgreichen Nachrichtenempfang.
  - MESSAGE_COMM_ERROR
    Roboter meldet Kommunikationsfehler.

  Das Spielfeld bestätigt keine Nachrichten. Es kann nur die Meldung eines leeren Akkus bekommen. Dies wird dem Anwender aber auch direkt durch den Roboter signalisiert.
- Nachrichten mit Bewegungsbefehlen an den Roboter
  - MESSAGE_MOVEMENT_FORWARD
  - MESSAGE_MOVEMENT_BACKWARD
  - MESSAGE_ROTATION_LEFT
  - MESSAGE_ROTATION_RIGHT
  - MESSAGE_ROTATION_UTURN
  - MESSAGE_POSITION_CORRECTION
    Löst eine starke Positionskorrektur des Roboters aus (vgl. Abschnitt 5.4.2).
- Nachrichten des Roboters

– MESSAGE_EMPTY_BATTERY
Roboter teilt Spielfeld mit, dass seine Batterie leer ist und er für weitere Aufgaben nicht zur Verfügung steht (vgl. Abschnitt 5.3.1).

4.3.2.3 Protokolldesign

Die Kommunikation zwischen Spielfeld und Roboter wird mit 7 Bit langen Nachrichten entworfen, hinzu kommt ein Bit zur Fehlerdetektion (Prüfsummenbit). Ein Kommunikationsvorgang besteht aus drei Phasen:

1. Verbindungsaufbau

   Beide Kommunikationspartner stellen genügend Ressourcen zur Verfügung und bereiten sich somit auf die Kommunikation vor. Sie schließen diese Phase ab, indem sie ihre Kommunikationsbereitschaft signalisieren.

2. Senden/Empfangen der Datenbits

   Die acht Datenbits werden jeweils zwischen einer Eins und einer Null übertragen (siehe Abbildung 4.11), der Empfänger kann den Wert des Datenbits dann per Majoritätsentscheidung ermitteln.

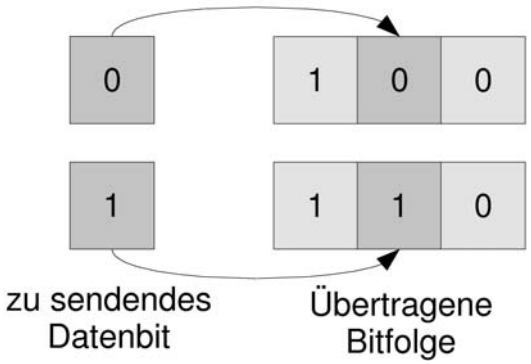


Abbildung 4.11: Einbettung der Datenbits

3. Verbindungsabbau

   Eine abschließende Eins teilt das Ende des achten Datenbits mit.

Die Bestätigung eines Nachrichtenempfangs sowie das erneute Senden einer Nachricht werden als eigenständige Kommunikationsvorgänge durchgeführt.

4.3.2.4 Allgemeine Kommunikationsparameter

Folgende Parameter sind für die Nutzung des Protokolls zu beachten:

- Sendefrequenz (soll): 4 kHz
- Sendefrequenz (max): 5 kHz
- Sendefrequenz (min): 2 kHz
- Empfängerabtastrate während aktiver Kommunikation (min): $8kHz$
- Abtastrate inaktive Kommunikation [Lauschen](min): beliebig

4.3.2.5 Ausgangssituation

Zu Beginn einer Kommunikation steht der Roboter auf einem Spielfeld (bzw. kann auch bei einer Spielfeld-initiierten Kommunikation auf dem Weg zu diesem sein, sich dann dort ausrichten und letztendlich eine äquivalente Situation schaffen). Sowohl das Spielfeld als auch der Roboter lauschen auf dem Kommunikationskanal, ob der Kommunikationspartner einen Datenaustausch initiiert.

Wird durch mindestens einen der Kommunikationspartner eine Kommunikation initiiert (z.B. durch den Roboter um eine Fehlermeldung à la Batterie-ist-leer oder durch das Spielfeld um Bewegungsbefehle zu senden), so werden die drei Schritte des Protokolls abgearbeitet.

4.3.2.6 Phase 1: Verbindungsaufbau

Beide Kommunikationspartner müssen genügend Ressourcen für die Kommunikation reservieren, d.h. insbesondere, dass der Empfänger über die anstehende Kommunikation informiert werden muss. Der Verbindungsaufbau wird abgeschlossen durch eine Bereitschaftssignalisierung beider Kommunikationspartner.

- Der *Sender* reserviert die notwendigen Ressourcen und beginnt mit einem ihm typischen Blinkmuster (s.u.).
- Der *Empfänger* erkennt bei seinem gelegentlichen Abtasten, dass der andere Kommunikationspartner eine Nachricht senden möchte, reserviert seinerseits die notwendigen Ressourcen und antwortet mit einem ihm typischen Blinkmuster.
- Der *Sender* erkennt am Blinkmuster des Empfängers, dass dieser zum Empfang bereit ist und beendet sein Blinken. Er wartet, bis auch der Empfänger sein Blinken beendet hat und beginnt dann mit seinem Nachrichtenversand.
- Der *Empfänger* erkennt am Ausbleiben des Blinkens des Senders, dass dieser seine Bereitschaftsbestätigung empfangen hat, hört seinerseits auf zu blinken und ist so für den Datenempfang bereit.

Blinkmuster Roboter

Das Blinkmuster des Roboters ist in Abbildung 4.12 dargestellt.

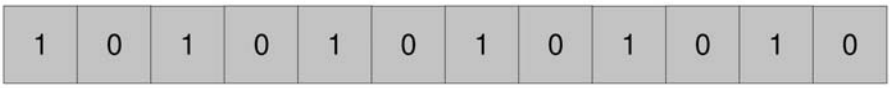


Abbildung 4.12: Blinkmuster Roboter

Zwei Gründe sprechen für dieses Muster:

1. Ein Blinkmuster mit schnellen Wechseln ermöglicht ein schnelles Erkennen einer Reaktion seitens des Spielfeldes. Dies gilt, da der Roboter nur während der Null-Phasen lauschen kann. Häufige Null-Phasen führen dementsprechend zu einer geringeren maximalen und durchschnittlichen Verzögerung.
2. Der schnelle Wechsel von Nullen und Einsen erlaubt dem Spielfeld, die Flanken zu zählen und an ihnen zu erkennen, ob ein Roboter eine Kommunikation zu initiieren versucht (Flankenfrequenz mind. 1 kHz) oder ob er durch Tageslicht oder Lampenlicht (Frequenz höchstens etwa 50 Hz) angeregt wird.

Blinkmuster Spielfeld

Abbildung 4.13: Blinkmuster Spielfeld

Da sich die Anforderungen an das Spielfeld von denen an den Roboter unterscheiden, unterscheidet sich auch sein Blinkmuster (siehe Abbildung 4.13). Dieses ist nämlich so zu wählen, dass es

1. eine möglichst lange Eins liefert, an der sich ein Roboter während der Fahrt ausrichten kann (Einzelheiten dazu siehe Abschnitte 5.4.6 und 5.4.7),
2. es sicher und schnell eine Reaktion seitens des Roboters erkennt.

Eine Dauer von 20 Einsen in Folge ist ein Kompromiss aus schneller Reaktion und hohem Einsanteil, zwei Nullen in Folge stellen sicher, dass ein Blinken des Roboters detektiert werden kann, aber auch, dass wenn keine Eins vom Roboter empfangen wird, dieser sicher nicht blinkt.

Beispiel Timing Verbindungsaufbau

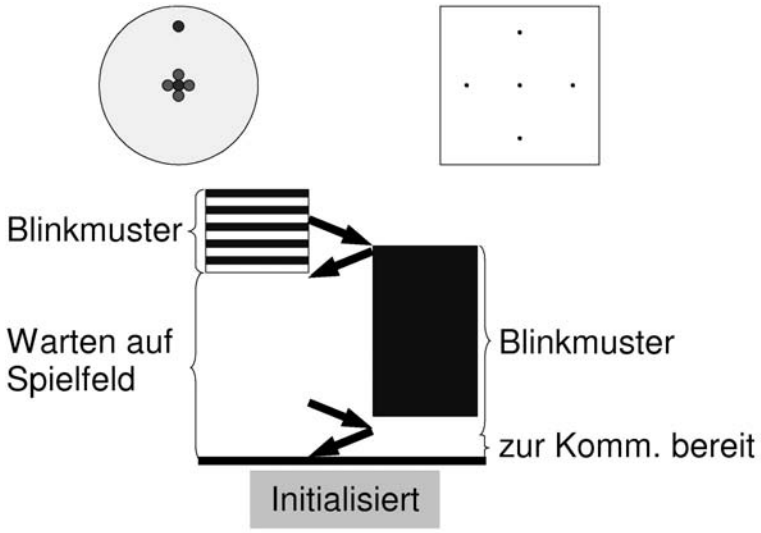


Abbildung 4.14: Beispiel eines Verbindungsaufbaus Roboter zu Spielfeld

In Abbildung 4.14 wird beispielhaft ein Verbindungsaufbau für eine Roboter-initiierte Kommunikation dargestellt.

4.3.2.7 Phase 2: Senden/Empfangen der Datenbits

Der Sender sendet mit seiner Zielsendefrequenz von etwa 4 kHz für seine acht Datenbits folgende Muster:

- für eine Datennull sendet er eine Eins gefolgt von zwei Nullen,
- für eine Dateneins sendet er zwei Einsen gefolgt von einer Null.

Das sich ergebene allgemeine Muster zeigt Abbildung 4.15.

Abbildung 4.15: Sendemuster für ein Datenbit

Zwischen zwei der Muster ist stets eine steigende Flanke. Diese Tatsache wird vom Empfänger genutzt, um die Daten auszuwerten. Von einer bis zur nächsten Flanke wird dazu regelmäßig das Signal abgetastet. Die Nullen und Einsen werden gezählt, und schließlich wird per Majoritätsentscheidung bestimmt, ob es sich bei dem transportierten Datenbit um eine Null oder eine Eins handelte. Ist die Anzahl der gezählten Nullen

größer als die der Einsen, so war das Datenbit eine Null, sonst handelte es sich um eine Eins.

Durch die Majoritätsentscheidung ist das Kommunikationsprotokoll gegenüber leichten Geschwindigkeitsschwankungen eines oder gar beider Kommunikationspartner tolerant. Insbesondere ist keine Synchronität der beiden notwendig.

4.3.2.8 Phase 3: Verbindungsabbau

Die Übertragung des letzten Datenbits endet gemäß Abbildung 4.15 sicher mit einer Null. Da der Empfänger nun aber die Daten von einer steigenden Flanke bis zur nächsten steigenden Flanke auswertet, ist diesem für das letzte Datenbit eine abschließende Eins zu senden.

Sobald der Sender seine Eins zum Verbindungsabbau gesendet hat und sobald der Empfänger sein achtes Datenbit durch diese letzte steigende Flanke bekommen hat, ist die eigentliche Kommunikation beendet und es können nun alle Ressourcen wieder freigegeben werden.

Auf Empfängerseite ist die Nachricht auszuwerten, unter anderem ist die Prüfsumme zu verifizieren. Dazu wurde als 8. Bit ein Paritätsbit gesendet. Ein Roboter als Empfänger hat eine Bestätigung (MESSAGE_ACK oder MESSAGE_COMM_ERROR) zu senden und ggf. eine Aktion auszuführen, ein Spielfeld als Empfänger muss ggf. eine Aktion (z.B. Wiederholen des Befehles bei einer Fehlermeldung) ausführen.

4.3.2.9 Fehlerbehandlung

An diversen Stellen in der Implementierung des Protokolls sind sowohl im Roboter als auch im Spielfeld Timeouts spezifiziert, so dass stets eine Deadlocksituation verhindert werden kann. Im Falle eines Timeouts oder bei starken Einflüssen von außen, die in einer Fehlinterpretation münden, geschieht nun folgendes:

Bei einer Spielfeld-initiierten Kommunikation (dies schließt Bestätigungsnachrichten vom Roboter mit ein) sendet das Spielfeld die Nachricht erneut. Da der Roboter durch ein Timeout wieder in den Wartezustand für die Kommunikation wechselt, kann eine normale Nachrichtenübermittlung erneut beginnen.

Bei einer Roboter-initiierten Kommunikation (hierbei kann es sich nur um die Warnung wegen eines leeren Akkus handeln) versucht der Roboter seine Nachricht so lange zu senden, bis er an einen erfolgreichen Sendevorgang glaubt. Danach signalisiert er dem Benutzer direkt sein Problem durch ein ständiges Blinken der Kopf-LED. Falls das Spielfeld die Nachricht nicht korrekt empfängt, verwirft sie diese. Ein Mechanismus zum erneuten Senden der Nachricht musste nicht entwickelt werden, da der Roboter die Warnung - wie beschrieben - auch direkt signalisiert.

Haben beide gleichzeitig eine Kommunikation initiieren wollen, so kann es dazu kommen, dass sie das Blinkmuster des Gegenübers als Antwort auf ihr Muster interpretiert haben. Folglich haben beide ihre Nachricht gleichzeitig gesendet ohne dass der Partner diese Empfangen konnte. Da jedoch nur das Spielfeld Nachrichten wiederholt sendet[1], kann es nicht zu einer erneuten Kollision kommen.

4.3.3 Spielfeld - Host

Die Feld zu Host Kommunikation verläuft über den 'Host-Adapter', beschrieben in Abschnitt 4.2.2. Bei diesem Adapter ist auf der einen Seite das Feld zu Feld Protokoll realisiert, und auf der anderen Seite steht eine serielle Schnittstelle (RS232) zum Anschluss an den PC zur Verfügung.

Über die serielle Schnittstelle können pro Einheit 5 bis 8 Bits übermittelt werden, eine Übertragungsgarantie gibt es auch hier nicht. Daher ist es sinnvoll, wie bei der Feld zu Feld Kommunikation auch hier einen Selbstsynchronierungs-Mechanismus einzubauen. Dazu erfolgt die Übertragung mit 5 Bits über die serielle Schnittstelle. Dabei werden die 5 Bits in 2 Felder unterteilt: Dem 'Sync'- und dem 'Data'-Feld. Das Syncfeld ist 1 Bit groß und das Datafeld ist 4 Bit groß. Das Syncbit ist standardmäßig nicht gesetzt. Zum Host-PC wird folgendes Kommunikationsprotokoll realisiert:

1. Übertragen der ersten 4 Bits der Paketlänge, dabei ist das Bit des Syncfelds gesetzt
2. Übertragen der letzten 4 Bits der Paketlänge
3. Übertragen der folgenden 4 Bits der Daten
4. Falls noch nicht alle Daten übertragen, wiederhole mit Schritt 3

Das Syncbit steuert damit die Synchronisierung des Protokolls. Dadurch geht maximal ein Paket verloren oder wird falsch übertragen wenn ein Bitfehler aufgetreten ist.

4.4 Software

4.4.1 Aufgaben

Die von R3D3 zur Benutzer-Interaktion verwendeten Host-PCs haben laut PG-Antrag keine Kenntnis der Spielfeld-Topologie. Die Intelligenz des Spiels ist als verteiltes System im Spielfeld realisiert, so dass die Host-PCs weder die *Ausführung der Befehle* noch die *Abarbeitung der Fabrikelemente* durchführen können. Dies muss im Spielfeld geschehen. Einer der Host-PCs (der sog. Master) sendet dem Client, der an der Reihe ist, lediglich eine Aufforderung, seinen Befehl auszuführen. Der Client sendet dann den eigentlichen

[1] In der Regel wird jedoch erst der Roboter auszuwechseln sein, da dieser einen entladenen Akku signalisieren wollte

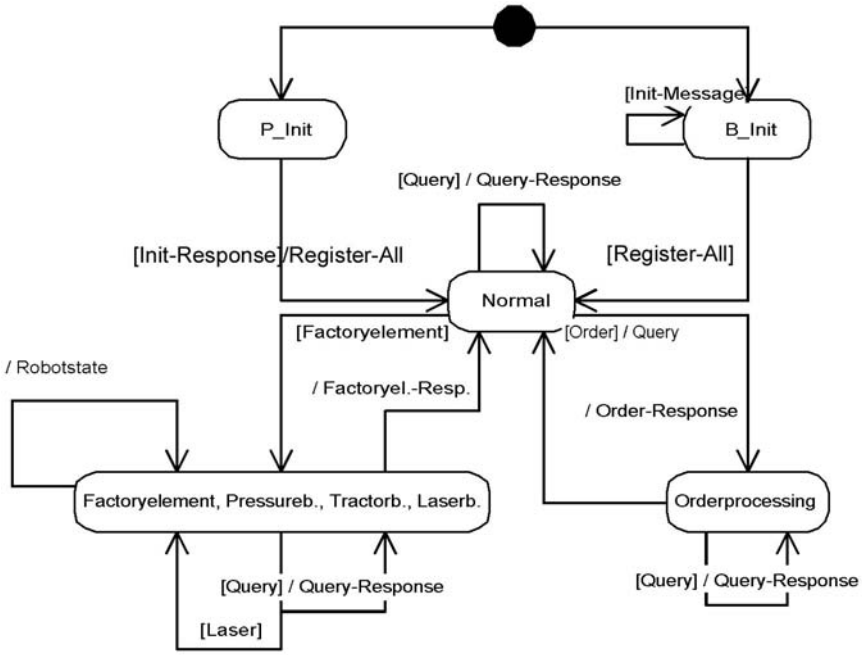


Abbildung 4.16: Zustandsübergangsmodell der Spielfeldsoftware

Befehl an das Feld, auf dem der Roboter steht. Das Feld verfügt im Gegensatz zu den Host-PCs über alle benötigten Informationen für die Befehlsausführung (dazu zählen Kenntnisse über Wände, Gruben, Spielfeldrand, usw.).

Die Abarbeitung der Fabrikelemente muss laut PG-Antrag von einem verteilten Algorithmus durchgeführt werden. Der Master stößt auch in diesem Falle die Abarbeitung durch eine Nachricht an. Die eigentliche Abarbeitung der Fabrikelemente erfolgt im Spielfeld.

Eine weitere Aufgabe der Spielfelder liegt im Bereich der *Kommunikation*. Denn nahezu jede Kommunikation erfolgt implizit über das Spielfeld: Die Host-PCs verfügen über keine direkte Kommunikationsmöglichkeit mit den Robotern, ihre Nachrichten werden durch das Spielfeld geleitet. Dies setzt eine Struktur voraus, die es erlaubt, Nachrichten an spezifizierte Empfänger zu senden. Diese Aufgabe übernimmt das Routing-Protokoll.

4.4.2 Modellierung der Spielfeldsoftware

Die Arbeitsweise der Spielfeldsoftware lässt sich am besten als *Zustandsübergangsmodell* beschreiben. Dieses Modell beinhaltet eine Menge von Zuständen, zwischen denen das System wechseln kann.

Diese Zustandsübergänge sind an gewisse Bedingungen geknüpft, wie zum Beispiel die Ankunft einer Nachricht. Das komplette Zustandsübergangsmodell wird in Abbildung 4.16 dargestellt.

Die Zustände in der Abbildung sind als Ovale dargestellt, die Zustandsübergänge durch Pfeile. Ist an einen Zustandsübergang eine Bedingung geknüpft, so wird sie in eckigen Klammern an den Pfeil geschrieben (vgl. Übergang von B_Init nach Normal).

Findet zusätzlich zum Übergang noch eine Aktion statt, so steht sie nach einem Querstrich (vgl. Übergang von Orderprocessing nach Normal). Ist an den Zustandsübergang sowohl eine Bedingung als auch eine Aktion geknüpft, wird die wie folgt notiert: [Bedingung]/Aktion.

4.4.2.1 Zustandsübergangsmodell

Nach dem Einschalten gelangt das System zuerst in einen Initialen-Zustand, wobei die Spielfelder in den Zustand *B_Init* kommen, während der Parkplatz in *P_Init* startet. In diesem Zustand wird auf der Ebene des Routing-Layers die Init-Nachricht bearbeitet (Eine komplette Übersicht aller Nachrichten ist [ZWB06] zu entnehmen).

Empfängt ein Host-PC eine Init-Nachricht, sendet er eine Init-Response an den Parkplatz zurück. Der Parkplatz sammelt die Responses und sendet diese mit einer Register-All an alle Felder und Hosts. Dies bewirkt im Spielfeld und im Parkplatz jeweils einen Übergang in den Zustand *Normal*.

Im Zustand *Normal* werden Querys mit einer Query-Response bearbeitet, dabei handelt es sich um Nachrichten, die während der Befehlsverarbeitung vorkommen. Einen Zustandsübergang löst die Nachricht Order aus, das System wechselt in den Zustand *Orderprocessing*, in dem der Befehl ausgeführt wird. Die Befehlsausführung endet mit dem Senden der Order-Response Nachricht, die dem Client alle neuen Daten über seinen Roboter mitteilt. Die genaue Ausführung eines Befehls ist im folgenden Abschnitt beschrieben.

Ähnlich zur Befehlsausführung läuft die Abarbeitung der Fabrikelemente ab. Eine Factory-Element-Nachricht führt das System in den Zustand *Factory-Element, Pressurebeam, Tractorbeam, Laserbeam*. Dieser Zustand besteht aus mehreren Unterzuständen, die aber zur Vereinfachung des Diagramms zusammengefasst wurden.

Der *Factory-Element* Zustand führt zur Abarbeitung der Fabrikelemente Expressförderband, Förderband, Schieber, Zahnrad, Presse. Danach folgen die Zustände Pressurebeam und Tractorbeam. Als letztes Fabrikelement treten die Laser der Roboter und der Felder in Kraft und fügen den Robotern Schaden zu. Die Ausführung von Fabrikelementen wird in 4.4.2.3 genauer beschrieben.

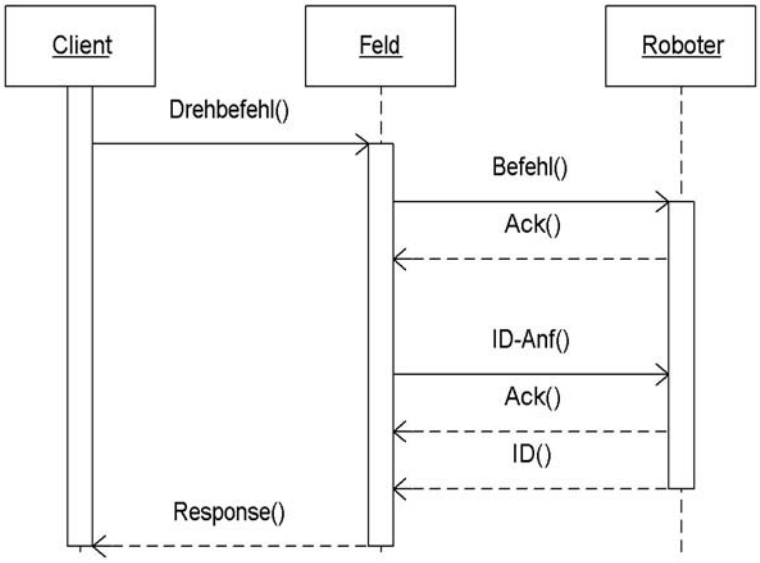


Abbildung 4.17: Ausführung einer Drehung

4.4.2.2 Befehlsausführung

Die Befehlsausführung beginnt, wenn das Feld, auf dem ein Roboter steht, vom Host-PC einen Befehl (z. B. Linksdrehung, Ein-Feld-Vorwärts) zugeschickt bekommt. Da der Host-PC über keinerlei Topologie-Kenntnisse verfügt, muss das Spielfeld zuerst die *Ausführbarkeit* des Befehls überprüfen, die von Wänden, Gruben, anderen Robotern etc. abhängig sein kann. Erst dann wird der Befehl an den Roboter weitergeleitet. Die beiden folgenden Abschnitte behandeln je eine beispielhafte Ausführung eines Drehbefehls und eines Bewegungsbefehls. Zuerst wird hier die Ausführung von Drehungen beschrieben, da sie einfacher ist.

Drehungen

Die Interaktion von Client, Feld und Roboter ist in Abbildung 4.17 dargestellt. Nach dem Empfang des Drehbefehls wird die neue *Blickrichtung* des Roboters berechnet und die entsprechende *LED* zur Navigationshilfe angeschaltet. Dadurch kann sich der Roboter nach der Befehlsausführung auf dem Zielfeld ausrichten (siehe Fehlerkorrektur 5.3.4). Jetzt kann der eigentliche *Befehl* zum Roboter geschickt werden. Der Roboter quittiert den Empfang dieser Nachricht mit einem “Acknowledge“ und führt die Drehung aus.

Das Spielfeld sendet nach dem Erhalt der Acknowledege Nachricht eine *ID-Anforderung*. Wenn der Roboter die Drehung beendet hat und somit wieder bereit für die Kommunikation ist, empfängt er diese Nachricht, antwortet mit einem Acknowledge und dann mit seiner ID. Durch den Empfang der ID des Roboters kann das Spielfeld sicher sein, dass

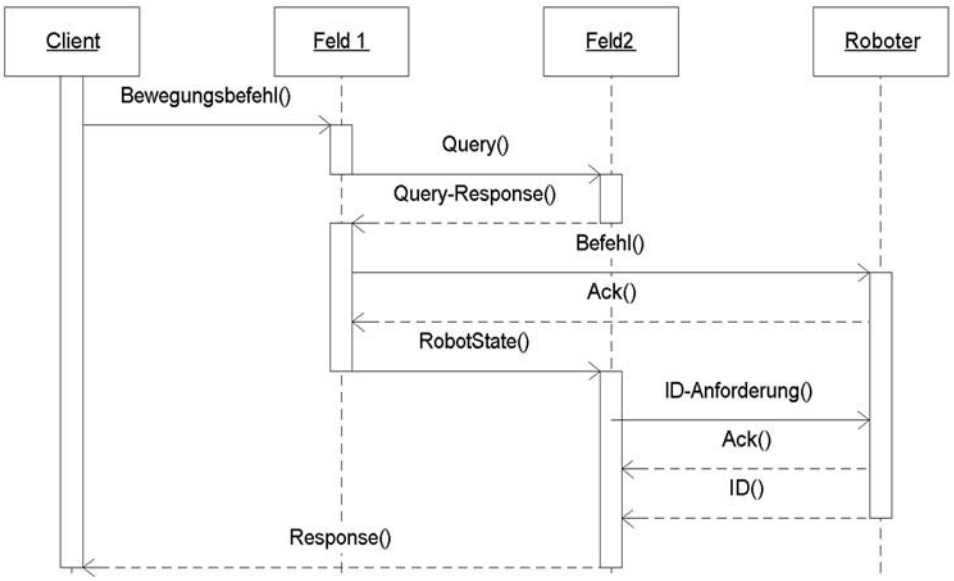


Abbildung 4.18: Ausführung eines Bewegungsbefehls

der Roboter seinen Befehl ausgeführt hat. Daher wird jetzt die LED zur Navigationshilfe ausgeschaltet und eine Response-Nachricht an den Host-PC gesendet. Damit ist die Bearbeitung des Befehls abgeschlossen.

Bewegungsbefehle

Die Ausführung von Bewegungsbefehlen ist deutlich komplexer als die Ausführung von Drehungen, was darauf zurückzuführen ist, dass bei Bewegungen immer mehrere Felder betroffen sind. Die Abarbeitung eines Ein-Feld-Vorwärts-Befehls ist in Abbildung 4.18 zu sehen und wird jetzt beschrieben:

Zuerst wird überprüft, ob der Befehl ausführbar ist, also ob sich eine Wand in Blickrichtung des Roboters befindet. Daraufhin wird ein *Query* an das Zielfeld gesendet, so dass dieses Feld prüft, ob eine Wand den Befehl blockiert. Das Zielfeld antwortet mit einer *Query-Response* Nachricht, in der eine positive oder negative Antwort enthalten ist. Befindet sich auf diesem Feld ein Roboter, wird dieser entweder verschoben und danach eine positive Antwort gesendet oder aber eine negative Antwort, falls sich der Roboter nicht verschieben lässt. Im Falle einer positiven Antwort werden auf diesem Feld die LEDs zur Navigationshilfe angeschaltet.

Nach dem Empfang einer Query-Reponse Nachricht mit positiver Antwort wird der *Bewegungsbefehl* zum Roboter geschickt. Der Roboter sendet seine Acknowledge-Nachricht

und beginnt zu fahren. Das Feld schickt eine Nachricht vom Typ *RobotState* an das Zielfeld. Der RobotState nimmt eine wichtige Rolle ein und wird im folgenden Abschnitt ausführlich beschrieben.

Empfängt das Zielfeld diese Nachricht, sendet es eine ID-Anforderung an den Roboter, der nach dem Eintreffen auf diesem Feld wieder kommunikationsbereit ist. Ähnlich zur Ausführung von Drehungen antwortet er zuerst mit einem Acknowledge und dann mit seiner ID. Das Feld kann jetzt die LEDs zur Navigationshilfe abschalten und eine Nachricht an das Startfeld schicken, so dass auch dieses seine LEDs zur Navigationshilfe abschaltet. Eine Response-Nachricht wird an den Host-PC gesendet, dadurch ist der Befehl ausgeführt.

RobotState

Das Design von R_3D_3 sieht vor, dass der Roboter nur sehr wenig Kenntnis von seinem *momentanen Zustand* hat. Diese Informationen müssen daher in dem Spielfeld gespeichert werden. Hierzu wird eine Hilfsdatenstruktur mit dem Namen RobotState verwendet, die den momentanen Zustand des Roboters repräsentiert. Der RobotState wird in demjenigen Feld gespeichert, auf dem der Roboter gerade steht. Wird der Roboter zu einem anderen Feld bewegt, wird sein RobotState an das neue Feld gesendet, wie bei der Befehlsausführung oben beschrieben. Dies passiert auch, wenn sich das neue Feld auf derselben Spielfeldbox befindet.

Der RobotState enthält die folgenden Informationen:

PreviousPosition Hier wird die vorherige Position des Roboters abgespeichert, so dass bei Ankunft des Roboters an das alte Feld eine Nachricht zum Ausschalten der LEDs zur Navigationshilfe gesendet werden kann.

UndoneMovement Hier werden nicht abgearbeitete Bewegungsbefehle abgespeichert. Erhält der Roboter beispielsweise einen Drei-Felder-Vorwärts-Befehl, wird dieser Befehl wie folgt in Ein-Feld-Vorwärts-Befehle zerlegt: UndoneMovement wird auf 2 gesetzt und dem Roboter wird ein Ein-Feld-Vorwärts-Befehl geschickt. Das Feld, auf dem der Roboter ankommt, vermindert die Anzahl um eins und schickt dem Roboter wieder nur einen Ein-Feld-Vorwärts-Befehl usw.

UndoneTurns Analog zu den nicht abgearbeiteten Befehlen werden in UndoneTurns die noch anstehenden Drehungen gespeichert. Dies geschieht, wenn der Roboter 90 Grad zu seiner Blickrichtung verschoben werden muss:

- Drehbefehl an Roboter schicken
- Ausführung des Befehls, der Roboter steht nun mit der Front zur Verschiebe-Richtung.
- Im RobotState wird die Drehung zur Wiederherstellung der alten Blickrichtung vermerkt

- Der Roboter erhält den Bewegungsbefehl
- Ausführung des Bewegungsbefehls
- Das Zielfeld schickt dem Roboter den Drehbefehl aus dem RobotState zu

Viewdirection Entspricht der Blickrichtung des Roboters. Sie wird durch sog. Orientierungsbits kodiert. Die Orientierungsbits werden im Routing-Protokoll verwendet. Es handelt sich um zwei Bits, die angeben, welche Koordinate sich ändert und in welcher Richtung (siehe dazu 4.4.3.2)

OptionCards Enthält einen Bitvektor, der angibt, ob der Roboter über die entsprechende Optionskarte verfügt. Hier werden nur die Optionskarten vermerkt, die für das Spielfeld relevant sind. So ist zum Beispiel die Optionskarte Extra-Memory, die dem Spieler eine zusätzliche Programmierkarte zur Verfügung stellt, für das Spielfeld unerheblich. Diese Optionskarte wird vom Client berücksichtigt. Die Optionskarte Rammbock hingegen, die einem geschobenen Roboter einen Schadenspunkt zufügt, kann nur vom Spielfeld berücksichtigt werden.

UnreportedDamage In diesem Feld wird der Schaden gespeichert, der einem Roboter während der Befehls- oder Fabrikelement-Bearbeitung zugefügt wird, aber noch nicht an den Client gesendet wurde.

Factory-Element-Stage Entspricht der vom Roboter abgearbeiteten Fabrikelementstufe (siehe folgender Abschnitt).

4.4.2.3 Fabrikelement-Ausführung

Die Fabrikelement-Gruppen

Die Fabrikelemente von R3D3 müssen von einem verteilten Algorithmus im Spielfeld abgearbeitet werden. Dabei ergibt sich aber ein Problem mit den Optionskarten *"Hochdruckstrahl"* und *"Traktorstrahl"*. Diese Optionskarten ersetzten den Standard-Roboter-Laser und fügen während der Ausführung der Laser keinen Schaden zu, sondern sie verschieben gegnerische Roboter. Dadurch ist es prinzipiell möglich, dass durch die Nachrichtenlaufzeiten unterschiedliche Ergebnisse entstehen, wenn ein Roboter vom Hochdruck und vom Traktorstrahl gleichzeitig getroffen wird. Um dies zu vermeiden, werden diese beiden Optionskarten in je einer getrennten *Fabrikelement-Gruppe* abgearbeitet.

Die Gruppen sind wie folgt eingeteilt:

1. Expressförderbänder, Förderbänder, Schieber, Zahnräder, Pressen
2. Hochdruckstrahl
3. Traktorstrahl
4. Laser, Brenner, Checkpoints und Reparaturfelder.

Der Master stößt diese Gruppen einzeln an, während die eigentlich Ausführung der Stufen in dieser Gruppe dann im Feld selbständig passiert. Wenn der Master zum Beispiel die erste Gruppe der Fabrikelemente aktiviert, führen alle Felder gleichzeitig ihre Fabrikelemente Expressförderbänder, Förderbänder, Schieber, Zahnräder, Pressen aus. Wie dies im Einzelnen geschieht, ist im folgenden Abschnitt nachzulesen.

Fabrikelement-Algorithmus

Wenn der Master dem Spielfeld mittels einer Factory-Element-Nachricht mitteilt, welche Gruppe der Fabrikelemente an der Reihe ist, arbeitet das Spielfeld die Stufen dieser Gruppe wie folgt ab:

Alle Fabrikelement-Stufen dieser Gruppe werden einmal angestoßen. Dabei kann es vorkommen, dass ein Roboter bewegt wird. Meldet sich dieser Roboter auf dem neuen Feld an, erkennt dass Spielfeld, dass es sich in der Fabrikelement-Bearbeitung befindet, und führt alle Fabrikelemente noch einmal aus. Daher ist es wichtig zu wissen, welcher Roboter welches Fabrikelement bereits "miterlebt" hat. Dies geschieht wie bereits erläutert mithilfe des RobotStates.

4.4.3 Protokoll-Stack

4.4.3.1 Einleitung

Im vorhergehenden Abschnitt wurde detailliert beschrieben, wie das Spielfeld seine Aufgaben innerhalb der R3D3-Realisierung wahrnimmt. Dabei wurde aus Gründen der einfacheren Beschreibung davon abstrahiert, dass viele dieser Aufgaben und Aktionen durch Kommunikation durchgeführt werden. In diesem Abschnitt wird die Kommunikation genauer betrachtet, weil sie nicht nur ein wichtiger Bestandteil der Ausführung von Befehlen und Fabrikelementen ist, sondern weil das Spielfeld auch die Kommunikation zwischen den Host-PCs und ihren Robotern zur Verfügung stellt.

Für eine Kommunikation sind *Protokolle* nötig, die die Semantik und das Format der auszutauschenden Informationen festlegen. Wie das ISO/OSI Modell zeigt, hat es sich bewährt, Protokolle in Schichten zu modellieren und zu einem so genannten Protokoll-Stack zusammenzufassen.

Jede Schicht, auch Layer genannt, benutzt dabei zur Erfüllung ihrer speziellen Aufgabe die jeweils nächst tiefere Schicht im Protokoll-Stack.

Der R3D3-Protokoll-Stack hat einen dreistufigen Aufbau wie in Abbildung 4.19 zu sehen.

- Application-Layer: Diese Schicht schließt den Protokoll-Stack nach oben ab. Sie verarbeitet Nachrichten, die für den Spielbetrieb gebraucht werden (Befehle an den Roboter, Fabrikelemente usw.).

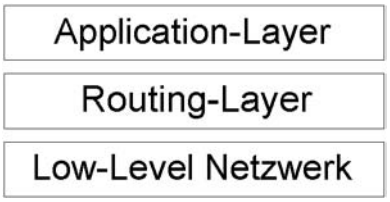


Abbildung 4.19: R_3D_3-Protokollstack

- Routing-Layer: Der Routing-Layer ist dafür zuständig, Nachrichten vom Sender an den richtigen Empfänger weiterzuleiten.
- Low-Level Netzwerk-Layer: Auf dieser Ebene werden Bits physikalisch übertragen und sichergestellt, dass der Empfangspartner diese korrekt empfängt.

Die Interaktion der beschriebenen Schichten soll an dem folgenden Beispiel verdeutlicht werden: Nachdem ein Roboter seinen aktuellen Bewegungsbefehl ausgeführt hat, muss der Host-PC des Spielers benachrichtigt werden. Dazu generiert der Application-Layer der Spielfeld-Box, auf der der Roboter sich befindet, eine Nachricht. Diese Nachricht wird zusammen mit der Adresse des Zieles an den Routing-Layer weitergeben.

Der Routing-Layer ermittelt anhand der Zieladresse den physikalischen Kanal, an den sie Nachricht zu senden ist. Bevor die Nachricht an die nächste Schicht weitergegeben wird, fügt der Routing-Layer den sog. Routing-Header vor die Nachricht. In dem Routing-Header stehen Daten wie zum Beispiel die Zieladresse, die für den Routing-Layer auf dem Zielsystem nötig sind. Die neue Nachricht, bestehend aus Routing-Header und alter Nachricht wird zusammen mit der Bezeichnung des physikalischen Kanals an den Low-Level Netzwerk-Layer übergeben, der die Nachricht dann über die physikalische Leitung überträgt.

4.4.3.2 Routing-Protokoll

Aufgabe

Die Aufgabe eines Routing-Protokolls besteht darin, Nachrichten so *weiterzuleiten*, dass sie am gewünschten Ziel ankommen. Das Routing-Protokoll bekommt von der übergeordneten Schicht eine Nachricht und eine Zieladresse übergeben und muss entscheiden, über welchen *physikalischen Kanal* die Nachricht zu senden ist, so dass sie letztendlich am Ziel ankommt.

Dazu wird eine *Routing-Tabelle* benötigt, mit deren Hilfe die übergebene Adresse einem Kanal zugeordnet werden kann. Hierzu verwendet das R3D3-Routing-Protokoll eine Koordinatensystem-Basierte Struktur.

Abbildung 4.20: Koordinatensystem

Dazu wird jedes Spielfeld in ein virtuelles, zweidimensionales *Koordinatensystem* eingeteilt. Jedes Spielfeld bekommt ein eigenes, eindeutiges Koordinaten-Paar bestehend aus x- und y-Koordinate zugeordnet.

Das bedeutet, dass jede Spielfeldbox insgesamt 9 Koordinaten-Paare bekommt (für jedes Feld ein Paar siehe Abbildung 4.20). In der Routing-Tabelle werden für jeden Nachbarn die Koordinaten seines mittleren Feldes verwendet. Eine beispielhafte Routing-Tabelle könnte so aussehen:

Nachbar	Koordinaten
0	(1,4)
1	(4,1)
2	(1,-2)
3	(-2,1)

Die Entscheidung, an welchen Kanal eine übergebene Nachricht zu senden ist, wird anhand des Euklidischen Abstands getroffen: Bekommt der Routing-Layer eine Nachricht und eine Adresse - die Adresse besteht aus einem Koordinaten-Paar - von der übergeordneten Schicht übergeben, beginnt er damit, für jeden seiner Nachbarn den Abstand zur Zieladresse zu berechnen. Die Nachricht wird dann an den Nachbarn gesendet, der den geringsten Abstand zur Zieladresse hat. Die folgende Tabelle zeigt den Abstand zur Adresse (2, 6):

Nachbar	Koordinaten	Abstand
0	**(1,4)**	**2,34**
1	(4, 1)	5,39
2	(1, -2)	8,06
3	(-2, 1)	6,40

In diesem Fall wird die Nachricht an Nachbar 0 weitergeleitet.

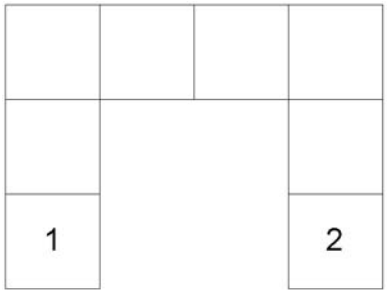


Abbildung 4.21: Spielfeld Aufbau

Das R3D3-Routing-Protokoll kann Nachrichten aber nicht nur an eine bestimmte, durch eine Adresse spezifizierte Spielfeldbox schicken, sondern es unterstützt auch Broadcast-Nachrichten, die im nächsten Abschnitt beschrieben werden.

R_3D_3-Broadcast-Nachrichten

Ein Broadcast ist eine Nachricht, die an *alle* Teilnehmer gesendet wird. Wenn eine Spielfeldbox eine Broadcast-Nachricht empfängt, leitet der Routing-Layer diese an den übergeordneten Layer weiter und sendet sie zusätzlich an alle Nachbarn, außer an den, von dem die Nachricht empfangen wurde. Das R3D3-Routing-Protokoll kennt im Gegensatz zu anderen Routing-Protokollen *zwei* verschiedene Broadcast-Nachrichten.

- Broadcast, der an alle Spielfelder gehen soll. Die Nachricht wird an den übergeordneten Layer weitergeleitet und zusätzlich an alle Nachbarn weitergesendet. Dies ist der übliche Fall, der zum Beispiel während der Init-Phase eintritt. Dabei wird das Adressfeld des Routing-Headers mit der BROADCASTADRESS belegt. Die BROADCASTADRESS ist als `-63` definiert und kommt in keinem möglichen Spielfeldaufbau vor.

- Broadcast, der aufgrund einer Fehlersituation passiert: Wenn eine Spielfeldbox eine kleinere Distanz zum Ziel hat als alle ihre erreichbaren Nachbarn, dann kann sie die Nachricht nur ans Ziel bringen, indem sie als Broadcast geschickt wird. Ein solcher Aufbau ist in Abbildung 4.21 dargestellt. Die Spielfeldbox mit der Nummer Eins versucht eine Nachricht an die Box Nummer zwei zu senden. Bei der Abstandsberechnung stellt sich aber heraus, dass der einizge Nachbar, den die Box Nummer Eins hat, eine größere Distanz hat zum Ziel hat als sie selbst. Die Nachricht muss also als Broadcast gesendet werden.
  Bei *diesem* Broadcast wird die Zieladresse ins Adressfeld der Nachricht eingetragen, so dass die Zielbox diese Nachricht an seinen oberen Layer weiterleitet, alle anderen Boxen diese Nachricht aber nicht an ihren oberen Layer weitergeben, sondern die Nachricht nur weiterleiten.

Wenn der Routing-Layer eine Broadcast-Nachricht empfängt, unterscheidet er die folgenden Fälle:

1. Message-ID der empfangenen Nachricht ist bekannt: Die Nachricht wurde schon einmal empfangen und bearbeitet, daher wird sie nun verworfen.
2. Zielkoordinaten-Paar besteht aus den Koordinaten (BROADCASTADRESS, BROADCASTADRESS): Die Nachricht ist ein Broadcast, der an alle Felder gehen soll. Die Nachricht wird sowohl an den übergeordneten Layer weitergeleitet als auch an alle Nachbarn gesendet.
3. Es handelt sich um einen Broadcast, der durch eine Fehlersituation entstanden ist. Die Nachricht wird an den übergeordneten Layer weitergeleitet, falls die Zieladresse der eigenen Adresse entspricht, sonst an alle Nachbarn weitergesendet.

Die nächsten Abschnitte beschäftigen sich zuerst mit der Init-Phase des Routing-Layers, in der die Koordinaten-Struktur aufgebaut wird, und mit der darauf folgenden Betriebsphase.

Init-Phase

In der Init-Phase wird die Koordinatensystem-Struktur aufgebaut. Die Belegung der Felder mit den Koordinaten geschieht wie folgt: Der *Parkplatz* weist sich selbst die Koordinaten (0, 0) zu und beginnt darauf, den anderen Spielfeldboxen - entsprechend ihrer Position - die Koordinaten zuzuweisen. Dazu sendet der Parkplatz eine Init-Nachricht an alle seine Nachbarn. Die Init-Nachricht enthält ein Koordinaten-Paar für die Spielfeldbox und zwei Bits, die als Orientierungsbits bezeichnet werden. Die Orientierungsbits geben an, welche Koordinate sich ändert, und ob sie erhöht oder vermindert wird. Ihre Belegung und die entsprechende Bedeutung ist der folgenden Tabelle zu entnehmen.

1. Bit	2. Bit	Beschreibung
0	0	x-Koordinate wird verringert
0	1	x-Koordinate wird erhöht
1	0	y-Koordinate wird verringert
1	1	y-Koordinate wird erhöht

Zusätzlich zu den Koordinaten speichert jede Spielfeldbox einen so genannten *Hop-Counter*, der die Entfernung zum Parkplatz angibt. Dies wird dazu benötigt, um den Roboter zum Parkplatz zurückzuleiten. Aus zeitlichen Gründen konnte das Fahren der Roboter zum Parkplatz nicht getestet werden (siehe hierzu 4.5.2).

Der aktuelle Hop-Counter wird in die Init-Nachricht aufgenommen, den jede Spielfeldbox speichert, um eins erhöht und weiterschickt. In der Init-Phase kommt es daher häufig vor, dass zusätzliche Init-Nachrichten geschickt werden müssen, um die Hop-Counter aller Spielfeldboxen richtig zu belegen.

Nach der Init-Phase wechselt das System in die Betriebsphase, die im folgenden Abschnitt beschrieben wird.

Betriebsphase

Nachdem die Init-Phase abgeschlossen ist, gelangt das System in die Betriebsphase. In dieser Phase ist der Routing-Layer passiv, er wartet auf *Eingaben* der über- und untergeordneten Schicht.

Empfängt der Low-Level Layer eine Nachricht, wird diese an den Routing-Layer nach oben weitergeleitet. Der Routing-Layer wird nun aktiv und führt die folgende Fallunterscheidung durch:

1. Die Ziel-Koordinaten der Nachricht stimmen mit den eigenen Koordinaten überein. Daher wird der Routing-Header abgeschnitten und die Nachricht an den übergeordneten Layer weitergegeben.
2. Die Nachricht wurde als Broadcast gesendet.
3. Wenn die Ziel-Koordinaten nicht mit den eigenen übereinstimmen und die Nachricht nicht als Broadcast gesendet wurde, wird die Nachricht an den Nachbarn mit dem geringsten Abstand zum Ziel weitergeleitet.

Falls der übergeordnete Layer eine Nachricht senden will, übergibt er sie zusammen mit der Adresse an den Routing-Layer. Der Routing-Layer führt hier die folgende Fallunterscheidung durch:

1. Die Ziel-Koordinaten der Nachricht stimmen mit den eigenen Koordinaten überein. Die Nachricht wird an den übergeordneten Layer zurückgeleitet.
   Das geschieht beispielsweise dann, wenn bei der Befehlsausführung des Roboters eine Nachricht an ein anderes Feld geschickt wird, wobei die Koordinaten auf derselben Spielfeldbox liegen.
2. Die Nachricht soll als Broadcast versendet werden.
3. Wenn die Ziel-Koordinaten nicht mit den eigenen übereinstimmen und die Nachricht nicht als Broadcast gesendet werden soll, wird die Nachricht weitergeleitet.

4.5 Änderungen Software

4.5.1 Neue Init-Phase

Im ersten Entwurf des Low-Level Netzwerk-Layers war vorgesehen, dass das Protokoll nach dem Einschalten erkennt, an welchen Seiten des Spielfeldes sich ein Nachbar befindet, und an welcher Seite kein Nachbar. Dieser Sachverhalt ist für das Routing-Protokoll von großer Wichtigkeit. Denn für die Abstandsberechnung des Routing-Protokolls dürfen

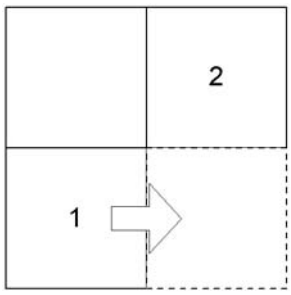


Abbildung 4.22: Routing-Fehler durch falsche Nachbarerkennung

nur tatsächlich vorhandene Nachbarn herangezogen werden. Abbildung 4.22 verdeutlicht das Problem: Spielfeldbox Eins soll eine Nachricht an Box Zwei weiterleiten. Sie hat nur die obenliegende Box als Nachbarn, die rechte Box (als gestrichelte Linie angedeutet) ist nicht vorhanden, obwohl der Routing-Layer glaubt, dass sie vorhanden sei. Sowohl die oben liegende als auch die rechts liegende Box hätten zu Spielfeldbox Zwei den gleichen Abstand, so dass der Routing-Algorithmus irgendeinen Nachbar auswählt. Wenn die Nachricht dadurch nach rechts gesendet wird, wie durch den Pfeil angedeutet, geht die sie verloren, was im Test auch tatsälich zu beobachten war.

Da die Erkennung der Nachbarn zu Beginn der Betriebsphase abgeschlossen sein muss, wurde die Init-Phase des Routing-Layers nachträglich angepasst. Der Routing-Layer verwaltet einen so genannten Nachbar-Status, der drei Werte annehmen kann: Gültig, Warte-Auf-Bestätigung und Ungültig. Eine Spielfeld-Box, die eine Init-Nachricht an einen Nachbar schickt, setzt dessen Nachbarstatus auf Warte-Auf-Bestätigung. Diese Spielfeld-Box empfängt nun die Init-Nachricht und sendet eine Bestätigungsnachricht zurück. Durch den Empfang der Bestätigungsnachricht setzt die Spielfeld-Box den Nachbarstatus auf gültig.

Nach einem festgelegten Timeout werden alle Nachbarn, deren Status auf Warte-Auf-Bestätigung steht, auf ungültig gesetzt.

4.5.2 Wegfall des Hop-Counter

Der Hop-Counter des Routing-Protokolls, der die Entfernung der Spielfeldbox vom Parkplatz angibt, wird nicht mehr benötigt. Dies hängt damit zusammen, dass es durch die kurze Testphase nicht mehr möglich war, das autonome Fahren der toten Roboter zum Parkplatz zu testen. Stattdessen wird der Benutzer aufgefordert, den Roboter vom Spielfeld zu nehmen und zu Beginn der neuen Runde auf das letzte Backup zu stellen, das mithilfe der LEDs markiert wird.

Die Implementierung des Hop-Counters wurde letztes Semester bereits abgeschlossen. Er ist fester Bestandteil der Init-Phase, so dass seine Entfernung nicht nur unverhältnismäßig aufwendig, sondern auch überflüssig wäre.

4.5.3 Neuer Fabrikelement-Algorithmus

Die bisherige Ausführung der Fabrikelemente wies im Feld-Test einige Schwächen auf (so funktionierte beispielsweise das Expressförderband nicht). Daher wurde der Algorithmus zur Abarbeitung der Fabrikelemente überarbeitet.

Die einzelnen Fabrikelemente werden so lange ausgeführt, bis ein Roboter *verschoben* werden muss (beispielsweise durch Förderband oder Schieber). In diesem Fall wird die Abarbeitung erst dann fortgesetzt, wenn der Roboter diese Verschiebung *ausgeführt* hat.

Wird ein Roboter beispielsweise durch ein Express-Förderband verschoben, führt er einen Bewegungsbefehl aus. Wenn er sich auf seinem neuen Feld anmeldet, führt das Spielfeld noch einmal die Expressförderbänder aus. Dadurch, dass im RobotState dieses Roboters die Expressbänder bereits vermerkt sind, bleibt er von dieser Ausführung unberührt. Hat ein weiterer Roboter, der auf dieser Spielfeldbox steht, die Expressförderbänder noch nicht abgearbeitet, ist er nun an der Reihe.

4.5.4 Abstürze des Spielfeldes

Die Spielfeldsoftware stürzt immer noch gelegentlich ab. Dies ist darauf zurückzuführen, dass der *Programm-Stack* so weit anwächst, dass er im Speicher liegende Variablen überschreibt. Dabei ist häufig die Routing-Tabelle betroffen, so dass infolge dessen die Weiterleitung von Nachrichten fehlschlägt. Das Problem konnte aus Zeitgründe vor dem Ende nicht mehr behoben werden, da eine große Umstrukturierung des Codes nötig wäre.

4.6 Zusammenfassung und Bewertung

Es wurden 16 Spielfeldboxen und ein Parkplatz gefertigt. Die vorherigen Abschnitte haben die verschiedenen Aspekte, wie die Mechanik, Elektronik, Kommunikation und Software, des Spielfeldes detailliert beschrieben.

Im Bereich der Mechanik hat es sich als gute Entscheidung erwiesen, die Spielfeldboxen selber zu konstruieren. Es zeigte sich, dass die Verwendung von Aluminium-Winkeln für die Unterkonstruktion in Verbindung mit Platinen als Seitenflächen eine sehr gute Kombination darstellt. Lediglich die Entscheidung, den Strom über die Winkel zu leiten, erwies sich als ungünstig, da die Kontaktflächen zu schnell oxidieren.

Die Gestaltung der Oberfläche der Spielfeldboxen durch durchsichtiges Makrolon mit darunter liegendem Papier für das grafische Design, erwies sich ebenfalls als vorteilhaft. Allerdings musste diese Konstruktion nachträglich abgestützt werden, da der überarbeitete Roboter ein höheres Gewicht hatte.

Die verwendeten Kommunikationsprotokolle sind speziell für den Einsatz in der R_3D_3 Realisierung konzipiert und implementiert worden. Dadurch wurde sichergestellt, dass

die an die Kommunikation gestellten Anforderungen durch die Protokolle erfüllt werden. Für die Feld zu Feld Kommunikation wurde ein interrupt-basiertes Protokoll mit einer Daten- und einer Clockleitung umgesetzt, das Pakete bis zu einer Größe von 255 Bits übertragen kann. Der Roboter kommuniziert durch LEDs und Phototransistoren mit dem Spielfeld. Da diese Kommunikation nur über eine Leitung verfügt, muss zu Begin eines jeden Übertragungsvorgangs eine ausgedehnte Synchronisation stattfinden. Lediglich bei der Kommunikation zwischen Host-PC und Spielfeld wurde auf die serielle RS232-Schnittstelle zurückgegriffen.

Das Design der Spielfeldsoftware ist durch eine Trennung von Kommunikationsschicht, die in Form eines Protokollstack implementiert ist, und Anwendungsschicht geprägt. Dadurch wurde eine einfache Portierbarkeit der Kommunikationsschicht auf die verschiedenen Komponenten wie Spielfeld, Hostschnittstelle und Host erreicht. Leider war der Code zu groß für die verwendeten Mikrokontroller im Spielfeld, so dass sich immer wieder während der Laufzeit Fehler durch einen Überlauf des Stacks ereignen. Dieses Problem konnte bis zum Ende der Projektgruppe nicht behoben werden und ist auf die folgenden Punkte zurückzuführen:

- Das Spielfeld verfügt über deutlich weniger Speicher als der Host-PC, muss aber dennoch eine viel größere Anzahl an Aufgaben bewältigen. Dadurch wurde der Code für die verwendete Hardware zu groß.
- Die Programmierung wurde, wie in der Softwaretechnik, üblich, im Hinblick auf ein gutes, leichtverständliches Design optimiert. Im vorliegenden Fall wäre es nötig gewesen, das Design auf die verwendeten Hardware-Resourcen vor allem im Bereich Codegröße zu optimieren.

5 Roboter

In diesem Kapitel werden die Roboter als Spielfiguren des Spieles beschrieben. Dazu wird in Abschnitt 5.1 zunächst auf den ersten Entwurf eingegangen und dessen Probleme, weshalb er verworfen werden musste. Der detaillierte Aufbau wird im Zwischenbericht [ZWB06] beschrieben.

Abschnitt 5.2 beschreibt den Aufbau des zweiten Entwurfes. Die Software des Roboters wird in den Abschnitten 5.3 und 5.4 vorgestellt, zuerst der Stand der Software für den ersten Entwurf, der im Zwischenbericht beschrieben wurde, und dann die Änderungen, die für den zweiten Entwurf vorgenommen werden mussten.

Abschnitt 5.5 gibt eine abschließende Zusammenfassung und Bewertung der Entwürfe und legt die Stärken und Schwächen der Konzepte dar.

5.1 Alter Roboter

Dieser Abschnitt gibt einen Überblick über den Aufbau des ersten Roboterentwurfes. Detallierte Informationen enthält der Zwischenbericht [ZWB06]. Abschließend wird eine Bewertung des ersten Ansatzes gegeben.

5.1.1 Beschreibung des Roboters

Im Laufe des Projekts musste aufgrund einiger Komplikationen das erste Konzept des Roboters verworfen und ein neues entwickelt werden. Einige Komponenten des zuerst entworfenen Roboters konnten hierbei jedoch für den neuen Entwurf übernommen werden. Die Ideen und Konzepte, die bei diesem Roboterentwurf angewendet wurden, werden in den folgenden Abschnitten kurz erläutert, und in Abschnitt 5.1.2 wird eine Bewertung des Roboterentwurfs vorgenommen werden.

5.1.1.1 Anforderungen

Zu den Anforderungen des Roboters gehört zum Einen, dass Spielzüge autonom ausgeführt werden. Dazu ist es notwendig, dass der Roboter so präzise wie möglich über das Spielfeld navigieren kann. Zum Anderen muss der Roboter dazu mit dem Spielfeld kommunizieren können, um sowohl Befehle empfangen, als auch Nachrichten an das Spielfeld senden zu können (vgl. Kapitel 5.3.2).

Da die Spielfeldboxen aus quadratischen, 5 cm großen Spielfeldern bestehen (vgl. Kapitel 4), und sich der Roboter auf diesen bewegen und vor Allem drehen können muss, ergibt sich eine weitere Anforderung an die Größe des Roboters. Um bei Bewegungen nicht mit Robotern auf anderen Feldern zu kollidieren, darf der Durchmesser des Roboters $4,5\,cm$ nicht übersteigen. Dies konnte sowohl bei dem ersten, als auch bei dem zweiten Entwurf des Roboters umgesetzt werden.

Des Weiteren wurden Anforderungen bezüglich der Energieversorgung definiert. Die Energiequelle muss für einen langen Spielbetrieb geeignet sein, um einen häufigen Batteriewechsel während des Spielbetriebs möglichst zu vermeiden. Hier wurde eine umweltfreundliche regenerierbare Lösung bevorzugt.

5.1.1.2 Komponenten

Im Zwischenbericht [ZWB06] wurden einige Standardlösungen für die einzelnen Komponenten und eine Begründung für die jeweilige Entscheidung ausführlich beschrieben. An dieser Stelle soll eine kurze Zusammenfassung der Komponenten erfolgen.

Um möglichst einfach die Anzahl der Schritte ermitteln zu können und um den Roboter präzise nach vorn steuern zu können, fiel die Wahl auf einen Schrittmotor. Einen kleinen und im Verhältnis zur Größe leistungsstarken und preisgünstigen Schrittmotor stellte der SP0618M0104 [Nan04] der Firma Nanotec mit einem Durchmesser von 6 mm dar.

Zusammen mit der Auswahl des Motors wurde ein Getriebe gewählt, das auf diesen Motor passte. Aufgrund der allgemeinen schlechten Verfügbarkeit von Getrieben in kleinen Stückzahlen erschien das Bausatzgetriebe G30 [Mik03] als geeignet. Jedoch ergaben sich Probleme mit dieser Lösung, sowohl was den Zusammenbau, als auch den Einsatz anging. Weitere Ausführungen zu dieser Problematik sind in 5.1.2 zu finden.

Die Auswahl der Räder, wie im Zwischenbericht [ZWB06] beschrieben, erwies sich für den dauerhaften Einsatz auf der Makrolonoberfläche der Spielfeldboxen als nicht geeignet, weil sie immer wieder rutschten. Aufgrund dessen wurde ein alternativer Ansatz entwickelt, der sich auch für den neuen Roboter als gute Lösung erwiesen hat. Es wurden Messingfelgen gedreht, in die jeweils in der Mitte ein Loch für die Achse gebohrt wurde. Auf diese Felgen wurden dann herkömmliche O-Ringe aus Gummi gezogen, die für eine deutlich bessere Haftung auf dem Makrolon sorgten.

Da sich die Getriebe nicht als strapazierfähig genug erwiesen, wurde in Erwägung gezogen, einen Radkasten einzusetzen, der die auf das Getriebe einwirkenden Kräfte reduzieren sollte. Da diese Radkästen aufgrund von Kosteneinsparungen jedoch nur aus Messing gefertigt werden konnten und ein präzises Arbeiten an der Fräse wegen der geringen Festigkeit des Materials im Vergleich zu Aluminium nicht möglich war, erwies sich auch der Radkasten als unzureichende Lösung. Es war vorgesehen, diese Kästen an der Bodenplatte festzuschrauben und die beiden Kastenteile miteinander zu verschrauben, um die Getriebe immer wieder auseinanderbauen zu können. Jedoch waren die Kastenteile

entweder nicht fest genug aneinander gesetzt, so dass die Zahnräder und die Schnecke des Getriebes nicht ineinandergriffen, oder sie waren so fest aneinandergeschraubt, dass die Zahnräder des Getriebes sich überhaupt nicht mehr drehten. Aufgrund dieser Probleme musste der Entwurf als zu empfindlich eingestuft und verworfen werden.

Für die Energieversorgung des Roboters wurden statt nicht wiederaufladbarer Batterien Akkus verwendet. Als die für uns günstigste Technologie in diesem Bereich stellten sich die Lithium Polymer Akkus heraus, da sie über eine hohe Energiedichte verfügen. Das Modell Kok 1200 der Firma Wes [Wes04] stellte auch aufgrund seiner günstigen Maße und der Möglichkeit, durch ein Ladegerät ULTRAMAT 12 [Gra04] Aufladungen mehrerer Akkus gleichzeitig vorzunehmen, eine gute Lösung dar. Diese konnten auch in dem neuen Roboterentwurf eingesetzt werden.

Eine Übersicht der Komponenten ist in Abbildung 5.1 zu finden.

Einheit	Baustein	Datenblatt
Mikrocontroller	AT91SAM7S32	[Atm05]
Motoren	SP0618M0104	[Nan04]
Motortreiber	HIP4020	[Int05]
Getriebe	Bausatzgetriebe G30	[Mik03]
Energie-Versorgung	Kok 1200	[Wes04]
Festspannung	KF33BD	[STM04]

Abbildung 5.1: Komponentenübersicht des ersten Roboterentwurfs

5.1.1.3 Mechanisches Konzept

Da sich der mechanische Aufbau im Vergleich zum Stand des Zwischenberichts [ZWB06], außer der Änderung der Räder und der Radkästen, wie bereits im vorherigen Abschnitt erwähnt, nicht gravierend verändert hat, wird hier eine kurze Zusammenfassung gegeben. Die $4\,mm$ dicke Bodenplatte aus Aluminium enthielt Aussparungen für die Räder und Getriebe, die Leuchtdiode und die Fototransistoren, den Akkuladestecker und den Schalter.

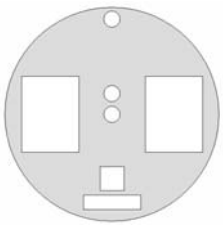


Abbildung 5.2: Grundriss der Bodenplatte mit Aussparungen

Da die beiden Räder getrennt voneinander angesteuert werden sollten, waren zwei Achsen vorhanden, die in der Mitte nicht verbunden waren. Diese wurden in die Bodenplatte, in ein dafür gebohrtes Loch eingelassen. Zur Führung der Achsen in den Aussparungen dienten Aderendhülsen.

Auf der Achse saß dann jeweils das Getriebe, auf dem der Motor steckte. Um den Motor, der einen Achsdurchmesser von 1 mm hatte, mit der Getriebeschnecke, die einen Innendurchmesser von $0,8$ mm hatte, verbinden zu können, musste deren Öffnung passend aufgebohrt werden.

Zur Gewährleistung der Kommunikation mit dem Spielfeld befand sich der erste Fototransistor genau in der Mitte der Bodenplatte und der zweite am vorderen Rand des Roboters. Die entsprechenden Löcher in der Bodenplatte hatten einen Durchmesser von circa $3,5mm$, damit die Leuchtdiode und die Fototransistoren passgenau in diese gesteckt werden konnten, um eine zusätzliche Befestigung zu sparen.

Für den Akkuladestecker wurde in die hintere Hälfte der Bodenplatte eine Aussparung mit den Maßen 5 mm x 5 mm gefräst. Direkt dahinter konnte der Schalter, mit dem der Roboter von außen an- und ausgeschaltet werden konnte, in einer $0,3cm$ x $1,2cm$ großen Aussparung in der Platte eingesetzt werden.

Damit der Roboter auf den zwei Rädern das Gleichgewicht halten konnte, wurde die Bodenplatte auf der Unterseite mit runden Filzgleitern versehen, die geviertelt und je zwei Viertel vorn und hinten zur Stützung aufgeklebt wurden. Um den Akku mittig in dem Roboter platzieren zu können, wurde eine Winkelkonstruktion aus Kunststoff gebaut, die an der Bodenplatte befestigt wurde.

5.1.1.4 Elektronisches Konzept

Die elektronischen Komponenten wurden auf drei verschiedenen Platinen angeordnet. Die Hauptplatine enthielt die Mikrocontrollerschaltung und die Steuerung eines Motors, während die Nebenplatine die Spannungsversorgung und die Steuerung des anderen Motors enthielt. Auf der Bodenplatine waren, wie bereits in dem vorherigen Abschnitt erwähnt, die optischen Kommunikationsschnittstellen untergebracht.

Die Haupt- und Nebenplatine wurden an den beiden Seiten des Akkus untergebracht, und die Bodenplatine wurde in die Bodenplatte gesteckt. Eine Übersicht der Aufteilung der Platinen ist in Abbildung 5.3 zu finden.

5.1.2 Bewertung des Konzepts

Mit dem vorgestellten Roboterkonzept ist es möglich, die gestellten Anforderungen größtenteils zu erfüllen: Durch die Steuerelektronik ist es möglich, autonom Bewegungen auszuführen, die Kommunikation mit dem Spielfeld erfolgt über den vorgestellten optischen Weg via LEDs und Fototransistoren, Navigierbarkeit garantieren die getrennt

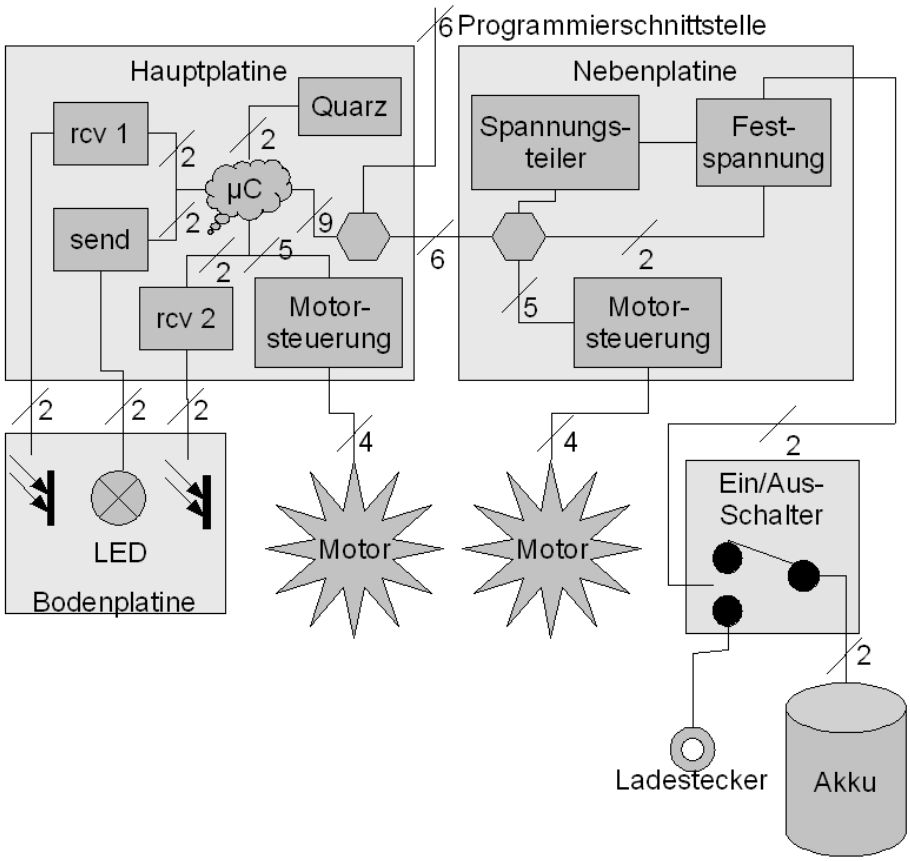


Abbildung 5.3: Platinenaufteilung

angesteuerten Achsen, Schrittweitenkontrolle ist durch den Einsatz der gut beherrschbaren Schrittmotoren möglich, lange autonome Laufzeiten durch die verwendete Akku-Technologie.

All diese Komponenten sind durch entsprechenden Entwurf und passende Anordnung auch zu einem Gesamtentwurf, der die geforderten Größenbeschränkungen einhält, kombiniert. Das einzige Problem bereitete die synchrone Steuerung der beiden Antriebsräder, was auf die schlecht beherrschbaren Laufeigenschaften der Bausatzgetriebe zurückzuführen war, womit die Navigierbarkeit und damit die Grundlage eines erfolgreichen Roboterkonzepts genommen ist.

Um in diesem Punkt ein anforderungserfüllendes Ergebnis zu erzielen, war es nötig, einen veränderten mechanischen Aufbau als Grundlage eines alternativen Entwurfes zu wählen, welcher im folgenden Abschnitt beschrieben wird.

5.2 Neuer Roboter

Dieser Abschnitt beschreibt die Entwicklung und den Aufbau des im zweiten Semester entwickelten Roboters. Zunächst wird die Mechanik dieses Roboters beschrieben. Diese unterscheidet sich zum Teil sehr stark von der vorherigen Entwicklung. Aber auch die Elektronik, die anschließend beschrieben wird, weist Unterschiede auf, so dass diese neu entwickelt wurde. Dabei konnte auf Erfahrungen, die während der Entwicklung der Elektronik des Roboters aus dem ersten Semester gesammelt wurden, zurückgegriffen werden. Darauf folgt die Beschreibung der Entwicklung und Umsetzung des Design des Roboters. Diese beinhaltet insbesondere das Design der Hülle des Roboters.

5.2.1 Mechanik des Roboters

Aufgrund eines Fehlkonzepts des ersten Roboterentwurfs (siehe Kapitel 5.1) war es nötig, einen komplett neuen Ansatz zu verfolgen. Dabei flossen viele Erfahrungen aus den vorherigen Roboteraufbauten in das neue Konzept ein. In den nachfolgenden Absätzen werden die mechanischen Details erläutert. Dabei wurde insbesondere auf folgende Punkte Wert gelegt:

- Robuste Konstruktion
- Einfache (De-)Montierbarkeit
- Lange Laufzeit

Der Motorblock stellt die Halterung für die Kommunikation, die LEDs für die Schrittberechnung und die Motoren dar. Desweiteren beinhaltet er die Elektronik und die Räder.

Für die Kommunikation mit dem Spielfeld werden 4 LEDs und ein Fototransistor benötigt (vgl. Absatz 5.2.2). Die LEDs werden in einem Kreis um den Transistor angeordnet und befinden sich auf der Unterseite des Blocks. Der Laufkorrektur-Algorithmus benötigt einen weiteren Fototransistor, dieser befindet sich abgesetzt am Ende des Blocks.

Die Schrittberechnung benötigt 2 LEDs und 2 Fototransistoren. Diese sind an der Blockseite nebeneinander angebracht. Beide LEDs sitzen auf jeder Seite am äusseren Rand.

Die Motoren befinden sich rechts bzw. links der Hauptachse und werden von oben angebracht. Zur Durchführung der Motorachse existieren auf der Oberseite 2 Löcher mit dem Durchmesser $5\,mm$. Die Befestigung eines Motors erfolgt durch zwei Senkkopfschrauben. Diese Löcher befinden sich über- und unterhalb des Motorachsenloches und sind für den Senkkopf der Schraube ausgespart.

Zur Befestigung der Radachse wird die Bohrung in der Seite des Motorblocks benötigt.

Für die Kabel des Transistors und der hinteren LED aus dem unteren Teil des Blocks werden Bohrungen bis zur Oberseite benötigt, da ein vorzeitiges seitwärtiges Herausziehen der Kabel aus dem Block, wie bei den anderen 3 LEDs, nicht möglich ist. Hierbei

Abbildung 5.4: Ansichten des Motorblocks

muss beachtet werden, dass sich auf der Oberseite die Akkus befinden. Somit wird ein vertiefender Kanal zur Kabelführung auf der Oberseite benötigt. Die Kabel des hinteren Fototransistors werden zwischen den Motoren verlegt. Der Kabelkanal wird wegen des geringen Platzes zwischen Motorblock und Gehäuse benötigt.

Die Platine des Roboters wird an dem Abstandshalter an der hinteren Seite festgeschraubt. Dieser hat abgeschrägte Kanten, um einen Kontakt der Leiterbahnen durch den Abstandshalter zu verhindern.

5.2.1.1 Antriebssatz

Der Antriebssatz ist der Kern des Roboters. Dies ist auch die Stelle gewesen, woran der erste Roboter gescheitert ist. Jeder dieser Roboter benötigt zwei Antriebssätze, einen für jedes Rad. Dabei besteht jeweils ein Antriebssatz aus folgenden Elementen:

- Antriebsmotor
- Getriebe
- Rad

Antriebsmotor

Als Antriebsmotor dient ein Gleichstrommotor mit bereits vom Hersteller aufgesetztem (Planeten-)Getriebe. Der Hauptgrund für die Wahl ist auf die schnelle Beschaffbarkeit eines Motors mit geeigneten Abmessungen zurückzuführen. Dabei hat ein Gleichstrommotor auch diverse technische Vorteile gegenüber einem Schrittmotor. Dazu zählen eine gute Motorleistung bei geringer Größe und niedrigem Stromverbrauch. Als bedeutenster Nachteil muss auf die von Hause aus fehlende präzise Schrittansteuerung hingewiesen werden. Dies ist bei dem neuen Robotor alternativ gelöst. Details sind dazu im Nächsten Paragraphen zu lesen.

Wie oben bereits kurz erwähnt, ist im Lieferzustand des Motors vorne am Motor ein Platengetriebe aufgesetzt. Dies hat eine Übersetzung von 4:1, dementsprechend hat die Antriebswelle hier schon eine Verstärkung von annähernd 4-facher Kraft.

Schrittzählung und Präzision

Für das Fahren des Roboters ist eine schrittweise Ansteuerung des Motors notwenig, da sonst der Roboter nicht gerade laufen würde. Das liegt daran, dass baugleiche Gleichstrommotoren nicht unbedingt gleich schnell laufen müssen. Die Software benötigt aber beim Fahren und besonders beim Fehler korrigieren eine relativ verlässliche Schrittansteuerung. Siehe dazu Paragraph 5.4.4.

Der Motor wird dabei um eine Messeinheit ergänzt, die es der Software ermöglicht, eine viertel Motorumdrehung zu erkennen. Dazu wird auf die Antriebswelle ein Vierkantspiegel angebracht, und von der Seite eine LED/Phototransistor-Kombination. Der Vierkantspiegel ist ein aus Aluminium gefräster 7x7x4mm Block. An den Kanten ist Spiegelfolie aufgeklebt, um für eine höhere Reflexion zu sorgen. Dabei strahlt die LED Licht senkrecht Richtung Antriebswelle. Der Angebrachte Spiegel lenkt das Licht in eine andere Richtung ab, je nachdem wie die Position der Welle gerade ist. Dabei trifft das Licht auch auf den Phototransistor, vier Mal pro Umdrehung, je Spiegel einmal. Diese Impulse kann der Controller erkennen und dadurch den Motor schrittweise ansteuern.

Getriebe

Die normale Aufgabe eines Getriebes ist die Kraftverstärkung und evtl. eine Geschwindigkeitsreduktion. Dies gilt auch für das bereits oben beschriebene Planetengetriebe, welches fest mit dem Motor verbunden ist. Im Gegensatz dazu dient das zweite Getriebe einem anderen Zweck: Die Erhöung der Fahrpräzision und der Umlenkung der Achs-Drehbewegung um 90°, da der Motor aufgrund der Baugröße senkrecht eingebaut werden muss. Das zweite Getriebe ist ein Schneckengetriebe mit einer Übersetzung von 20:1, welches genau diese Anforderungen erfüllt.

Durch die in den letzten beiden Abschnitten beschriebenen Methoden wird damit eine Präzision je Schritt von ungefähr 1,2mm Bewegung je Rad erreicht.

Rad

Der Roboter muss über die glatte Oberfläche des Makrolons fahren können, daher benötigt das verwendete Rad eine entsprechende Bodenhaftung. Zum einen ist ein Vorteil des neuen Roboters dabei das enorm erhöhte Gewicht. Zum andern konnten die Erfahrungen des ersten Roboters genutzt werden. Als Reifen werden dabei gedrehte Messingfelgen, aus eigener Fertigung, mit aufgespannten Gummiringen genutzt. Diese werden mit Zinn auf die Zahnräder des Schneckenrades gelötet.

Fazit

Durch die oben beschriebenen Methoden der Antriebsrealisierung sind sehr solide Fahreigenschaften geschaffen.

Die gute Haftbarkeit und Antriebskraft des Roboters zeigt sich sogar in subjektiven Versuchen. Dabei kann der Roboter sogar aus eigener Kraft, Lasten vor sich her schieben wie Kartons, Dosen o.ä.

5.2.1.2 Gehäuse

Zum Zusammenhalten und Verstauen der Einzelteile dient ein rundes Aluminium-Gehäuse. Dieses stammt noch aus dem ersten Roboteraufbau. Da der Roboter die Spielfelder nicht überschreiten darf, ist der Äussere Durchmesser des Gehäuses $45mm$. Das etwa $70mm$ hohe Gehäuse wird als externer Auftrag aus einem ganzen Stück gedreht, da sich bei Recherchen kein passendes Material finden liess. Entsprechend für den neuen Roboter angepasst werden 2 Löcher gebohrt, um die Radachse und den damit verbundenen Rest am Gehäuse zu befestigen. Weitere Befestigungspunkte sind nicht notwendig, da der restliche Teil des Roboters sehr gut in das Gehäuse passt und es dabei nicht zu Verwacklungen kommt.

5.2.1.3 Akku

Für den neuen Roboter werden aus Gründen der Wiederverwendung dieselben Akkus wie beim ersten Roboter verwendet. Lediglich werden hier jeweils 2 Stück benötigt, um die erforderliche Spannung für die Motoren zu erreichen. Die Akkus sind in Abschnitt 5.1.1.2 näher erläutert.

5.2.1.4 Montage der Einzelteile

Zuletzt wird der Roboter noch zusammengebaut. Dazu sind zusammenfassend folgende Einzelteile notwendig, die jeweils in den referenzierten Kapiteln ausführlich beschrieben sind.

1. Motorblock (1x) (siehe Kapitel 5.2.1)
2. Antriebssatz (2x) (siehe Abschnitt 5.2.1.1)
3. Elektronik (1x) (siehe Abschnitt 5.2.2)
4. Gehäuse (1x) (siehe Abschnitt 5.2.1.2)
5. LiPo Akku (2x) (siehe Abschnitt 5.2.1.3)

Als erstes wird die Radachse in den Motorblock eingesetzt, an der später die Räder aufgehängt werden. Dabei sollte der Motorblock schwarz angespritzt sein, um später Reflexionen im Messystem zu vermeiden.

Nun müssen die Leuchtdioden von unten in den Motorblock geschoben werden. Die Anschlussdrähte werden vorher mit etwas Schrumpfschlauch isoliert. Der Schrumpfschlauch und die starren Drähte sorgen auch für den guten Halt im Motorblock, so dass die Leuchtdioden nicht herausfallen können.

Darauf folgend werden die Motoren von oben auf den Motorblock montiert und von unten festgeschraubt. Dabei ist auf lockeren Sitz der Drehachse zu achten. Nun kommen die mit den Vierkantspiegeln bestückten Getriebeschnecken auf die Motorachse. Diese werden mit einer Madenschraube an der Seite befestigt. In Abbildung 5.5 ist der daraus entstehende Aufbau zu sehen.

Abbildung 5.5: Motorblock mit Motor, Achse und einer Getriebeschnecke

Als nächster Schritt werden die Räder auf die Achsen gesteckt und mit einem Sprengring in Position gehalten. Dabei sollte mit etwas Getriebefett gearbeitet werden, um den Lauf des ganzen Aufbaus zu erleichtern. Daraufhin wird die Platine auf der Rückseite des Motorblocks befestigt. Zusätzlich ist es nötig, die Leuchtdioden und Phototransistoren an die Elektronik anzuschliessen. Nun ist der eigentliche Innenaufbau des Roboters fertiggestellt, wie er in Abbildung 5.6 zu sehen ist.

Um nun einen fahrfähigen Roboter zu bekommen, muss nun noch der Innenaufbau des Roboters in das Aluminiumgehäuse geschoben werden. Der Akku wird von oben zwischen Elektronik und Motoren fixiert und angeschlossen. Das Ergebnis ist dann in Abbildung 5.7 zu sehen.

Abbildung 5.6: Das komplett aufgebaute Innenleben des Roboters, noch ohne Akku

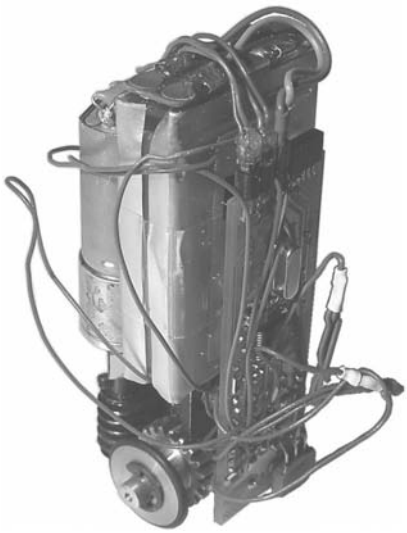


Abbildung 5.7: Der komplett aufgebaute Roboter ohne Gehäuse

5.2.2 Elektronik des Roboters

Die Elektronik des neuen Roboters enthält als Hauptkomponente den schon im ersten Entwurf verwendeten Mikrocontroller AT91SAM7S32 [Atm05] von Atmel. Dieser Mikrocontroller benötigt Versorgungsspannungen von $1,8\,V$ und $3,3\,V$. Zur Bereitstellung der $1,8\,V$ besitzt er einen internen Spannungsregler, so dass dem Mikrocontroller nur noch die $3,3\,V$ bereitgestellt werden müssen. Der entsprechende Aufbau für die Spannungsversorgung wurde aus der Schaltung des vorigen Roboters übernommen. Es wird der

Spannungsregler KF33BD [STM04] eingesetzt, der zum Betrieb noch zwei Kondensatoren von $1\,nF$ im Eingang und $2,2\,\mu F$ im Ausgang benötigt.

In der Schaltung (siehe Abbildung 9.3) befindet sich ein Stecker für die Verbindung mit dem Akku, eine Buchse für die Verbindung mit dem Akkuladegerät und ein Schalter. Der Schalter ermöglicht die Umschaltung von normalen Betrieb auf die Ladung des Akkus. Der Akku wurde ebenfalls aus dem früheren Konzept für den Roboter übernommen. Es ist der Lithium Polymer Akku [Wes04] Kok 1200 von Kokam mit einer Spannung von $4,2\,V$. Für diesen war das Ladegerät ULTRAMAT 12 [Gra04] schon am Lehrstuhl vorhanden. Für die Kontrolle des Akkuladestands ist ein Spannungsteiler vorgesehen, dessen Eingangsspannung die Akkuspannung ist und dessen Ausgangsspannung durch den AD-Wandler des Mikrocontrollers gemessen wird.

Zum Betrieb des Mikrocontrollers sind weitere Komponenten in der Schaltung enthalten, wie ein Quarz mit einer Frequenz von $12\,MHz$ und weitere Kondensatoren für den internen Spannungsregler zur Reduktion von Ripple[1]. Außerdem sind Entstörkondensatoren nahe am Mikrocontroller zwischen der Versorgungsspannung und Masse geschaltet.

Für die Ansteuerung der beiden Gleichstrommotoren (siehe 5.2.1.1) werden zwei Motortreiber HIP4020 [Int05] von Intersil eingesetzt. Diese Motortreiber wurden auch schon in der ersten Entwicklung des Roboters eingesetzt und getestet. So existierten schon Erfahrungen mit der Ansteuerung der Motortreiber durch den Mikrocontroller, auf welche zurück gegriffen werden konnte. Der Unterschied ist nur, dass keine Schrittmotoren, die jeweils zwei HIP4020 benötigten, eingesetzt wurden, sondern Gleichstrommotoren. Diese werden nur mit jeweils einem HIP4020 angesteuert.

Der Nebeneffekt beim Einsatz von Gleichstrommotoren ist die nicht wie bei Schrittmotoren vorhandene Kontrolle über die Umdrehung der Motorwelle. Zur Kontrolle der Umdrehung wird eine optische Lösung eingesetzt. Dabei wird ein auf der Welle montierter quaderförmiger Aluminiumblock durch eine LED beleuchtet. Die Reflexion an dem Block wird durch einen Fototransistor registriert. Das reflektierte Licht zum Fototransistor ist am stärksten, wenn eine Seite des Blocks senkrecht zum Fototransistor liegt. Die entstehenden Pegel bei der Rotation des Blocks können direkt vom Mikrocontroller registriert werden. Durch diesen Aufbau ist eine Zählung von viertel Umdrehungen der Motorwelle möglich.

Für die optische Kommunikation mit dem Spielfeld und die Navigation sind jeweils ein Fototransistor in der Mitte des Motorblocks (siehe auch 5.2.1) und vorn senkrecht zur Achse der Räder angebracht. Für die Kommunikation sind um den Fototransistor in der Bodenmitte des Roboters vier LEDs positioniert. Vier LEDs sind nötig, weil eine einzelne LED, die sich wegen des Fototransistors nicht mittig befinden kann, sich abhängig von der Blickrichtung des Roboters nicht zwingend über dem im Feld angebrachten Fototransistor befinden muss. Die vier LEDs werden mit entsprechendem Vorwiderstand direkt durch den Mikrocontoller angesteuert. Die Fototransistoren beeinflussen den Zustand der ihnen

[1] Welligkeit

nachgeschalteten Transitoren, die für gut registierbare Pegel für den Mikrocontroller sorgen.

Weiterhin enthält die Schaltung eine Buchsenleiste für die Progammierung des Mikrocontollers. Die Programmierung findet über eine RS232-Schnittstelle eines PC statt. Der benötigte Pegelwandler von RS232-Pegeln zu $3,3\,V$ Pegeln wurde auf eine kleine Programmierplatine ausgegliedert. Außerdem ist ein Pin des Mikrocontollers über einen Widerstand mit einer zweiten Buchsenleiste verbunden, deren anderer Anschluss an die Versorgungsspannung von $3,3\,V$ gelegt ist. Hier wird eine LED, die in der Haube (siehe Abschnitt 5.2.3) des Roboters befestigt ist und abhängig von der Farbe der Hülle blau, gelb, rot oder grün ist, mit dem Mikrocontroller verbunden. Abhängig von der Farbe der LED werden auch unterschiedliche Vorwiderstandwerte benötigt.

Alle Bauelemente sind auf einer doppelseitigen Platine mit einer Fläche von $22x65\,mm$ untergebracht. Die benötigten Platinen wurden in Eigenarbeit durch die PG-Mitglieder geroutet, belichtet, geätzt und bestückt, so dass auf einen professionellen Platinenservice verzichtet werden konnte. Das Layout und die Bestückung befinden sich im Anhang 9.2.

5.2.3 Design des Roboters

Das Design des Roboters sollte sich an den Titel unserer PG R_3D_3 und damit an dem Aussehen des Roboters R2D2 aus Star Wars orientieren. Darum wurde eine Zylinderform für die Aussenhülle des Roboters verwendet.

Der Zylinder wurde aus Aluminium gedreht und entspricht mit einem Aussendurchmesser von $45\,mm$ den Anforderungen an die Maße des Roboters, wie durch den PG-Antrag vorgegeben. Um einen oberen Abschluß des Rohres zu finden und die herausragenden Roboterteile zu schützen und zu verdecken, wird eine Haube auf den Zylinder gesetzt. Die Haube sollte leicht abnehmbar sein, da dies für den Aufbau des Roboters und eventuelle Veränderungen an diesem nötig ist. Trotzdem soll sie fest auf dem Roboter sitzen. Für diese Haube wurde der untere Teil eines eiförmigen Styroporstücks verwendet, der ausgehöhlt und farbig gestaltet wurde. Die Aushöhlung war aufgrund der über das Aluminiumrohr hinausragenden Teile des Roboters notwendig. In diesem Zusammenhang wurde jedoch nur soviel Material aus dem Styropor herausgenommen, dass der feste Sitz der Haube gewährleistet werden konnte. Mit Acrylfarben erhielten die Hauben einen silberfarbenen Anstrich und zur Unterscheidung jedes einzelnen Roboters wurde am unteren Rand der Haube ein farbiger Streifen in den Farben rot, gelb, grün und blau aufgetragen.

Des Weiteren wurde der Aluminiumzylinder mit einer Folie beklebt, die mit einem Aufdruck versehen wurde, der sich an das Aussehen des R2D2 anlehnt. Auch hier finden sich die Farben der Hauben des Roboters wieder.

Um im Spielbetrieb gut erkennen zu können, wo sich die Vorder- und Rückseite des Roboters befindet und um sehen zu können, ob der Roboter Daten empfängt, wurde zusätzlich

eine LED in der Farbe des Roboters in die jeweilige Haube eingesetzt, die bei der Kommunikation zwischen Roboter und Spielfeld zu blinken beginnt. Die Abbildung 5.8 zeigt den fertigen Roboter.

Abbildung 5.8: Der fertige Roboter

5.3 Software

Obwohl sich die Mechanik des Roboters - wie in den vorangegangenen Abschnitten beschrieben - vom ersten Prototypen bis zur endgültigen Version drastisch änderte, waren an der Software nur geringe Änderungen notwendig. Während im Abschnitt 5.4 die Änderungen an der Software näher betrachtet werden, umreißt dieser Abschnitt kurz die verbliebenen Softwareelemente, wie sie schon im Zwischenbericht [ZWB06] vorgestellt wurden.

5.3.1 Zustandsmaschine des Roboters

Grundlegende Designentscheidung der Projektgruppe R_3D_3 war, die Spielintelligenz in den Spielfeldern zu realisieren. Dementsprechend sind die Funktionalitäten, die der Roboter bereitstellen muss, überschaubar. Die Implementierung einer einfachen Zusandsmaschine ermöglicht die Wahrnehmung aller Aufgaben und erlaubt zugleich das dynamische Austauschen von Robotern während des Spielbetriebs (notwendig z.B. infolge eines leeren Akkus).

Ein Roboter steht zu Beginn einer jeden Spielaktion auf einem Spielfeld und wartet auf einen Befehl. Sobald dieser eintrifft, d.h. sobald das Spielfeld einen Befehl an den Roboter sendet, wird dieser bestätigt und ausgeführt. Nach Ausführung beginnt ein erneutes Warten.

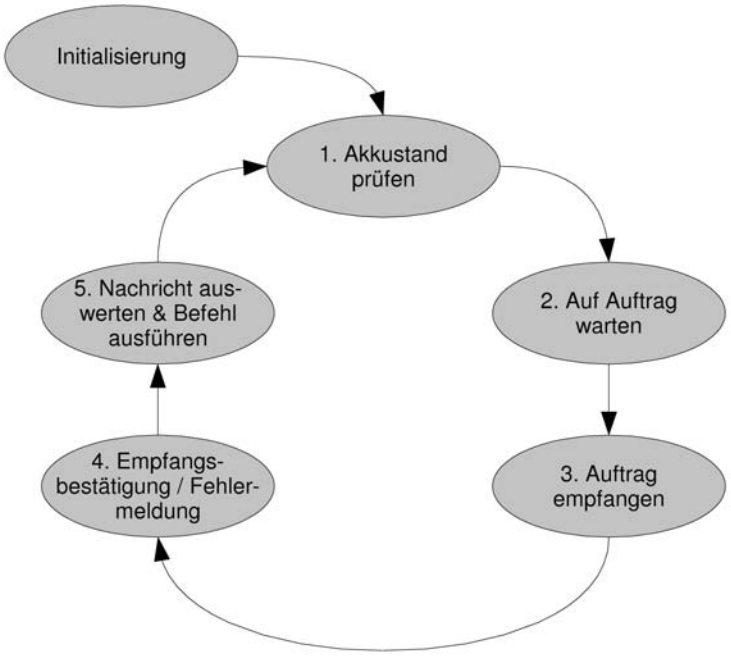


Abbildung 5.9: Zustandsmaschine des Roboters

Es ergibt sich eine Zustandsmaschine wie sie in Abbildung 5.9 zu sehen ist. In den einzelnen Zuständen werden folgende Arbeiten ausgeführt:

Initialisierung:

Um die LEDs, den ADC (Analog/Digital Converter) und die Motoren nutzen zu können, ist ihre Initialisierung im Mikrocontroller notwendig. In dieser Initialisierungsphase werden z.B. die General-Purpose-Pins des Mikrocontrollers, an denen die LEDs angeschlossen sind, als Ausgabepins deklariert (dies geschieht durch Ansteuerung spezieller Register im Mikrocontroller, näheres ist dem Datenblatt zu entnehmen [Atm05]). Zudem wird jeweils ein initialer Zustand gesetzt.

Akkustand prüfen (1):

Damit der Roboter korrekt arbeitet, wird vor jedem Arbeitszyklus einmal geprüft, ob genügend Energie zur Verfügung steht. Ist die Spannung des Akkus kritisch, so wird dies erkannt. Der Roboter übermittelt dem Spielfeld eine entsprechende

Warnung, da ein fehlerfreies Arbeiten nicht mehr länger sichergestellt ist. Zudem wird dies dem Benutzer durch Blinken der Kopf-LED signalisiert.

Auf Auftrag warten (2):

Während der Roboter auf einen Auftrag wartet, wird lediglich eine Empfangseinheit zur Kommunikation betrieben. Durch die Initiierung eines Kommunikationsprozesses durch den Kommunikationspartner wird dieser Zustand verlassen.

Auftrag empfangen (3):

Eine Nachricht des Spielfeldes wird gemäß des Kommunikationsprotokolls beschrieben in Abschnitt 4.3.2 empfangen.

Empfangsbestätigung / Fehlermeldung (4):

Der Empfang der Nachricht wird mit einer *MESSAGE_ACK* bei erfolgreicher Übertragung und einer *MESSAGE_COM_ERROR* bei einem detektierten Kommunikationsfehler bestätigt.

Nachricht auswerten und Befehl ausführen (5):

Zuletzt ist die in der Nachricht codierte Anweisung auszuführen.

Einige wichtige Aspekte der Programmierung, die sich im zweiten Teil des Projektes nicht geändert haben, werden im Folgenden näher betrachtet. Details sind im Zwischenbericht [ZWB06] zu finden.

5.3.2 Kommunikation

Das Kommunikationsprotokoll wird in zwei Schichten implementiert. Eine Schicht realisiert hardwarenahe Prozeduren zum Senden und Empfangen von Einsen und Nullen sowie für den Verbindungsaufbau und -abbau. Eine zweite Schicht realisiert Steuerroutinen, die die Kommunikation kontrollieren. Dieses Design führt dazu, dass durch die Änderung des Kommunikationsprotokolls lediglich einzelne Parameter anzupassen sind, die Implementierung selbst jedoch nicht modifiziert werden muss.

Das Protokoll selbst wird in Abschnitt 4.3.2 vorgestellt.

5.3.3 Motorsteuerung

Die Bewegungen des Roboters - genauer die Ansteuerungen der beiden Motoren, die den Roboter fortbewegen - werden in drei Schichten realisiert. Die oberste Schicht stellt Funktionen für den Spielablauf bereit (s.u.) und integriert die Fehlerkorrektur. Die mittlere Schicht setzt eine elementare Roboterbewegung (z.B. den Roboter um einen Motorschritt links drehen) in Motorbefehle um (z.B. linker Motor einen Schritt vor, rechter Motor

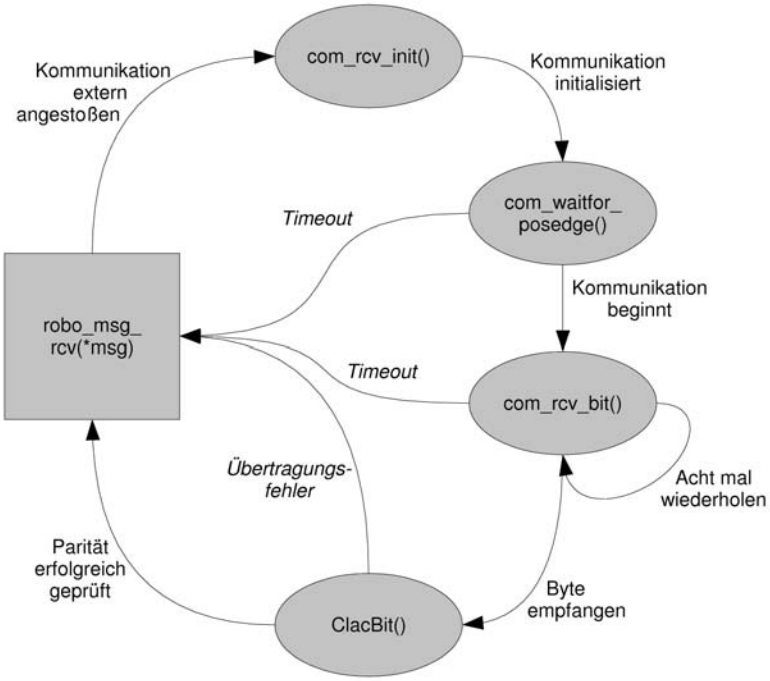


Abbildung 5.10: Nachrichtenempfang

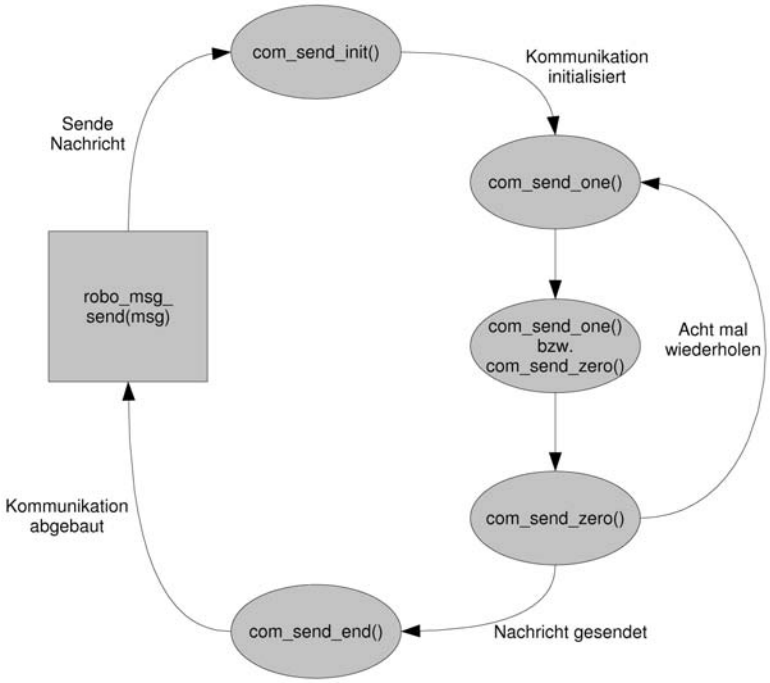


Abbildung 5.11: Nachrichtenversand

einen Schritt zurück). Die unterste Schicht lässt einen Motor um einen Motorschritt in die vorgegebene Richtung drehen.

Da die Motorsteuerung sich jedoch durch die Verwendung von Gleichstromgetriebemotoren statt von Schrittmotoren wesentlich geändert hat und zudem die Einbindung der Fehlerkorrektur überarbeitet wurde, werden die Schichten in den Abschnitten 5.4.4 und 5.4.7 genauer beschrieben.

Wichtig festzuhalten ist jedoch, dass die Schnittstellen der obersten Schicht unverändert bleiben und folgende spielrelevante Funktionalitäten bereitstellen:

- Vorwärtsbewegung,
- Rückwärtsbewegung,
- Linksdrehung sowie
- Rechtsdrehung (alle mit einem Übergabeparameter für die Anzahl).

So wirken sich die notwendigen Änderungen in den unteren Schichten nicht auf den weiteren Programmablauf aus.

5.3.4 Fehlerkorrekturalgorithmus

Die Bewegungen des Roboters können auf unterschiedlichste Art gestört werden. So entstehen kleine Fehler in der Positionierung des Roboters. Alle Fehler für sich genommen sind sehr klein, in Summe aber können sie das Spielvergnügen erheblich beeinträchtigen, sobald der Roboter nicht mehr an seinem Ziel ankommt, mit einem anderen Roboter kollidiert oder gar von der Spielfläche stürzt. Folglich ist es notwendig, diese Fehler zu korrigieren.

Ziel der Fehlerkorrektur ist, den Roboter - möglichst unauffällig in die Bewegung integriert - nach jedem Spielzug genau zu platzieren.

Grundlage der Korrekturalgorithmen

Damit ein Roboter nach einer Bewegung seine Position relativ zur Spielfeldmitte bestimmen kann, wurde jedes Spielfeld mit 5 LEDs und jeder Roboter mit zwei Fototransistoren ausgestattet (vgl. Abschnitte 4.1.2 und 5.2.2). Infolge dessen kann ein Roboter durch Messen der Lichtintensität am Fototransistor bei leichter Veränderung seiner Position bestimmen, ob er sich zur LED hin oder von ihr weg bewegt.

Grundlage für die Korrekturalgorithmen bilden nun zwei Suchen.

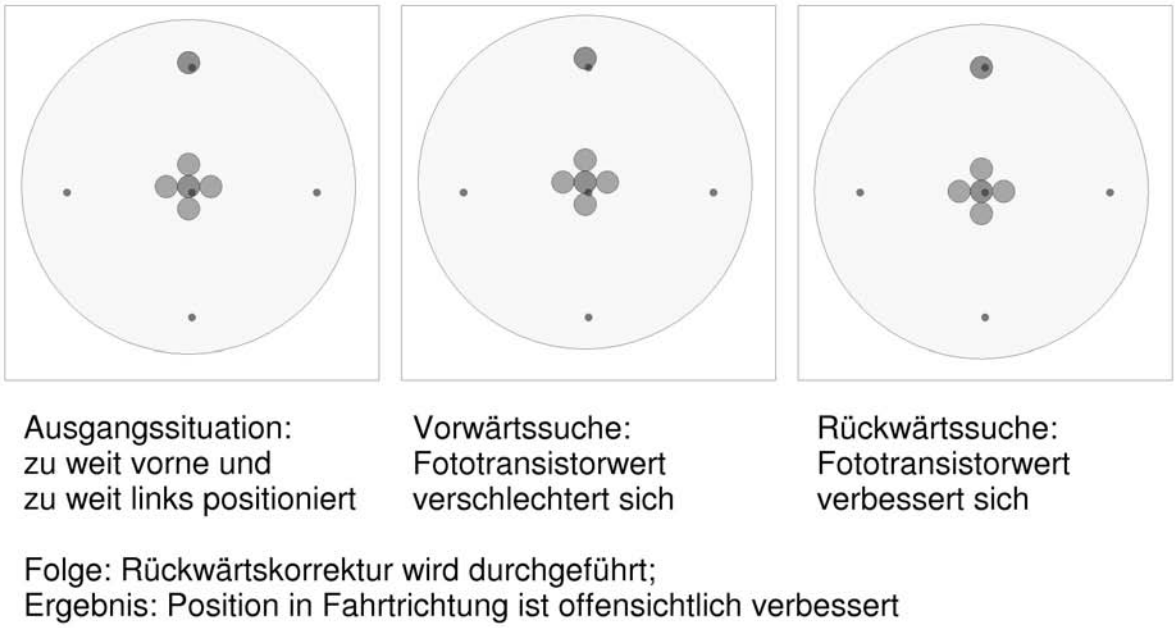


Abbildung 5.12: Vorwärts-Rückwärts-Suche

Eine Vorwärts-Rückwärts-Suche (vgl. Abbildung 5.12), bei der der Roboter sich ein wenig vorwärts und ein wenig rückwärts bewegt, um die Lichtintensität an verschiedenen Stellen zu messen, erlaubt Rückschlüsse über die relative Position des Roboters zur LED in Fahrtrichtung.

Analog erlaubt eine Links-Rechts-Suche (durch Drehbewegungen des Roboters) Rückschlüsse über die relative Position des Roboters zur LED in seitlicher Richtung.

Optimierungen der Korrekturalgorithmen unter Prüfung mittels praktischer Tests führten zu leichten Modifikationen. Daher sind weitere Details im Abschnitt 5.4.7 zu finden.

5.3.5 Fazit

Obwohl im Roboter keine Spielintelligenz zu implementieren, sein Funktionsumfang stark begrenzt und deren Komplexität überschaubar ist, erweist sich die Programmierung des Roboters als schwierig, da an dieser Stelle alle Fäden der Roboterkonstruktion zusammenlaufen. Intensives Testen von Softwarebestandteilen am ersten Prototypen der Roboter und in eigens erstellten Testaufbauten sowie eine hohe Separation von Funktionalitäten in der Implementierung ermöglichten aber beim Wechsel der Roboterplattform eine Übernahme großer Softwarebestandteile.

5.4 Änderungen Software

Einige Änderungen an der Software des Roboters sind durch die neue Roboterplattform sowie das geänderte Kommunikationsprotokoll notwendig geworden. Ferner zeigten Integrationstests mit dem Spielfeld sowohl den Bedarf neuer Funktionalitäten als auch die Notwendigkeit der Verbesserung vorhandener Funktionen.

Folgende Bereiche der Software wurden überarbeitet:

- Einführung einer initialen Ausrichtung

  Da ein Anwender zum Beginn eines Spieles den Roboter von Hand auf das erste Spielfeld stellen muss, ist davon auszugehen, dass diese Positionierung relativ ungenau ist. Folglich wird eine sehr starke Korrektur notwendig. Diese geschieht durch die *initiale Ausrichtung (Abschnitt 5.4.1)*.

- Einführung einer Ausrichtung bei Bedarf

  Wenn ein Roboter im Spiel stirbt und noch über weitere Leben verfügt, so muss er vom Benutzer auf ein bestimmtes Backup-Feld gestellt werden (einige Spielfelder erlauben es dem Spieler ein Backup anzulegen, so dass dieser nach einem Totalschaden nicht erneut von seinem Startfeld aus starten muss), von dem aus er wieder ins Spielgeschehen eingreifen kann. Eine zur initialen Ausrichtung analoge Situation entsteht. Die *Ausrichtung bei Bedarf (Abschnitt 5.4.2)* wird jedoch durch eine Kommunikation angestoßen.

- Änderung des Kommunikationsprotokolls

  Eine Beschreibung des geänderten Kommunikationsprotokolls ist in Abschnitt 4.3.2 zu finden. Leichte Änderungen waren notwendig, um die Tageslichterkennung im Spielfeld zu unterstützen.

- Einführung einer Radumdrehungsmesstechnik

  Durch die Verwendung von Gleichstromgetriebemotoren statt Schrittmotoren muss die Fortbewegung des Roboters nun gesondert gemessen werden, eine *Radumdrehungsmesstechnik (Abschnitt 5.4.3)* ist notwendig.

- Änderung der Motoransteuerung

  Ebenfalls durch die Verwendung von Gleichstromgetriebemotoren statt Schrittmotoren ist eine *Änderung der Motoransteuerung (5.4.4)* unter Einbezug der neuen Radumdrehungsmesstechnik notwendig.

- Erweiterung der Nachrichtenvalidierung

  Da das Spielfeld nun auf Fehlermeldungen des Roboters reagiert, ist vor der Bestätigung einer empfangenen Nachricht seitens des Roboters die Validität genau zu prüfen. Dazu ist die *Nachrichtenvalidierung (5.4.5)* zu erweitern.

- Qualitätssicherung von Messwerten für die Fehlerkorrektur

  Zur *Qualitätssicherung von Messwerten für die Fehlerkorrektur (5.4.6)* sind zum einen eine genauere Bestimmung von Messergebnissen seitens des ADC und zum anderen eine Wertekorrektur notwendig.

- Optimierung der Fehlerkorrektur

  Beim Testen konnten einige *Optimierungen der Fehlerkorrektur (5.4.7)* vorgenommen werden, um deren Qualität, deren Geschwindigkeit sowie deren Integration in den Spielablauf zu verbessern.

Die Änderungen werden im Folgenden genauer beschrieben.

5.4.1 Initiale Ausrichtung

Zu Beginn eines Spieles wird der Anwender durch den Master aufgefordert, einen Roboter nach dem anderen auf gekennzeichnete Spielfelder zu stellen. Dabei ist davon auszugehen, dass diese erste Platzierung durch Menschenhand relativ ungenau ist und folglich eine starke Fehlerkorrektur notwendig wird. Da auf der anderen Seite diese Ausrichtung (auch nicht durch die Wiederverwendung in der *Ausrichtung bei Bedarf (Abschnitt 5.4.2)*) sehr selten im Spiel passiert und zudem stets durch den Eingriff eines Anwenders bedingt ist, ist eine unauffällige Integration in den Spielablauf nebensächlich.

So besteht die initiale Ausrichtung nach einer Vorwärts-Rückwärts-Korrektur aus einer Linksdrehung und einer anschließenden Rechtsdrehung (je um 90°) incl. Korrekturen.

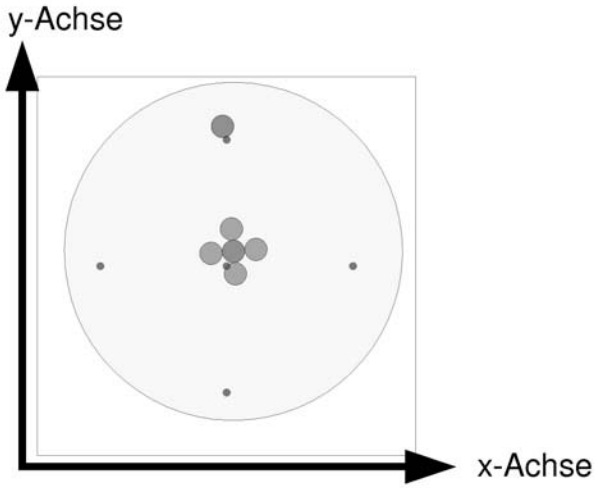


Abbildung 5.13: Fehlersituation

Die Vorwärts-Rückwärts-Korrektur eines Roboters, der wie in Abbildung 5.13 auf einem Spielfeld platziert wurde, sorgt dafür, dass dieser vor seiner Drehung eine möglichst kleine Abweichung zur Idealposition aufweist.

Die anschließende Linksdrehung reduziert durch ihre Fehlerkorrektur (Abschnitt 5.4.7) insbesondere den Fehler in x-Richtung, die abschließende Rechtsdrehung (analog) den Fehler in y-Richtung. Anschließend meldet der Roboter dem Spielfeld seine Spielbereitschaft, welches diese durch die Anforderung einer Identifikationsnachricht beständig einfordert.

Die initiale Ausrichtung wird nach dem Einschalten des Roboters automatisch ausgeführt.

5.4.2 Ausrichtung bei Bedarf

Im Spielverlauf sind eine Reihe von Situationen denkbar, in denen ein Roboter sterben kann. Gemäß der Spielregeln [AMI99] wird dieser dann - sofern er noch über weitere Leben verfügt - auf ein bestimmtes Backup-Feld zurückgesetzt. Dieses wird durch seine LEDs hervorgehoben. Zugleich wird der Benutzer informiert, dass der Roboter auf dem markierten Feld zu platzieren ist.

Damit ergibt sich hier eine ähnliche Situation wie in der *initialen Ausrichtung (Abschnitt 5.4.1)*. Allerdings ist hier dem Roboter durch das Spielfeld - dem im Gegensatz zum Roboter bekannt ist, dass dieser manuell umgesetzt wurde - mitzuteilen, dass eine starke neue Ausrichtung durchzuführen ist. Dazu wurde eine neue Nachricht generiert.

5.4.3 Radumdrehungsmesstechnik

Im ersten Prototypen waren für den Antrieb der Roboter Schrittmotoren vorgesehen, die den Vorteil haben, dass bei ihnen schon durch die Ansteuerung der Motoren feststeht, wie weit sich der Roboter fortbewegt. Der Wechsel der Antriebstechnik hin zu Gleichstromgetriebemotoren schafft den Bedarf einer Messtechnik, wie sie in Abschnitt 5.2.1.1 vorgestellt wird.

Nun kann die Vierteldrehung der Achse am Motor wie folgt beobachtet werden:
Zu Beginn einer Vierteldrehung steht die Achse so, dass am Fototransistor eine *Eins* gemessen wird. Das Rad ist so lange zu drehen, bis eine *Null* und eine erneute *Eins* gemessen werden konnten.

1.) Solange eine Eins gemessen wird warte auf eine Null
```
while (*AT91C_PIOA_PDSR & photo_pin );
```
2.) Solange eine Null gemessen wird warte auf eine Eins
```
while (!(*AT91C_PIOA_PDSR & photo_pin));
```

5.4.4 Änderung der Motoransteuerung

Einen Überblick über das zugrunde liegende Schichtenmodell gewährt Abschnitt 5.3.3. Hier werden die untere und mittlere Schicht beschrieben, da diese maßgeblich durch die Hardwareänderungen beeinflusst wurden.

Untere Schicht der Motoransteuerung

In dieser Schicht wird eine Methode zur Verfügung gestellt, die es erlaubt, einen Motor so lange vorwärts bzw. rückwärts laufen zu lassen, bis sich die Achse um eine Vierteldrehung in entsprechender Richtung gedreht hat. Die Vierteldrehung ist dabei der kleinste mögliche Schritt und wird durch die Hardware (vgl. Abschnitt 5.2.1.1) vorgegeben.

1.) Bestimme die Motordrehrichtung (vorwärts oder rückwärts)
```
if (direction>0) *AT91C_PIOA_SODR = direction_pin;
else *AT91C_PIOA_CODR = direction_pin;
```
2.) Starte Motor (gleichbedeutend mit: löse die Bremse)
```
*AT91C_PIOA_CODR = enable_pin;
```
3.) Warte eine Vierteldrehung per busy waiting gemäß Abschnitt 5.4.3
4.) Stoppe Motor (gleichbedeutend mit: ziehe Bremse an)
```
*AT91C_PIOA_SODR = enable_pin;
```

Mittlere Schicht der Motoransteuerung

Die vier Funktionen, die in der oberen Schicht benötigt werden, sind in Befehle für die untere Schicht umzusetzen:

- Roboterbewegung um einen Elementarschritt vorwärts;
- Roboterbewegung um einen Elementarschritt rückwärts;
- Roboterdrehung um einen Elementarschritt linksherum;
- Roboterdrehung um einen Elementarschritt rechtsherum.

Exemplarisch wird aus einer „Roboterbewegung um einen Elementarschritt vorwärts“ eine „Drehung des linken Rades um einen Elementarschritt vorwärts“ und eine „Drehung des rechten Rades um einen Elementarschritt vorwärts“. Der unteren Schicht wird dabei die Drehrichtung des Rades direkt und das Rad durch Übergabe der zugehörigen Anschlüsse indirekt mitgeteilt.

Da das Warten auf die Raddrehung eines Motors mit einem busy waiting einhergeht, ist es nicht möglich, die Motoren parallel zu nutzen. Um dennoch für den menschlichen Beobachter eine flüssige und für die Fehlerminimierung eine möglichst präzise gerade Bewegung zu erhalten, sind die Motoren so anzusteuern, dass sich ein Rad je genau zwei Mal nacheinander dreht (vgl. Abbildung 5.14).

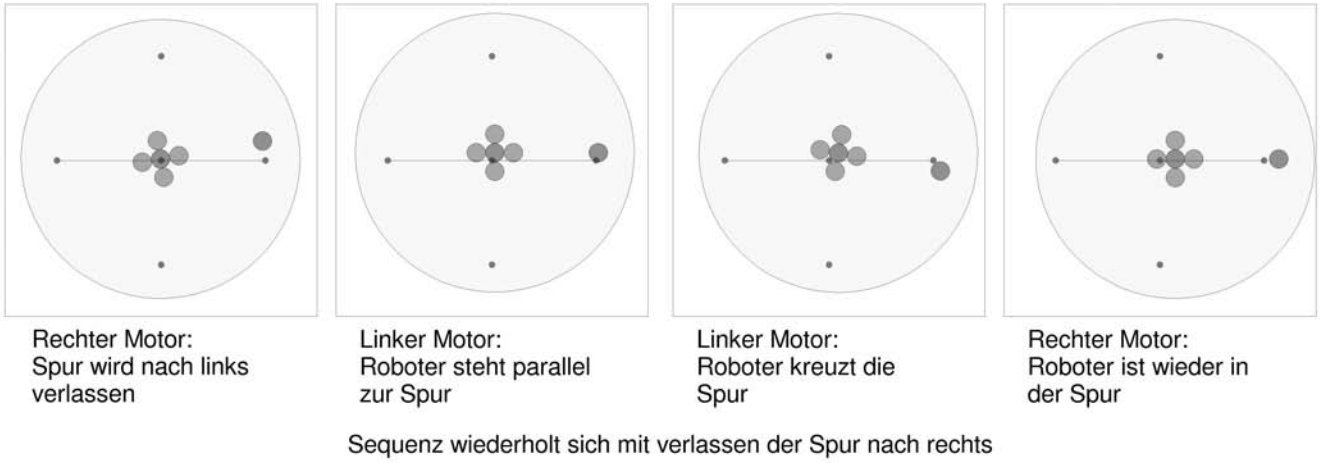


Abbildung 5.14: Vorwärtsbewegung eines Roboters

5.4.5 Erweiterung der Nachrichtenvalidierung

Empfängt der Roboter eine Nachricht, die er nicht auswerten kann, sei es, ein Empfangsfehler - detektiert durch ein Timeout - tritt ein, die Prüfsumme einer Nachricht stimmt nicht, oder die Nachricht ist dem Roboter nicht bekannt, so kann er dem Spielfeld eine Fehlernachricht (eine sogenannte *MESSAGE_COM_ERROR*) senden. Das Spielfeld wiederholt daraufhin die Nachricht.

In der Überprüfung war lediglich der Aspekt, ob der Roboter eine Nachricht (vollständig empfangen und mit richtiger Prüfsumme) kennt, zu ergänzen. Eine Abfrage, ob eine empfangene Nachricht aus dem Pool der bekannten Nachrichten stammt, entscheidet, ob eine *MESSAGE_ACK* oder eine *MESSAGE_COM_ERROR* zu senden ist.

5.4.6 Qualitätssicherung von Messwerten für die Fehlerkorrektur

Die Praxis zeigt, dass die Messwerte, die der ADC für den Fototransistor[2] liefert, auch bei konstanten Außenbedingungen (Lampenlicht, fixe Position des Roboters) leicht schwanken. Dies kann zu fehlerhaften Interpretationen führen. Ein vorhandener Fehler kann verstärkt werden, d.h. der Fehlerkorrekturalgorithmus verbessert die Fahrtgenauigkeit des Roboters nicht, sondern verschlechtert diese (vgl. Beispiel in Abbildung 5.15). Folglich sind Qualitätssicherungsmaßnahmen für die Messwerte des Fototransistors notwendig.

[2]Für die Fehlerkorrektur wird stets der vordere Fototransistor verwendet.

ADC Einstellungen

Auf Kosten der Geschwindigkeit einer Messung ist es möglich, vom ADC bessere Messwerte zu bekommen, denn all seine Funktionen werden in Abhängigkeit von einer eigenen *Clock* durchgeführt, wobei gilt: Je langsamer der ADC bzw. seine Clock läuft, desto höhere Qualität haben seine Ergebnisse.

Es ergeben sich jedoch zwei Probleme:

1. Die Kommunikation braucht die hohe ADC-Geschwindigkeit, damit ein Abtasten mit 12 kHz möglich ist. Da sie jedoch keine hohe Messwertqualität für eine Interpretation der Messwerte als binäre Werte braucht, ist folglich mit zwei Qualitätsstufen zu arbeiten. Bei nur einem vorhandenen ADC bedeutet dies, dass die Geschwindigkeit des ADCs regelmäßig umzuschalten ist:

   ```
   if (mode == ADC_IS_COMMUNICATION)
   *AT91C_ADC_MR = 0x0F1D0000;
   else if (mode == ADC_IS_MOVEMENT)
   *AT91C_ADC_MR = 0x0F1D3F00;
   ```

2. Obwohl die Qualität der Messergebnisse wesentlich besser wird, unterliegen sie dennoch auch weiterhin leichten Schwankungen (bis zu 10 Einheiten bei einem Wertebereich von 0 bis 1023). Folglich ist hier noch eine Wertenachbesserung notwendig.

Filtern des Blinkmusters der Spielfelder

Erste Aufgabe dabei ist, das Blinken der LED, die in der Mitte des Spielfeldes ist, zu filtern. Zu diesem Blinken kommt es, da das Spielfeld - um die Ankunft eines Roboters mitgeteilt zu bekommen - eine Identifizierungsnachricht anfordert, also eine Kommunikation zu initiieren versucht.

Die Filterung ist simpel: Wenn der Messwert eindeutig auf eine ausgeschaltete LED schließen lässt, wird dieser Wert verworfen. Dazu muss ein hoher Messwert vorliegen (der Grenzwert wird als THRESHOLD_KILLINGZEROS definiert) und ein Blinken der Ziellampe überhaupt möglich sein:

```
if ( (photo_value>THRESHOLD_KILLINGZEROS) && (blinking) ) ...
```

Dabei wird protokolliert, wie oft die Werte während eines Lesezyklus (siehe nächsten Abschnitt) verworfen werden. Falls es zu einem ständigen Werteverwurf kommt, wird eine Fehlermeldung generiert und zurückgegeben, um einen Deadlock zu verhindern:

```
if (error_count > MAX_ZERO_ERRORS) return 1023;
```

Mitteln eines Wertes

Dieser Schritt besteht darin, dass an jeder Position, die der Roboter untersucht, nicht nur ein Wert, sondern mehrere Werte eingelesen werden und über diese Werte ein Mittelwert gebildet wird. Die Mittelwertbildung (ein Lesezyklus) filtert einen Großteil der Schwankungen raus und liefert so recht stabile Werte.

Glätten der Werte

Ein letzter Schritt zur Verbesserung der Werte ist in die Fehlerkorrektur integriert und wird auch dort (in Abschnitt 5.4.7) beschrieben.

5.4.7 Optimierung der Fehlerkorrektur

Die Fehlerkorrektur besteht aus zwei Schichten. Eine Korrekturschicht bestimmt einen Korrekturwert und korrigiert in Vorwärts-Rückwärts- (*vr-Korrektur*) bzw. Links-Rechts-Richtung (*lr-Korrektur*). Eine Korrektursteuerungsschicht bietet Fehlerkorrekturvarianten für verschiedene Einsatzsituationen. Sie legt fest, wann welche Korrektur zu erfolgen hat. Bei schwerwiegenden Korrekturen ist gar eine vollständige Steuerung des Roboters (z.B. eine Linksdrehung) möglich. Die Korrektursteuerungsschicht wird von der obersten Schicht der Motorsteuerung aus aufgerufen.

Korrekturschicht

Die Korrekturschicht stellt je eine Funktion für eine Vorwärts-Rückwärts- und für eine Links-Rechts-Korrektur zur Verfügung, deren einziger Unterschied in der Suchrichtung besteht (erstere sucht durch Vorwärts- und Rückwärtsfahren und reduziert so den Abstand zur idealen Position, zweitere sucht durch Links- und Rechtsdrehungen und reduziert so den Winkel zur idealen Ausrichtung).

1.) Werte einlesen

Entgegen dem im Zwischenbericht [ZWB06] geplanten Ansatz - der Roboter sollte in die erste Suchrichtung bewegt werden, bis die Messwerte schlechter werden, und falls sie auf diesem Weg nicht besser wurden (und damit schon das Optimum gefunden wurde) ebenso in die zweite Richtung bewegt werden - ist ein fester experimentell optimierter Suchradius implementiert worden, da:

- eine Glättung der Werte untereinander sonst nicht möglich (siehe unten), diese aber für eine qualitativ hochwertige Korrektur notwendig ist, und
- die Spielfeldgrenzen sonst schon bei der Suche regelmäßig verletzt werden.

Ein fester Suchradius wird abgefahren und qualitätsverbesserte Messwerte (siehe oben) werden für jeden Punkt in ein Messwerte-Array eingetragen.

2.) Glättung der Werte

Richtung in Schritten:	2 links	1 links	geradeaus	1 rechts	2 rechts
Optimale Korrektur			x		
Messwerte-Array:	35	22	23	23	30
ermittelte Korrektur:		x			
geglättete Werte:	57	80	68	76	53
ermittelte Korrektur:					x
geglättete Werte mit Dopplung:	92	80	68	76	83
ermittelte Korrektur:			x		
Anm.: Je tiefer der Wert desto heller die Lichteinstahlung.					

Abbildung 5.15: Beispiel einer ADC-Auswertung

Obwohl die Werte im Messwerte-Array recht stabil sind, müssen wegen möglicher Fehlersituationen diese untereinander geglättet werden. Ein Beispiel dazu zeigt Abbildung 5.15.

Hier sind für einen Messradius von zwei Schritte nach links bis zwei Schritte nach rechts relativ zu bisherigen Fahrtrichtung die Messwerte des Fototransistors (Zeile 3) und die daraus resultierende Korrektur um einen Schritt nach links (Zeile 4) dargestellt. Der hohe Anstieg der Messwerte zu seinen Messbreichsgrenzen legt jedoch eine Geradeausfahrt - also keine Korrektur - nahe. In den verbleibenden Zeilen wird der Einfluss der im Folgenden erläuterten Glättung dargestellt.

Eine Glättung aller

$$Wert\ (i)\ =\ Wert\ (i-1)\ +\ Wert\ (i)\ +\ Wert\ (i+1)$$

verbessert die Situation, liefert jedoch bei Randwerten (nach Abfangen von Speicherzugriffsfehlern) zu niedrige Werte (siehe Abbildung 5.15, Zeilen 5 und 6). Dies kann auf folgende Arten verhindert werden:

1. Links und rechts einen weiteren Messwert erheben, diese Korrektur jedoch nicht in die Menge der möglichen Korrekturen aufnehmen.

   Eine Erweiterung des Suchraumes birgt eine höhere Gefahr der Feldgrenzverletzung.

2. Die äußeren Messwerte aus der Menge der möglichen Messwerte entfernen.

   Dies reduziert die Korrekturmächtigkeit der Fehlerkorrektur, insbesondere bei kleinen Suchradien.

3. Die äußeren Werte werden einfach wiederholt.

   Dies ist in obiger Situation optimal. Bei sehr dicht beieinander liegenden Messwerten kann diese überproportionale Gewichtung der Randwerte jedoch zu einer Bevorzugung entsprechender Korrekturen führen.

Die letzte Lösung ist für den Einsatz in den Robotern am Besten geeignet und wird dementsprechend umgesetzt (siehe Abbildung 5.15, Zeilen 7 und 8).

3.) Richtungspräferenz

Um nicht durch eine Korrektur Fehler zu erzeugen, ist es sinnvoll, im Zweifelsfall die eingeschlagene Richtung beizubehalten. Deswegen wird bei gleichen geglätteten Werten stets die Korrektur bevorzugt, die am dichtesten am bisherigen Kurs liegt.

4.) Fehlerbehandlung per Plausibilitätsprüfung

Liegt der Messwert der ermittelten Korrektur außerhalb des zu erwartenden Wertebereiches (weil z.B. die Messwerterhebung stets fehl schlug und immer eine 1023 zurücklieferte), so wird die geplante Korrektur verworfen und der eingeschlagene Kurs beibehalten.

Letztendlich wird so stets eine Korrektur ermittelt und ausgeführt.

Korrektursteuerungsschicht

In dieser Schicht werden Methoden zur Verfügung gestellt, die situationsbezogen optimale Korrekturen durchführen:

- *pos_intermediate_correction*

  Dies ist eine unauffällige Ausrichtungskorrektur für einen Einsatz während einer Vorwärtsbewegung. Sie besteht nur aus einer Links-Rechts-Korrektur, ihre Mächtigkeit wird im Zwischenbericht [ZWB06] ausführlich dargestellt.

- *pos_final_turn_correction*

  Dies ist eine Ausrichtungskorrektur für den Abschluss einer Drehung. Eine Links-Rechts-Korrektur bereitet eine anschließende Vorwärts-Rückwärts-Korrektur dahingehend vor, dass diese dann die Position in Blickrichtung optimal korrigiert. Die orthogonale Richtung wurde bereits vor der Drehung mit dem Abschluss des vorangegangenen Bewegungsbefehls korrigiert. Eine abschließende Links-Rechts-Korrektur verfügt potentiell über bessere Messwerte als die einleitende Links-Rechts-Korrektur, da der Fototransistor nach bereits erfolgten Korrekturen besser zur LED steht. Sie stellt sicher, dass eine nächste Bewegung in die richtige Richtung erfolgt. Dies ist z.B. bei einer folgenden Vorwärtsbewegung eine ähnlich mächtige Korrektur wie eine *pos_intermediate_correction*.

- *pos_final_light_correction*

  Dies ist eine leichte Fehlerkorrektur nach einer Vorwärts-Bewegung. Sie ist identisch zur *pos_final_turn_correction*, die zur Blickrichtung orthogonale Richtung wurde dabei bereits durch die vorausgegangenen *pos_intermediate_corrections* korrigiert.

- *pos_final_grave_correction*

  Dies ist eine schwerwiegende Fehlerkorrektur nach einer Rückwärtsbewegung. Sie unterscheidet sich dahingehend von den anderen Korrekturen, dass sie zum einen den Suchraum vergrößert (unter höherer Gefahr, Spielfeldgrenzen zu verletzen) und zum anderen in Blickrichtung als auch in orthogonaler Richtung korrigiert.

  Die Korrektur selbst besteht aus drei Phasen:

  – Eine vorbereitende Vorwärts-Rückwärts-Korrektur verbessert die Qualität der folgenden Korrektur, da diese mit einer Drehung beginnt.

  – Eine Rechtsdrehung incl. aller Korrekturen korrigiert in zur eigentlichen Blickrichtung orthogonalen Richtung.

  – Eine Linksdrehung incl. aller Korrekturen korrigiert letztendlich nach einer inversen Drehung in Blickrichtung.

  Wegen der deutlich größeren Spielverzögerung, der Offensichtlichkeit ihrer Arbeit und der höheren Gefahr der Spielfeldgrenzverletzung ist sie sparsam einzusetzen, sie vermag jedoch wesentlich größere Fehler zu beheben als die übrigen vorgestellten Methoden.

Einsatz der Korrekturen

Die möglichen Korrekturen sind nun in die Bewegungsabläufe des Roboters zu integrieren. Dies geschieht wie in Abschnitt 5.3.3 beschrieben auf der obersten Motorsteuerungsebene.

- Vorwärtsbewegung

  An jeder Weg-LED wird eine *pos_intermediate_correction* durchgeführt. Die Bewegung wird - da nur geringe Fehler zu erwarten sind (vgl. [ZWB06]) - durch eine *pos_final_light_correction* abgeschlossen.

- Rückwärtsbewegung

  Eine zielgerichtete Fehlerkorrektur an den Weg-LEDs wie bei der Vorwärtsbewegung ist leider durch die Anordnung der Fototransistoren im Roboter nicht möglich, denn dazu wäre ein Fototransistor am hinteren Ende des Roboters notwendig. Daher ist der gesamte Fehler der Bewegung auf dem Zielfeld zu korrigieren. Wegen der erhöhten Wahrscheinlichkeit größerer Fehler wird dazu die *pos_final_grave_correction* verwendet.

- Drehbewegungen

  Die Drehbewegungen werden durch die eigens konzipierte *pos_final_turn_correction* berichtigt.

5.4.8 Fazit

Da der Roboter in all seinen Aktionen im besonderen Maß mit seiner physikalischen Umwelt interagieren muss, stellen gerade die praktischen Tests der entwickelten Ideen und Algorithmen eine wichtige Grundlage für die Verbesserung der Techniken dar.

Auch wenn der Eindruck entsteht, dass nahezu alle Komponenten der Software des Roboters überarbeitet werden mussten, konnte doch dank der modularisierten Implementierung die Grundstruktur der Software stets beibehalten werden. Einzelne Änderungen waren zudem schnell und einfach möglich und erlaubten, ihre Auswirkungen in anschließenden Tests sofort zu beobachten.

So konnte die Software sowohl an den neuen Roboter zügig angepasst als auch effizient optimiert werden.

5.5 Zusammenfassung und Bewertung

Bei der Entwicklung der Roboter als Spielfiguren des Spieles galt es viele Anforderungen zu berücksichtigen: So war neben steuerbarer, aber sonst autonomer Bewegung und bidirektionaler Kommunikation mit dem Spielfeld zum Erhalt von Steuerbefehlen vor allem die Einhaltung der Größenbeschränkungen wichtig. Deswegen wurde bei der Auswahl der Komponenten des ersten Entwurfes vor allem auf ihre Größe geachtet [ZWB06].

Dieser erfüllt die meisten Anforderungen mit der zentralen Steuereinheit eines Mikrocontrollers, zwei separat ansteuerbaren Schrittmotoren, die über jeweils ein Bausatzgetriebe eines der beiden Räder antreiben. Die Kommunikation wird auf optischem Wege mit LEDs und Fototransistoren über ein selbstentworfenes Protokoll abgewickelt. Für die Energieversorgung wird ein Modellbauakku eingebunden (für Details zu den einzelnen Komponenten siehe Abschnitt 5.1.1). Mit diesen Komponenten entsteht ein voll funktionstüchtiger Roboter, der die an ihn gestellten Anforderungen erfüllt. Das Problem dieses Entwurfes liegt in der synchronen Ansteuerung der beiden Räder, was auf die fehlende Laufruhe der beiden Bausatzgetriebe zurückzuführen ist. Auch nach Maßnahmen zur Stabilisierung war das Ergebnis nicht brauchbar, so dass eine neue Lösung gefunden werden musste.

Beim zweiten Ansatz wurde die Einhaltung der Größenbeschränkungen beachtet, aber es stand nicht mehr die Miniaturisierung im Vordergrund, sondern die Verwendung eines robusten Antriebes. So wurden wegen der besseren Verfügbarkeit größere Gleichstrommotoren mit störungsärmeren aufgesetzten Getrieben gewählt und der restliche Aufbau

diesen angepasst. Ein Motorblock wurde entworfen, auf dem die Komponenten befestigt sind. An den übrigen Punkten wurde auf die bereits im ersten Entwurf funktionierenden Komponenten zurückgegriffen. So wird der eingesetzte Akku weiterverwendet, die Art der Kommunikation beibehalten, die Grundstruktur der Elektronik kann weiterverwendet werden und auch weite Teile der Software finden sich im zweiten Entwurf wieder. Die Teile, die mit dem Antrieb zusammenhängen, müssen angepasst werden, so ist zum Beispiel bei Gleichstrommotoren eine Schrittzählung notwendig, um die Navigierbarkeit aufrecht zu erhalten (für Details siehe Abschnitte 5.2, 5.3 und 5.4).

Der zweite Ansatz erfüllt die ihm gestellten Anforderungen bis zur Einsatztauglichkeit in der Praxis. Die einzige Schwäche zeigt der Entwurf beim Rückwärtsfahren, die aber beim ersten Entwurf aufgrund des analogen Aufbaus auch zu erwarten gewesen wäre. Hier wäre ein zusätzlicher Fototransistor zur besseren Ausrichtung an den Navigations-LEDs des Spielfeldes hilfreich.

6 Host

6.1 Einleitung

Die Hosts, bzw. die Host-Software, bildet das Bindeglied zwischen dem Spieler und dem Spiel. Wo bei gewöhnlichen Brettspielen der Spieler den Spielablauf manuell durchführen muß, werden die Schritte der Spielfiguren bei R_3D_3 in die Host-Software eingegeben, und das Spielfeld führt den Spielablauf daraufhin automatisch aus. Zusätzlich präsentiert die Software dem Spieler alle Informationen, die für den Spielablauf benötigt werden, wie zum Beispiel die Programmkarten, die der Spieler erhalten hat, die erhaltenen Schadenspunkte und die bereits erreichten Checkpoints des Spielers.

Damit die Spieler nicht nacheinander ihre Befehle an die Software geben müssen, wurde die Software darauf ausgelegt, daß pro Spieler ein Host zum Einsatz kommt. Die Hosts sind über das Spielfeld miteinander verbunden und können so alle nötigen Informationen austauschen, zum Beispiel die Schadenspunkte der anderen Spieler.

Die folgenden Abschnitte beschreiben zunächst die Aufgaben der Host-Software genauer. Darauf folgt eine Beschreibung der technischen Umsetzung inklusive einer kurzen Beschreibung der verwendeten Architektur. Abschließend wird eine Zusammenfassung und eine Bewertung der Umsetzung der Host-Software gegeben.

6.2 Aufgaben

Die Hauptaufgabe der Host-Software besteht in erster Linie darin, dem Spieler alle für den Spielablauf benötigten Informtaionen zu liefern, und dem Spieler die Möglichkeit zu geben, seine Kommandos, repräsentiert durch eine Sequenz von Programmierkarten, an den Roboter zu senden. Um das zu gewährleisten, wurde für die Host-Software eine möglichst intuitive GUI (Graphical User Interface) erstellt.

Die GUI übernimmt außerdem die Aufgabe, den Spieler durch den Programmablauf zu führen (siehe Kapitel 7). Dazu werden dem Spieler je nach Spielphase unterschiedliche Programmoberflächen angezeigt, die nur die zu diesem Zeitpunkt benötigten Informationen darstellen. So werden z.B. in der Programmierphase nur die Informationen des jeweiligen Spielers dargestellt, und während der Programmausführungsphase die Informationen über Schadenspunkte, Backups, den nächsten zu erreichenden Checkpoint und die erhaltenen Optionskarten für alle Spieler angezeigt.

Zustätzlich zur reinen Visualiserung von Information und der Interaktion zwischen den Spielern und dem Spiel muß ein Host eine zentrale Rolle einnehmen, um den Spielablauf

zu regeln. Obwohl es sich bei dem Spiel um ein verteiltes System handelt, gibt es dennoch Aufgaben, die aus Konsistenz- und Effizienzgründen nicht verteilt gelöst worden sind. Dazu gehört z.B. das Austeilen von Karten und die Auswertung der Prioritäten für die Befehlsabarbeitung. Außerdem übernimmt der Host, der diese zentrale Rolle als *Master* zugewiesen bekommt, auch die Initialisierung des Spielfelds mit Checkpoints und fordert zum Aufstellen der Roboter auf das Spielfeld auf.

6.3 Umsetzung

6.3.1 Architektur

Die Software-Architektur (Abb. 6.1) kann grob in zwei Bereiche eingeteilt werden:

- Die Host-Architektur, die nur in den Host-PCs zum Einsatz kommt und
- die Kommunikations-Architektur, die auch in den Host-Schnittstellen und im Spielfeld eingesetzt wird.

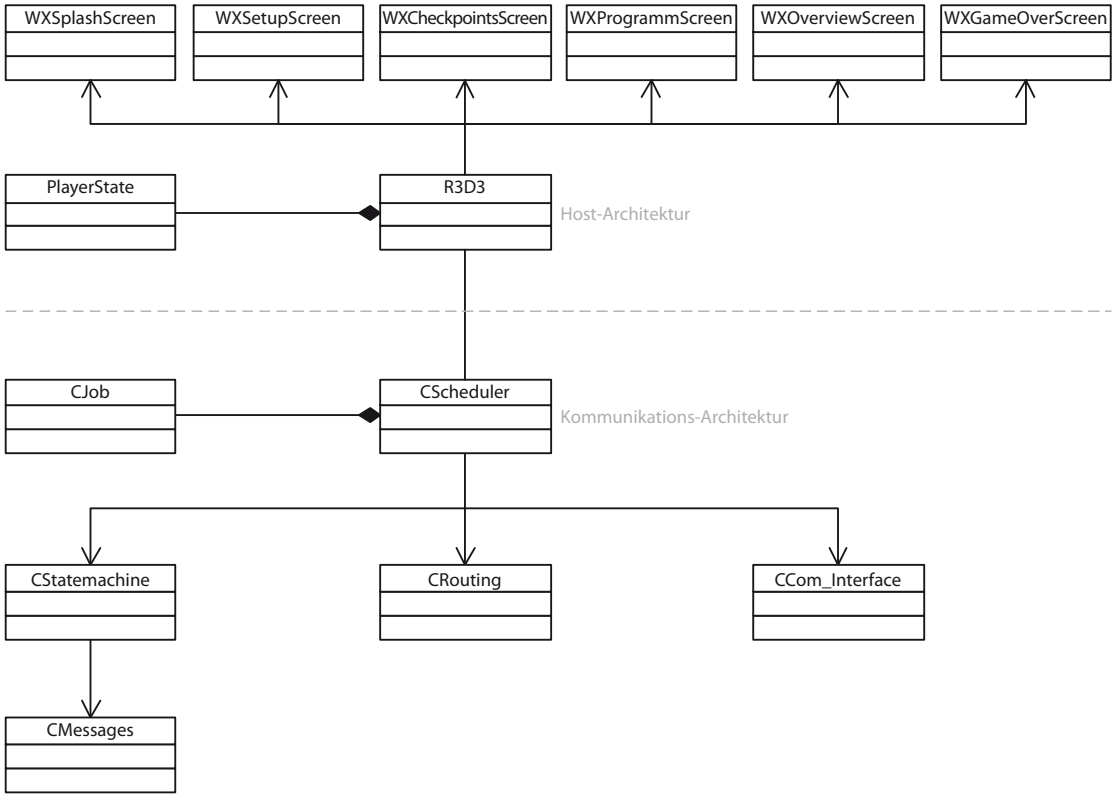


Abbildung 6.1: Architekturüberblick

Die Kommunikations-Architektur bildet die Grundlage für den Nachrichtenaustausch zwischen den Komponenten. Für die Host-Software ist über zusätzliche Job-Typen eine Anbindung an den Scheduler der Kommunikations-Architektur implementiert worden. Ausgelöst durch den Scheduler werden so die Funktionen und Darstellungsroutinen der Host-Architektur aufgerufen. Die Host-Architektur hängt ihrerseits die Ergebnisse von Operationsaufrufen, als neuen Job verpackt, in die Job-Queue des Schedulers ein.

Auf die Details der Host-Software wird im folgenden eingegangen. Die Details zur Kommunikations-Architektur im allgemeinen sind in Abschnitt 4.4.3 erläutert. Eine Beschreibung der Anpassung der Kommunikations-Architektur für den Host findet sich in Abschnitt 6.3.5.

6.3.2 Die *wxWidgets*-Bibliothek

Die Host-Software ist in C/C++ geschrieben. So kann der in reinem C geschriebene Quelltext der Kommunikations-Architektur auch in den Hosts wiederverwendet werden. In C/C++ stehen jedoch nicht wie in Java durch die *Swing*-Pakete eigene Klassen zum Erstellen grafischer Benutzeroberflächen zur Verfügung. Es wurde daher die *wxWidgets*-Bibliothek[1] eingesetzt, eine Open-Source Bibliothek zur Generierung von GUIs.

Die Vorteile von *wxWidgets* gegenüber anderen GUI-Bibliotheken liegen vor allem in der Plattformunabhängigkeit und der plattformnativen Darstellung. Die Plattformnativität wird dadurch erreicht, dass *wxWidgets* intern plattformspezifischen Code einsetzt. Daher müssen die GUI-Bibliotheken von *wxWidgets* zunächst für die Zielplattform selbst compiliert werden. Außerdem unterliegt *wxWidgets* der L-GPL (Library General Public Licence). Details finden sich bei [Sma05].

6.3.3 Datenaustausch zwischen Host- und Kommunikations-Architekur

Um eine möglichst einfache Anbindung zwischen der in allen Spielfeldkomponenten benutzten Kommunikations-Architektur und der Host-Architektur zu schaffen, wurde der Scheduler um zusätzliche Job-Typen erweitert. Zum Einen wurden Jobs hinzugefügt, um in der Host-Architektur die zur Spielphase passenden Bildschirme aufzurufen, und zum Anderen, um Daten auszutauschen, beispielsweise die erhaltenen Programmierkarten oder Schadenspunkte. Zusätzlich wurden Jobs eingefügt, um den Master aufzufordern Karten zu geben, Optionskarten zu verteilen oder die Spielerreihenfolge anhand der Prioritäten auszurechnen.

Der Datenfluß wurde in Abbildung 6.2 als Sequenzdiagramm für die Spielsituation „Kartengeben“ dargestellt. Der Scheduler ruft in regelmäßigen Abständen die Methode *GetFirstJob()* auf. Dieser Aufruf liefert den als nächstes zu bearbeitenden Job aus der Job-Queue des Schedulers zurück. In diesem Beispiel einen Job mit dem Typ *GIVE_CARDS*.

[1] http://wxwidgets.org/

Daraufhin wird durch die Methode *ProcessJob(Job)* dieser Job abgearbeitet. In diesem Fall wird duch die *CallAction(JobType, Job)*-Methode die Abarbeitung an die *R3D3*-Instanz der Host-Architektur abgegeben und durch den Aufruf von *AddPendingEvent(Job)* für die weitere Bearbeitung vorgemerkt. Durch *ProcessAction(Job)* wird dann die *DealProgramCards(Job)*-Methode aufgerufen. Der übergebene Job enthält die Informationen für wieviele Spieler Karten gegeben werden sollen und wieviele Karten jeder Spieler aufgrund seiner Schadenspunkte überhaupt noch erhält.

Nachdem die Karten per Zufall aus dem „Deck“ gezogen worden sind, wird beim Scheduler ein neuer Job angefordert. In diesen Job werden nun die Karten eingehängt, damit sie vom Scheduler weiterverarbeitet werden können. Die Methode *GetFreeJob()* gibt dabei nicht nur einen neuen Job zurück, sondern hängt diesen auch gleich für die weitere Bearbeitung in die Job-Queue ein. Der Scheduler wird nun in einem der nächsten *GetFirstJob()*-Aufrufe den neuen Job mit den Karten abarbeiten und an die Statemachine weiterreichen.

Die Lösung der Anbindung durch zusätzliche Jobs hat sich als sehr gut erweiterbar erwiesen, da die benötigten Daten für die entsprechenden Funktionen gleich mit an den Job angehängt werden konnten. So konnten auch nachträglich noch Funktionen problemlos hinzugefügt werden, die eigentlich ohne Benutzerinteraktion laufen sollten, wie zum Beispiel das Zurücksetzen der Roboter auf ihr letztes Backup-Feld.

6.3.4 Parallelität von Host- und Kommunikations-Architekur

Da C/C++ und die *wxWidgets*-Bibliothek eine einfach Unterstützung für nebenläufige Kontrollflüsse durch Threads bieten, wurde das Konzept des Schedulers, wie sonst in der Kommunikations-Architektur verwendet, für die Host-Software etwas aufgeweicht. Statt die Parallelität der Host-Architektur, des Schedulers und des Com_Interface nur über die im Scheduler verarbeiteten Jobs laufen zu lassen, wurde für den Host jeweils ein Thread für diese drei Programmteile gestartet.

Dieser Schritt war nötig, um eventuelle Nachrichtenverluste zu vermeiden, die aufgeträten wären, wenn der Kontrollfluß für eine längere Operation in der Host-Architektur geblieben wäre, während gleichzeitig eine Nachricht vom Com_Interface verarbeitet werden sollte.

Um den Kontrollfluß des Schedulers zusätzlich zu entkoppeln, werden die Methoden der Host-Architektur nicht direkt aufgerufen. Stattdessen wurde auf eine asynchrone Abarbeitung durch den Kontrollfluß der Host-Architektur gesetzt. Um nicht eine weitere Job-Warteschlange im Host implementieren zu müssen, wird auf die Event-Queue zurückgegriffen, die für die Abarbeitung der *Windowevents* zuständig ist, die von den *wxWidgets*-Klassen generiert werden. Diese Events werden in einem *Eventloop* abgefragt, so daß das Programm auf Mausklicks oder Tastatureingaben reagieren kann.

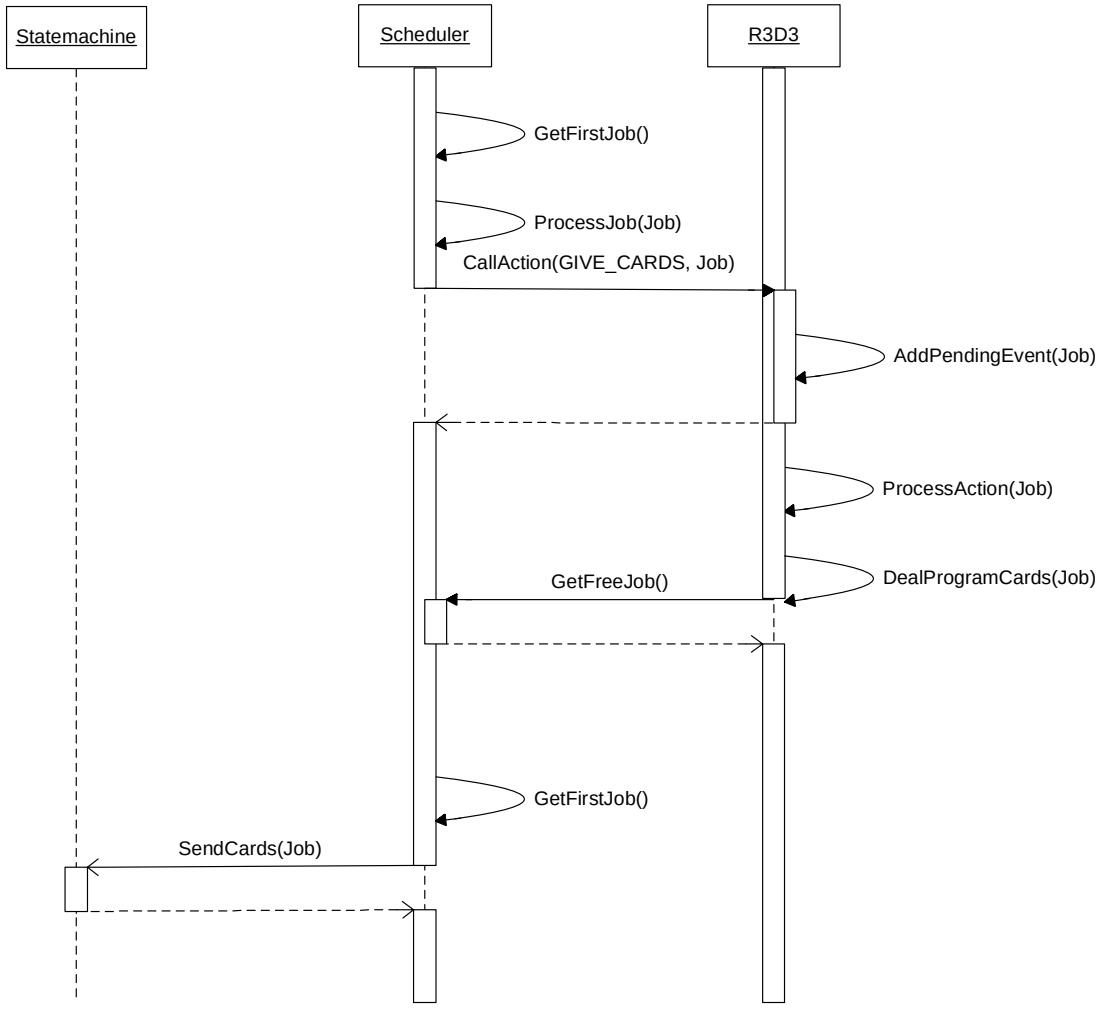


Abbildung 6.2: Datenaustausch

In dem die Jobs, die von der Host-Architektur verarbeitet werden sollen, auch in diesen Eventloop eingehängt werden, kann der Kontrollfluß sofort zum Scheduler zurückkehren, und die eigentliche Operation wird daraufhin durch den Thread der Host-Architektur, in der der Eventloop läuft, ausgeführt.

In Abbildung 6.2 ist ist diese Trennung zu erkennen. Die Methode *CallAction(Job)* löst intern die Methode *AddPendingEvent(Job)* aus, die den Job in ein Event kapselt und dann in den Eventloop einhängt. Diese Events werden dann vom Kontrollfluß der R3D3-Klasse durch eine spezielle Eventbehandlung abgefangen, der Job aus dem Event ausgelesen und an die Methode *ProcessAction(Job)* deligiert, die die weitere Auswertung des Jobs vornimmt und die nächste Operation entsprechend des Job-Typs auslöst.

6.3.5 Statemachine der Kommunikations-Architekur im Host

Im Folgenden wird das Zustandsübergangsdiagramm (Abbildung 6.3) des Hosts beschrieben. In diesem Diagramm lassen sich auch die unterschiedlichen Aufgaben und Abläufe der Hosts, die als Client agieren, und des Hosts, der die Rolle des Masters übernimmt, erkennen. Die Syntax der Abbildung ist die gleiche wie in Abschnitt 4.4.

Nach dem Einschalten gelangt das System in den Zustand *Init*. In diesem Zustand werden Nachrichten vom Typ Init bearbeitet, indem eine Init-Response an den Parkplatz gesendet wird. Der Host, dessen Init-Response zuerst erhalten wurde, wird zum Master erklärt. Der Parkplatz sendet nach dem Erhalt aller Init-Response-Nachrichten eine Register-All-Nachricht. Diese Nachricht enthält die Adressen aller am Spiel teilnehmenden Spieler, wobei die Adresse des Masters als erste in dieser Nachricht steht.

Die Clients wechseln nun in die Abläufe, die durch die Client-Statemachine beschrieben werden, und der Master führt die Abläufe entsprechend der Master-Statemachine aus.

6.3.5.1 Client-Statemachine

Der Client wechselt in den Zustand *C_Init*, in dem er darauf wartet, vom Master mittels einer Cards-Nachricht die Karten zugeschickt zu bekommen. Durch diese Nachricht wechselt er in den Zustand *C_Neue_Runde*.

Nachdem der Benutzer die Spielkarten zu einer Programmsequenz zusammengestellt hat, wird diese mit der Programmsequence-Nachricht an den Master gesendet und der Zustand auf *C_WarteAufBefehl* gesetzt. Wenn der Client eine NextOrder-Nachricht empfängt, wird dem Spielfeld der nächste Befehl, den der Roboter auszuführen hat, gesendet und im Zustand *C_Ausführung* auf die Antwort in Form einer NextOrder-Response-Nachricht gewartet.

Durch die Nachricht Order-Response wechselt der Client in den Zustand *C_Fabrikelemente*. In diesem Zustand werden Factoryelement-Response Nachrichten mit den Nachrichten Statusupdate und Factory-Done beantwortet. Dadurch wird allen Hosts mitgeteilt, wie weit die Abarbeitung der Programmsequenzen fortgeschritten ist.

Wenn der Client in der nächsten Runde an der Reihe ist, seinen Befehl auszuführen, bekommt er wieder eine NextOrder-Nachricht und wechselt in den Zustand *C_Ausführung* zurück. Sobald alle Befehle abgearbeitet sind, bekommt der Client eine EndOfRound-Nachricht und sendet daraufhin alle Programmierkarten, die nicht festgebrannt sind, an den Master zurück und wechselt in den Zustand *C_Rundenende*. In diesem Zustand wird darauf gewartet, dass der Master durch das Senden der Programmierkarten die nächste Runde einleitet.

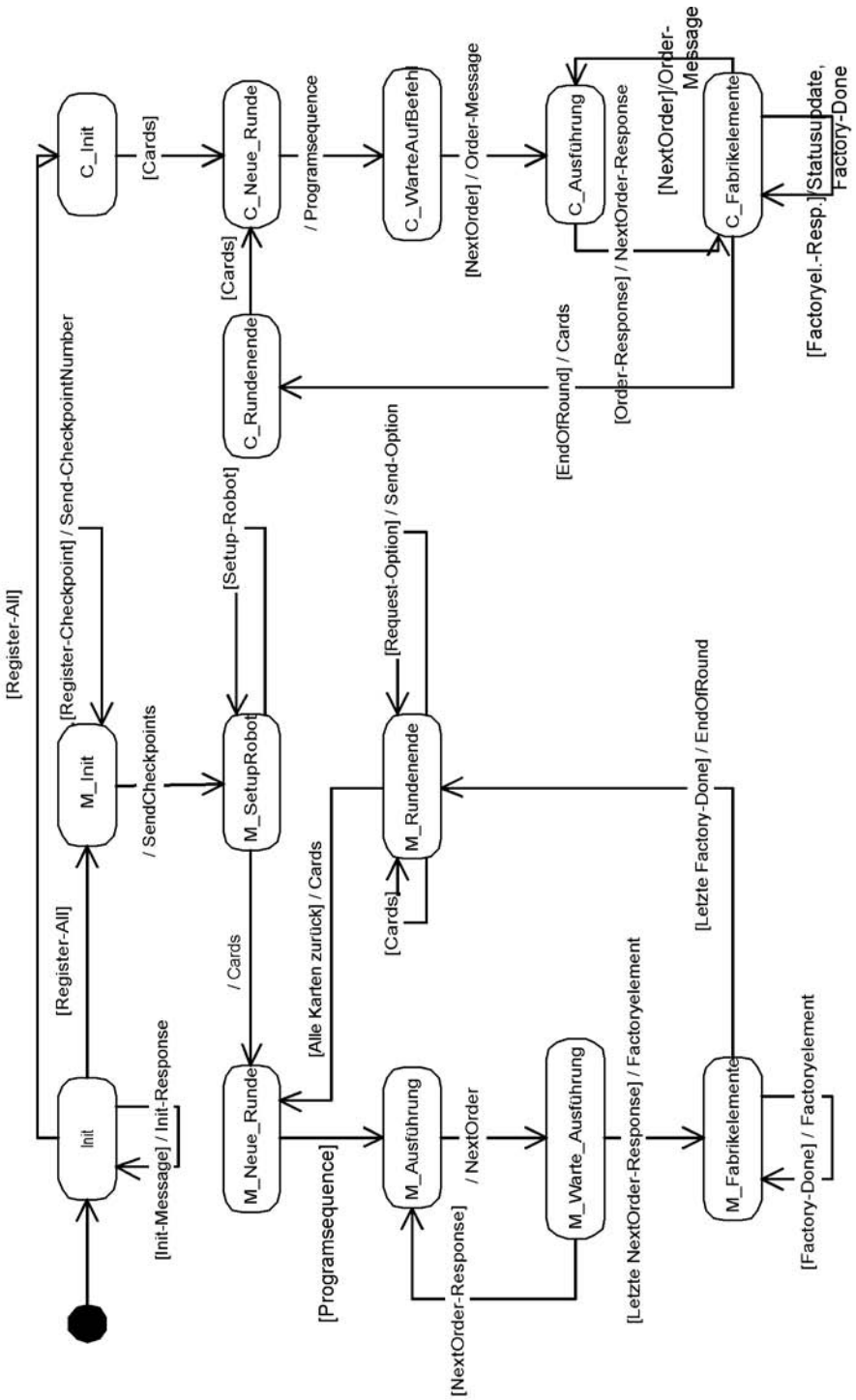


Abbildung 6.3: Host-Statemachine

6.3.5.2 Master-Statemachine

Der Master wechselt nach dem Erhalt der Register-All-Nachricht in den Zustand *M_Init*. Dieser Zustand dient dazu, dass der Benutzer die Checkpoints einsteckt. Wenn ein Spielfeld bemerkt, dass ein Checkpoint eingesteckt wurde, sendet es eine Register-Checkpoint-Nachricht an den Master. Da das Spielfeld nicht entscheiden kann, der wievielte Checkpoint eingesteckt wurde, wird daher die aktuelle Checkpointnummer beim Master angefragt. Der Master schickt dem Spielfeld mit einer Send-Checkpointnumber-Nachricht die Nummer des Checkpoints zu. Wenn der Benutzer das Einstecken der Checkpoints beendet hat, sendet der Master die Adressen der Checkpoints mit einer SendCheckpoints-Nachricht an alle Clients.

Darauf folgt die Phase des Spiels, in der die Roboter initialisiert werden. Dazu sendet der Master dem Parkplatz eine Setup-Robot-Nachricht. Dieser antwortet ebenfalls mit einer Setup-Robot-Nachricht, wenn die Initialisierung und Positionierung abgeschlossen ist.

Nachdem alle Roboter positioniert wurden, sendet der Master Cards-Nachrichten an alle Clients, um ihnen Karten zu geben und die erste Runde zu beginnen. Dabei wechselt der Master in den Zustand *M_Neue_Runde*. Haben alle Clients dem Master eine Programmsequence-Nachricht gesendet, geht der Master in den Zustand *M_Ausführung* über. In diesem Zustand wird geprüft, ob alle Befehle dieser Stufe abgearbeitet sind. Zur Befehlsverarbeitung wird dem entsprechenden Client eine NextOrder-Nachricht geschickt und in den Zustand *M_Warte_Ausführung* übergegangen. Nachdem dieser Befehl abgearbeitet und der Client mit einer NextOrder-Response geantwortet hat, geht der Master in den Zustand *M_Ausführung* zurück.

Falls es sich um den letzten Befehl dieser Runde gehandelt hat, beginnt nun die Ausführung der Fabrikelemente. Dazu sendet der Master eine Factoryelement-Nachrichtund wechselt in den Zustand *C_Fabrikelemente*. Wenn alle Clients mit Factory-Done geantwortet haben, schickt der Master erneut eine Factoryelement-Nachricht, um die nächste Stufe der Fabrikelemente anzustoßen. Sind alle Fabrikelemente abgearbeitet, sendet der Master eine EndOfRound-Nachricht, um die Clients aufzufordern, ihre Karten zurückzuschicken. Dabei wechselt der Master in den Zustand *M_Rundenende*. Steht der Robter eines Clients auf einem Doppelreperaturfeld, kann dieser Client in diesem Zustand vom Master mittels einer Request-Option-Nachricht eine Optionskarte anfordern, die dann mit einer Send-Option beantwortet wird. Nachdem der Master alle Karten zurückbekommen hat, wechselt er in den Zustand *M_Neue_Runde*.

6.4 Zusammenfassung und Bewertung

Die Entscheidung, die Host-Software in C/C++ zu schreiben, hat sich als richtig erwiesen. Der Zeitgewinn, der sich durch den hohen Grad der Wiederverwendung der Kommunikations-Architektur ergeben hat, hat dafür gesorgt, daß der Zeitplan für die

Entwicklung der Software eingehalten werden konnte. Nur zum Ende der Programmierung konnte die Testphase nicht wie geplant anlaufen, da sich, bedingt durch die Neukonstruktion des Roboters, die Tests mit der engültigen Hardware verzögerten.

Außerdem wurden durch den Einsatz von C/C++ Portierungsfehler des Protokoll-Stacks nach z.B. Java oder komplizierte Wrapperklassen für den Einsatz der Kommunikations-Architektur vermieden.

Durch die Verwendung der *wxWidgets*-Bibliothek konnte auf einfache und effiziente Weise eine intuitive GUI implementiert werden. Screenshots der enstandenen GUI finden sich in Kapitel 7.

Außerdem sollte durch die *wxWidgets*-Klassen die Plattformunabhängigkeit der Software sichergestellt werden. Wie in Abschnitt 6.3.2 beschrieben, bietet diese Bibliothek die Möglichkeit, die Software für die gängisten Zielplattformen zu kompilieren. Die Portierbarkeit der Host-Software auf ein Linuxsystem wurde letztendlich nicht mehr überprüft, da die Tests nach der Fertigstellung der Roboter eine höhere Priorität hatten.

Durch die gute Modularisierung der Kommunikations-Architektur konnten die Anpassungen, die für die Host-Software nötig waren, einfach durchgeführt werden. Auch die Trennung von Master und Client konnte so ohne größeren Aufwand nur durch unterschiedliche *Statemachines* realisiert werden.

Die simple Erweiterbarkeit der Anbindung an die Host-Architektur erwies sich ebenfalls als vorteilhaft. Wie in Abschnitt 6.3.3 beschrieben, konnten so Features, die nicht mehr realisiert oder nicht ausreichend getestet werden konnten, durch zusätzliche Dialoge und Benutzerinteraktion auf einfache Weise ersetzt werden.

Die Programmierung der Host-Software erwies sich letztendlich als relativ problemlos und fehlerarm. Bei den Tests wurden nur wenige Fehler im Programmablauf und dem Zusammenspiel der Softwarekomponenten gefunden. Dies begründet sich höchstwahrscheinlich dadurch, dass es einfach und früh möglich war, Tests dieser Software durchzuführen. Alle Komponenten wurden erst für sich und später in einem monolithischen Test in einer Testumgebung auf ihr Zusammenspiel überprüft. Somit wurde eine weitgehende Fehlerfreiheit der Host-Software erreicht.

Zusätzlich hat die Verständigung und Absprache der Entwickler untereinander bei der Implemtierung der Software problemlos funktioniert. So konnten die Entwickler unabhängig voneinander größere Teile der Software programmieren und die Entwicklung zügig voran treiben, ohne auf tägliche Zusammenarbeit angewiesen zu sein.

7 Spielablauf

In den nun folgenden Abschnitten wird auf das realisierte Robo-Rally-Spiel eingegangen. Hierbei wird neben dem Spielfeldaufbau, welcher aus mehreren Spielfeldboxen besteht, auch die Horsteinstellung erläutet.

7.1 Spielvorbereitung

Das Spiel wurde so realisiert, dass maximal vier Spieler miteinander spielen können. Vor dem Spielbeginn werden zuerst die Spielfeldboxen zu einem individuellen Spielfeld zusammengesteckt. Die Spielfeldboxen werden über die Parkplatz-Spielfeldbox mit Strom versorgt. Hierfür befindet sich ein spezieller Anschluss, welcher mit dem Netzteil verbunden wird. Die Konsolen werden mit dem Spielfeld über Spezielle Host-Schnittstellen verbunden. Die Roboter werden auf die Parkplatzpositionen gesetzt.

7.2 Initialisierungsphase

Nachdem alle Komponenten angeschlossen wurden, kann das Programm auf den jeweiligen Konsolen gestartet werden. Durch das Starten des Programms auf den Konsolen findet in den angeschlossenen Spielfeldboxen eine Initialisierung statt. Dabei wird im Spielfeld ausgehend vom Parkplatz ein Koordinatensystem aufgebaut, so dass die einzelnen Spielfeldboxen direkt adressiert werden können. Dasselbe gilt auch für die angeschlossenen Konsolen. Hinzu kommt jedoch, dass sich die Konsolen beim Parkplatz zurückmelden. Die erste Konsole, die sich beim Parkplatz zurückmeldet, wird als Master ausgezeichnet. Die oben erwähnten Etappen werden den Spielern angezeigt durch den Splashscreen (siehe Abbildung 7.1).

Abbildung 7.1: SplashScreen

Sobald sich alle vier Konsolen zurückgemeldet haben oder der Timer abgelaufen ist, wird der Spieler auf der Master-Konsole dazu aufgefordert, in aufsteigender Reihenfolge

nacheinander die Checkpoints auf dem Spielfeld auszuwählen. Die Auswahl der Checkpoints erfolgt durch das Drücken einer Taste auf der Spielfeldbox. Jede Spielfeldbox kann maximal vier Checkpoints haben. Dieser Vorgang wird durch das folgende Bild (siehe Abbildung 7.2) auf dem Bildschirm verdeutlicht.

Abbildung 7.2: CheckpointScreen

Nachdem die Checkpoints gewählt wurden, werden die Roboter initialisiert. Die Initialisierung der Roboter erfolgt, indem die Roboter auf die entsprechenden Positionen auf dem Parkplatz gesetzt werden. Durch die Kommunikation zwischen Master und Roboter, werden die Roboter initialisiert und den jeweiligen Konsolen zugewiesen. Dieser Vorgang wird den Spielern verdeutlicht durch das folgende Bild auf der Konsole.

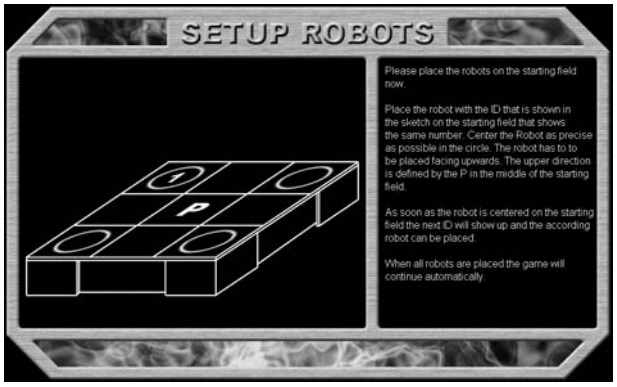


Abbildung 7.3: RobotSetupScreen

7.3 Spielanfang

Nach der Initialisierungsphase wird eine Programmoberfläche „Hauptbildschirm“ auf jeder Konsole angezeigt. Der Hauptbildschirm enthält (siehe Abbildung 7.4):

- Die von der Master-Konsole verteilten Karten,

- Die Programmslots für die Karten, die im nächsten Zug verwendet werden
- Den Status, ob und ggf. welche Karten bereits festgebrannt sind
- Den Shalter, um den Roboter für die nächste Runde abzuschalten und
- Informationen über erlittene Schadenspunkte, verbliebene Backups und den nächsten zu erreichenden Checkpoint.

Die Karten werden von links nach rechts mit fallender Priorität angeordnet, um den Spielern einen besseren Überblick zu ermöglichen. Jeder Spieler wählt fünf Karten nacheinander und platziert sie in die unteren leeren Felder. Die Reihenfolge der Karten entspricht auch der Ausführungsreihenfolge der Befehle. Die Idee des Spiels besteht darin, die Karten so zu positionieren, dass der Roboter am schnellsten den nächsten Checkpoint erreicht. Nachdem die Spieler die Karten sortiert haben, geben sie der Master-Konsole Informationen, indem sie auf „Ready“ Klicken. Anhand der Kartenprioritäten entscheidet die Master-Konsole über die Ausführungsreihenfolge des Spielzuges.

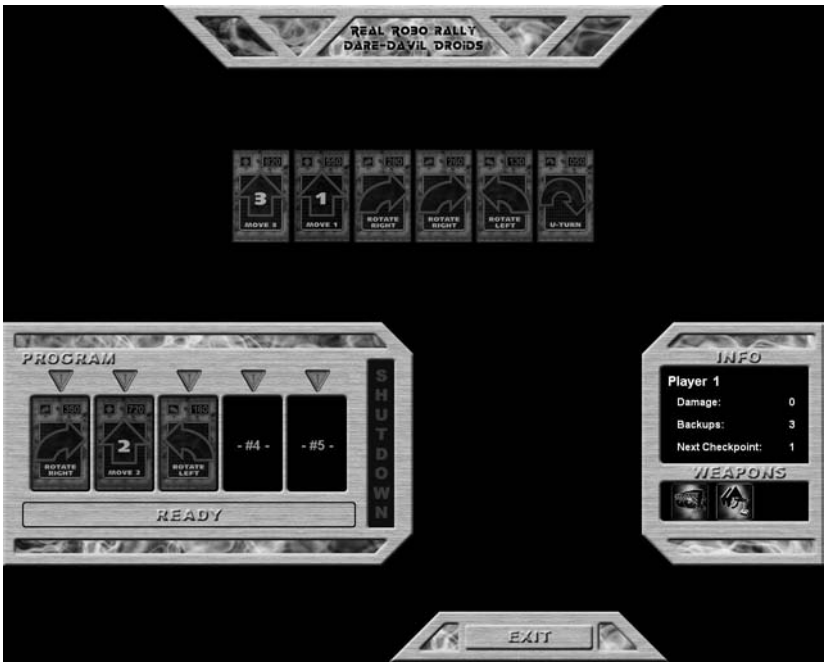


Abbildung 7.4: ProgramScreen

Während die Konsolen die Befehle an den Roboter übertragen, wird den Spielern eine Übersicht präsentiert (siehe Abbildung 7.5) die dem Spieler übermittelt, wie weit die Befehlsabarbeitung fortgeschritten ist, und wie der Status der anderen Roboter aussieht.

Nachdem die jeweiligen Roboter ihren Spielzug ausgeführt haben, werden die Fabrikelemente aktiviert. Die Aktivierung erfolgt nach der folgenden Reihenfolge:

Abbildung 7.5: Übersichtsbildschirm

1. Expressbänder
2. Förderbänder
3. Schieber
4. Zahnräder
5. Presse
6. Grube
7. Brenner
8. Laser

Nachdem eine Spielrunde abgeschlossen ist und noch kein Spieler die Checkpoints in der richtigen Reihenfolge erreicht hat, werden die Karten neu verteilt. Die Karten werden anhand der Schadenspunkte der jeweiligen Roboter verteilt. Spieler, deren Roboter mit zunehmenden Schadenspunkten belegt sind, bekommen weniger Karten.

7.4 Spielende

Sobald ein Roboter das Zielcheckpointfeld in der resultierenden Reihenfolge erreicht hat, wird der Spieler durch eine Gewinnmeldung auf der Spielerkonsole informiert.

Abbildung 7.6: Spielende-Bilschirm

8 Zusammenfassung und Ausblick

8.1 Status

Der Status-Abschnitt soll als kurze Zusammenfassung der vorhergehenden Kapitel dienen, um einen groben Überblick der Fortschritte der Projetkgruppe R3D3 zu bekommen. Die Struktur des Abschnitts orientiert sich an dem Aufbau der zusammengefassten Kapitel.

8.1.1 Spielfeld

Da keine Fertiglösung alle Anforderungen an das Spielfeld zufriedenstellen konnte, musste zuerst eine Entscheidung getroffen werden, welche Materialien verbaut werden. So bestehen die Seitenwände aus Epoxydharz-Platinen, die mit 4 Aluminium-Winkeln verschraubt werden. Eine durchsichtige Makrolon-Platte bildet die Oberfläche. Davon ausgehend hat es regelmäßig Kontaktprobleme gegeben, da Nachbarn unter Umständen nicht mit Srom versorgt wurden, wenn die Auflagefläche der stromführenden Winkel nicht groß genug war. Ein exakter Zusammenbau und eine Reinigung hat hier geholfen. Auf unebenem Boden kann der Kontakt auch alternativ durch Unterlegscheiben zwischen den Winkeln sichergestellt werden, die dank der Magneten leicht haften.

8.1.2 Roboter

Da auch für den Roboter keine Fertiglösung gewählt werden konnte, mussten aus einer Vielzahl von Bauteilen die richtigen ausgewählt werden. Als besonders aufwändig erwies sich die Suche und der Zusammenbau der Getriebe und das Auffinden eines geeigneten Akkus. Dies ist auf die besonderen Anforderungen bezüglich der Größe zurückzuführen. Fast alle Komponenten müssen zuerst auf ihre Größe bezüglich der Verwendbarkeit geprüft werden.

Schließlich hat sich gezeigt, dass der erste Entwurf nicht zum Ziel führt. Die beiden Sebstbaugetriebe liefen nicht dauerhaft zuverlässig und konnten auch durch weitere Stabilisierungen nicht nutzbar gemacht werden. Deswegen war es nötig, einen alternativen Entwurf neu zu entwickeln. Von diesem wurden vier Roboter samt Hüllen gefertigt, kalibriert und im Spiel getestet.

8.1.3 Host

Der Schwerpunkt der Aufgaben für dieses Semester lag vor allem in der Anbindung an die unterliegende Kommunikations-Architektur und im Testen des tatsächlichen Spielverlaufs. Der Host wurde an verschiedene Aufbauten angeschlossen und getestet. Fehler wurden korrigiert und fehlende Bildschirmseiten in der Benutzerschnittstelle eingefügt. Der Host-Adapter wurde in Betrieb genommen und erlaubt nun ein unkompliziertes Anschließen von Host-Rechnern an beliebiger Stelle. Der Mehrspielerbetrieb funktioniert, und die Regeln des Spiel werden eingehalten.

Abschließend lässt sich feststellen, dass die PG-Ziele erreicht wurden.

8.2 Ausblick

Das Spiel ist in der jetzigen Form durchaus spielfähig. Trotzdem gibt es in allen Bereichen noch einige Punkte, mit denen man das Spiel weiter ausbauen könnte. Einige davon sind schon zu Anfang der PG angesprochen worden, weitere erschließen gegebenenfalls neue Schwerpunktfelder.

Der Host:

- Erweiterung des R_3D_3 Spiels um Regeln und Spiel-Varianten

  Während der Konzeptphase wurden aus Gründen des hohen Aufwandes einige dieser ausgeschlossen oder nicht betrachtet. Dazu gehören beispielsweise einige Optionskarten.

- Entwicklung externer Spiel-Konsolen.

  Diese sollen die Interaktion der Spielfläche mit dem Host-PC während des Spielbetriebs ablösen. Anstelle eines PC erhält jeder Spieler eine dedizierte Konsole, die an beliebiger Stelle am Spielflächenrand angeschlossen wird. Die Steuerbefehle eines Spielers für seinen eigenen Roboter werden dann von der Konsole an die Spielfläche übertragen.

- Entwicklung einer selbständig agierenden KI

  Diese könnte beispielsweise fehlende Mitspieler ersetzen. Jedoch müsste zuvor ein Mechanismus implementiert werden, der das Aussehen des Spielfeldes und die Position aller Roboter an diese KI liefert.

Roboter:

- Veränderung des Unterbaus

Da die Lichtverhältnisse unter dem Roboter sehr gut sind, wäre beim Neubau von Robotern ernsthaft zu überlegen, den Fototransistor aus der Mitte an das hintere Ende zu verschieben. Dies erlaubt eine exaktere Fehlerkorrektur, da über die Längsachse Lichtwerte gelesen werden können. Zudem würde dies auch die vier Kommunikations-LEDs des Roboters auf eine reduzieren, da nun die eine LED in der Mitte genau über einem - dann genau in der Mitte angebrachten - Fototranistor auf dem Feld kommunizieren kann.

- Erhöhung der Genauigkeit des Antriebs

  Auch wenn der Roboter sehr präzise fährt, könnte diese durch Änderung der Viereckspiegel, die zur Messung von Radumdrehungen benutzt werden, auf acht Ecken gegebenenfalls die Präzision noch weiter erhöhen und so die Bewegungen weiter verfeinern.

Das Spielfeld:

- Erweiterung des Roboter-Routing zur Minimierung der Benutzereingriffe

  Im Falle eines Roboter-Todes könnte der Roboter zum letzten Checkpoint selbständig zurückfahren. Auch wenn der Roboter durch Stromausfall seinen Zustand verliert, könnte er dann auf den Parkplatz gestellt und von dort automatisch an die richtige Stelle gefahren werden.

- Vergrößerung des Speichers

  Sollten Boxen neu gebaut werden, würde sich unter Umständen eine Vergrößerung des Speichers lohnen, um auch Spielflächen wesentlich größerer Dimension aufbauen zu können. Die aktuellen Mikrocontroller sind fast am Limit.

9 Anhang

9.1 Schaltpläne

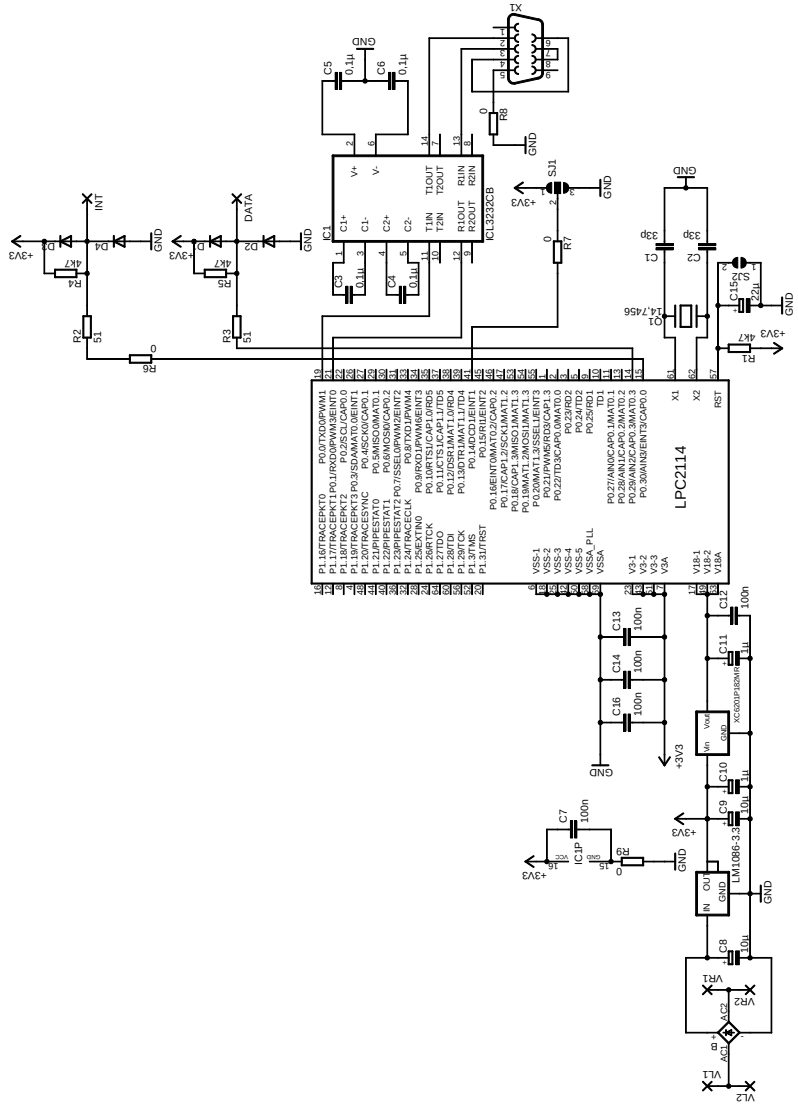


Abbildung 9.1: Schaltplan Host-Schnittstelle

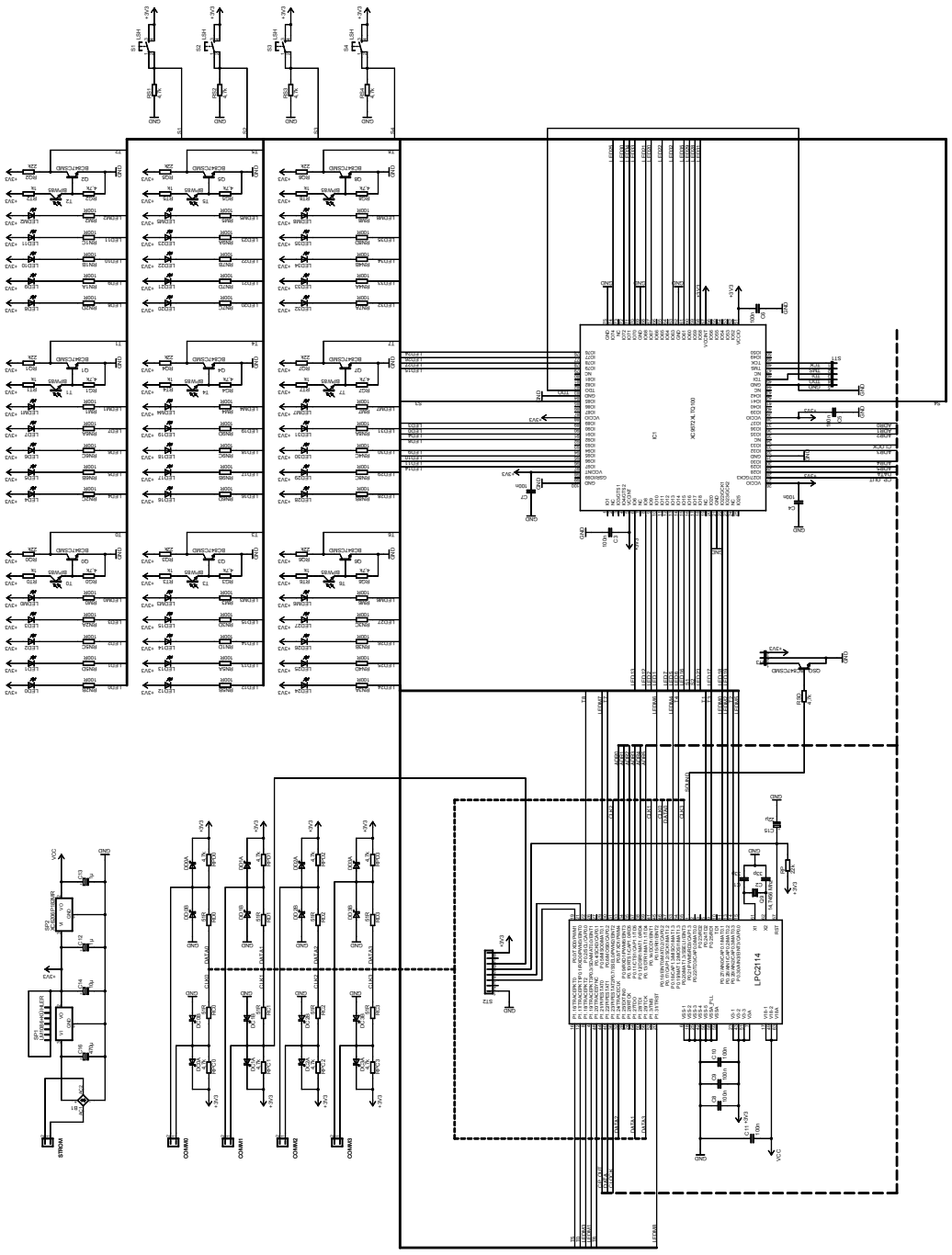


Abbildung 9.2: Schaltplan Feldplatine Version 2.0

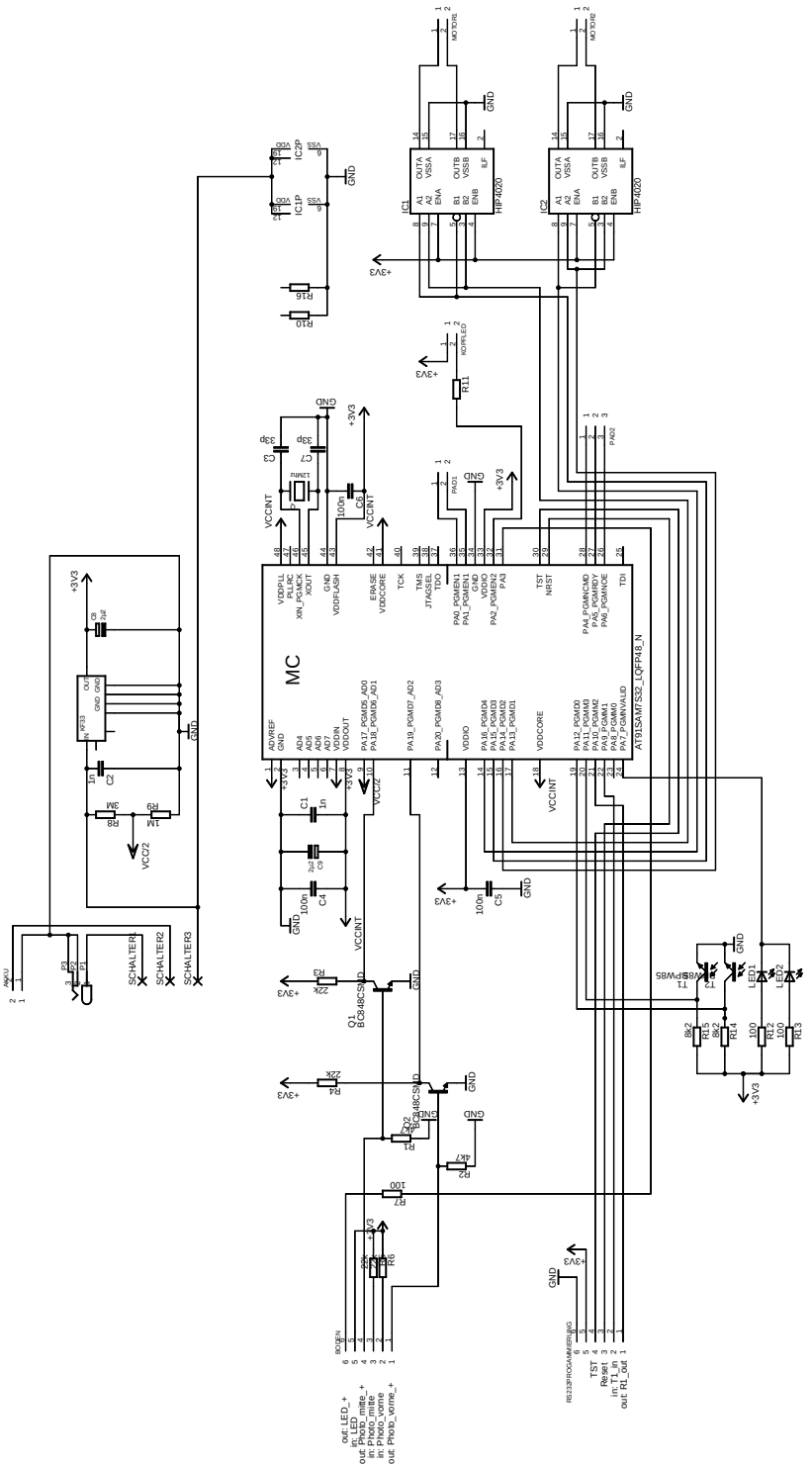


Abbildung 9.3: Schaltplan Roboter

9.2 Platinenlayouts

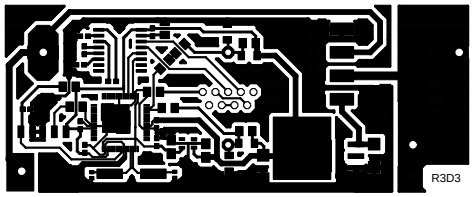


Abbildung 9.4: Platinenlayout Host-Schnittstelle - Routing - Oberseite

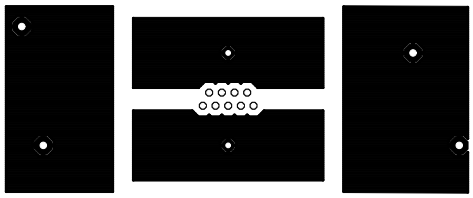


Abbildung 9.5: Platinenlayout Host-Schnittstelle - Routing - Unterseite

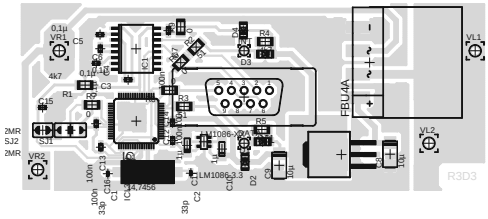


Abbildung 9.6: Platinenlayout Host-Schnittstelle - Bauteile - Oberseite

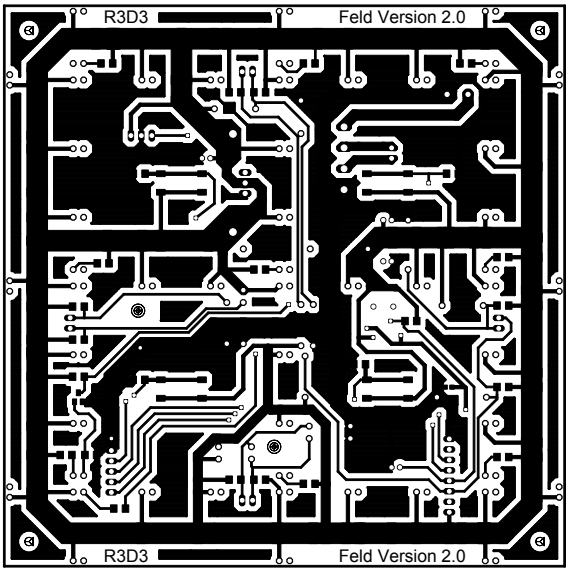


Abbildung 9.7: Platinenlayout Feldplatine Version 2.0 - Routing - Oberseite

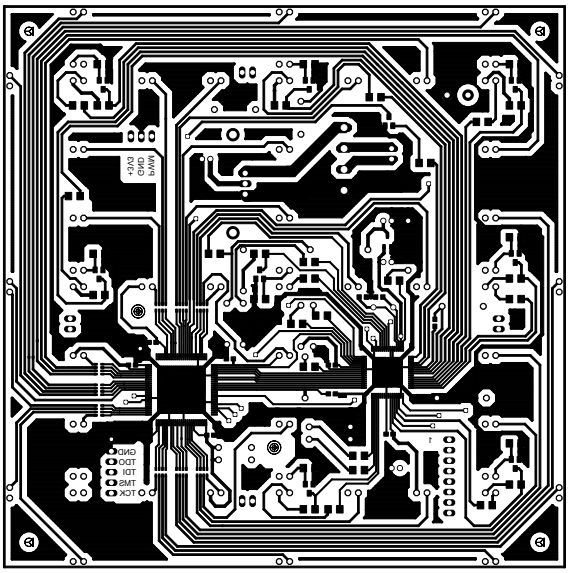


Abbildung 9.8: Platinenlayout Feldplatine Version 2.0 - Routing - Unterseite

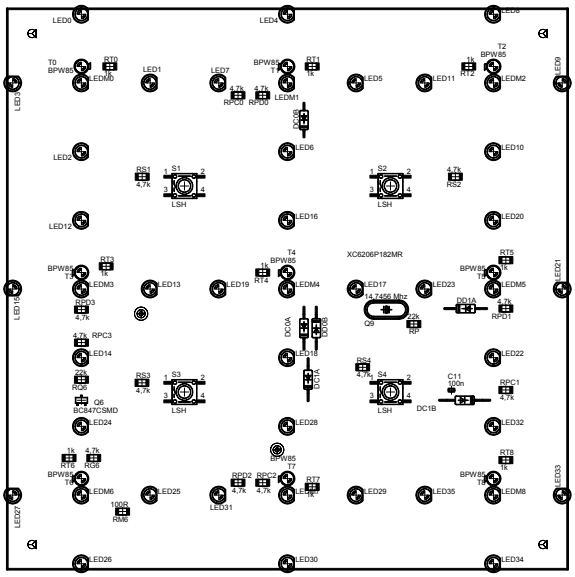


Abbildung 9.9: Platinenlayout Feldplatine Version 2.0 - Bauteile - Oberseite

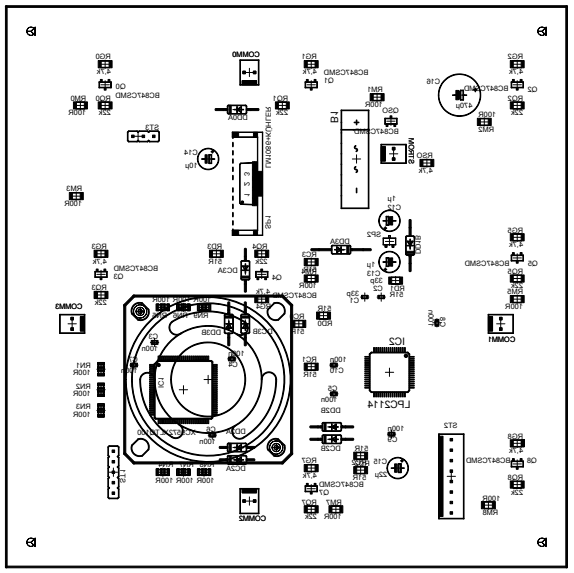


Abbildung 9.10: Platinenlayout Feldplatine Version 2.0 - Bauteile - Unterseite

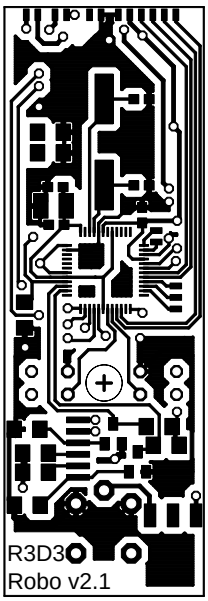


Abbildung 9.11: Platinenlayout Roboter - Routing - Oberseite

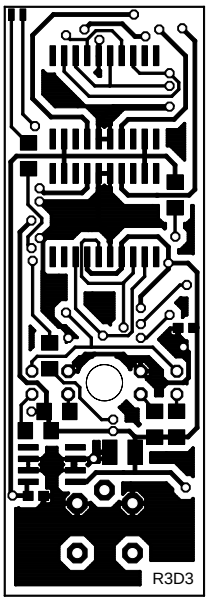


Abbildung 9.12: Platinenlayout Roboter - Routing - Unterseite

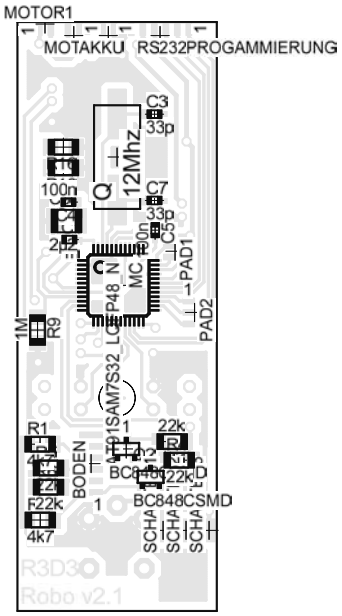


Abbildung 9.13: Platinenlayout Roboter - Bauteile - Oberseite

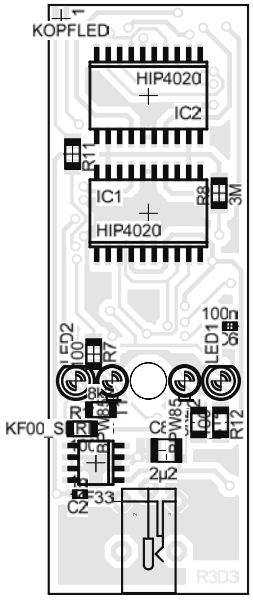


Abbildung 9.14: Platinenlayout Roboter - Bauteile - Unterseite

9.3 Mechanischer Aufbau des Spielfeldes

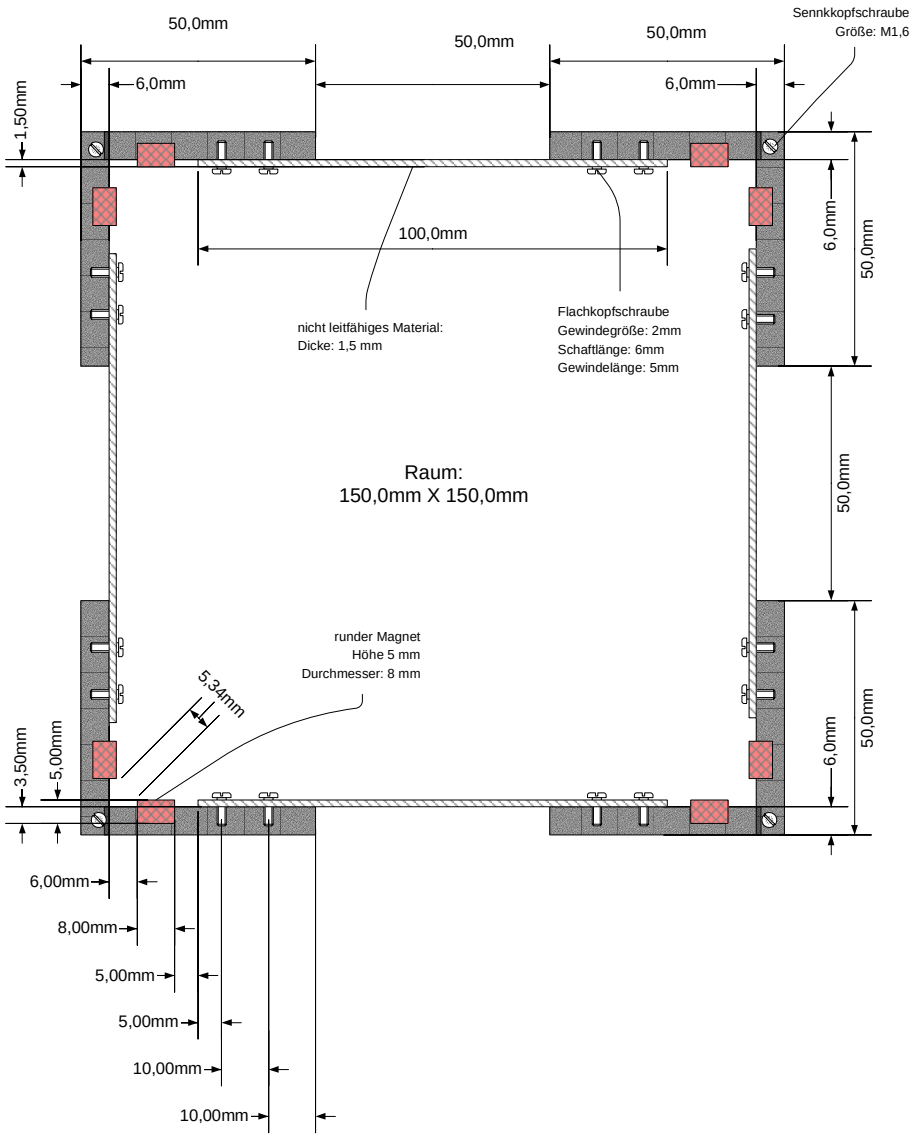


Abbildung 9.15: Spielfelfbox von oben gesehen

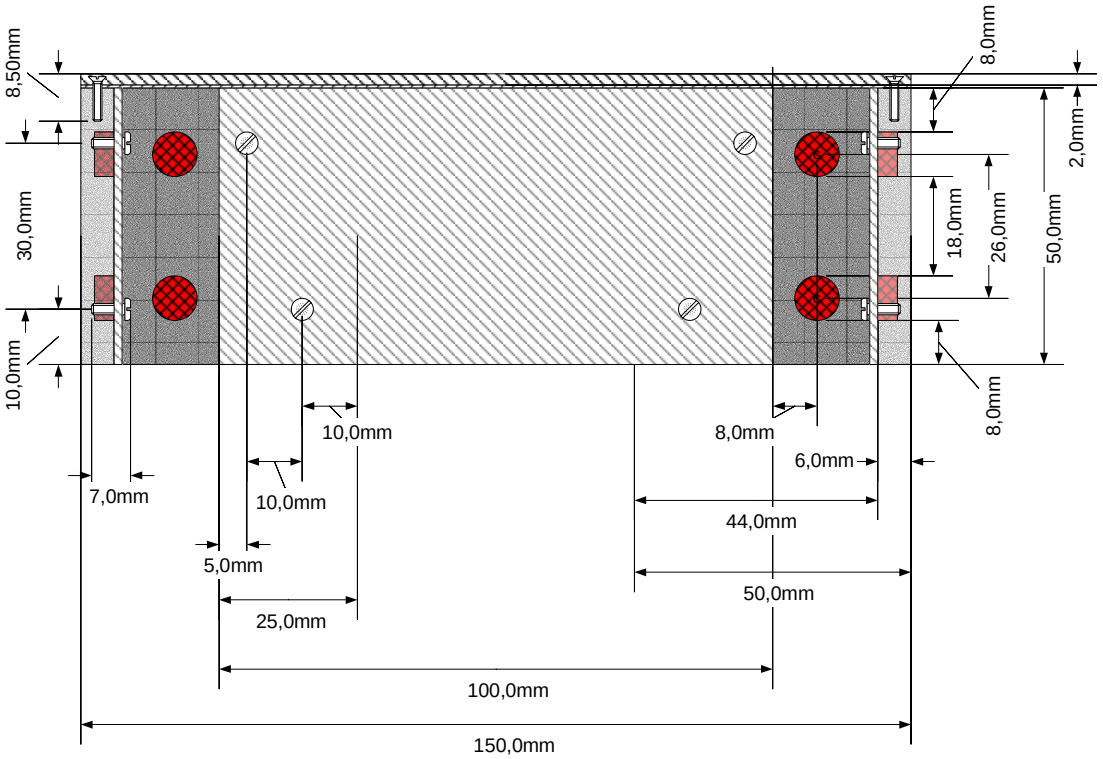


Abbildung 9.16: Spielfeldbox von der Seite gesehen

9.4 Mechanischer Aufbau des Roboters

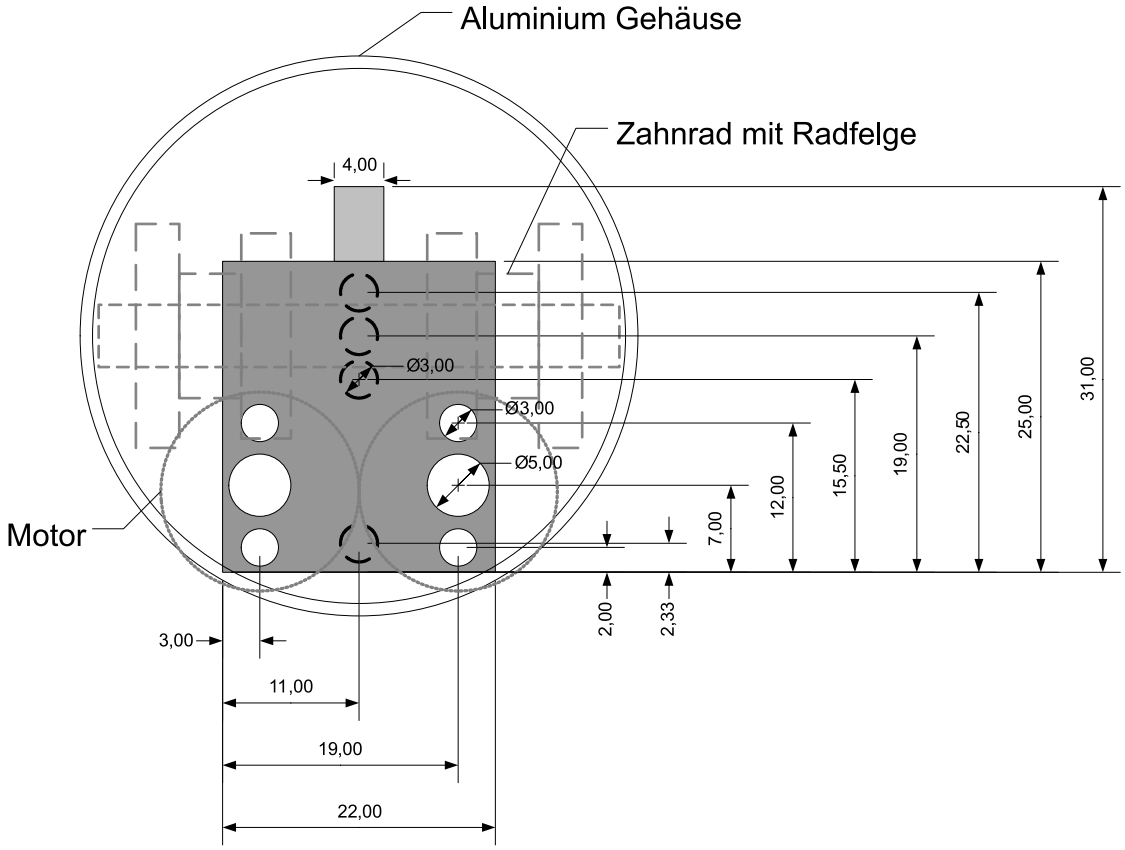


Abbildung 9.17: Motorblock von oben gesehen

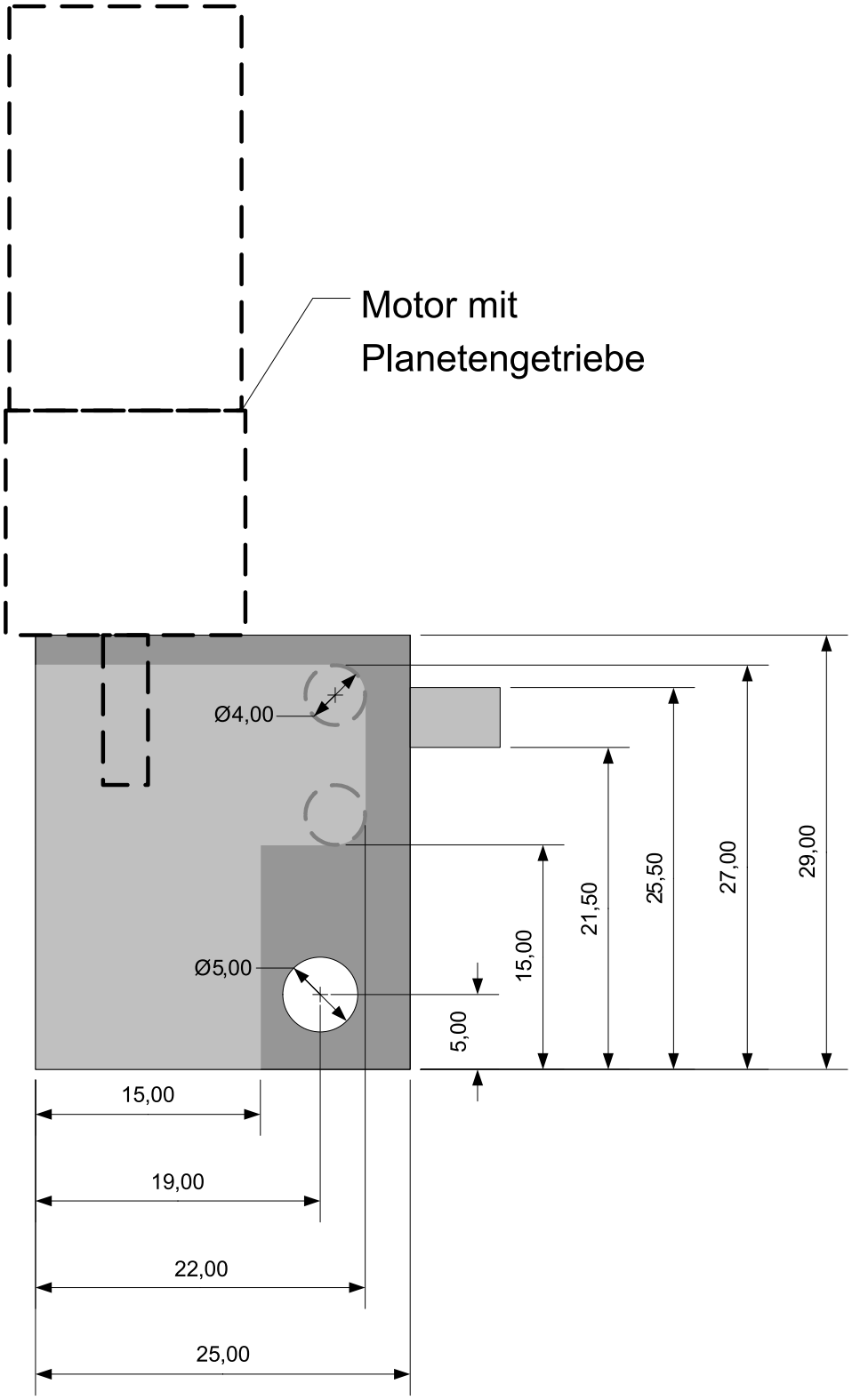


Abbildung 9.18: Motorblock von der Seite gesehen

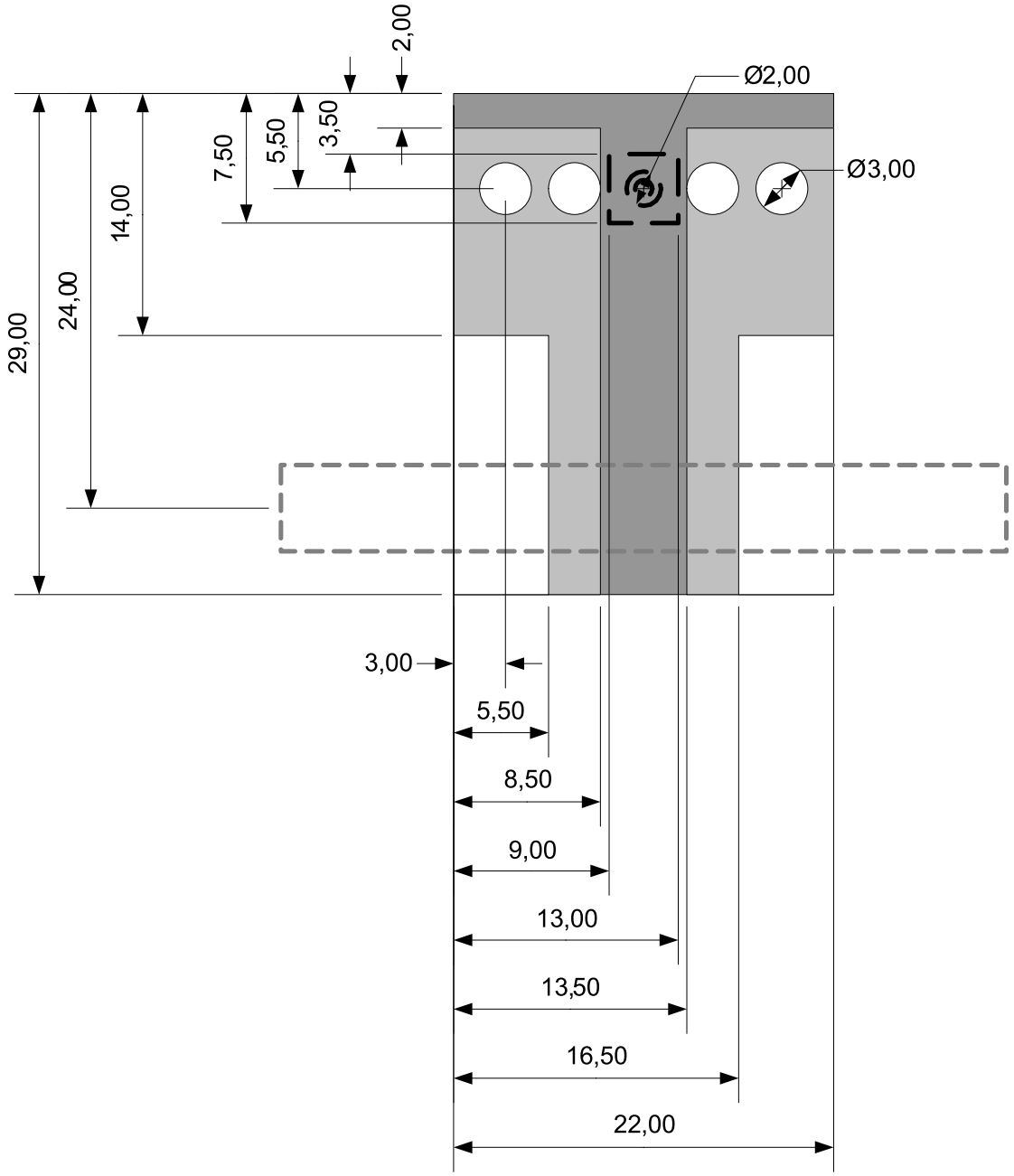


Abbildung 9.19: Motorblock von vorne gesehen

Literaturverzeichnis

[AMI99] AMIGO Spiel + Freizeit GmbH: *Robo Rally - Das rasante Rennen! - Spielanleitung*, 2. Auflage, 1999, http://www.amigo-spiele.de/upload/Roborally_1091.pdf.

[AMI00] AMIGO Spiel + Freizeit GmbH: *Robo Rally Erweiterung - Crash and Burn - Spielanleitung*, 1. Auflage, 2000, http://www.amigo-spiele.de/upload/RoboRally2_1092.pdf.

[Atm05] Atmel Corporation: *Atmel AT91SAM7S32 Datasheet Rev. 6175A*, Jul 2005, http://www.atmel.com/dyn/resources/prod_documents/doc6175.pdf.

[Dio05] Diotec: *KBU4A...KBU4M*, October 2005, http://www.diotec.com/pdf/kbu4.pdf.

[Gra04] Graupner GmbH & Co KG: *Bedienungsanleitung ULTRAMAT 12*, November 2004, http://www.graupner.de/fileadmin/downloadcenter/anleitungen/6412.69.pdf.

[Int02] Intersil Corporation: *ICL3221, ICL3222, ICL3223, ICL3232, ICL3241, ICL3243*, March 2002, http://www.intersil.com/data/fn/fn4805.pdf.

[Int05] Intersil: *Half Amp Full Bridge Power Driver for Small 3V, 5V and 12V DC Motors*, December 2005, http://www.intersil.com/data/fn/fn3976.pdf.

[Mik03] Mikroantriebe: *Getriebemotor DC Bausatz (zum Kleben)*, Juli 2003, http://www.mikroantriebe.de/download/g30bs_a.pdf.

[Nan04] Nanotec: *Stepping Motor - SP0618M0104-A-Z01*, July 2004, http://www.nanotec.de/data/zweiphasen/sp1555/media/sp0618m0104.pdf.

[Nat05] National Semiconductor: *LM1086* 1.5*A Low Dropout Positive Regulators*, June 2005, http://cache.national.com/ds/LM/LM1086.pdf.

[Phi03] Philips Semiconductors: *LPC2114/2124/2212/2214 USER MANUAL*, May 2003,

http://www.semiconductors.philips.com/acrobat/usermanauls/UM_LPC2114_2124_2212_2214_2.pdf.

[Poe36] Poe, Edgar Allan: *Maelzel's Chess-Player*. Southern Literary Journal, April 1836.

[Sma05] Smart, Julian: *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall Professional Technical Reference, 2005.

[STM04] STMicroelectronics: *KF00 SERIES*, 2004, http://www.st.com/stonline/products/literature/ds/4337/kf33b.pdf.

[Tor] Torex: *XC6201 Series Positiv Voltage Regulators*, http://www.torex-usa.com/product/pro02/pdf/XC6201_E.pdf.

[Wes04] *Katalog-D*, April 2004, http://www.wes-technik.de/Download/Katalog-D.pdf.

[Wiz05] Wizards of the Coast, Inc.: *Robo Rally - Rulebook*, 2005, http://www.wizards.com/avalonhill/rules/roborally.pdf.

[Xil03] Xilinx Inc.: *Xilinx XC9500 CPLD Series*, November 2003, http://www.xilinx.com/products/95xlsh.pdf.

[ZWB06] PG477: *R3D3 - Real Robo Rally Dare-Devil Droids - Zwischenbericht*. Technical report, University of Dortmund, April 2006.