

RFID gestützte Roboter Navigation

Bachelorarbeit

vorgelegt von Michael Antemann am: 31. August 2010

**im Studiengang Angewandte Informatik
an der Fakultät IV - Wirtschaft und Informatik
Fachhochschule Hannover**

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum

Unterschrift

Autor

Michael Antemann
Hindenburgstr. 26c
30851 Langenhagen

Tel.0511 8667120
E-Mail: Michael.Antemann@gmail.com

Erstprüferin

Prof. Dr.-Ing. Elisabeth Dennert-Möller
Fakultät IV
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover

Zweitprüfer

M.Sc. Oliver Pawlowski
Fakultät IV
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover

Inhaltsverzeichnis

1 Über diese Arbeit	4
1.1 Motivation und Aufgabe	4
1.2 Aufbau der Arbeit	5
2 RFID	6
2.1 Funktionsweise	6
2.2 Anwendungsgebiete	6
3 Navigation	8
3.1 Formen der Navigation	8
4 Verwendete Hardware	9
4.1 Der Lego NXT Roboter	9
4.2 RFID-Tags	12
4.3 Bluetooth	13
5 Verwendete Software-Technologien	14
5.1 Java Implementierung: LeJos API	14
5.2 Java API: NxtJLib und NxtJLibA	14
6 Beschreibung des Szenarios	16
6.1 Das Raster	18
6.2 Ablauf und Aufgabe	18
6.3 Der Roboter	20
7 Software	21
7.1 Anforderungen	21
7.2 Architektur	23
7.3 Roboter-Schichten im Detail	23
7.4 Server-Schichten im Detail	24
7.5 Das Programm	25
7.6 Technische Konzeption	27
8 Implementierung	31
8.1 Bestimmung der Richtung und Ausrichtung	32
8.2 Lesen und auswerten von Sensordaten	34
8.3 Abbildung des RFID-Rasters	36
8.4 Kommunikation	37
8.5 Zusammenfassung	37
9 Erkenntnisse	37
9.1 Vergleich mit ähnlichen Technologien	37
9.2 Fazit	38

Abbildungsverzeichnis

1	Links: Ein im handel erhältliches Lesegerät. Rechts: Ein RFID-Tag, außen die gewundene Antenne, in der Mitte ein kleiner Chip	7
2	NXT-Stein ohne Zubehör.	9
3	HiTechnic Kompasssensor, montiert	10
4	NXT Lichtsensor, nicht montiert	11
5	Codetex RFID-Sensor, montiert	11
6	RFID-Tag in Form eines Schlüsselanhängers(4cm x 3cm). Links: Rückseite mit logischer Referenznummer Rechts: Weiße Vorderseite	12
7	Ein Kreuzungspunkt, das Klebeband bildet die Kanten des Rasters Der Kreuzungspunk enthält unter dem Klebeband einen RFID-Tag	18
8	Grundaufbau des Roboters	20
9	Fertige Konstruktion des Roboters	21
10	Darstellung der Schichtenarchitektur	24
11	Grober Ablaufplan der Robotersoftware	26
12	Klassendiagramm der Geschäftslogik	28
13	Klassendiagramm der Aktions- und Sensorenschicht	29
14	Klassendiagramm der Kommunikationsschicht	30
15	Verlieren der Markierung und Gegenmaßnahmen	34
16	Formel zur Berechnung des arithmetische Mittels	35

1 Über diese Arbeit

1.1 Motivation und Aufgabe

Um von A nach B zu kommen stehen heutzutage sehr viele Hilfsmittel wie Kompassen, Karten, GPS Satelliten, Navigationscomputer und Markierungen oder Schilder zur Verfügung. Für Menschen reicht unter Umständen die Nutzung von Karten und Lageplänen, aber mit technischen Hilfsmitteln wie dem GPS-Handy oder einem Navigationscomputer ist eine weitaus bessere und einfachere Orientierung möglich. Besonders GPS ist sehr beliebt geworden wenn es um die Unterstützung für das Auto in Städten und auf dem Land geht. Mit der Einschränkung, dass diese Technologie nur auf einige Meter genau ist und einen ständigen Kontakt zu den Satelliten erfordert, sind jedoch auch noch andere Systeme denkbar.

Da die RFID-Technologie eine Ortung bis hin zu wenigen Millimetern Genauigkeit erlaubt und überall einsetzbar ist, bietet diese eine sehr gute Möglichkeit sich dort zurechtzufinden, wo keine Satelliten Signale verfügbar sind. Das könnte in Gebäuden oder auch in Metropolen sein, wo Hochhäuser den Satelliten Kontakt behindern.

Bei einem möglichen Szenario sind im Gebäude an Schlüsselpositionen RFID-Tags platziert. Kommt ein Lesegerät in die Nähe des Tags, erkennt es ihn und weiß, über den Kontakt zu einer Datenbank, wo im Gebäude man sich gerade befindet. Mit dieser Information lässt sich nun bestimmen, welche Route zum Zielort führt.

Solche Systeme könnten sich eventuell für die Unterstützung von älteren oder ortsfremden Menschen als nützlich erweisen. So könnten sie zum Beispiel einem blinden Menschen akustisch mitteilen wo er sich gerade befindet und welche Räume gerade in der Nähe sind. Einem Hörgeschädigtem könnte andererseits ein Lageplan dargelegt werden, welcher seine aktuelle Position im Gebäude wiedergibt. Würde solch ein System weiter vernetzt werden, kann es auch sinnvoll eingesetzt werden um im Notfall den schnellsten oder sichersten Ausweg zu berechnen und anzuzeigen. Es könnte auch festgestellt werden, ob alle das Gebäude verlassen haben, oder wo Rettungskräfte am ehesten gebraucht werden. Der IWalker ist solch ein System[iwalker].

Ein weiteres Einsatzgebiet wäre, die Navigationsgrundlage für Roboter, in der Logistik. Durch eine Lagerhalle die mit RFID-Tags ausgestattet ist, hätten Roboterflotten die Möglichkeit Routen und Positionen zu koordinieren und so die Verteilung und Lagerung von Waren zu übernehmen.

1.2 Aufbau der Arbeit

Ziel der Arbeit, ist es einen autonomen mobilen Roboter, durch ein Raster an RFID-Tags navigieren zu lassen. Dabei werden zunächst in Kapitel 2 und 3 die zugrunde liegenden Technologien und Verfahren vorgestellt.

Danach wird in Kapitel 4 die Hardware vorgestellt, die für die Durchführung dieses Projektes zur Verfügung steht. Das Folgende Kapitel 5 beschreibt die eingesetzten Software-Technologien und Standards.

Das Versuchszenario welches für das Projekt entworfen wurde und die Konstruktion des Roboters, wird in Kapitel 6 vorgestellt und beschrieben.

In Kapitel 7 wird eine Software-Architektur entwickelt, die in der Lage ist die in Kapitel 6 geforderten Leistungen zu erbringen.

Die Umsetzung der Software-Architektur wird in Kapitel 8 untersucht. Zum Schluss, findet sich im letzten Kapitel 9 noch ein Fazit und ein Vergleich mit ähnlichen Projekten und Technologien.

2 **RFID**

RFID steht für "Radio Frequency Identification"[rfid], was so viel bedeutet, wie die Identifizierung mittels Funkwellen. Die Technologie besteht hauptsächlich aus 2 Komponenten. Einem Lesegerät, dem *Reader* oder *Leser* und einem Auszeichnungs-Transponder, welcher allgemein hin als Tag bezeichnet wird.

Reader existieren in verschiedensten Größen und Ausführungen, sie unterscheiden sich maßgeblich in der Reichweite, in welcher sie die Signale bzw Antworten der Tags aufnehmen können. So können sie eine Reichweite von ein paar hundert Meter bis hin zu wenigen Millimeter aufweisen.

Die *Tags* oder auch *Transponder* genannt existieren in allen möglichen Formen, im Grunde bestehen sie aus einem kleinen Mikrochip und einer Antenne. Der Chip enthält in der Regel eine kleine Datenfolge zur eindeutigen Identifizierung, aber in teureren Ausführungen auch Möglichkeiten für eine Authentifizierung mittels Passwort, oder die Fähigkeit Daten aufzunehmen und zu speichern. Darüber hinaus existieren auch aktive Transponder welche zum Mikrochip und der Antenne noch eine, wie eine Batterie, besitzen. Ein Beispiel für ein Tag und einen Reader sind in Abbildung 1 zu sehen.

2.1 **Funktionsweise**

Der Leser erzeugt ein elektromagnetisches Hochfrequenzfeld, dieses Feld dient zur Datenübertragung und versorgt gleichzeitig die Tags mittels Induktion mit Energie. Die gewundene Antenne nimmt dabei die Induktionsenergie auf und versorgt den angebundenen Mikrochip. Dieser verändert je nach Programmierung und Ausstattung das Empfangene Feld und erzeugt somit eine Antwort die das Lesegerät empfängt. Das Lesegerät erkennt also, wie das ausgesandte Feld verändert oder geschwächt wurde, und kann diese Änderungen interpretieren. Dieses Verfahren macht ein RFID-System zu einer berührungslosen Technologie. Je nach Anwendungszweck und Budget ist die Ausstattung und Anwendung frei skalier- und anpassbar.

2.2 **Anwendungsbereiche**

Meist wird die RFID Technologie im Logistikbereich eingesetzt. Sie löst zunehmend den alten Barcode ab, dort ist sie besonders geeignet weil es eine berührungslose Technologie ist. Außerdem sind die Tags mittlerweile für wenige Cent pro Stück erhältlich. Die Ware muss nur



Abbildung 1:

Links: Ein im handel erhältliches Lesegerät.

Rechts: Ein RFID-Tag, außen die gewundene Antenne,
in der Mitte ein kleiner Chip

noch in die Nähe eines Readers oder an diesem vorbei gebracht werden um im Logistik-System registriert zu werden.

Beispiele die dabei auch schon öfter in der Öffentlichkeit diskutiert werden, sind unter anderem der Intelligente Einkaufswagen, welcher immer weiß welche Artikel gerade in ihm sind, wie teuer der gesamte Wageninhalt ist und zum bezahlen braucht man sich nur noch an eine entsprechende Station stellen und der Wageninhalt wird automatisch gebucht. Im Kaufhaus oder Supermarkt selbst, könnte man auf einfache Weise Verfallsdaten kontrollieren und eine aufwändige Inventur überflüssig machen. Beim Thema Logistik, ist es wichtig zu wissen welcher Artikel sich wo befindet und in welche Richtung er sich gerade bewegt. Auch solche Informationen lassen sich mit entsprechender Ausstattung ermitteln. Eine weitere größere Anwendungssparte sind elektrische Schlosser, mit Lesegeräten welche sich mit einem Transponder entsperren lassen.

In den folgenden Abschnitten dieser Arbeit geht es um Navigation bzw Wegfindung mittels RFID. Das ist ein relativ neues Gebiet wo diese Technologie Verwendung findet bzw finden kann. Bei aktuellen Ausstellungen findet man Objekte wie den intelligenten Teppich, mit einem integrierten Raster aus RFID-Tags. Es existieren außerdem diverse Forschungsarbeiten zu dem Thema, für die Navigation und Unterstützung im Gebäude, wo RFID zum Einsatz kommt, da dort GPS-Satelliten Signale nicht benutzbar oder zu ungenau sind.

3 Navigation

Der Begriff Navigation, beschreibt ein Verfahren, bei dem es darum geht von einem Punkt A aus, sicher zu Punkt B zu gelangen. Dazu gehören zunächst die Positionsbestimmung. Dabei bestimmt man an welcher Stelle in einem gegebenen Koordinaten System man sich gerade befindet. Ist diese Position ermittelt, geht es darum eine mögliche Route zum Zielort oder Bestimmungsort zu ermitteln. Der letzte Schritt befasst sich dann mit dem reisen oder steuern und vor allem dem einhalten der Route.

3.1 Formen der Navigation

Geschichtlich gesehen hat die Navigation meist bei der Schifffahrt bzw. der Nautik eine Rolle gespielt. Aus der geschichtlichen Entwicklung heraus, sind einige sehr unterschiedliche Formen der Navigation entstanden. Eine der ältesten ist das Navigieren anhand von Landmarken, bzw. markanten Landschaftsmerkmalen, die es erlauben, einem seine Position bestimmen zu lassen. Diese Form heißt terrestrische Navigation. Bezogen auf das Projekt könnte man so die umfangreichere Markierung der Tags, oder einer Linie welcher der Roboter folgen soll als Landmarke bezeichnen.

Eine Navigationsform, welche im Rahmen dieser Arbeit an Bedeutung gewinnen wird ist die Koppelnavigation. Dabei geht es darum anhand von Kurs, Geschwindigkeit und Zeit zu schätzen welche Strecke zurück gelegt wurde und wo man sich befindet. Dabei ist natürlich nur ein mehr oder weniger genaues raten möglich, deswegen wird eine so ermittelte Position auch als *gegisster* bzw *vermuteter Ort* oder *Koppelort* bezeichnet. Dieser Ort wird durch fortlaufende Messungen ständig bestimmt.

Weitere Formen sind unter anderem, die Positionsbestimmung anhand der Gestirne, mittels Satelliten oder per Funkwellen.

4 Verwendete Hardware

4.1 Der Lego NXT Roboter

Das Lego NXT System ist der Nachfolger des seit 1998 erhältlichen Lego-RCX. Beide sind Teil des Lego Mindstorms Education Konzeptes "LME"[lme]. Somit steht eine voll funktionsfähige und frei Programmierbare Plattform zur Verfügung, um das Verhalten von Robotern zu simulieren und Programme für diese zu schreiben und zu testen.

Das NXT-System ist flexibel ausrüstbar und besteht zumindest aus dem NXT-Stein (siehe Abbildung 2) selbst und einer beliebigen Kombination aus drei Motoren und vier Sensoren. Alle Komponenten, werden auf die Lego typische Weise zusammengeführt. Ein fertiger Roboter, besteht dann idealerweise aus einem Lego-Fahrgestell und einer Aufhängung aus Lego für die Sensoren. Bei den Sensoren ist man auch nicht auf die im Handel erhältlichen beschränkt, so ist es möglich mittels des I2C Busses auch sämtliche anderen Sensoren zu benutzen die sich mit diesem Standard vertragen[i2c].

Zum LME-Konzept gehört außerdem eine bereits im NXT integrierte grafische Programmiersprache NXT-G, bietet darüber hinaus aber auch die Möglichkeit, andere Sprachen wie C++ oder Java zu benutzen. Informationen zu der Java Implementierung sind in Kapitel 5 zu finden.

Aufgrund der flexiblen Hardware Konfigurationsmöglichkeiten und der freien Wahl der Programmiersprache, bietet der NXT eine ausgezeichnete Grundlage für dieses Ausarbeitung.

Die folgende Auflistung zeigt eine Darstellung der verwendeten Sensoren und die technischen Spezifikationen des NXT[lins]



Abbildung 2: NXT-Stein ohne Zubehör.

NXT-Stein

- 32-bit ARM7 Mikroprozessor
- 256kbyte FLASH- und 64kbyte RAM-Speicher
- Unbegrenzte Anzahl an Programmspeicherplätzen
- Bluetooth Technologie ermöglicht die kabellose Kommunikation zwischen NXT Stein, Computern, PDAs und Mobiltelefonen
- USB 2.0 Anschluss
- 4 Eingänge für Sensoren
- 3 Ausgänge für Motoren und Lämpchen
- Programmierbares, graphisches Display (60 x 100 Pixel)
- Lautsprecher
- Design und Handhabung ist auf LEGO TECHNIC Konstruktionsystem abgestimmt

HiTechnic Kompasssensor

Abbildung 3: HiTechnic Kompasssensor, montiert

Beschreibung aus dem Legoportal: "Mit dem HiTechnic Kompasssensor können LEGO MINDSTORMS NXT Roboter präzise navigiert werden. Der digitale Kompass misst das Erdmagnetfeld, um eine exakte Navigation durch das NXT-Roboter Projekt zu gewährleisten.

Im Lese-Modus kann der Sensor die aktuelle Position des Roboters ermitteln. Im Kalibrier-Modus kann der Sensor angepasst werden, um zum Beispiel extern verursachte Schwankungen des Magnetfelds, wie sie z.B. in der Nähe von Batterien oder Elektromotoren auftreten, auszugleichen. Der Sensor hat eine Auflösung von 1° und gibt einen Wert zwischen 0 und 359 aus. Der Wert wird 100mal pro Sekunde aktualisiert."(HiTechnic Sensoren(Kompasssensor): [lins])

Der NXT Lichtsensor



Abbildung 4: NXT Lichtsensor, nicht montiert

Beschreibung aus dem Legoportal: "Der Neue NXT Lichtsensor ist eine Weiterentwicklung des Lichtsensors des klassischen LEGO MINDSTORMS für Schulen.

Der Sensor ist deutlich empfindlicher und erlaubt präziserer Lichtmessungen auf einer Skala zwischen 0 (kein Licht) und 100 (sehr hell). Es ist ebenfalls möglich das Infrarotlicht unterhalb dem Sensor auszuschalten, um nur das Umgebungslicht zu messen." [lins]

RFID-Reader

Der RFID-Sensor(rechts) von Codetex, erlaubt es die 5 Byte große Identifikationsnummer aus dem Tag aus zu lesen. Er arbeitet in einem Frequenzbereich von 125kHz und liest Transponder vom Typ EM 4102. Er hat eine Reichweite von 30mm und verfügt, neben einem Energiesparmodus, auch über verschiedene Lesemodi.[coda]



Abbildung 5: Codetex RFID-Sensor, montiert

Kommunikation

Um mit anderen Systemen wie Computern oder anderen NXT Blöcken zu Kommunizieren, besitzt der NXT einen USB Anschluss und einen Bluetooth Adapter. Mittels des USB-Adapters lässt sich eine Datenverbindung zu einem Computer aufbauen, mit welcher sich Daten austauschen lassen und Programme auf den Block gespielt werden können. Da diese Verbindung allerdings kabelgebunden ist, bietet die Bluetooth Variante mehr Möglichkeiten. Bluetooth ermöglicht eine drahtlose Verbindung über mehrere Meter. Mehr zum Thema Bluetooth stehe in Abschnitt 4.3

4.2 RFID-Tags

Für die Durchführung der Versuche, stehen 20 RFID-Tags in Form von Schlüsselanhängern(siehe Abbildung 6) zur Verfügung. Auf jedem Tag ist eine eindeutige Nummernfolge vom Typ "long" gespeichert, welche mit dem RFID-Leser ausgewertet werden kann. Ein Schreiben oder Ändern von Werten auf den Tags ist mit der derzeitigen Ausstattung nicht möglich.



Abbildung 6: RFID-Tag in Form eines Schlüsselanhängers(4cm x 3cm).

Links: Rückseite mit logischer Referenznummer

Rechts: Weiße Vorderseite

4.3 Bluetooth

Bluetooth ist ein Standard, der eine drahtlose Funkverbindung beschreibt. Dabei wird ein Punkt zu Punkt Netzwerk eingerichtet, welches zum Senden und Empfangen von Daten beider maßen in der Lage ist. Je nach Bluetooth Version lassen sich unterschiedliche Reichweiten und Datenraten erzielen.

Ein kurzer Überblick über die derzeitigen 3 Klassen[Tabelle 1], zeigt das die Reichweiten und Datenraten sehr weit auseinander liegen.

Klasse	Max. Leistung	Max. Leistung	Reichweite im Freien
Klasse 1	100 mW	20 dBm	ca. 100 m
Klasse 2	2,5 mW	4 dBm	ca. 10 m
Klasse 3	1 mW	0 dBm	ca. 1 m

Tabelle 1: Überblick Bluetooth Standard[bt]

Der NXT-Stein verfügt laut Spezifikationen über einen Bluetooth Chip, der den Bluetooth 2.0 + EDR¹ Standard erfüllt und hat somit eine Reichweite von 10 Meter und Verfügt über eine Datenrate von 2,1 MBit/s.

¹EDR (Enhanced Data Rate), Technologie zur Erhöhung der Datenrate durch Optimierung des Bluetooth Standards

5 Verwendete Software-Technologien

Wie bereits in Kapitel 4 erwähnt besteht die Möglichkeit, Programme auch in einer anderen Sprache, als der vorgegebenen NXT-G Sprache zu schreiben. Da die Lego Firmware von "Lego" als Open Source öffentlich gelegt wurde, existieren eigentlich für alle gängigen Programmiersprachen, wie C/C++ oder auch Java, Implementierungen die relativ einfach zu installieren sind. Dabei wird die vorhandene Firmware überschrieben und durch eine neue ersetzt. Durch eine bereitgestellte API, können dann Programme geschrieben werden die auf dem NXT hoch geladen, gestartet und umgesetzt werden.

5.1 Java Implementierung: LeJos API

Für dieses Projekt, wurde zunächst die vorhandene Firmware durch eine neue ersetzt. Dabei wurde auf die Versionen: 0.85 der LeJos[lejos] Implementierung zurückgegriffen. Diese Firmware ist eine Java Implementierung für den NXT. Sie erlaubt es Java Programme die mit der TinyVM[tmv] gelinkt wurden, auf den NXT zu laden und diese auszuführen.

Die TinyVM stellt eine minimale Java VM² bereit. In ihr sind weniger Implementierungen enthalten, so fehlen die meisten *Collection Implementierungen* und *Util Klassen*. Sie bietet aber eine volle Threadfähige Umgebung und genau auf den Roboter zugeschnittene Funktionalitäten.

Mit LeJos ist man in der Lage, moderne objektorientierte Software für den NXT zu schreiben. Für alle im Handel erhältlichen Sensoren sind bereits Klassen vorhanden. Es existieren außerdem so genannte *Pilot-Klassen*, welche die Ansteuerung der Motoren erleichtern und ein Paket welches einem die Nutzung von Bluetooth und USB-Verbindungen ermöglicht.

Entwickelt werden kann in der gewohnten Entwicklungsumgebung.

5.2 Java API: NxtJLib und NxtJLibA

Zusätzlich zu der LeJos Firmware, wurde noch eine zusätzliche Bibliothek eingebunden. Bei der NxtJLibA[aplu] Bibliothek, handelt es sich um Klassen die eine erweiterte Nutzung der Java Möglichkeiten erlauben. So überschreibt diese Bibliothek teilweise Implementierungen

²Die Java VM ist eine sogenannte Virtual Machine, und ermöglicht das Platform-unabhängige ausführen von Java Code.

von LeJos und ermöglicht so, für die meisten Sensoren, die Benutzung des *Java Event Handling Systems*.

Eine weitere von aplu.ch angebotene Bibliothek(NxtJLib) bietet noch die Möglichkeit den NXT nicht autonom laufen zu lassen, sondern direkt über die Bluetooth Verbindung zu steuern. Bei dieser Variante übernimmt der PC oder gar das Handy alle Berechnungen und übermittelt lediglich Steuerbefehle. Aus Kompatibilitätsgründen mit dem Bluetooth 64bit Treiber konnte dieses Verfahren nicht getestet werden.

6 Beschreibung des Szenarios

Das Versuchsszenario sieht vor, dass ein Roboter, welcher mit einem RFID-Reader ausgestattet ist, sich durch ein vorgegebenes Muster an RFID-Tags, eine Route zu seinem Ziel sucht. Dabei sollte der Roboter sich möglichst autonom verhalten und somit auf möglichst wenig Resourcen zurückgreifen. Geplant ist ein Aufbau bei dem sich der Roboter durch ein Gitter-Raster bewegen soll. Bei diesem Raster gilt es den Kanten zu folgen, die Knoten stellen Kreuzungen dar welche durch die RFID-Tags identifiziert werden können. Durch die Identifikation lässt sich dann die Position im Raster bestimmen und eine Route berechnen.

Damit lässt sich ein ähnliches Konzept abbilden wie es mittlerweile im *Intelligenten Fußboden*³ angewandt wird.

Mit solch einer Infrastruktur, ist es möglich, dass einzelne Roboter oder auch ganze Roboterflotten bestimmte Routen abfahren und gezielt Aufgaben erledigen können.

Diese können vom Transport von Waren bis hin zur Überwachung der Umgebung reichen. Für dieses Projekt wird keine konkrete Aktion ausgeführt werden. Sie wird lediglich durch Nutzung des Lautsprechers angedeutet werden. Da aber je nach Ausstattung so ziemlich jede Aufgabe für einen Roboter Möglich ist, sind an dieser Stelle der Fantasie keine Grenzen gesetzt.

In der einfachsten Ausführung, besteht das beschriebene Projekt aus folgenden Teilkomponenten.

- Einem in den Boden integrierten Raster, das den Arbeitsbereich des Roboters darstellt.
- Dem Roboter selbst, welcher sich auf dem Boden bewegt und "Aufgaben" ausführt
- Einem Server oder einer Gegenstelle, welche den Roboter mit der Karte des Rasters initialisiert

³ Bei dem Intelligenten Fußboden oder auch *smart floor*, handelt es sich um einen speziellen Teppich Belag, bei dem ein Raster aus RFID Chips integriert ist. Dies soll es ermöglichen autonome Reinigungs-Roboter zu betreiben.[sfloor]

Weiter besteht die Aussicht, das Projekt um die folgenden Punkte zu erweitern, was allerdings aus zeitlichen Gründen leider nicht möglich war.

- Dynamische Verwaltung und Steuerung, des oder der Roboter mittels Bluetooth über einen Server
- Das Raster in verschiedene Teilbereiche erweitern, z.B. gesonderte Ladestationen zum Akku aufladen oder Einbahnstraßen.
- Die Form des Rasters ändern und so einen von 90° zu 45° Abzweigungen.
- Statt eines einzelnen Roboters, mehrere auf dem gleichen Raster einsetzen.

6.1 Das Raster

Der Arbeitsbereich des Roboters ist durch ein Raster aus RFID-Tags definiert. Die Tags sind dabei in regelmäßigen Abständen auf dem Boden platziert(siehe Abbildung 7). Jeder RFID-Tag, stellt einen Kreuzungspunkt oder auch Knoten dar. Auf diese Weise ergeben, sich für jeden inneren Knoten vier Wege oder auch Kanten die zu dem jeweils benachbarten Knoten führen.

Durch das Raster, kann die Position eines Knoten jetzt logisch durch ein kartesisches Koordinatensystem der Form (x,y) dargestellt werden. Die Position(0,0) entspricht dabei dem Knoten in der untersten linken Ecke.

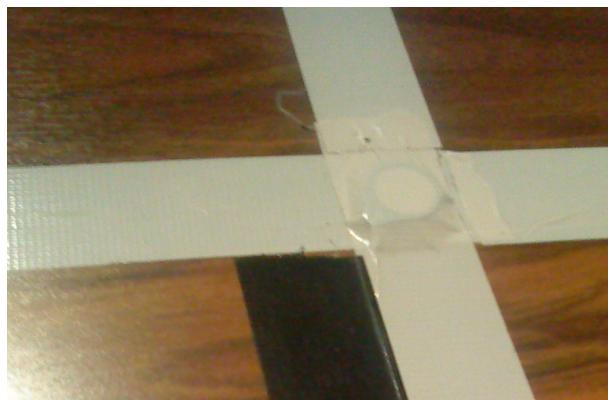


Abbildung 7: Ein Kreuzungspunkt, das Klebeband bildet die Kanten des Rasters

Der Kreuzungspunkt enthält unter dem Klebeband einen RFID-Tag

Die Größe des Rasters, ist durch die vorhanden 20 Tags auf 5 x 4 beschränkt. Legt mal die Tags in einem Abstand von jeweils 30cm aus, so ergibt sich ein Raster der Größe 1,5m x 1,2m in dem der Roboter Navigieren kann.

Durch die hier beschriebene Anordnung des Rasters und die Tatsache, dass jeder Knoten durch RFID eindeutig zu identifizieren ist, ergibt sich eine fest vorgegebenen Umgebung. Diese ist dem Roboter bekannt, was die Navigation und Routenplanung wesentlich erleichtert, da er die Umgebung nicht erst kennen lernen und analysieren muss.

6.2 Ablauf und Aufgabe

Der Roboter wird zu Beginn mit dem Koordinaten-Muster des Rasters initialisiert. Dies kann per Bluetooth passieren oder fest in den Code integriert sein, da es sich um eine statische Umgebung handelt die sich in der Regel nicht verändert.

Ist die Initialisierung abgeschlossen wird der Roboter auf einem Knoten oder einer Kante des Rasters platziert und das Programm durch drücken einer Taste gestartet. Die Blickrichtung beim Start, wird von dem Roboter jetzt als Norden erfasst. Sofern er sich jetzt noch nicht auf einem Knoten befindet, folgt er nun der Kante bis er einen Knoten erfasst hat. Anhand der erfassten Daten kann er seine genaue Position im Raster ermitteln.

Wenn die Position ermittelt ist, wird der Roboter auf Aufträge warten. Trifft ein solcher ein, wird er aktiv und berechnet eine Route um den Auftrag zu erfüllen.

6.3 Der Roboter

Der Roboter der zum Einsatz kommt ist ein Lego-NXT, der aus den in Kapitel 4 beschriebenen Komponenten besteht. Motoren und NXT-Stein wurden nach einer Anleitung aus dem Internet zu dem so genannten "Castor Wagen" [castor] zusammengebaut. Hier rechts zu sehen, auf Abb. 8.



Abbildung 8: Grundaufbau des Roboters

Diese Konstruktion bietet zwei unabhängig voneinander steuerbare

Räder, was es dem Roboter erlaubt auf der Stelle zu wenden. Die Montage der Sensoren ist bei diesem Design auch einfach und stabil. An diesem Grundaufbau sind dann, die in Kapitel 4 beschriebenen Sensoren befestigt. Zu sehen auf Abbildung 9.

Der Kompass ist etwas erhöht angebracht und befindet sich genau über dem Mittelpunkt des NXT. Er weist nach vorne und gibt somit immer die Fahrtrichtung in Grad an.

Die beiden anderen Sensoren sind an einer Aufhängung vor dem Roboter befestigt und weisen beide nach unten. Der vorderste Sensor ist der Lichtsensor. Auf diese Weise ist es möglich eine Bodenmarkierung, wie z.B. verschieden farbiges Klebeband, zu erkennen und ihr zu folgen. So kann auch erkannt werden, dass der Roboter die vorgegebene Strecke verlassen hat. Aufgrund von Ungenauigkeiten in der Motoransteuerung, ist dies relativ häufig der Fall.

Der RFID-Sensor befindet sich in einer Linie mit dem Lichtsensor und ermöglicht so das erkennen von RFID-Tags, zu einem Zeitpunkt an dem sich der Mittelpunkt des Roboters fast über dem Tag befindet.

Mit der beschriebenen Ausrüstung ist es dem Roboter also möglich:

- Linien bzw. Markierungen zu folgen,
- die Blick-Richtung zu bestimmen,
- die Knoten Id's aus zu werten,
- mit einem Server zu kommunizieren.

Mit diesen Fähigkeiten, ist der Roboter technisch in der Lage alle in Kapitel 6 beschriebenen Tätigkeiten und Aufgaben ausführen zu können.

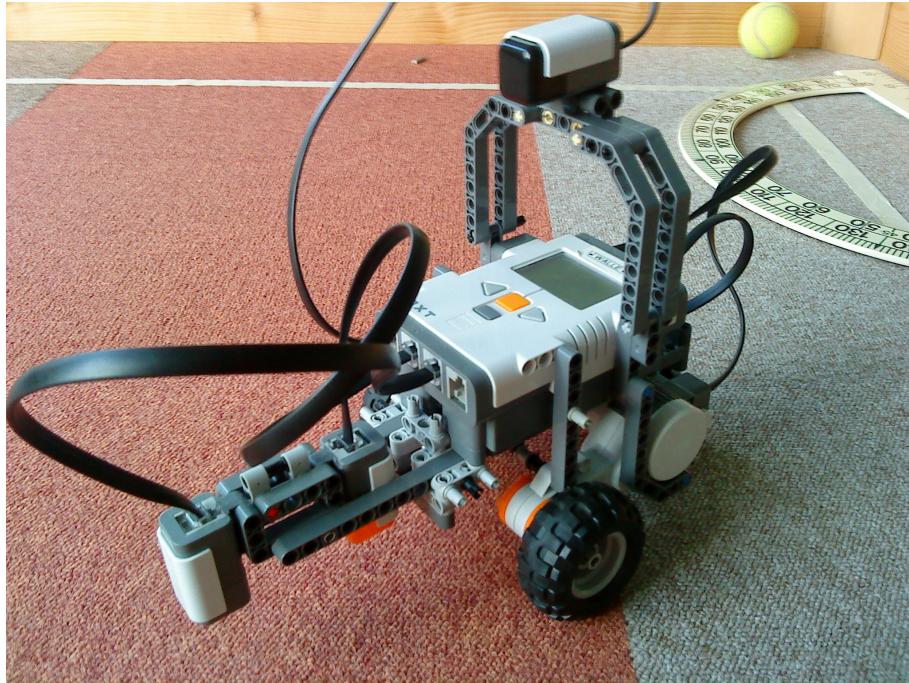


Abbildung 9: Fertige Konstruktion des Roboters

7 Software

7.1 Anforderungen

Da der Roboter in einer hauptsächlich statischen Umgebung agiert, muss dieser kaum aufwändige Berechnungen anstellen. Die Hauptaufgabe besteht darin, den Markierungslien zu folgen und an den Knoten über die Richtung zu entscheiden. Parallel dazu sollte nebenbei noch ständig die eingehenden Kommunikationsdaten verarbeitet werden. Dazu müssen während der Bewegung ständig die Sensordaten überwacht werden. So kann dann unverzüglich auf unerwünschte Änderungen der Umgebung reagiert werden. Deswegen wurde eine reaktive Steuerungsarchitektur⁴ gewählt.

Eine reaktive Architektur zeichnet sich dadurch aus, dass ständig aktuelle Sensordaten ausgewertet und verarbeitet werden. So kann sofort auf Ereignisse reagiert werden.

Dies wird realisiert, indem die Sensordaten in einer zentralen Logikeinheit ständig gegen ein Satz an bedingten Verhaltensregeln geprüft werden. Durch die hohe Frequenz dieser Überprüfung, wird sicherge-

⁴(Dazu Kapitel 4: [rosteu]): "Unter einer Steuerungsarchitektur versteht man die Struktur der Steuerung eines Systems. [...] Hierzu gehören die Anforderungen, die durch ungenaue Sensorik und Aktorik entstehen, sowie Zeitanforderungen durch die Interaktion mit einer realen Umwelt."

stellt das der Roboter keine entscheidenden Ereignisse verpasst. Solch ein Ereignis könnte eintreten, wenn der Roboter zum Beispiel gerade einen Knoten passiert und der RFID-Sensor in diesem Moment gerade nicht empfangsbereit ist.(vgl. Seite 68 vorletzter Absatz[rosteu]) Um Routen berechnen zu können oder anderen Aufgaben vorrang zu geben, benötigt der Roboter noch einen gewissen Grad an Intelligenz und Planungsfähigkeit. Um dies mit dem reaktiven Konzept zu vereinbaren, wird dazu in den Regeln noch eine Liste von Aktionen eingeführt. Zum Beispiel, muss der Roboter, während er gerade eine Route berechnet, nicht ständig den RFID-Tag interpretieren.

7.2 Architektur

In Abbildung 10 wird das Projekt in die zwei Systeme Server und Roboter unterteilt. Das Schichtenmodell sieht vor, dass nur Schichten miteinander kommunizieren können welche übereinander liegen. Eine Kommunikation zwischen Komponentenschicht und Kommunikationsschicht ist also nicht möglich, genauso wenig wie eine Kommunikation zwischen Datenerfassung und Aktionsschicht. Durch diese Einschränkung wird die Wartbarkeit bzw. Erweiterbarkeit der Software erhöht, da die Kommunikation innerhalb der Software überschaubar bleibt.(vgl. [sarchl])

7.3 Roboter-Schichten im Detail

Kommunikationsschicht

Die Kommunikationsschicht kümmert sich um den Erhalt von Aufgabenpaketen. Dabei übermittelt der Server eine Reihe von Werten an den Roboter, die von der Kommunikationsschicht entgegengenommen werden. Die Daten werden dann an die Logik zur Verarbeitung weiter vermittelt. Die Kommunikationsschicht übernimmt auch das senden von Mitteilungen an den Server. Dabei kann es sich um Info- oder Debugmeldungen handeln.

Logikschicht

In der Logikschicht sind alle Klassen angesiedelt die Entscheidungen zu treffen haben. Eingehende Aufgabenpakete müssen verarbeitet werden. Die Routenberechnung muss durchgeführt werden. Auch alle Sensordaten werden hier logisch interpretiert und entsprechende Aktionen veranlasst.

Komponentenschicht

Die Komponentenschicht ist noch einmal in die Teilgebiete für "Datenerfassung" und Aktionsschicht unterteilt.

Die Datenerfassung kümmert sich um stetige Bereitstellung von Sensordaten. Gewisse Daten werden hier auch statistisch verarbeitet, um Messfehlern vorzubeugen.

Die Aktionsschicht kümmert sich um die Ausführung von kleinen Teilaufgaben. Diese können parallel ablaufen und die Ausführung wird jeweils durch die Logikschicht veranlasst oder gestoppt. Hier erfolgt

auch der einzige Zugriff auf die Aktoren. Die Logikschicht trägt allerdings die Verantwortung, dass Teilaufgaben nicht gleichzeitig auf geteilte Ressourcen wie Motoren zugreifen.

7.4 Server-Schichten im Detail

Der Server besteht hauptsächlich aus der Applikation selbst die im Grunde frei skalierbar ist. In der Grundausrüstung sollte die Applikation in der Lage sein, sich mittels der Kommunikationsschicht mit einem oder mehreren Robotern zu verbinden. Den verbundenen Robotern können dann Aufgaben zugewiesen werden. Ob es sich dabei um eine Kommandozeilen Interpretation handelt oder eine hochwertige GUI Anwendung ist nicht näher definiert.

Wichtig ist lediglich die Kommunikationsschicht, welche das Interface für die Kommunikation implementieren muss. Es sorgt für eine ordnungsgemäße Kommunikation mit den verbunden Robotern. Da falsch übermittelte Daten das Programm des Roboters stören könnten.

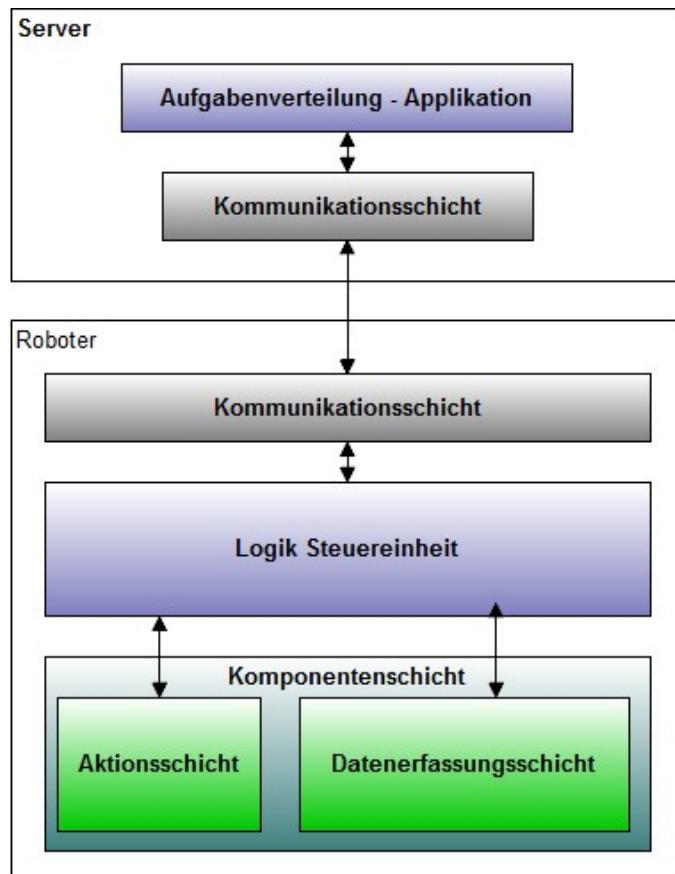


Abbildung 10: Darstellung der Schichtenarchitektur

7.5 Das Programm

Einen groben Ablaufplan was das Programm eigentlich macht, wird in Abbildung 11 dargestellt. Aus dem Ablaufplan kann man mehrere unterschiedliche Zustände heraus lesen. Nach dem Zustand der *Initialisierung*, geht er in eine *Wartende Phase* über wo er auf Aufträge wartet. Wird solch ein Auftrag erhalten, beginnt die Phase der *Lokalisierung*. Bei der *Lokalisierung* wird die aktuelle Position bestimmt. Ist das Ziel noch nicht erreicht folgt die *globale Navigationsberechnung*, in welcher die Route zum Ziel und die Richtung zum nächsten Knoten berechnet wird. Zum Schluss befindet sich der Roboter in dem Abschnitt, in dem er sich um die *lokale Navigation* kümmert. Dabei geht es darum der Bodenmarkierung zu folgen bis der angepeilte Knoten erreicht ist, von wo aus der Vorgang bei der *Lokalisierung* wieder von vorn beginnt.

Die Kommunikation und das lesen von Sensordaten, taucht in der Darstellung des Ablaufes nicht auf da es sich dabei um nebenläufige Prozesse handelt die nach der Initialisierung ständig im Hintergrund mit laufen.

Ein Endpunkt wird ebenfalls nicht dargestellt um die Abbildung 11 einfach zu halten. Es besteht allerdings jederzeit die Möglichkeit das Programm über einen Steuerbefehl oder die Escape Taste des NXT zu beenden. Wie die einzelnen Komponenten diesen Ablauf realisieren und zusammenarbeiten, wird in den nächsten Abschnitten beschrieben.

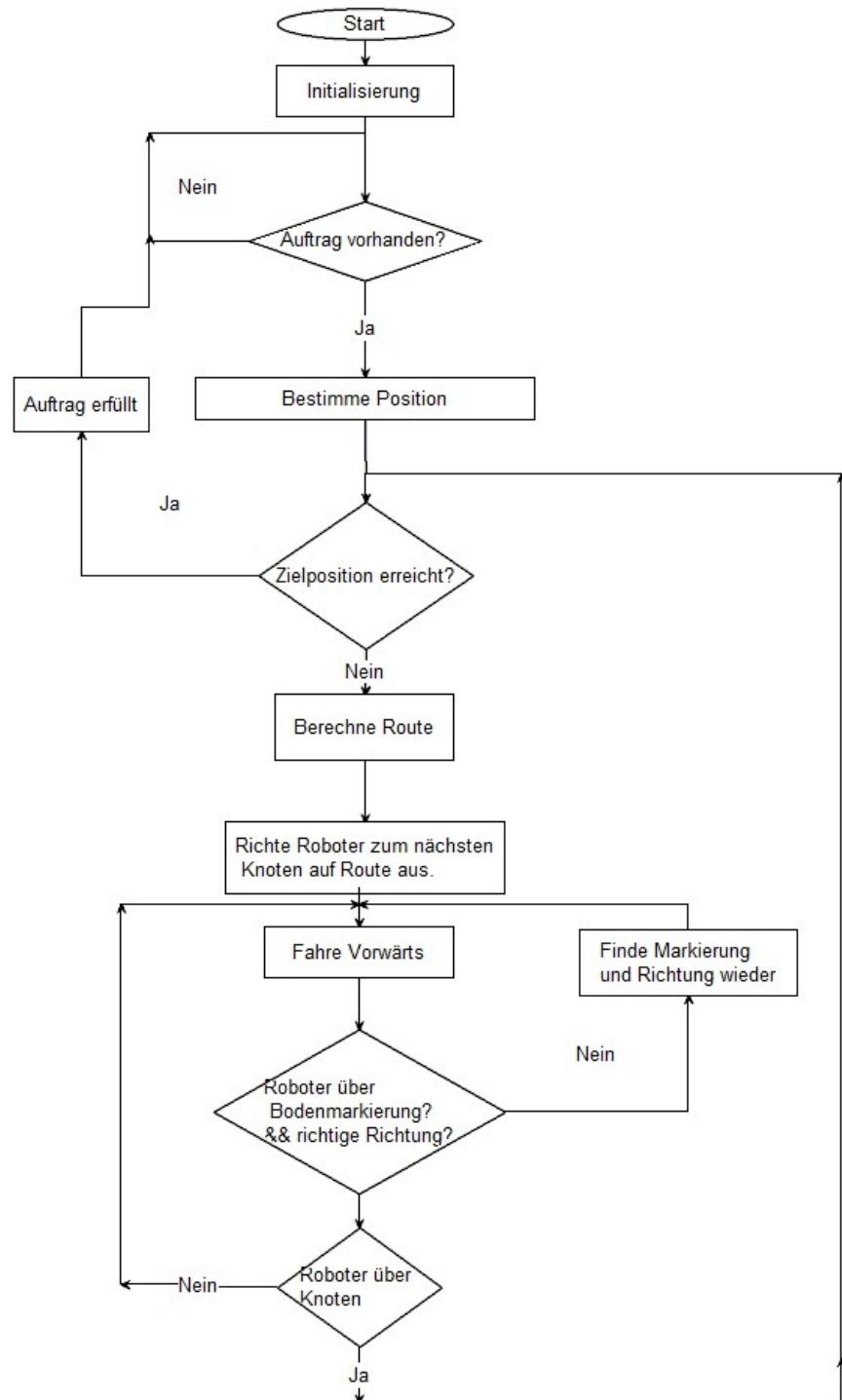


Abbildung 11: Grober Ablaufplan der Robotersoftware

7.6 Technische Konzeption

Die oben beschriebene Architektur wird mit Java Implementiert werden. Bei dem Entwurf findet sich hauptsächlich das im vorherigen Abschnitt beschriebene Schichtenmodell, in Form von Java Klassen, bzw. Paketen wieder. Dabei spielt die Logikschicht die wohl wichtigste Rolle. Da hier alle wichtigen Entscheidungen und Berechnungen ange stellt werden.

Die Java Pakete haben alle den gemeinsamen Präfix
de.fhannover.inform.lejos.

Klassen der Logikschicht sind im Paket *.application* zu finden. Im Pa ket *.comm* werden alle Klassen abgelegt die für die Kommunikation relevant sind. Die Beiden anderen Schichten befinden sich sinngemäß im *.action* und *.sensor* Paket.

Da nebenläufige Programmteile zum Einsatz kommen, müssen viele Klassen Thread-sicher implementiert⁵ werden.

Geschäftslogik

In dem Logikpaket wird es eine Klasse geben, welche sich um die Datenhaltung kümmert. In ihr existieren Datenstrukturen, die speichern welche Aufgaben zu erledigen sind und in welchem Zustand der Roboter sich gerade befindet. Außerdem sind hier die derzeitigen Sollwerte hinterlegt. Mögliche Sollwerte sind, die angepeilte Richtung und die Identifikationsnummer des gesuchten RFID-Tags. Im Klassendiagramm als *Controller* bezeichnet, siehe Abbildung 12.

Die eigentliche ausführende Klasse, ist ein Thread mit dem Namen *DecisionThread*(siehe Abbildung 12). Mit aktuellen Sensor- und Soll werten durchläuft er eine Endlosschleife, in der bedingte Abfragen die Verhaltensregeln darstellen. Auf diese weise lässt sich der momentane Zustand bestimmen. Je nach Sensordaten wird unterschiedlich reagiert. Im Klassendiagramm aus Abbildung. 12) sind die unterschiedlichen Zustände des Roboters durch die Methoden der Klasse *Decision Thread* angedeutet.

Die Enumerationen *Direction* und *Action* stellen in gewisser weise Hilfsklassen dar. *Direction* sorgt für eine einfache Umsetzung des Rich tungswinkels in eine abstrakte Richtungsangabe wie Nord oder Ost. Die verschiedenen Aktionen die in *Action* definiert sind ermöglichen neben den Zuständen eine genauere und logische Bestimmung der ak tuellen Lage. Stellt der Entscheidungsträger(*DecisionThread* fest das z.B. die Bodenmarkierung verschwunden ist, wird das Global in der

⁵Ressourcen auf die mehrere Threads gleichzeitig zugreifen, müssen synchronisiert werden.

Datenhaltungsklasse festgehalten mittels `actions.add(Action.OFFTRACK)`. So kann die Entscheidungsfähigkeit gesteigert werden, da das Weltbild⁶ des Roboters verbessert wird. Beide Klassen sind ebenfalls in Abbildung 12) zu sehen.

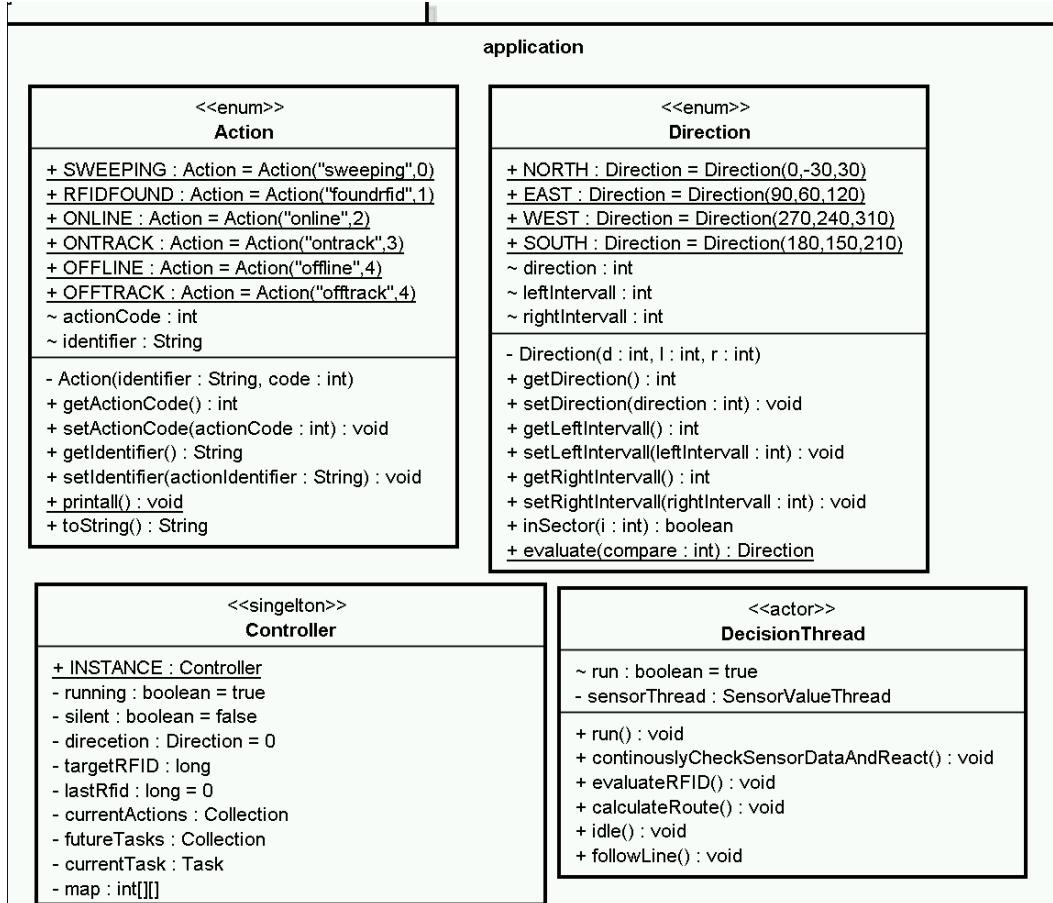


Abbildung 12: Klassendiagramm der Geschäftslogik

Komponenten

Zu den Komponenten gehören die Teilkäpfe `.action` und `.sensor`(siehe Abb. 13). Das `.action` Paket kümmert sich um die Ansteuerung der Akteuren. In diesem Fall existieren lediglich 2 Motoren. Allerdings müssen diese Koordiniert werden. Dazu dient die Klasse `Mover`. Sie bietet der Geschäftslogik Methoden für Suchmuster und Bewegungsabläufe an. Diese Methoden sind auf die entsprechenden Zustände und Aktionen zugeschnitten. Bei einer möglichen Erweiterung des Projektumfangs

⁶Weltbild, in diesem Fall, Informationen die sich aus der Interpretation verschiedener Sensordaten ergeben und länger als einen Entscheidungszyklus(Ein Schleifendurchlauf im Entscheidungsträger) erhalten bleiben.

würde hier auch die Bedienung von Werkzeugen oder einem Greifer hineinfallen.

Das Zweite Paket `.sensor`, kümmert sich um die Verwaltung der Sensoren. Diese haben bestimmte Grenzwerte, wie oft auf sie zugegriffen werden darf. Deswegen existiert hier ein nebenläufiger Thread (`SensorValueThread`) welcher in regelmäßigen Abstand die Werte ausliest und für die Geschäftslogik bereit hält. Zusätzlich besteht hier die Möglichkeit die Daten noch statistisch zu verarbeiten und auf zubereiten. Die Klasse `RFIDSensorListener` implementiert ein Interface aus der NxtJLibA Bibliothek welches das Java Event Handler Pattern auch für den NXT verfügbar macht. Die Implementierung ist auch mit einem Thread gelöst der dann die Hook-Methode `detected()`⁷ aufruft.

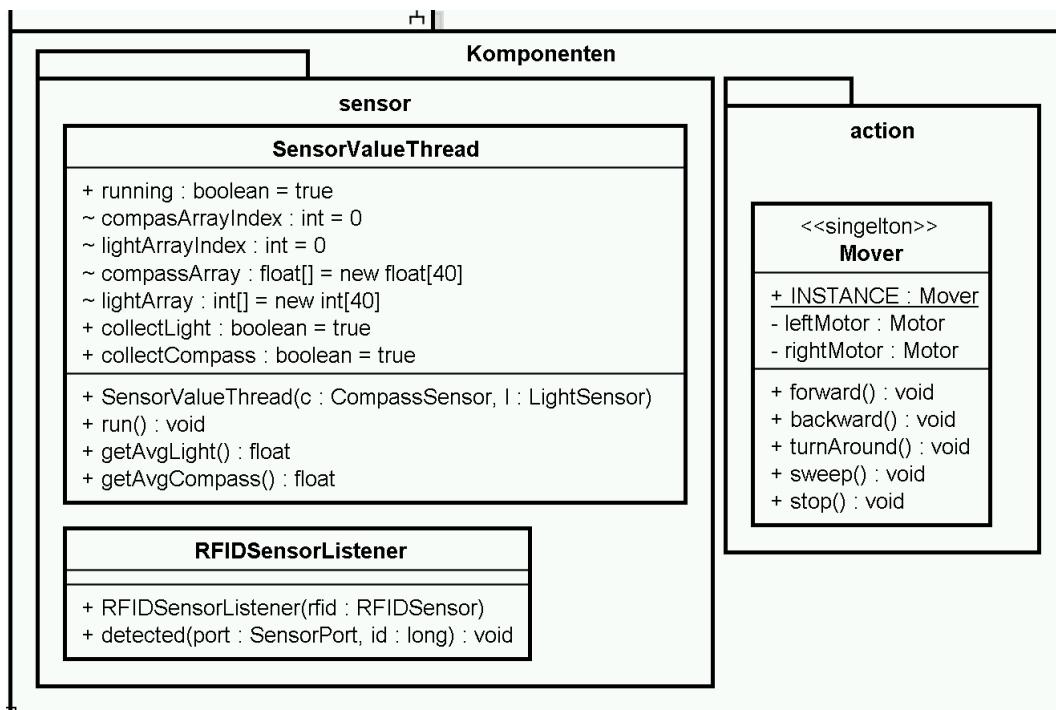


Abbildung 13: Klassendiagramm der Aktions- und Sensorenschicht

Kommunikation

Das Kommunikationspaket ist so geplant, dass ein Interface entworfen wird, das es einem erlaubt über eine Bluetooth Verbindung zu kommunizieren. Dabei wird ein fester Satz an Kommandocodes definiert. Werden Mitteilungen vom `.comm` Paket empfangen werden diese mit einem Thread entgegengenommen (siehe Abb. 14) und unverzüglich in

⁷Methode die über ein Interface oder abstrakte Klasse definiert wird und von außen aufgerufen wird.

einer Queue⁸ zwischengespeichert. Auf diese weise können alle übermittelten Daten, in der selben Reihenfolge später wieder verarbeitet werden. Durch die Zwischenspeicherung, wird der Puffer der Bluetooth Verbindung nicht unnötig belastet.

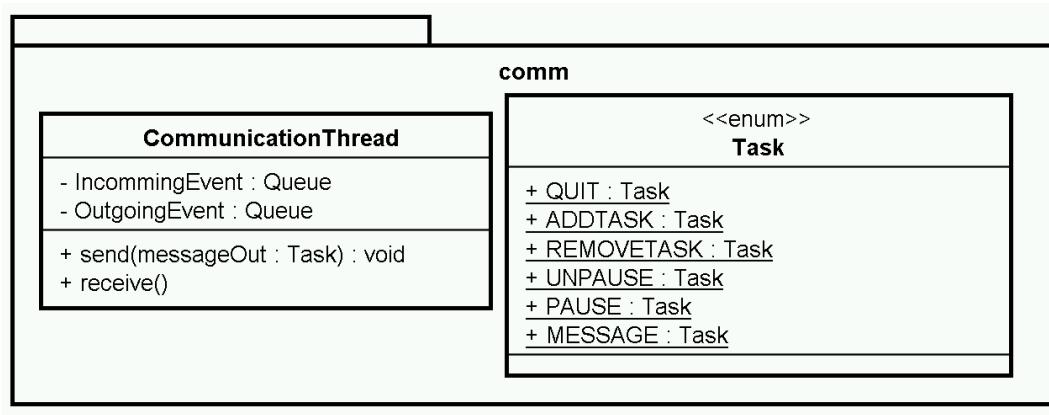


Abbildung 14: Klassendiagramm der Kommunikationsschicht

⁸Datenstruktur die nach dem FIFO(first in first out) Prinzip arbeitet.

8 Implementierung

Vorweg sei hier gesagt, dass es leider nicht möglich war alle Teile der Software komplett zu Implementieren. Jedoch konnten die einzelnen Komponenten davon soweit getestet werden, um sich ein Urteil zu erlauben und den Entwurf und die Architektur zu bewerten.

Zunächst wurde die Anwendungsschicht soweit implementiert, dass alle Sensoren und Aktoren initialisiert werden. Danach wurde ein leerer *DecisionThread* erstellt (siehe Listing: 1). Dieser konnte dann durch einfaches hinzufügen oder entfernen von Verhaltensregeln erweitert werden. Auf diese Weise ist es möglich einzelne Teilkomponenten einfach zu separieren und zu testen, was sehr hilfreich für das Debuggen ist. Auch können so einzelne Algorithmen schnell gegen andere ausgetauscht werden ohne das restliche Programm zu beeinflussen. So kann man schnell verschiedene Implementierungen gegeneinander testen.

Listing 1: Leerer DecisionThread

```

1 public class DecisionThread extends Thread {
2     // Durch drücken von Escape wird "run" auf "false" gesetzt
3     // und das Programm beendet.
4     boolean run = true;
5
6     public DecisionThread() {
7     }
8     public void run{
9         while(run){
10            // @todo Regeln überprüfen und Reaktionen veranlassen.
11
12            this.sleep(50);
13        }
14    }
15 }
```

An der Stelle des "Todo"-Kommentars in Zeile 10, von Listing 1, können jetzt bedingte Abfragen, bestimmte Verhaltensmuster auslösen. Ein einfaches Beispiel dafür sind in Listing 2 zu sehen. Dort sorgt die Abfrage in Zeile 3 von Listing 2, dafür das nach der verlorenen Bodenmarkierung gesucht wird. Ist die Bodenmarkierung bei einem späteren Schleifendurchlauf wieder vorhanden, wird der Suchvorgang gestoppt und andere Operationen bekommen wieder Vorrang. Auf die Darstellung der übergeordneten Status-Eigenschaften wurde hier verzichtet, diese stellen eine weitere Abstraktions-Ebene dar. Die in Listing 2 gezeigten Regeln, befinden sich dann in Blöcken und werden nur überprüft wenn der entsprechende Status vorliegt.

Listing 2: Regel definition in DecisionThread

```

1 [...]
2     while(run){
3         if(Controller.getActions().contains(Action.OFFTRACK)){
4             sweepForTrack();
5         }
6         if (light.getNormalizedLightValue() > avgLight -10) {
7             stopSweeping();
8             Controller.getActions().remove(Action.OFFTRACK);
9             Controller.getActions().add(Action.ONTRACK);
10        }
11        [...]
12        if(Controller.getActions().contains(Action.ONTRACK)){
13            Mover.forward();
14        }
15        this.sleep(50);
16    }
17 [...]

```

Den wichtigsten Punkt, der oben beschriebenen Implementierung, spielte die korrekte Zusammenarbeit der nebenläufigen Prozesse. Viele Zugriffe mussten synchronisiert werden damit keine Fehler auftreten. Jedoch konnte die Architektur erfolgreich getestet werden. So war es unter anderem möglich den Roboter mit einem ähnlichen Regelwerk eine Markierung folgen zu lassen und über einem RFID-Tag anhalten zu lassen und diesen auszuwerten.

Mit dieser Grundlage können einzelne Algorithmen entwickelt werden, die sich um die kleinen Teilaufgaben kümmern. Diese können dann in die oben beschriebenen Logik integriert werden.

8.1 Bestimmung der Richtung und Ausrichtung

Da in mehreren Versuchen der Kompass und die Motoren sich als zu ungenau erwiesen, kann dieser nicht als alleiniges Mittel zur Navigation eingesetzt werden. Er wird nun eingesetzt um eine grobe Himmelsrichtung zu Bestimmen. So lässt sich sagen in welche Richtung *Norden*, *Osten*, *Süden* oder *Westen* der Roboter gerade ausgerichtet ist. Bei dem Start des Systems, wird die *nördliche* Richtung des Rasters festgelegt. Die Enumeration *Direction* ermöglicht somit per Objektreferenz die Himmelsrichtung zu bestimmen. Somit lässt sich im Raster leicht bestimmen, ob die derzeitige Linie in die richtige Richtung weist.

Ausgerichtet wird der Roboter dann mit einer von Lejos bereitgestellten Methode, die den Roboter mit Hilfe des Kompass richtig ausrichtet.

Die Methode *rotateTo(degree)*, richtet den Roboter mithilfe des Kompass in die übergebene Richtung aus.

Liniенverfolgung

Während der Roboter von einem Knoten zum nächsten fährt, benutzt er Bodenmarkierungen in Form von Klebeband als Navigationshilfe. Das Klebeband erzeugt im Lichtsensor andere Werte als der Umliegende Fußboden. Eine Liniенverfolgung kann auf dieser Grundlage erreicht werden, indem der Roboter sich nur vorwärts bewegt, wenn er sich über einem entsprechenden Lichtwert befindet. Für die Vorwärtsbewegung sorgt bereits der oben beschriebene DecisionThread. Stellt der Thread fest, das die Linie verlassen wurde, so stoppt die Vorwärtsbewegung und der Suchalgorithmus übernimmt die Kontrolle über die Aktoren.

Der Suchalgorithmus schwenkt den Roboter, abwechselnd nach rechts und links. Der Winkel um den geschwenkt wird, wird dabei mit jedem erfolglosen Schwenk vergrößert. Da er zuvor noch auf der Markierung war, ist diese noch in der Nähe und kann so wieder gefunden werden. In Abbildung 15 ist schematisch dargestellt wie der Roboter erst um α nach rechts schwenkt und dann um β nach links. Nach dem Linksschwenk befindet er sich wieder auf der Spur und kann weiter fahren.

Durchgeführt wird das Schwenken durch die Methode `rotate(int degree)`, die von Lejos bereitgestellt wird. Sie dreht den Roboter um die übergebene Gradzahl.

Listing 3: Aufrufe von `rotate()` um die Linie wieder zu finden

```

1 int alpha = 5;
2 while(sweeping){
3     // Im ersten Durchlauf, Rechtsdrehung da alpha positiv.
4     // Im zweiten Durchlauf, Linksdrehung da alpha negativ.
5     rotate(alpha, false);
6     alpha = alpha * -2;
7 }
```

Dieser Suchalgorithmus erwies sich als Zuverlässig, auch wenn die Erkennung der Linie mittels Lichtsensor, sich teilweise als problematisch erwies. So kann bei ungleichmäßig beleuchteten Räumen der Anfang der Strecke einen hohen Sensorwert liefern und das Ende einen viel geringeren. Teilweise waren diese Unterschiede so extrem das eine gewisse Toleranz vom Initialwert zwar etwas half aber keine Lösung darstellte. In der folgenden Beschreibung der Sensoren Implementierung wird darauf etwas genauer eingegangen und eine andere Lösung vorgestellt.

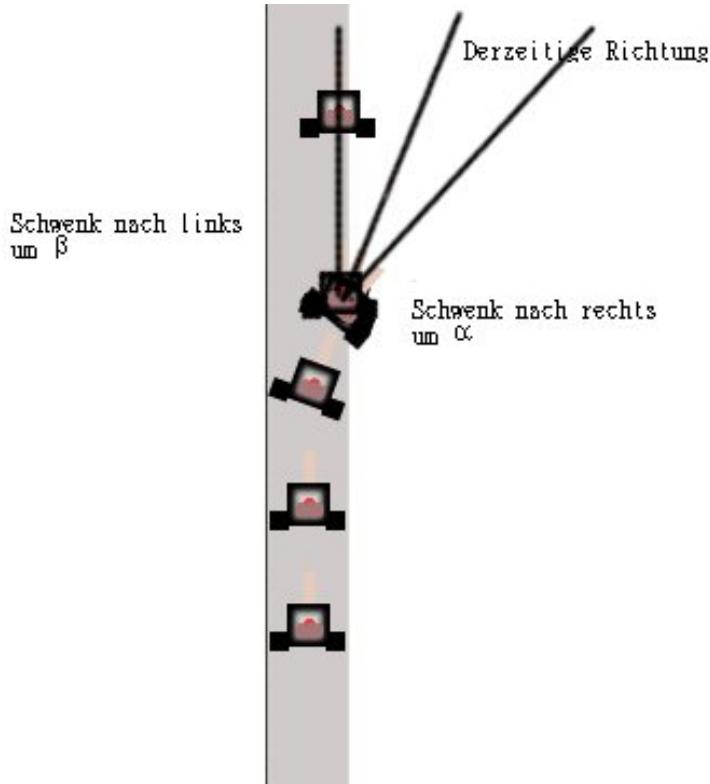


Abbildung 15: Verlieren der Markierung und Gegenmaßnahmen

8.2 Lesen und auswerten von Sensordaten

Der im `.sensor` befindliche Thread `SensorValueThread` ist für aktuelle Sensordaten verantwortlich. Bei diversen Tests hat sich gezeigt das auf diese zentral zugegriffen werden sollte, da manche Sensoren eine gewisse Zeit brauchen bis sie wieder abgefragt werden können.

Der `SensorValueThread` ermittelt alle 100 Millisekunden den aktuellen Wert für Licht und Richtung und hält diese, für Anfragen aus der Logikschicht bereit.

Durch die Probleme die sich beim Linien folgen ergaben, wurde diese Klasse außerdem dahingehend erweitert, dass sie statistische Informationen über die letzten gelesen Werte ermittelt. So lassen sich zum Beispiel die letzten 100 gelesenen Licht Werte Speichern. Durch das arithmetische Mittel dieser Werte, können abnehmende oder steigende Werte der Fahrbahnmarkierung Einfluss auf deren Schwellwert nehmen.

Denn wie bei der *Linienverfolgung* angedeutet, kann der zu Beginn eingelesen Schwellwert am Ende der Linie stark abweichen.

Um die Rechenleistung gering zu halten, wird der Durchschnittswert nur bei Anfrage berechnet.

Für die Daten des Kompass werden die gleichen statistischen Werte gesammelt. So können Ungenauigkeiten und falsche Werte ausgeschlossen werden. Allerdings gilt hier bei der Berechnung des Mittelwerts noch eine kleine Besonderheit. Das arithmetische Mittel ergibt sich durch die Summe der Werte geteilt durch deren Anzahl.(siehe Abb. 16) Der Kompass liefert Werte im Bereich von 0° bis 359° . Be-

$$m := \frac{1}{n} \sum_{i=1}^n X_i$$

Abbildung 16: Formel zur Berechnung des arithmetische Mittels

nutzt man nun die direkten Werte ergibt sich folgender Umstand:
 $(350^\circ + 10^\circ)/2 = 360/2 = 180^\circ$.

Richtig wäre aber das Ergebnis " 0° ". Da 350° unter Berücksichtigung des Intervalls nur nur -20° von $+10^\circ$ entfernt ist.

Richtig wäre also: $(-10^\circ + 10^\circ)/2 = 0^\circ$.

Erreicht wurde dies dadurch, dass beim Aufaddieren der Werte jeweils 360° dazu addiert wurde. Auf das dadurch entstehende arithmetische Mittel wird der Modul Operator angewandt, mit einem Wert von 360. Dadurch entsteht der richtige Mittelwert.(zu sehen im Listing 4

Beispiel: $((350+360)+(10+360))/2 = 1080 \% 360 = 0$

Listing 4: Mittelwert Berechnung für Kompass Werte

```

1 public float getAvgCompass() {
2     float sum = 0;
3     for (int i = 0; i < compassArray.length; i++) {
4         sum = sum + compassArray[i] + 360;
5     }
6     return (float) (sum/compassArray.length)%360;
7 }
```

Die statistischen Daten, haben das Folgen der Linie um ein Vielfaches verbessert. Da jetzt mit einem dynamischen Schwellwert gearbeitet wird, spielen die Lichtverhältnisse keine große Rolle. Wichtig ist für sich daraus ergebende Änderung des Schwellwertes, dass noch eine Sperre eingebaut werden muss. Schließlich dürfen keine Daten in die Statistik einfließen, die zum Beispiel beim Suchen nach der Linie entstehen. Also dann wenn sich der Lichtsensor über dem Boden befindet. Die Logikschicht ist dafür verantwortlich das die Datenerfassung zu diesem Zeitpunkten nicht stattfindet.

Die Werte des dritten Sensors, des RFID-Sensors werden mit einer Klasse ermittelt die das *NxtJlibA* Paket zur Verfügung stellt. Mit der bereitgestellten Klasse kann ein aus Java bekannter *EventListener* für den Sensor registriert werden. Der so erfolgte Statuswechsel kann

dann im Controller durch setzen des neuen Status bekannt gemacht werden.

Für die Erfassung der RFID-Tags spielt die Konstruktion der Knoten eine Rolle, sie müssen auch erkannt werden, wenn der Roboter sich genau am Rand der Linie bewegt, was zu Problemen führen kann. Auch musste die Sensoraufhängung noch weiter nach unten gesetzt werden. Zwar ist die Reichweite des Sensors mit 3cm beziffert, erfolgreiches Lesen fand aber erst ab einer Boden- Entfernung von 2cm statt.

8.3 Abbildung des RFID-Rasters

Eine konkrete Implementierung dieser Funktion konnte während der Bearbeitung des Projekts leider nicht erfolgen. Allerdings hätte eine Umsetzung wie folgt ausgesehen.

Die logische Abbildung des Rasters würde in einem 2 dimensionalen Array vom Typ *long* gespeichert. Initialisiert wäre jede Koordinate mit der Identifikationsnummer des entsprechenden RFID-Tags. Durch die Größe des Arrays, ergibt sich auch die Abmessungen. So lässt sich der Rand der Karte bestimmen.

Listing 5: Erstellung der logischen Karte

```

1 ...
2 long map[x][y];
3 map[0][0] = 12345678912345;
4 map[0][1] = 8912351561616172;
5 map[0][2] = 12455236568769;
6 ...

```

Die Koordinate mit dem Indexwert x und $y = 0$, würde dann der untersten linken Ecke entsprechen. Oder auch dem süd- westlichsten Punkt im Koordinaten System.

Navigieren lässt sich dann mittels der Himmelsrichtung, welche den Bezug zum realen Raster herstellt.

Soll der Roboter die Koordinate $(x + 1, y)$ ansteuern, so muss er einen Knoten nach Norden fahren. Die Erhöhung der y Komponente ergibt dementsprechend eine Fahrt nach Osten.

Ein einfacher Wegfindungsalgorithmus wäre dann, erst die x Komponente auf den richtigen Wert anzufahren, also sich erst in Nord-Süd Richtung zu bewegen und danach die richtige Y Komponente an zu steuern.

8.4 Kommunikation

Aufgrund von Kompatibilitätsproblemen der Bluetooth- Treiber mit einem 64-bit Betriebssystem, wie sich später herausstellte, war es mir leider nicht möglich die Kommunikationsschnittselle zu diesem Zeitpunkt noch entsprechend zu implementieren.

8.5 Zusammenfassung

Bei der Implementierung konnten leider nur Teilerfolge verbucht werden, da der Aufwand etwas unterschätzt wurde. Durch gewisse Schwachstellen und Ungenauigkeiten der NXT Hardware, steigt der Kalibrierungs- und Testaufwand enorm.

Dennoch konnte die unter Kapitel 7 entworfene Architektur zumindest als durchführbar bewertet werden. Zum Abschluss der Programmierung ist der Roboter in der Lage der Bodenmarkierung zu folgen und RFID-Tags zu identifizieren.

Die geplante Umsetzung des Rasters in der Software, basiert auf einer reinen Software-Lösung und sollte somit keine Probleme bei der Realisierung schaffen.

9 Erkenntnisse

9.1 Vergleich mit ähnlichen Technologien

Während der Recherchen für diese Ausarbeitung und der Bearbeitung des Projektes, fällt hauptsächlich auf das es nicht sehr viele praktische Beispiele gibt, die sich mit dem gleichen Thema befassen. Zu nennen wäre da der RFID-Rover von Herr Anders [anders]

Der RFID-Rover wurde auch mit einem Lego NXT realisiert und hat mich auch maßgeblich bei diesem Projekt inspiriert. Allerdings bewegt dieser sich frei im Raum von Wegpunkt zu Wegpunkt nur mit Hilfe des Kompass.

Von Interesse ist ebenfalls das Projekt des IWalker[iwalker] welches schon in der Einleitung erwähnt wurde. Dabei handelt es sich um einen Rollator, der per Sprachausgabe mitteilt, welche Räume gerade in der Nähe sind. Dabei kommt auch RFID-Technologie und ein Kompass zum Einsatz.

Das kommerzielle System was am meisten Ähnlichkeit zu dem aufweist, was in dieser Arbeit versucht wurde zu realisieren ist wohl Kiva von Kiva Systems[kiva] Auch dort existiert im Arbeitsbereich der Roboter ein Raster an Identifikationsmarken. Allerdings handelt es

sich dabei nicht um RFID-Tags sondern um auf den Boden gedruckte DataMatrix-Codes⁹. Das System scheint sehr effizient zu sein und wird auch praktisch eingesetzt.

9.2 Fazit

Wie das Kiva-System zeigt wurde in dieser Arbeit ein sehr interessanter Ansatz verfolgt. Ich denke gegenüber Barcodes, bieten RFID-Tags eine bessere und sicherere Positionsbestimmungsmöglichkeiten. Da je nach Ausstattung die Fähigkeiten der Tags ausgebaut werden kann, sind die Tags auch nicht nur auf die Identifikationsnummer beschränkt. In einer Lagerhalle, wo unter umständen Schmutz oder Flüssigkeiten den Boden verschmutzen können, können Bild gestützte Navigationsystem an ihre Grenzen kommen. Die Kommunikation per Funk lässt sich dagegen nicht so leicht stören. Darin sehe ich den größten Vorteil.

Durch die beschriebenen Technologien und Systeme wurden meine Erwartungen an RFID weitestgehend bestätigt. Auch wenn RFID schon relativ alt ist, ist mit modernen günstigen RFID-Tags ziemlich viel möglich. Es ist auf jedenfall ein Forschungsfeld wo noch sehr viel neues hinzukommt.

Zumindest in dieser Arbeit spielen die RFID-Tags lediglich die Rolle einer kleinen Bodenmarkierung zur Positionsbestimmung. Wenn man sich aber die ausbaubaren Fähigkeiten der Tags vor Augen hält, die sich um beliebige Komponenten erweitern lassen handelt es sich doch um eine sehr flexible Technologie.

Soweit mir die Implementierung bisher gelungen ist, würde ich auch sagen das, die auf Seite 17 unter "Aussicht" genannten Punkte mit genügend Zeit auf jedenfall realisiert werden könnten.

Während der Arbeit mit dem NXT sind mir jedenfalls bis auf die Ungenauigkeiten, kaum Beschränkungen aufgefallen die eine komplexe Software Implementierung verhindern würde.

Ein Problem das bei der Entwicklung viel Zeit gekostet hat, stellt sicherlich die Fehlerbehandlung dar. Auch wenn man mit den auf dem Display angezeigten Informationen, die Art des Fehlers und welche Methode ihn verursacht hat herausfinden kann, ist die Interpretation doch wesentlich komplizierter als in einer "normalen" Umgebung.

Die Durchführung dieses Projekts zeichnete sich hauptsächlich dadurch aus, das man gegen die Ungenauigkeit in der Aktorik des NXT ankämpfte. Auch scheinen manche Klassen aus Lejos[lejos] und Aplu

⁹Ein Barcode in Form eines 2d Musters

[aplu] sehr verbuggt zu sein, was den Implementierungsaufwand sehr stark steigerte.

Auch wenn kein komplettes und vollständiges System geschaffen werden konnte, so konnte doch festgestellt werden, dass es grundsätzlich möglich ist RFID-Tags als Navigationsgrundlage zu nutzen.

Literatur

[rfid] Allgemeines zu RFID

<http://de.wikipedia.org/wiki/RFID>

<http://www.rfid-journal.de/>

[sfloor] Der Intelligente Fußboden

http://www.vorwerk-teppich.de/sc/vorwerk/vorwerk_cebit_2006_rfid.html

[lins] Lego Mindstorms LME, von Lego in der Schule

<http://www.nxt-in-der-schule.de/>

[lme] Offizielle Lego Mindstorms Internetpräsenz

<http://mindstorms.lego.com/en-us/Default.aspx>

[coda] Lego RFID-Sensor Anleitung

http://www.codatex.com/picture/upload/image/Short_Manual_Web.pdf

[bt] Offizielle Bluetooth Infoseite

<http://www.bluetooth.com>

[tmv] TinyVM für Lejos

<http://tinyvm.sourceforge.net/>

[lejos] Lejos Implementierung

<http://lejos.sourceforge.net/>

[i2c] Beschreibung des I2C standardes

<http://www.mikrocontroller.net/articles/I2C>

[aplu] NxtJLibA - Referenzseite

<http://www.aplu.ch/home/apluhome.jsp?site=27>

[castor] Bauanleitungen zu Lego-NXT Robotern

<http://www.nxtprograms.com>

[rosteu] Steuerungsarchitekturen für autonome mobile Roboter

<http://darwin.bth.rwth-aachen.de>

[/opus3/volltexte/2002/408/pdf/Stenzel_Roland.pdf](http://opus3/volltexte/2002/408/pdf/Stenzel_Roland.pdf)

[sarch] Artikel zum Thema Schichtenarchitektur

<http://de.wikipedia.org/wiki/Schichtenarchitektur>

[anders] Anders@Mindstorms, diverse NXT Projekte, unter anderem
der RFID Rover

http://www.norgesgade14.dk/rfid_rover.php

[iwalker] Der IWalker, Assistent für ältere Menschen in einem RFID getaggten Haus

http://digital.cs.usu.edu/vkulyukin/vkweb/research/rollator_mounted_wayfinding.html

[kiva] Kiva Systems bieten eine logistische Roboter Lösung für Warenhäuser an.

<http://www.kivasytems.com/index.html>

[koeth] Michael Köther, Kooperatives Verhalten autonomer mobiler Roboter,

FH-Hannover Fakultät IV, 2008

Listings

1	Leerer DecisionThread	31
2	Regel definition in DecisionThread	31
3	Aufrufe von rotate() um die Linie wieder zu finden	33
4	Mittelwert Berechnung für Kompass Werte	35
5	Erstellung der logischen Karte	36