Dokumentation

zum Softwarepraktikum Teamrobotik SS 2007

Von Rene Kann, Ina Grischau und Tina Probst

mindstorms

Einführung

Diese Dokumentation ist eine Ergänzung zu unseren im Softwarepraktikum gehaltenen Vorträge, welche unsere Herangehensweise, Durchführung, sowie Erfolge und Probleme genauer beschreibt. Zusätzlich gehen wir auf das Hauptszenario und unsere Teilaufgaben ein, welche im Verlauf des Studienhalbjahres gelöst wurden.
Das Softwarepraktikum Teamrobotik wurde schon mehrere Semester hintereinander Angeboten. Wir waren die ersten Gruppen, welche das Praktikum mit den neuen LEGO NXT Robotern absolvieren durfte, welcher der Nachfolger des RCX ist. Da uns allen, sowohl Teilnehmern als auch Betreuer, dieser Bausatz neu war, wurden die Aufgaben in der Veranstaltung dynamisch entwickelt. Insgesamt absolvierten wir 4 Meilensteine (Teilaufgaben) auf dem Weg zum Hauptszenario. Dabei sollten die Vor und- Nachteile der jeweiligen Entwicklungsumgebungen getestet und der Umgang mit analogen Messdaten geübt werden. Das Praktikum dauerte ein Semester und fand im Sommersemester 2007 statt. Unser Betreuer war Dipl.-Inform. Manfred Deutscher-Tiemann.

Team

Unser Team besteht aus 3 Mitgliedern. Wir studieren Computer Visualistik im 4. Fachsemester (stand 2007).
Wir sind:

René Kann

Tina Probst

Ina Grischau

Inhaltsverzeichnis

Meilensteine

Aufgabe1

Zuerst sollten wir uns mit Hard- und Software befassen, unter Zuhilfenahme der im Internet gebotenen Quellen und des NXT Bausatzes. Wir programmierten die erste Steuerung, welche das Fahrzeug in alle Richtungen manövrierbar machte und testeten die Sensoren und Motoren.

Austesten der Sensoren

Wir testeten mit der „Try-Me-Funktion“ des NXT's Licht-, Tast-, Sound-, Ultraschallsensor, sowie den Motor. Der Tast- und Lichtsensor funktionierten auf Anhieb und es traten keinerlei Probleme auf. Nach einigem Austesten war auch der Ultraschallsensor nutzbar und es kamen keine weiteren Probleme zum Vorschein. Auch beim Soundsensor traten keine Schwierigkeiten auf.
Der Motor lief beim ersten Austesten mit der „Try-Me-Funktion“ erst gar nicht, doch nach einigem Probieren konnten wir unser erstes NXT-Modell anhand eines Motors zum fahren bringen.

Bild 1 : Erstes NXT Modell

Erste Schritte

Zunächst probierten wir unseren Roboter an einem Beispiel aus. Er sollte einer Schwarz/ weißen Kante aus Karton folgen. Er erkannte die Kannte sogar mit einer rötlichen Fließe statt des weißen Kartons, aber bei dem matten schwarzen Karton bekam er Schwierigkeiten. Die Zustände wurden sehr schlecht erkannt und obwohl der Austausch des schwarzen Kartons die Situation verbesserte, war kein Geradeausfahren möglich. Er bewegte sich im Kreis, sodass unser Fazit hier gilt: deutlich verbesserungswürdig.
Des Weiteren testeten wir das Fahren auf schwerem Gelände. Nachdem wir die vorderen Räder erhöhten und zwei Räder hinten befestigten, statt des Einen beweglichen (siehe Bild), überwand er auch Tierfelle. Jedoch war eine Stabilisierung nur bedingt möglich, so dass er etwas hin und her wackelte. An dieser Stelle hätten wir uns in dem Baukasten mehr Teile gewünscht, um die hintere Achse besser ausbauen zu können, um somit mehr Stabilität zu erhalten.

Aufgabe 2

Für unseren zweiten Vortrag beschäftigten wir uns mit Sensorenschnittstellen und deren Formen und Typen. Die Steuerung mobiler Systeme, die auf Umgebungsdaten und private Energieressourcen basieren, erfordern Kenntnisse über Sensoreneigenschaften, Motorparameter und Energiereserven. Die Ergebnisse der Sensorentests lagen uns in folgender Form und folgendem Typus vor.

	Form	**Typus**
Lichtsensor	%	Helligkeit (Ambient, Reflektion)
Servomotor	R, °	Weg (Umdrehungen, Grad)
Ultraschallsensor	cm, inch	Entfernung
Soundsensor	%	Unangepasste Dezibelwerte (dB), angepasste Dezibelwerte (dBA)
Tastsensor	0/1	Boolean

Tabelle 1 : Messdaten der einzelnen Sensoren

Nachfolgend finden sich die Sensoren in der Reihenfolge Lichtsensor (Ambient), Lichtsensor (Reflektion), Soundsensor(dbA), Soundsensor(bA), Ultraschallsensor und Servomotor. Zu jedem aufgeführten Diagramm wurden die datierten Zwischenergebnisse in einer Extra Tabelle erfasst und diesem Dokument im Anhang beigefügt.
Insgesamt wurden 468 Messdaten aufgenommen. Je 50 Messdaten für Ambient Licht, Reflected Licht, Ultraschall für Licht, Glas und Styropor und für den Soundsensor 10%, 50% und 100% Lautstärke unter dB und einmal 100% für dBA. Es wurde in den Abständen 0, 5, 10, 15, 20, 30, 40, 50, 60 und 70cm gemessen. Um den Extremwert herauszufinden, wurde eine weitere Messung durchgeführt. Für die Motoren wurden à 3 Messung unter 250g, 500g und keine Last durchgeführt.

Lichtsensor

Ambient Light

Der Lichtsensor hat eine Reichweite von ca. 2 m und die kritische Leistungsgröße befindet sich bei Entfernungen unter 20 cm und über 70 cm. Wie in dem folgenden Diagram erkennbar ist. Der konstanteste Werte wurde bei 40cm erzielt.

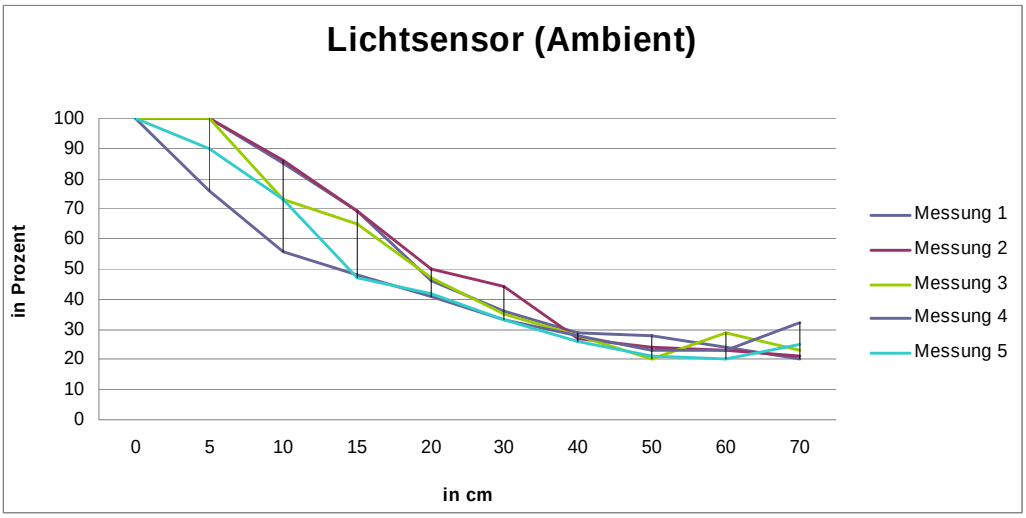

Diagramm 1 : Messdaten der Ambient Aufnahme des Lichtsensors

Auswertung des Diagramm 1:
Langwelliges Licht wird gut erkannt. Anhand der Leistungsgrößen lässt sich schlussfolgern, dass dieser Sensor einsetzbar ist, wenn man Objekte in der Nähe detektieren muss, da die Werte in der Ferne sehr schwanken.

Reflected Light

Dieser Sensor besitzt ebenfalls eine Reichweite von 2 m. Die kritische Leistungsgröße liegt unter 15cm und über 70cm. Der konstanteste Werte wurde bei 20cm aufgezeichnet.

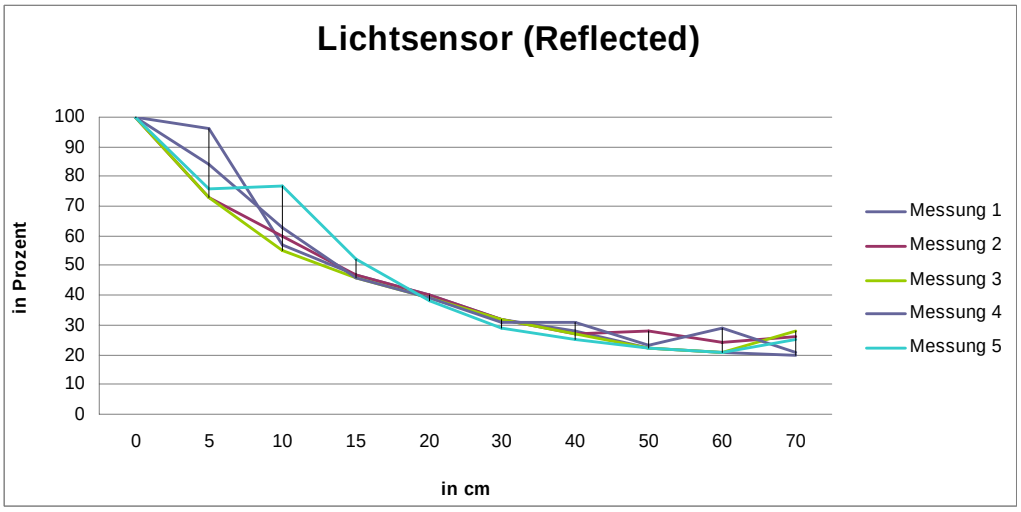

Diagramm 2 : Messdaten der Reflected Aufnahme des Lichtsensors

Auswertung des Diagramm 2:
Die Werte der Messungen schwanken eindeutig weniger als bei Ambient Light. So kann man sagen, dass dieser Sensor zuverlässiger misst. Dem Diagram kann man außerdem entnehmen, dass er für Werte zwischen 15 cm und 40 cm ideal einsetzbar ist.

Soundsensor

Dieser Sensor hat eine Reichweite von ca. 1 m und misst den Schalldruckpegel in dB und den bewerteten Schalldruckpegel in dBA. In jedem Versuch mit dem Soundsensor nutzen wir den Sound einer Sinuskurve als zu messenden Ton.

Soundsensor in dBA

Bei der folgenden Messreihe ließen wir uns die Messdaten des Soundsensors in dBA ausgeben.

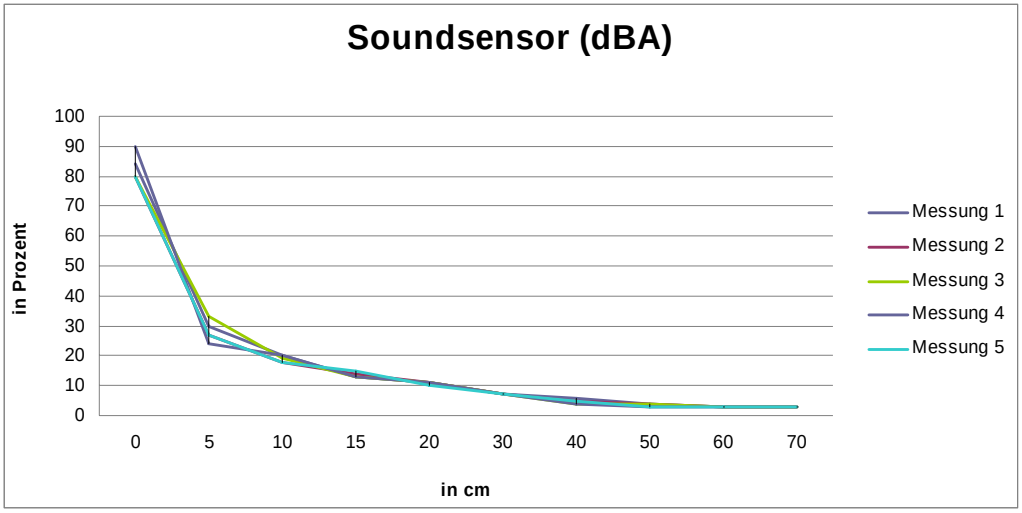

Diagramm 3 : Messdaten des Soundsensors in dBA

Auswertung des Diagramm 3:

Der Soundsensor liefert die stetigsten Werte aller Sensoren. Wie dem Diagramm zu entnehmen ist, wird er erst unter 20 cm ungenau und zwischen 5 cm und 0 cm ist keine brauchbare Aussage möglich.

Soundsensor in dB

In 3 weiteren, analogen Testreihen wurden unsere Ergebnis in dB ausgegeben, deren Voraussetzungen nur in der Lautstärke (10%, 50% und 100%) variiert wurden.

Auswertung der (folgenden) Diagramme 4-6:

Je leiser das Signal wird, umso weniger hat der Sensor Probleme einen festen Wert zu finden. Bei lauteren Tönen werden durch die Schwingungen der Membran, in unseren verwendeten Boxen, die Ergebnisse verfälscht. Der Sensor misst mehrere verschiedene Töne und kann dadurch keinen eindeutigen Wert messen. Dies ist sehr gut erkennbar im Diagramm 4, bei den Werten zwischen 0 und 5 cm Abstand zur Lautsprecherbox.

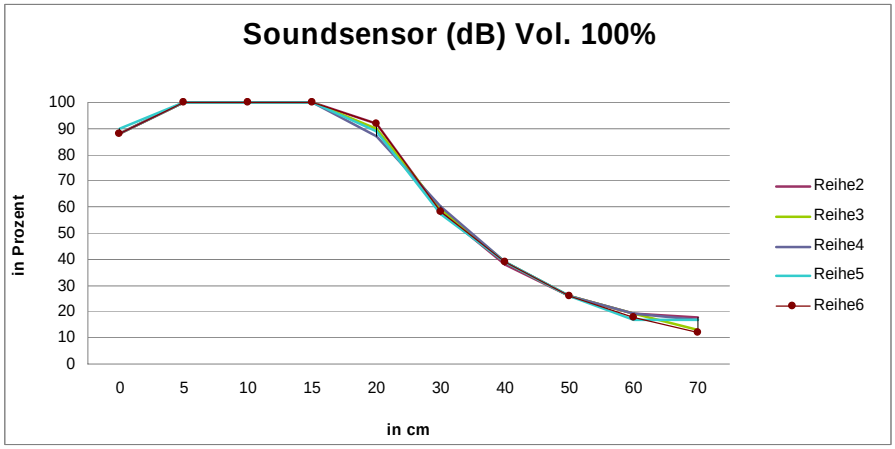

Diagramm 4 : Messdaten des Soundsensors in dB bei einer Lautstärke von 100%

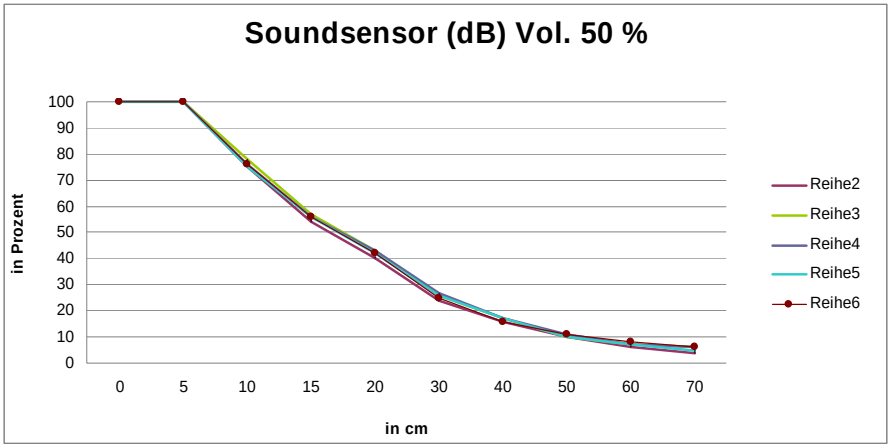

Diagramm 5 : Messdaten des Soundsensors in dB bei einer Lautstärke von 50%

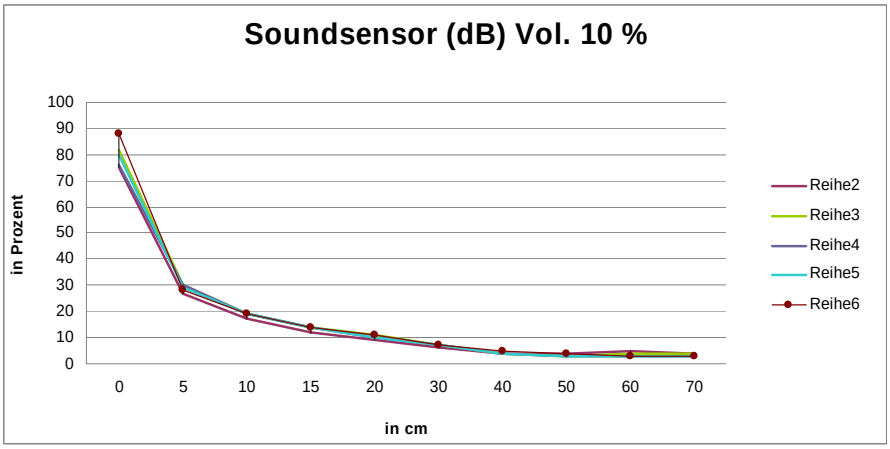

Diagramm 6 : Messdaten des Soundsensors in dB bei einer Lautstärke von 10%

Ultraschallsensor

Dieser Sensor hat bis zu 2,5 m eine Abweichung von ± 3 cm in seinen gemessenen Werten. Er misst die Zeit, die dessen ausgesendetes Signal braucht, um von einem Gegenstand reflektiert zu werden und wieder auf den Sensor zu treffen. Wir testeten den Ultraschallsensor im Folgenden mit drei unterschiedlichen Materialien.

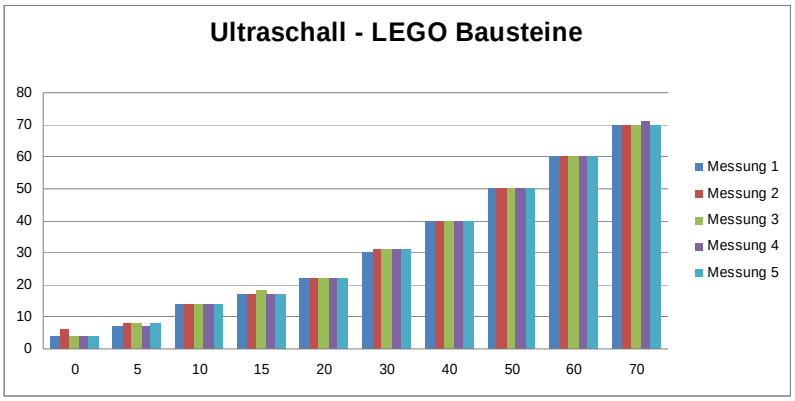

Diagramm 7 : Messdaten mit LEGO Bausteinen

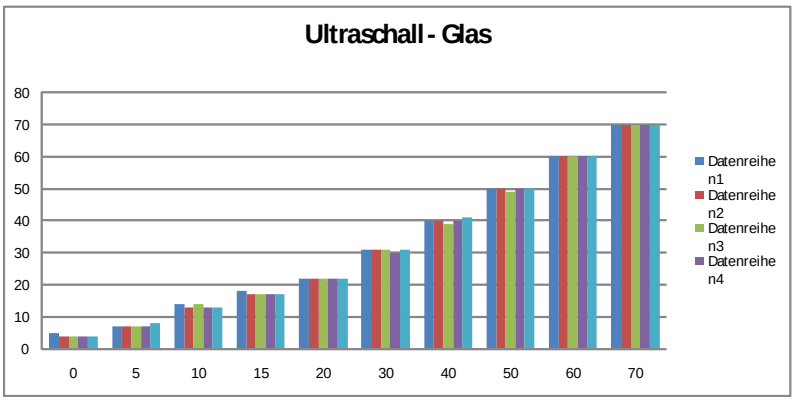

Diagramm 8 : Messdaten mit Glas

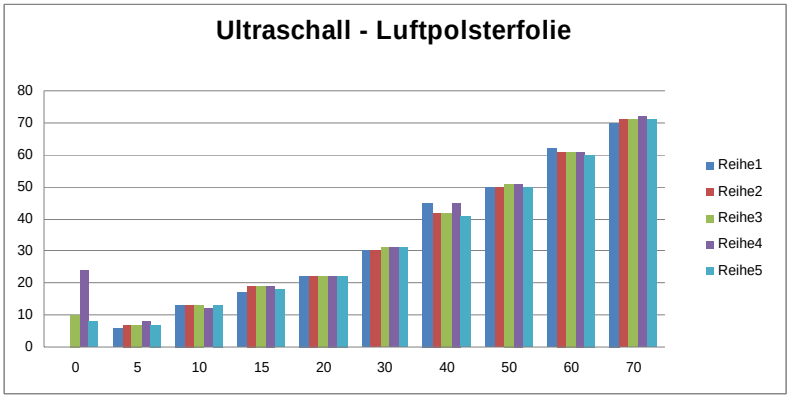

Diagramm 9 : Messdaten mit Luftpolsterfolie

Auswertung der Diagramme 7-9:
Bei Abständen zwischen 40 cm und 70 cm arbeitet der Sensor sehr genau. Jedoch hat er deutliche Probleme bei unebenen Oberflächen, wie Luftpolsterfolie (siehe Diagram). Bei glatten Oberflächen treten Probleme auf, wenn der Ultraschall in einem Winkel der gering größer oder kleiner 90° ist, auftrifft. Einfallwinkel ist gleich Ausfallwinkel und der reflektierte Ultraschall trifft nicht mehr vollständig auf den Sensor und verfälscht damit die Messungen.

Motoren

Motorenbelastung

Den Motor testeten wir anhand eines Belastungstests einmal ohne Gewicht, einmal mit 250 g und 500g. Wir missten die Sekunden, die er brauch, um eine Stecke von 10 m zurückzulegen.

Bild 2 : LEGO Modell für die Motorenbelastung

	Strecke	Messung 1 in sek	Messung 2 in sek	Messung 3 in sek	Ø Zeit in sek
Motoren Ohne Last	10 m	87	86	84	85, 67
Motoren mit 250 g	10 m	93	94	94	93, 67
Motoren mit 500 g	10 m	102	99	102	101

Tabelle 2 : Messdaten des Motorenbelastungstests

Auswertung des Motorenbelastungstests:
Es ist deutlich anhand der Tabelle ablesbar, dass die Motoren unter Last langsamer arbeiteten. Zwischen dem Ergebnis ohne Last und mit 250 g Belastung besteht ein Zeitunterschied von 8 Sekunden, wobei es zwischen unbelastet und 500 g Belastung ca. 15,33 Sekunden Unterschied sind und sich somit stark erhöht hat.

Akkutest

In einem zweiten Versuch wollten wir den Akku testen. Dafür haben wir die Ergebnisse des Motorenbelastungstests auf 8 Stunden Akkulaufzeit hochgerechnet.
Daraus ergeben sich die folgenden Werte:

Ohne Last	~ 3348 m
Last: 250 g	~ 3096 m
Last: 500 g	~ 2851 m

Tabelle 3 : Zurückgelegte Stecke bei einer Akkulaufzeit von 8 h und vollem Akku

Aufgabe 3

Austausch von aktuellen Daten zwischen NXT und Laptop

Wir sollten einen unserer letzten Versuche (Ultraschall und/oder Licht) wiederholen und unter den neuen Anforderungen demonstrieren, wie Messdaten vom NXT an Laptop gesendet, gespeichert und visualisiert werden. Das Senden und Speichern gelang uns und wird als Funktion in dem folgenden Programm für die Balldetektion verwendet. Auch in weiteren Funktionen findet sie ihre Anwendung.

Balldetektion

Wir sollten unseren NXT Roboter den IR-Ball finden lassen. Ab welcher Distanz wird er mit Infrarot detektierbar? Des Weiteren sollten wir einen aktiven oder passiven Greifer montieren, damit der NXT den Ball zur Lichtquelle transportieren kann. Darüber hinaus sollten wir versuchen den Ball per Farbenwert anzusteuern.
Unsere Funktion zur Balldetektion läuft folgenden Algorithmus ab. Der NXT dreht sich um 360° im Uhrzeigersinn und misst dabei mit dem Lichtsensor die Intensitäten. Er merkt sich die höchste Intensität und die dazugehörige Gradzahl. Dann dreht er sich entgegen dem Uhrzeigersinn um die Differenz zwischen 360° und der gemerkten Gradzahl wieder zurück. Aus Hardwaretechnischen Gründen, wie den Rädern oder Ketten, tritt hierbei eine geringe Ungenauigkeit auf. Somit fährt der NXT nicht direkt auf den Ball zu, so dass noch eine Korrektur nach einer bestimmten Stecke eingebaut werden musste. Wir haben uns überlegt, dass er nach jedem „Schritt“ die Lichtstärke misst. Weicht diese von einem bestimmten Wert ab, wird der Ball erneuert gesucht.
Wir haben herausgefunden, dass der Ball ab ca. 2 m Entfernung detektierbar wird, je nachdem auf welche Lichtintensität der NXT reagieren soll. Dieser Abstand hat sich als gut erwiesen, da bei diesem Schwellwert das Umgebungslicht nicht mit dem Ball verwechselt wird.

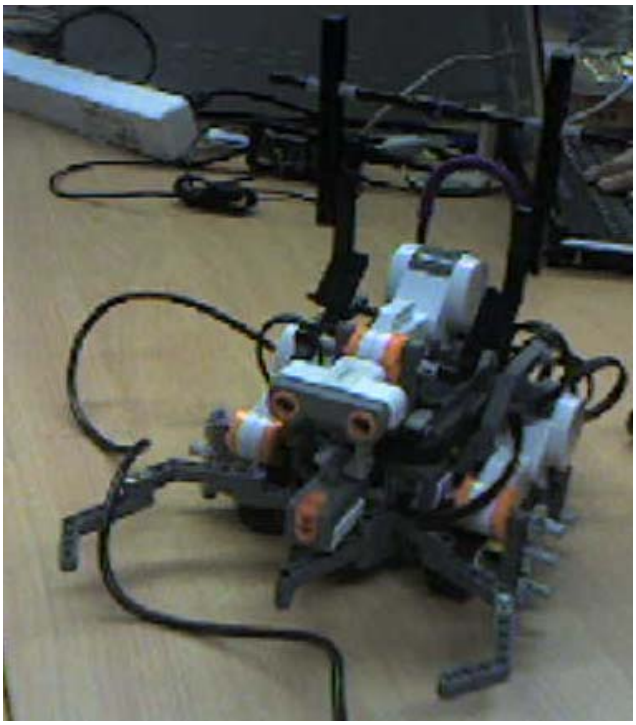

Bild 3 : Letzte Version unseres LEGO Modells mit Greifer

Aufgabe 4

Umfahren von Hindernissen

Es sollte ein Kreis- oder Linienförmiger Hindernisparcours aufgebaut werden, dessen Zwischenräume jeweils der doppelten breite des Roboters entsprechen. Durch ihn sollte der NXT mit Hilfe von Tastensteuerung oder unter Hilfenahme der Sensoren gesteuert werden. Als weitere Aufgabe war herauszufinden, ob Runde Hindernisse anders war genommen werden, als Eckige. Zuletzt sollten wir die Balldetektion durchführen, d.h. den Ball greifen und zu einer Lichtquelle die abseits der Barriere positioniert sein soll transportieren.

Hindernisunterschiede
Zunächst testeten wir die Erkennung verschiedener Hindernisse. Mit Hilfe des Ultraschallsensors sollte unser Roboter eine Cappuccinodose (rundes Hindernis), einen kleinen Sarg (glatte Oberfläche) und eine Raufaserwand (unebene Oberfläche) suchen. Die ersten Probleme traten bereits bei der runden Dose auf. Wie in der Auswertung der Messdaten zum Ultraschallsensor bereits erwähnt, hat er Probleme mit Einfallswinkeln über und unter 90°. Somit wurde der reflektierte Schall nicht vollständig von dem Sensor aufgenommen und es kam zu einer Verfälschung der Ergebnisse. So auch bei der Raufasertapete, weil diese eine unebene Oberflächenstruktur aufweist, so wie Luftpolsterfolie (siehe Diagramm 9).Letztendlich ist zu sagen, dass die besten Ergebnisse bei Hindernissen mit einer glatten Oberfläche erzielt werden, welche der Sarg aufweist. Diese Hindernisse sollten jedoch nahezu senkrecht zum Untergrund stehen, da der Schall (wie oben beschrieben) sonst nicht richtig reflektiert wird.

Bild 3 : Letzte Version unseres LEGO Modells mit Greifer

Balldetektion und Transport zur Lichtquelle
Als erstes wird die Balldetektion, wie in Aufgabe 3 erläutert, ausgeführt. Stellt der Lichtsensor einen Wert über einen angegebenen Schwellwert fest, so lässt er den Greifer herunter und der Ball ist gefangen. Nun wird der selbe Algorithmus (Balldetektion) ein weiteres Mal, nur mit einem anderem Schwellwert, aufgerufen. Dieser Schwellwert ist auf die Taschenlampe eingestellt. Da Ball und Taschenlampe mit dem Lichtsensor wahrgenommen werden, ist es nahe liegend, dafür den selben Algorithmus zu verwenden. Am Ziel angekommen, wird das Programm beendet.

Hauptszenario

Aufgabenstellung

Ziel des Softwarepraktikums war die kollisionsfreie Navigation des NXT durch einen Parcour von variabel positionierbaren Hindernissen. Dabei soll ein Signal gebendes Objekt (hier: Infrarotball) identifiziert, angesteuert und aus dem Parcour zu einem Zielpunkt (hier: Lichtquelle) befördert werden.

Lösungsweise

Dazu schrieben wir zunächst die Funktion „ballsuchen()“, die in der 3. Aufgabe unter dem Punkt „Balldetektion“ näher beschrieben wurde. Um das Problem des „aus den Augen Verlierens“ zu lösen, welches unter diesem Punkt aufgeworfen wurde, fügten wir die Funktion „korrigiereLinksRechts()“ hinzu. Diese misst permanent die Lichtintensität und speichert sie in einem Feld. Weicht sie zu stark ab, dann wird der NXT entweder nach links oder rechts korrigiert, je nachdem, wo das Licht herkommt. In der Funktion „fahreZumBall()“ wird die eben genannte Funktion eingebaut.

Hindernissen umfahren

Eine erste Idee zum Umfahren der Hindernisse war die Speicherung dieser in einer „Landkarte“. Diese Karte sollte durch ein 2-dimensionales Feld repräsentiert werden, welches den Größeneinheiten des Parcours entspricht. Findet der NXT nun ein Hindernis, dann wird die Position in der Landkarte vermerkt (siehe Bild 4). Die Nullposition befindet sich links oben, so dass man der Karte entnehmen kann, dass 2 cm nach rechts und 2 cm nach unten ein Hindernis steht. Die Detektion der Hindernisse sollte zufällig erfolgen, so dass im schlechtesten Fall nicht alle Hindernisse gefunden bzw. gespeichert werden. Ein Vorteil wäre aber, dass man auch die Lichtquelle speichern (hier rot gekennzeichnet) und den kürzesten Weg zu dieser berechnen könnte. Dann müsste aber gewährleistet sein, dass alle Hindernisse vermerkt sind.

	1	2	3	4
1				
2		X	X	
3				X
4	X		O	

Bild 4: Beispiel an einer 4x4 kleinen Landkarte, schwarze Kreuze sind erkannte Hindernisse, das rote Kreuz ein Licht und das O unser Roboter

Nach dem wir anfingen diesen Kartenalgorithmus zu Programmieren, wurde schnell klar, das wir dafür länger als die uns verbleibenden Wochen brauchen würden. So entschieden wir uns für eine nicht so effektive Variante, welche jedoch nicht Wirkungsloser war. Wenn der Roboter kein Licht gefunden hat, geht er über in einen Discovery Modus. Hier sucht er nach dem am weitesten entferntem Hindernis und speichert sich die Gradzahl genau wie bei dem Lichtsuchalgorithmus ab und fährt nach dem Zurückdrehen auf den gespeicherten Wert los. Er fährt solange, bis er vor einem Hindernis steht oder bis er eine eingegebene Anzahl von Umdrehungen zurückgelegt hat. Dies gewährleistet, dass er sich nicht zu weit vom Geschehen entfernt. Dann wird wieder nach dem Licht gesucht.

Ablauf des Hauptszenarios

Das Hauptszenario lief für uns sehr enttäuschend. Immer wieder traten schon während der Probefahrten Unregelmäßigkeiten beim durchlauf des Programms auf. So erhielten wir sogar ein paar Mal einen anderen Ablauf bei gleich bleibenden Programmcode, bzw. unterschiede bei gleichen Codes und anderen NXT Robotern. Diese Unstimmigkeit trat auch bei anderen Gruppen auf, so wie wir im Gesprächen miteinander erfuhren. Bei der Finalen Vorführung fand unser Roboter den Ball und schloss den Greifer, drehte sich daraufhin ein paar mal und fuhr dann nicht weiter, selbst wenn sich die Lichtquelle genau vor seinem Sensor befand. Wir versuchten diesen plötzlichen Fehler zu beheben, fanden ihn aber nicht.

Fazit

Obwohl wir auf sehr viele Probleme mit der Programmierung der NXT gestoßen sind, hatten wir sehr viel Spaß mit den Robotern. Wir haben uns mit Sensoren auseinandergesetzt, was uns sonst nicht möglich gewesen wäre. Dank diesen Einblick hat sich unseren Horizont erweitert. Hoffentlich lassen sich einige Hindernisse für die nächsten Gruppen beseitigen, so dass diese mit einer besseren Grundlage diese Aufgabe besser bewältigen können.

Tipps für die Arbeit mit den nächsten Gruppen

Für die nächsten Gruppen würden wir uns Wünschen nicht so genau auf die Sensorik einzugehen, um damit mehr Zeit für die Programmierung zu erhalten. Deren Aufgabenstellung könnte dann auch genauer (Einheitlicher Parcour) und kleiner aufgegeben werden. Ein Abgabesystem (vielleicht mit Abgabedatum) würde es ermöglichen die Lösungen jeden Meilensteins von den Gruppe ins Internet zu stellen, um so die Gruppen untereinander zu synchronisieren und auf eine Ebene zu bringen.

Quellenangaben

http://www.ortop.org/NXT_Tutorial/html/essentials.html /NXTTutorial– Essentials/ **LEGO**® Group / letzter Zugriff: 03.07.07

http://www.nxt-mindstorms.com/wiki/Main_Page / Lego Mindstorms Wiki/ letzter Zugriff: 03.07.07

http://www.mindstorms.com / Homepage von Mindstorms/ **LEGO**® Group / letzter Zugriff: 03.07.07

http://www.mindstormsforum.de / Lego Mindstorms und NXT Forum/ Frank Engeln/ letzter Zugriff: 03.07.07

http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/sa_nxt/download/sa-2006.18.pdf / LEGO Mindstorms NXT– Next Generation / Claudia Frischknecht & Thomas Other / letzter Zugriff: 03.07.07

Anhang

Quellcode

Hier sei als Beispiel einer unsere Quellcodes aufgeführt. Auf Nachfrage kann der Finale Code nachgereicht werden.

```
#include "NXCDefs.h"

bool ballGefunden;
bool steheVorBall;

void ballSuchen()
{
    SetSensorType(IN_4,IN_TYPE_LIGHT_INACTIVE);
    SetSensorMode(IN_4,SENSOR_MODE_RAW);
    int drehanzahl = 360;
    int speed = 60;
    int warteZeit = 10;

    int lichtstart = 0;

    if(!ballGefunden)
    {
                repeat(drehanzahl)
                {
                  OnFwd(OUT_B,speed);
                  OnRev(OUT_C,speed);
                  Wait(warteZeit);

                 if(Sensor(IN_4) > lichtstart)
                            lichtstart = Sensor(IN_4);
                }

                Wait(500);

                repeat(drehanzahl)
                {
                  OnFwd(OUT_B,speed);
                  OnRev(OUT_C,speed);
                  Wait(warteZeit);

                 if(Sensor(IN_4) >= lichtstart)
                 {
                            ballGefunden = true;
                            Off(OUT_BC);
                            break;
                 }
                }
    //int drehWinkelZurueck = drehWinkel*drehanzahl - drehWinkel*pos;
```

```
    //drehWinkelZurueck = drehZeit*drehanzahl - drehWinkelZurueck;
    }
}
```

void korrigiereLinksRechts()

```
{
    Off(OUT_BC);
    SetSensorType(IN_4,IN_TYPE_LIGHT_INACTIVE);
    SetSensorMode(IN_4,SENSOR_MODE_RAW);
    int arrayWerte[];

    arrayWerte[1] = Sensor(IN_4);
    int drehWinkel = 50;
    //RotateMotorEx(OUT_BC,60,drehWinkel,-100,true,true);
    arrayWerte[0] = Sensor(IN_4);
    //RotateMotorEx(OUT_BC,60,drehWinkel*2,100,true,true);
    arrayWerte[2] = Sensor(IN_4);

    /** Links größter Wert **/
    if(arrayWerte[0] > arrayWerte[1] && arrayWerte[0] > arrayWerte[2])
                //RotateMotorEx(OUT_BC,60,drehWinkel*2,-100,true,true);

    /** Mitte größter Wert **/
    if(arrayWerte[0] < arrayWerte[1] && arrayWerte[1] > arrayWerte[2])
                //RotateMotorEx(OUT_BC,60,drehWinkel,-100,true,true);
    /** Sonst steht er schon auf rechts = größter **/

     OnFwd(OUT_BC,50);
}
```

void fahreZumBall()

```
{
        SetSensorType(IN_4,IN_TYPE_LIGHT_INACTIVE);
        SetSensorMode(IN_4,SENSOR_MODE_RAW);

        int wertAlt = Sensor(IN_4);
        int speed = 70;
        bool sucheBallneu;

        while(true)
        {
                OnFwd(OUT_BC,speed);
                Wait(100);

                if((wertAlt-Sensor(IN_4)) > 0)
                {
                        ballSuchen();
                }

                if(Sensor(IN_4) > 750)
                {
```

```
                        steheVorBall = true;
                        Off(OUT_BC);
                        break;
                }

                wertAlt = Sensor(IN_4);
        }


        if(!sucheBallneu && steheVorBall)
        {
                Off(OUT_BC);
        }
}

task main()
{
    Wait(2000);
    while(true)
    {
            ballSuchen();
            if(ballGefunden && !steheVorBall)
            {
                Wait(1000);
                fahreZumBall();
            }
    }
}
```

Messdaten der Sensoren

Lichtsensor Ambient

	Abstand (cm)	%	%	%	%	%	
1	**0**	100	100	100	100	100	100
2	**5**	100	100	100	76	90	93,2
3	**10**	85	86	73	56	73	74,6
4	**15**	69	69	65	48	47	59,6
5	**20**	46	50	47	41	42	45,2
6	**30**	36	44	35	33	33	36,2
7	**40**	29	27	28	28	26	27,6
8	**50**	28	24	20	23	21	23,2
9	**60**	24	23	29	23	20	23,8
10	**70**	20	21	23	32	25	24,2

Lichtsensor Reflected

	Abstand (cm)	%	%	%	%	%	
1	**0**	100	100	100	100	100	100
2	**5**	96	73	73	84	76	80,4
3	**10**	57	60	55	63	77	62,4
4	**15**	47	47	46	46	52	47,6
5	**20**	40	40	39	39	38	39,2
6	**30**	32	32	32	31	29	31,2
7	**40**	28	27	27	31	25	27,6
8	**50**	22	28	22	23	22	23,4
9	**60**	21	24	21	29	21	23,2
10	**70**	20	26	28	21	25	24

Ultraschallsensor Legowand

	Abstand (cm)	cm	cm	cm	cm	cm	cm
1	**0**	4	6	4	4	4	4,4
2	**5**	7	8	8	7	8	7,6
3	**10**	14	14	14	14	14	14
4	**15**	17	17	18	17	17	17,2
5	**20**	22	22	22	22	22	22
6	**30**	30	31	31	31	31	30,8
7	**40**	40	40	40	40	40	40
8	**50**	50	50	50	50	50	50
9	**60**	60	60	60	60	60	60
10	**70**	70	70	70	71	70	70,2

Ultraschallsensor Glas

	Abstand (cm)	cm	cm	cm	cm	cm	cm
1	**0**	5	4	4	4	4	4,2
2	**5**	7	7	7	7	8	7,2
3	**10**	14	13	14	13	13	13,4
4	**15**	18	17	17	17	17	17,2
5	**20**	22	22	22	22	22	22
6	**30**	31	31	31	30	31	30,8
7	**40**	40	40	39	40	41	40
8	**50**	50	50	49	50	50	49,8
9	**60**	60	60	60	60	60	60
10	**70**	70	70	70	70	70	70

Ultraschallsensor Styropor

	Abstand (cm)	cm	cm	cm	cm	cm	cm
1	**0**	n/A	n/A	10	24	8	8,4
2	**5**	6	7	7	8	7	7
3	**10**	13	13	13	12	13	12,8
4	**15**	17	19	19	19	18	18,4
5	**20**	22	22	22	22	22	22
6	**30**	30	30	31	31	31	30,6
7	**40**	45	42	42	45	41	43
8	**50**	50	50	51	51	50	50,4
9	**60**	62	61	61	61	60	61
10	**70**	70	71	71	72	71	71

Soundsensor dB Lautstärke 100%

	Abstand (cm)						
1	**0**	88	88	88	90	88	88,4
2	**5**	100	100	100	100	100	100
3	**10**	100	100	100	100	100	100
4	**15**	100	100	100	100	100	100
5	**20**	92	90	87	89	92	90
6	**30**	58	59	60	57	58	58,4
7	**40**	38	39	39	39	39	38,8
8	**50**	26	26	26	26	26	26
9	**60**	19	19	19	17	18	18,4
10	**70**	18	13	17	17	12	15,4

Soundsensor dB Lautstärke 50%

	Abstand (cm)						
1	**0**	100	100	100	100	100	100
2	**5**	100	100	100	100	100	100
3	**10**	75	78	76	75	76	76
4	**15**	54	57	56	56	56	55,8
5	**20**	40	43	43	42	42	42
6	**30**	24	26	27	26	25	25,6
7	**40**	16	17	17	17	16	16,6
8	**50**	10	10	11	10	11	10,4
9	**60**	6	7	7	7	8	7
10	**70**	4	6	6	5	6	5,4

Soundsensor dB Lautstärke 10%

	Abstand (cm)						
1	**0**	75	82	76	80	88	80,2
2	**5**	27	30	30	29	28	28,8
3	**10**	17	19	19	19	19	18,6
4	**15**	12	14	14	14	14	13,6
5	**20**	9	11	10	10	11	10,2
6	**30**	6	7	7	7	7	6,8
7	**40**	4	4	4	4	5	4,2
8	**50**	4	3	3	3	4	3,4
9	**60**	5	4	3	3	3	3,6
10	**70**	4	4	3	3	3	3,4

Soundsensor dBA Lautstärke 100%

	Abstand (cm)						
1	**0**	84	80	80	90	80	82,8
2	**5**	30	27	33	24	27	28,2
3	**10**	20	18	19	20	18	19
4	**15**	13	14	13	13	15	13,6
5	**20**	11	11	11	11	10	10,8
6	**30**	7	7	7	7	7	7
7	**40**	6	5	4	4	5	4,8
8	**50**	4	4	4	3	3	3,6
9	**60**	3	3	3	3	3	3
10	**70**	3	3	3	3	3	3