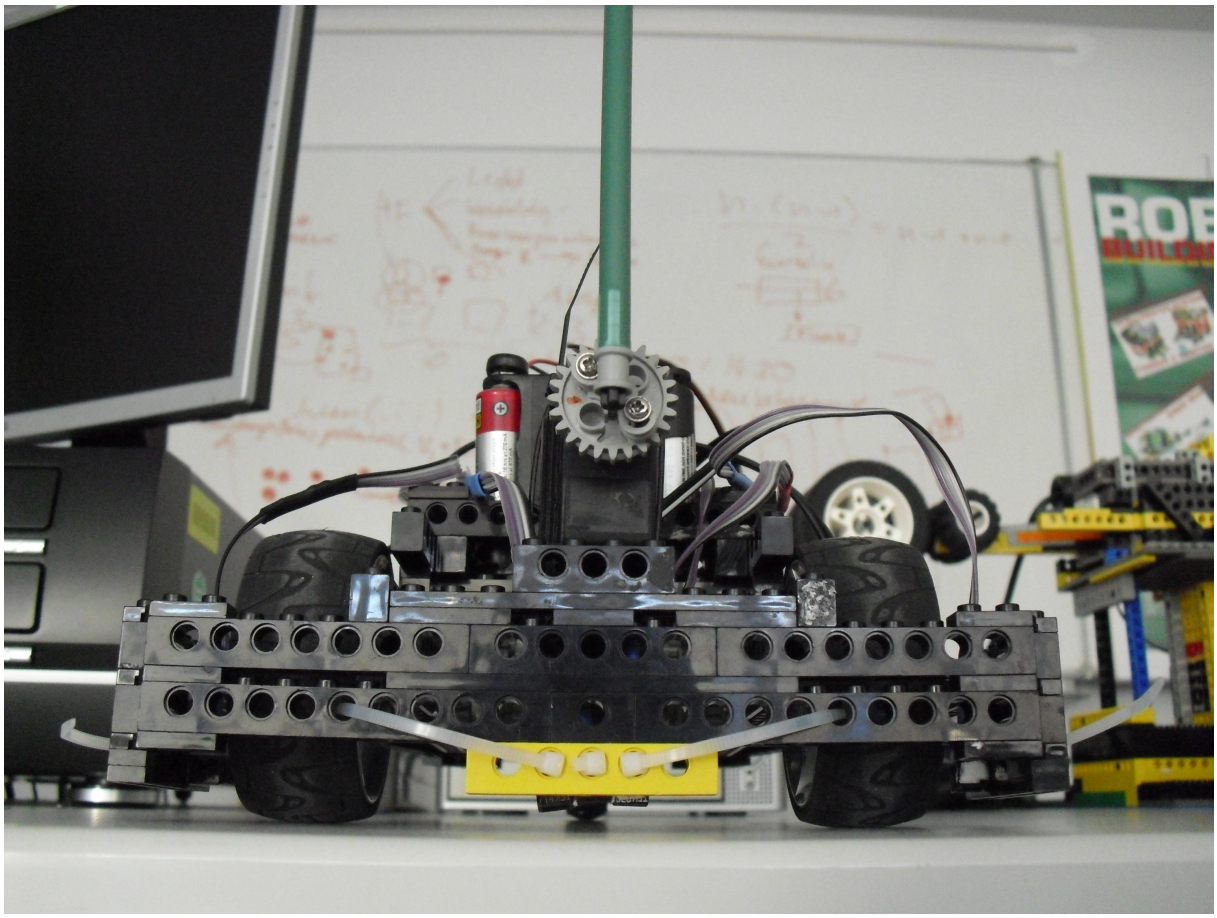
Dokumentation
Lego®-Roboter „Dancing Queen“



Projektaufgabe - Autonome mobile Systeme

Entwickler: Christian Kohls, Helge Scheel
Betreuer: Dipl.-Inf. I. Boersch, Prof. J. Heinsohn

Fachhochschule Brandenburg
Wintersemester 2010/2011

Inhaltsverzeichnis

1. Lösungsansätze
    1. Lösungsansatz Nr. 1 Seite 3
    2. Lösungsansatz Nr. 2 Seite 4
    3. Lösungsansatz Nr. 3 Seite 4
    4. Lösungsansatz Nr. 4 Seite 5

2. Hardware
    1. Sensorik Seite 7
    2. Motoren und Getriebe Seite 8
    3. Servomotor und Arm Seite 9
    4. Fahrgestell Seite 10

3. Software Seite 12

4. Quellenverzeichnis Seite 14

5. Anhang Seite 15

1. Lösungsansätze

Lösungsansatz Nr. 1

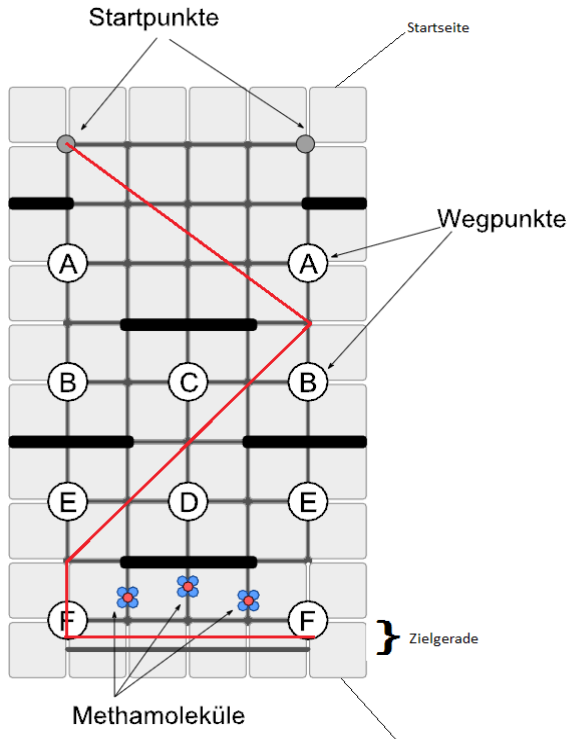


Abb. 1: geplanter Pfad des ersten

Im ersten Lösungsansatz sollte der Roboter vom Startpunkt diagonal zwischen die Punkte A und B der gegenüberliegenden Seite fahren. Von dort wiederum sollte er sich diagonal zwischen die Punkte E und F bewegen und sich rechtwinklig zur Zielseite ausrichten. Nach einer kurzen Strecke welche „Dancing Queen“ zeitgesteuert geradeaus fahren sollte, mussten die beiden Linien der Zielgerade erkannt werden und mit den Rädern auf diesen entlang gefahren werden. Während der Fahrt auf der Zielgeraden sollte mittels eines Servomotors ein schwenkbarer Stab in Position gebracht werden, um die Kugeln hinunter zu stoßen. Die Wegpunkte wurden bei diesem Ansatz mit Ausnahme von F nicht überquert, somit konnten während der Anfahrt auf das Ziel keine Punkte gesammelt werden. Jedoch ging man von einem schnellen Getriebe und wenigen zeitaufwendigen Kurvenmanövern aus, sodass die Annahme getroffen wurde, man würde die Moleküle schneller erreichen als die Konkurrenz. Die Punkte für zwei erfolgreiche Spaltungen (20 Punkte) und einem oder zwei Wegpunkte (drei bzw. sechs Punkte) würden aber schon für einen Sieg ausreichen, wenn die Konkurrenz gehindert werden konnte, die dritte Spaltung erfolgreich durchzuführen (max. 3 * 6 Punkte). Dies erschien recht wahrscheinlich, da der Roboter dem gegnerischen Roboter auf Höhe des dritten Moleküls im Weg gestanden hätte, die C-Atome aus ihrer Position gebracht oder die Spaltung selbst vorgenommen hätte.

Nachdem sich das erste Getriebe als völlig ungeeignet erwiesen hatte, musste nicht nur das Getriebe komplett neu gebaut werden, auch der Rahmen des Roboters musste stark angepasst werden. Die nötigen Änderungen erwiesen sich als so umfangreich, dass ein kompletter Neubau des Roboters nötig war, bevor der erste Test mit einem fahrbaren Roboter möglich war.
Auch bei der komplett neuen Version stellte sich schnell heraus, dass der Roboter bei gleicher Leistung auf beiden Motoren nicht reproduzierbar gerade aus fuhr, sondern zu ellipsenartigen Bahnen neigte. Die gefahrenen Bahnen neigten zu starken Schwankungen, die nicht auf längeren Strecken weder tolerierbar noch korrigierbar waren.

Lösungsansatz Nr. 2

Dem Problem der unpräzisen Fahrweise sollte mit Verbesserung des Getriebes und einer Umstellung der Taktik entgegen gewirkt werden. Der zweite Lösungsansatz beinhaltete die Erkennung der Hindernisse mittels Infrarotsensoren. Der Roboter sollte bei seiner Fahrt durch den Parcours die aufgestellten Hindernisse frühzeitig erkennen, sich an ihnen orientieren und seine Fahrt entsprechend anpassen. So sollte eine weiche Kurvenfahrt um die Hindernisse realisiert werden und gleichzeitig das Problem der ungenauen Fahrt überwunden werden.

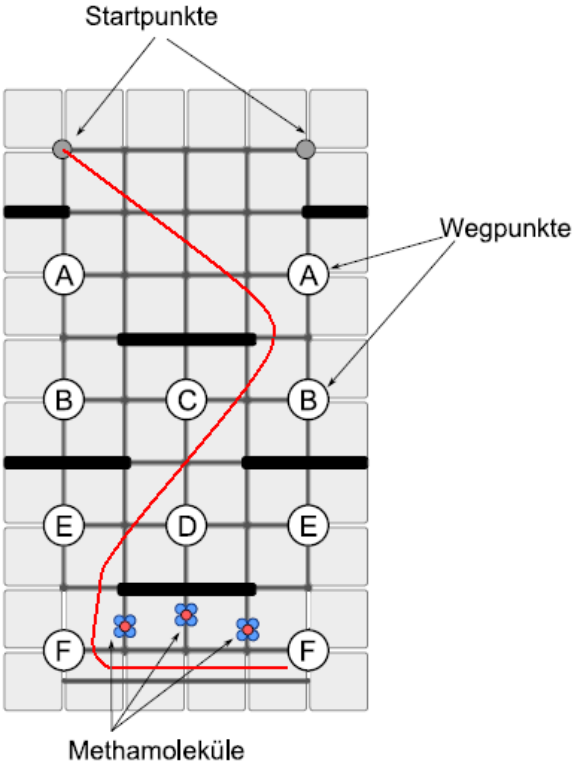


Abb. 2: dicht an den Hindernissen vorbei auf elliptischen Bahnen

Bei weiteren Tests wurde offensichtlich, dass die Fahrpräzision des Roboters so ungenau war, dass auch die Infrarotsensoren bei längeren Wegstrecken keine Hilfe sein würden und dieser Fahrplan nicht realisierbar war. Die Verbesserungen am Getriebe entpuppten sich nur als scheinbar, als bei Tests an einen anderen Arbeitstag die Fahrweise ebenso unberechenbar war wie zuvor.

Lösungsansatz Nr. 3

Da die unpräzise Fahrweise des Roboters sich vor allem bei längeren Strecken deutlich äußerte, wurde beim dritten Lösungsansatz ein Fahrplan gewählt, der aus kleineren Strecken bestand, die ohne Unterbrechung möglichst gerade aus gefahren werden mussten. Die abgebildete Route kam ohne lange Diagonalen aus und ermöglichte die Orientierung an mehr Hindernissen. Der Vorteil der wenigen Kurvenfahrten der vorherigen Ansätze musste zwar aufgegeben werden, dennoch schien der Lösungsweg aussichtsreich, da Tests zeigten, dass die Geschwindigkeit in den Kurven nicht sehr stark reduziert werden musste. Zudem ging man von sehr weichen Kurvenbewegungen aus, sodass die Geschwindigkeit nicht von abrupten Manövern stark verringert werden würde. Ein weiterer Vorteil dieses Fahrplans war die Nähe zu den Wegpunkten, sodass die Punkte für Überquerung gesammelt werden konnten.

Die Verwendung der Infrarotsensoren erwies sich allerdings als wesentlich schwieriger als ursprünglich gedacht. Vor allem die Aktivierung über die Software war ungewöhnlich zeitaufwendig und warf das Projekt erneut zurück. Ein defekter Sensor komplettierte dabei die Verwirrung und sorgte für seinen Anteil an der zeitlichen Verzögerung.
Nachdem der Einbau und die Steuerung von mit Infrarot arbeiteten Sensoren gelang, erwiesen sich diese als völlig untauglich für die ihnen zugedachte Aufgabe. Die Reichweite der Sensoren, bei der verlässliche und für die Verarbeitung geeignete Werte gemessen werden konnten, war wesentlich geringer als zur Entwicklung des Lösungsansatzes angenommen.
Kurz wurde noch über eine Ausführung des Plans ohne Unterstützung der Infrarotsensoren nachgedacht, dieses Unterfangen wurde nach der Erinnerung an die Fahrweise des Roboters jedoch schnell wieder verworfen.

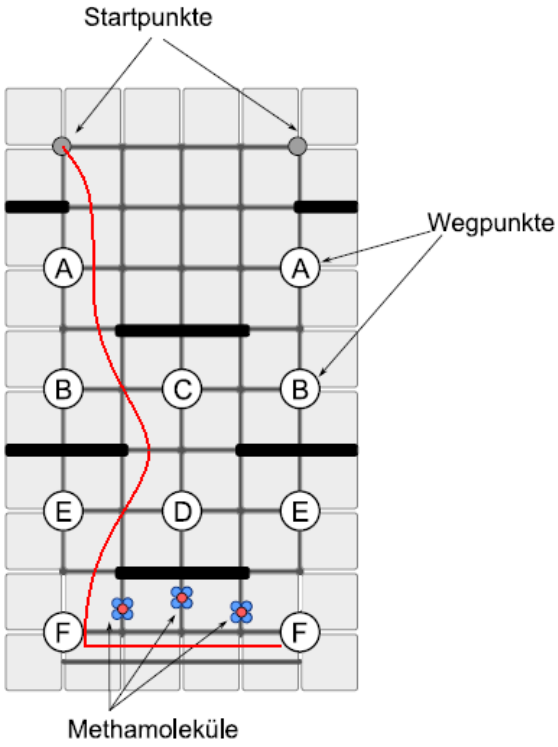


Abb. 3: kürzere Strecken zwischen den Orientierungspunkten

Lösungsansatz Nr. 4

Nach dem Scheitern der bisherigen Ansätze fand die Rückbesinnung auf das Linienfolgen statt. Der bereits vor dem Bau des Roboters als ineffizient betrachtete und nicht weiter verfolgte Gedanke schien die einzig verbleibende, in der restlichen Zeit realisierbare Möglichkeit zu sein, die unpräzise Fahrweise zu kompensieren.
Der zunächst provisorische Einbau der nötigen Sensoren und die Programmierung des Verfolgens der Linie ging für bisherige Verhältnisse schnell und führte zu mittlerweile unerwartetem Erfolg, vor allem in Bezug auf die Reproduzierbarkeit.
Das Abbiegen auf Kreuzungen stellte sich wiederum als Schwierigkeit heraus. Dem Roboter gelang es nicht, sich verlässlich genug nach der Drehung auf die Linie einzufädeln, dieses Problem konnte nur teilweise behoben worden.

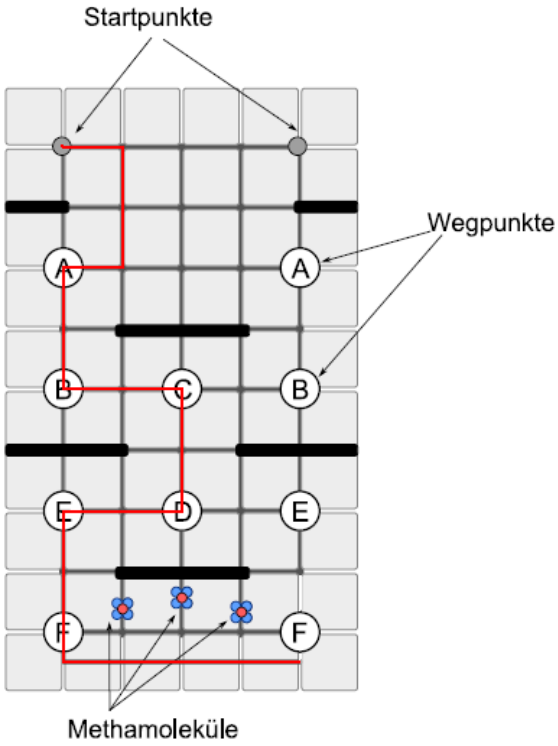


Abb. 4: folgen der dunklen Linien bis kurz vor dem Ziel

Für den Abschnitt ohne Führungslinie ab Wegpunkt E konnten die für das Fahren von Kurven, Folgen der Linie und Zählen der Querlinien verantwortlichen Programmteile modifiziert wiederverwendet werden. Der Roboter soll am Punkt E eine normale 90 Grad Kurve fahren, für die Steuerung wird allerdings statt der Frontsensoren der Hecksensor genutzt. Danach soll der Roboter möglichst gerade aus fahren und bei der Linie nach Wegpunkt F wieder eine Kurve fahren und der Linie bis zum Ende folgen. Das Erkennen der letzten Linie erfolgt durch zählen der überquerten Linien von Punkt E aus, bei der dritten Linie muss die Drehung erfolgen. Das Aufziehen auf die Linie und Folgen stellte sich aufgrund der ungenauen Position auf Höhe des Wegpunktes F als problematisch dar, deshalb wurde der folgen-Algorithmus angepasst.

2. Hardware

Sensorik

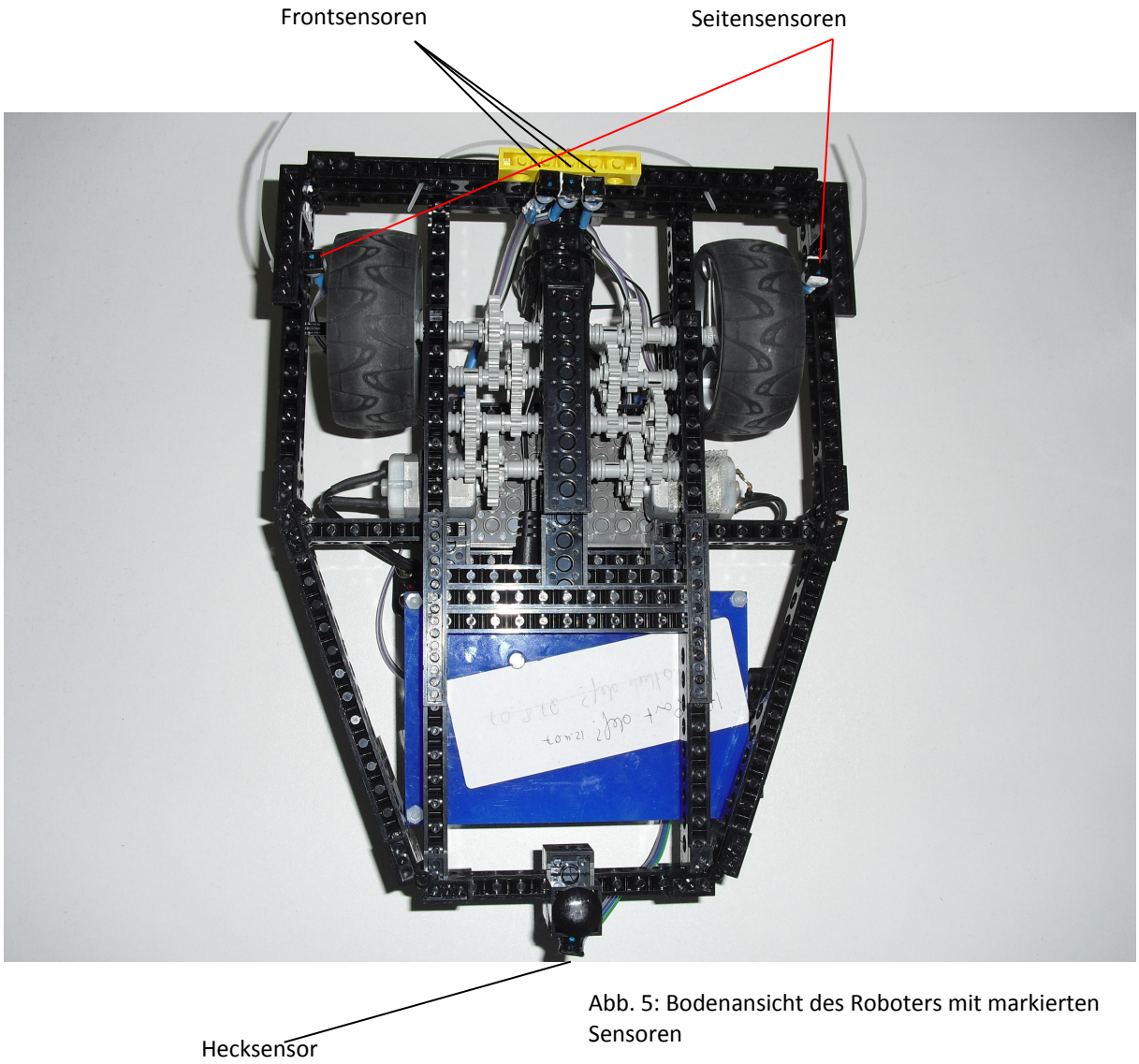


Abb. 5: Bodenansicht des Roboters mit markierten Sensoren

Der Roboter verwendet sechs Sensoren, es handelt sich dabei ausschließlich um Optokoppler. Diese Bauteile bestehen aus einem Sender und einem Empfänger und dienen der Übertragung von elektrischen Signalen zwischen zwei getrennten Stromkreisen. Genutzt werden sie bei diesem Roboter zur Helligkeitsunterscheidung des Bodens, so können die schwarzen Linien von dem hellen Boden unterschieden werden. Dies ermöglicht das Erkennen und Folgen der Linien.
Die drei Frontsensoren sind dabei für das Linienfolgen zuständig, sie sind so angeordnet, dass gerader Fahrt auf der Linie alle die dunkle Farbe erkennen. Wenn einer der beiden äußeren Sensoren einen Wert empfängt, der einen hellen Untergrund repräsentiert, kann entsprechend gegen gelenkt werden und die Fahrt auf der Linie fortgesetzt werden. Weiterhin werden die Frontsensoren bei den Kurvenfahrten benötigt, sie geben dabei Auskunft, wann die die Drehung beendet werden kann.

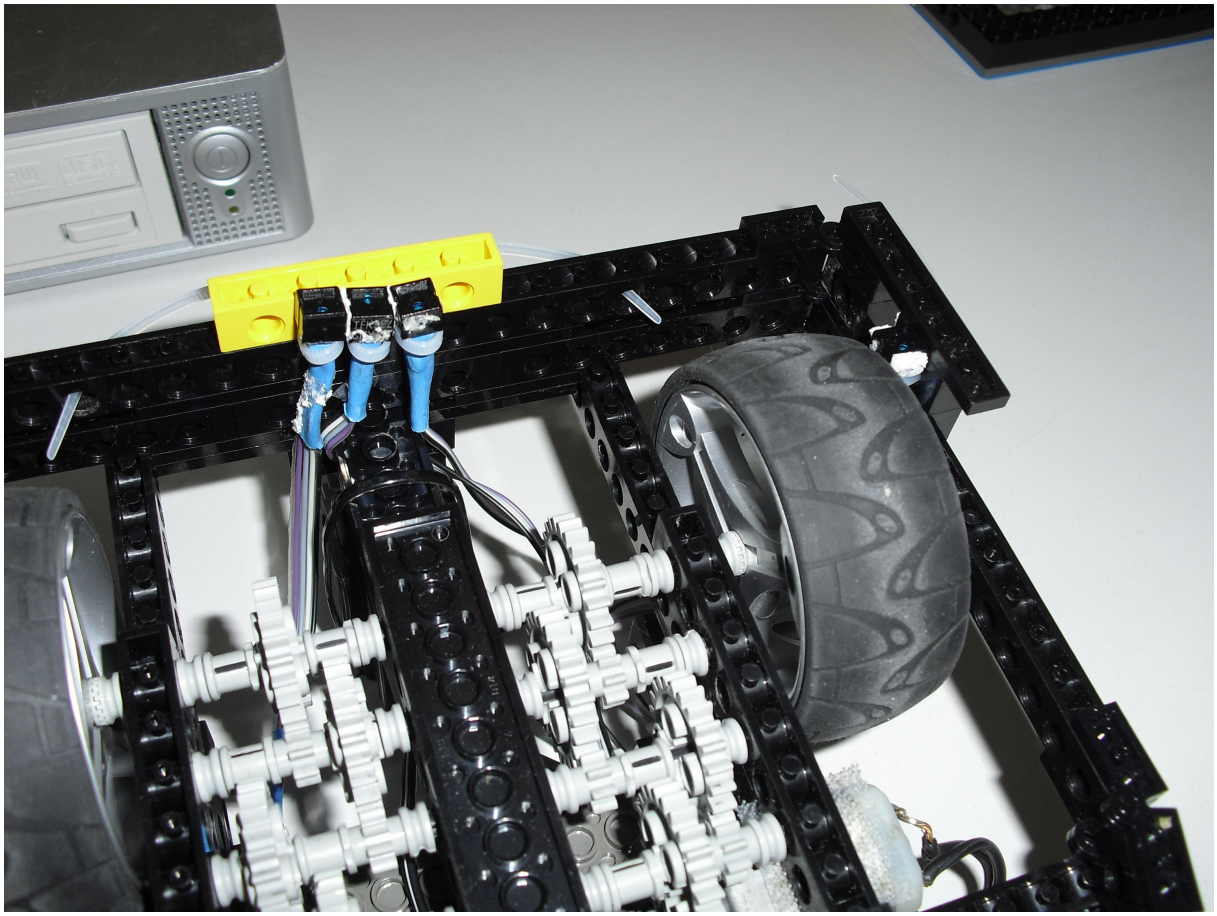


Abb. 6: Nahaufnahme der Frontsensoren und einem der Seitensensoren

Die seitlichen Sensoren werden zur Erkennung der quer zur Fahrtrichtung verlaufenden Linien genutzt. So können Kreuzungen erkannt und gezählt werden und entsprechend zum vorgegebenen Fahrplan eine Kurvenfahrt eingeleitet werden.
Der Hecksensor dient zur Erfassung des Startsignals. Der Startpunkt wird durch eine durchsichtige Scheibe markiert, die beleuchtet werden kann, der resultierende Helligkeitsunterschied wird als Start für das Rennen interpretiert. Der Sensor wird außerdem zur Fahrt der vorletzten Kurve (Punkt E) genutzt. An dieser Stelle endet die Führungslinie in Fahrtrichtung, sodass die Frontsensoren nicht zur Kurvenfahrt genutzt werden können.

Motoren und Getriebe

Der Roboter wird von zwei kleinen Elektromotoren angetrieben, die keine Legoteile sind und durch Ankleben von dünnen Puffern ober- und unterhalb der Motoren an die Größenverhältnisse der restlichen Bauteile angepasst wurden. Die Motoren besitzen eine hohe Drehzahl und müssen zum Antrieb der Räder übersetzt werden.
Das Getriebe besteht aus zwei Teilen, die spiegelverkehrt gebaut wurden und gegenüber an der Längsachse des Roboters angebracht sind, wie in Abbildung sechs gut sichtbar. Die Räder sind wie die Motoren zwischen Getriebe und Rahmen angebaut. Ein Getriebeteil setzt sich aus vier Achsen pro Motor zusammen, die jeweils die gleiche Übersetzungszahl haben. Dabei wird das Drehmoment jeweils von einem großen Zahnrad mit 24 Zähnen auf ein kleines

Zahnrad mit acht Zähnen übertragen, sodass sich insgesamt ein Übersetzungsverhältnis von 1 zu 81 (3^4) ergibt.

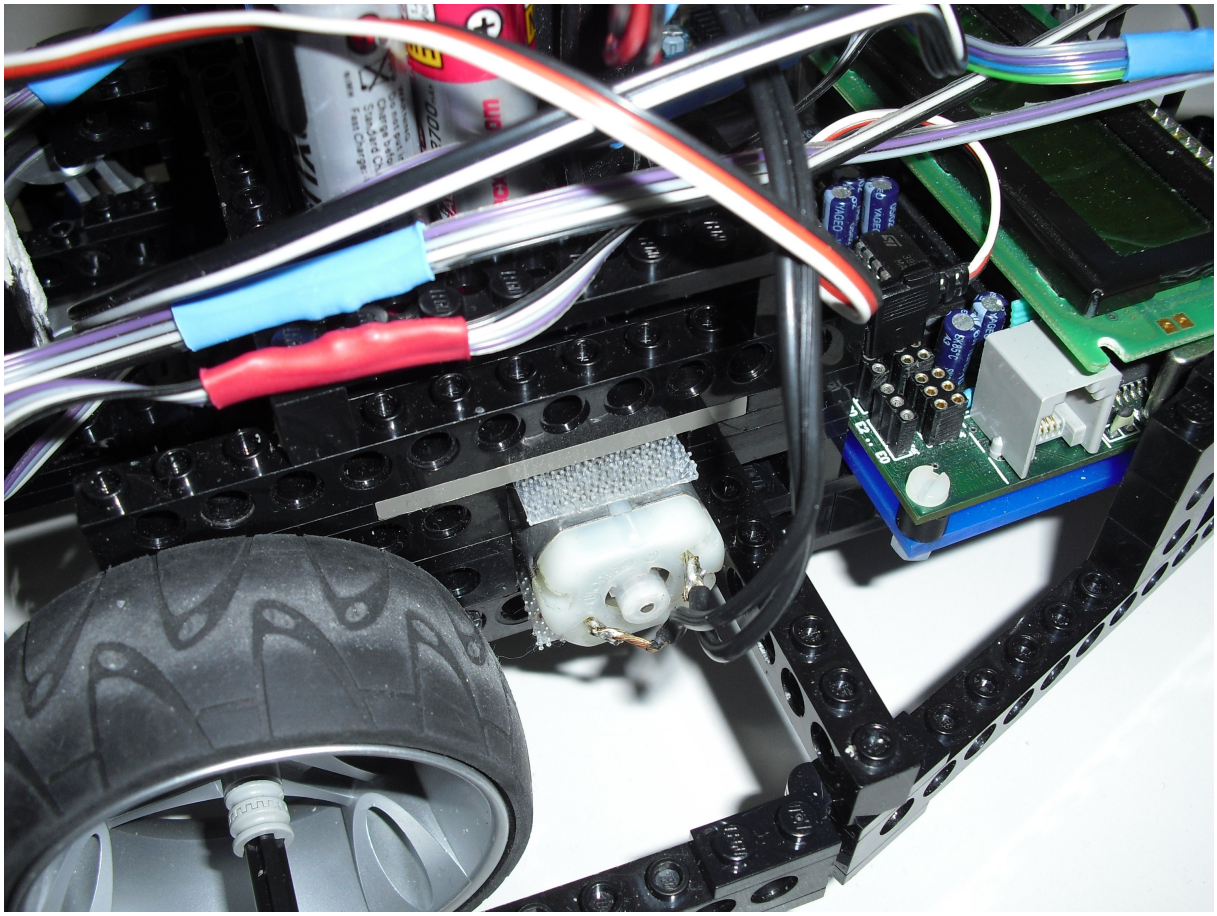


Abb. 7: Der gepufferte Motor und das Rad zwischen Rahmen und Getriebe

Damit sollte eine relativ schnelle Fahrt ermöglicht werden, ohne dass kleine Anstiege durch Unebenheiten der Fahrbahn zu einem Problem für den Roboter werden. Da in der Beschreibung der Lösungsansätze bereits mehrfach auf die unpräzise Fahrweise des Roboters eingegangen wurde, liegt die Folgerung nahe, dass die theoretischen Eigenschaften des Getriebes nur theoretischer Natur sind und sich nicht in der Praxis wiederfinden lassen.

Bei den absolvierten Fahrten zeigte sich neben der mangelnden Präzision immer eine starke Geräuschentwicklung, welche auf hohe Reibung im Getriebe schließen lassen. Dies äußert sich auch in dem hohen Energiebedarf des Roboters, wodurch die Abhängigkeit zum Akkustand vergrößert wird und auch leichte Steigungen zu einer starken Verlangsamung bis hin zu einem kompletten Stopp führen können. Die genauen Ursachen für dieses Verhalten sind nicht bekannt. Ein Grund wird bei der Anbringung der Motoren am Getriebe vermutet, wobei den Motoren wahrscheinlich zu viel Bewegungsraum ermöglicht wird. Eine stärkere Bindung führte bei Tests allerdings zu einer starken Verschlechterung des Fahrverhaltens.

Servomotor und Arm

Um die oberen Kugeln von den darunter liegenden herab zu stoßen, wird ein langer Plastikarm verwendet. Dieser wird während der Fahrt des Roboters auf der Zielgeraden quer

zur Fahrtrichtung ausgerichtet, sodass der Roboter nur gerade aus fahren muss, um die Moleküle zu spalten.

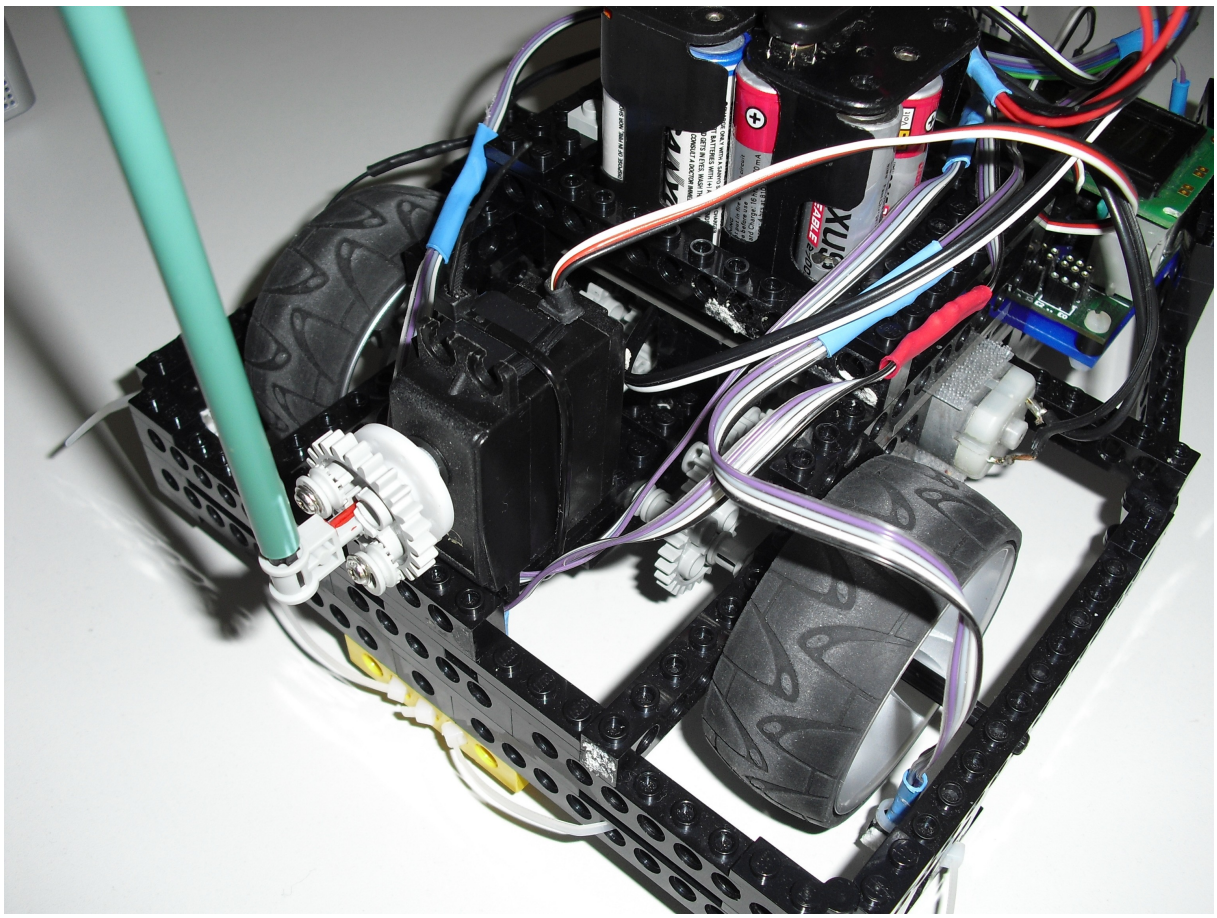


Abb. 8: Nahaufnahme des Servomotors mit befestigtem Arm

Der Arm kann dabei nicht bereits vor Beginn des Rennens ausgerichtet werden, da er sonst aufgrund seiner Länge an Hindernissen hängen bleiben würde. Die Ausrichtung erfolgt daher erst auf der Zielgeraden mittels eines Servomotors. Dieser schwenkt den Arm entsprechend zu Beginn des Rennens festgelegten Richtung, abhängig von der Startposition des Roboters. Aufgrund von Einschränkungen bei der Steuerung des Servomotors muss der Arm vor dem Rennen abhängig der Startpunktes auf den Servomotor aufgesteckt werden. Ein wählbarer Programmmodus kann den Arm vor dem Rennen entsprechend der eingestellten Startposition für die Fahrt senkrecht mittig ausrichten.

Fahrgestell

Grundlage für das Fahrgestell des Roboters ist Bausatz des Getriebes. Um das Getriebe herum wurde ein Rahmen konstruiert, der das System stabilisiert und den Einbau von Sensoren an nahezu jeder Position des Roboters ermöglichte. Gestützt wird der Rahmen von zwei Längsachsen, die den vordersten Teil des Rahmens durchgehend mit hintersten Abschnitt verbinden. Für zusätzlichen Halt sorgen zwei Bauteile, die mit dem Getriebe eine darauf aufliegende Querachse bilden.

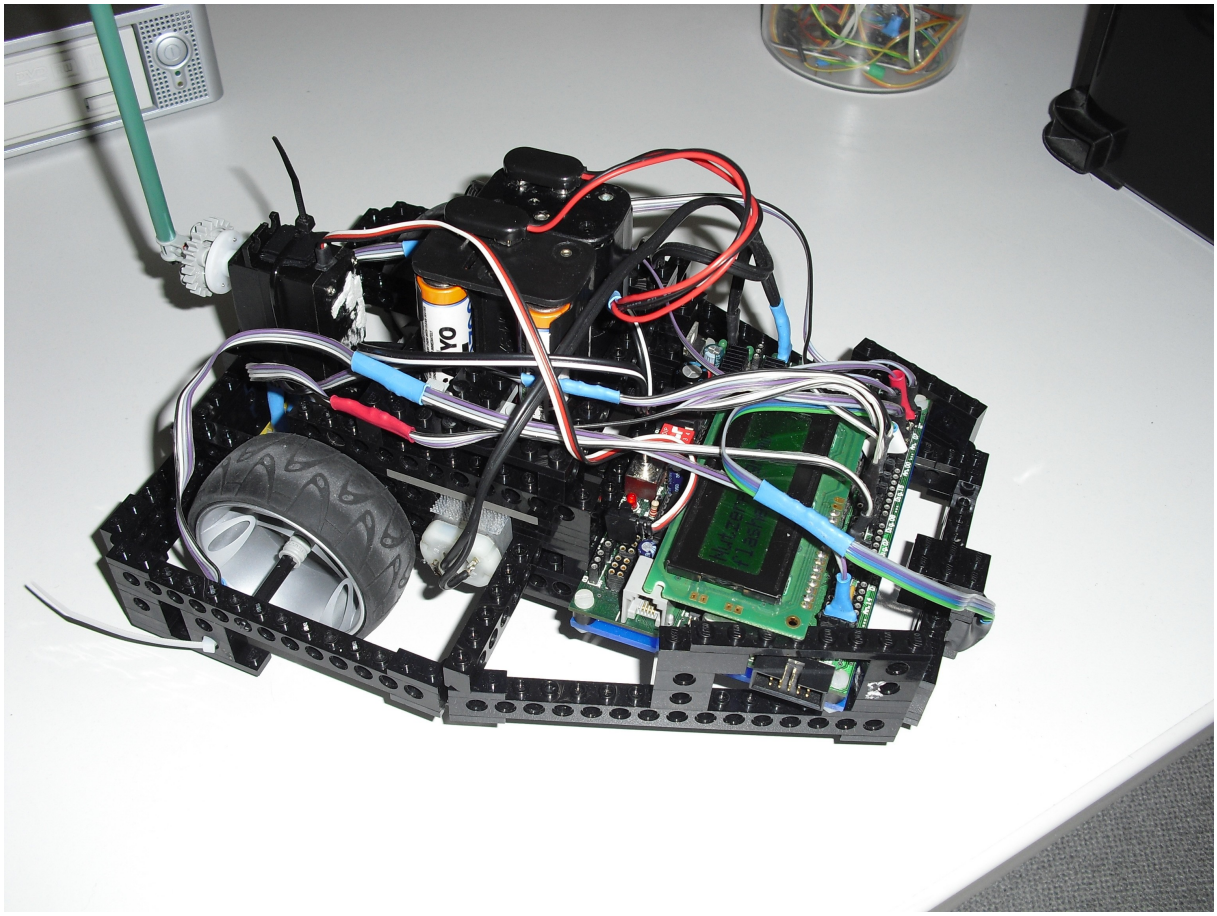


Abb. 9: Seitenansicht des Roboters

Die von dem Rahmen eingeschlossene Oberfläche wurde relativ groß gewählt, sodass sich die nachträgliche Montage von großen Bauteilen wie dem AKSEN-Board, dem Akkumulator und dem Servomotor einfach gestalten ließ und Änderungen an diesen Bauteilen schnell und ohne viel Demontagearbeiten möglich wurden. So ließ sich die benötigte Umstellung auf zwei Akkumulatoren kurz vor Ende des Projektes schnell und unkompliziert durchführen.

3. Software

Die Codestruktur der Dancing Queen ist so angelegt, dass eine Hauptfunktion (AksenMain) das Grundsätzliche Verhalten steuert, indem eine vorgegebene Route abgefahren wird und dabei entsprechende Unterprogramme gestartet werden, welche den Roboter ans Ziel führen sollen.
Es wird nun im Einzelnen auf die verschiedenen Funktionen mit ihren Funktionalitäten eingegangen.

AksenMain(void)
In der Hauptfunktion des Programmes werden als erstes benötigte Parameter und Variablen deklariert, mittels denen eine Route vorgegeben wird und die Position auf dieser Route abgebildet werden kann. Je nach DIP-Schalter Belegung wird die Route entsprechend der Startposition geändert oder auch der Stoßarm ausgerichtet. Auch grundlegende Eigenschaften wie das Starten bei Licht und das Stoppen nach einer bestimmten Zeit wird in der AksenMain implementiert.
Nach dem Startsignal wird die Funktion *followLine(int)* ausgeführt um der Linie zu folgen. Sobald ein Sensor meldet das eine neue Kreuzung gefunden ist, werden entsprechende Parameter wie *nextTurn* aktualisiert und anhand einer If-Else Struktur ausgewählt welches Unterprogramm ausgeführt werden soll. Dazu wird die Nummer der Kreuzung überprüft und mit dem Inhalt des *turns[]* Arrays verglichen. Bei einer gefundenen Übereinstimmung wird entsprechend reagiert: bei einer 0 oder 2 wird *turn(int)* mit der Kenn-Nummer der gerade befahrenen Kreuzung aufgerufen. Ergibt die Übereinstimmung eine 1 startet das Hauptprogramm die Funktion *forward (void)* und bei einer 3 oder 5 wird *forwardGoal(int)* mit der Kenn-Nummer aufgerufen. Am Ende wird immer die *stop(void)* Funktion aufgerufen.

followLine(int speed)
Anhand von den seitlichen in der Mitte befindlichen Optokopplern wird erkannt wann der Roboter droht von der Linie abzukommen und dann mittels einer If-Else Struktur die Motoren so gesteuert werden das sich der Roboter wieder auf die Linie ‚raufziehen' kann.

stop(void)
Diese Funktion stoppt die Motoren und startet eine, nicht durch das Programm stopbare, Endlosschleife.

turn(int direction)
Anhand der übergebenen Kenn-Nummer wird die Richtung bestimmt in die sich die Motoren drehen müssen. Die Motoren drehen sich gegensätzlich zueinander wodurch eine Drehung um die eigene Achse ermöglicht wird. Mithilfe der Sensoren wird erkannt wann der mittlere Sensor das Weiß des Spielfeldes erkennt, der Roboter dreht sich bis zu dem Zeitpunkt weiter bis der mittlere Sensor und der seitliche Sensor in Abbiegerichtung beide eine schwarze Linie erkannt haben. Die Richtung der Motoren wird zum Schluss wieder synchronisiert.

lastTurn(int direction)
Wie bei *turn(int)* wird auch hier die Richtung der Motoren bestimmt und eine Drehung um die eigene Achse erwirkt. Der Roboter dreht sich solange bis der hintere Sensor eine

schwarze Linie erkennt und fährt dann mit wieder synchronisierten Motoren ‚blind' geradeaus. Es wird mithilfe der vorher definierten Variable *offSite* nachdem auf welcher Seite des Spielfeldes man sich befindet der linke oder rechte Sensor überprüft und kontrolliert so wann der Roboter die dritte Linie, also die letzte Linie des Spielfeldes, erkannt hat. Der Roboter dreht sich in diesem Moment in die entsprechende Richtung bis der mittlere Sensor und der Sensor in Drehrichtung schwarz erkannt haben. Die Motoren werden wieder auf die gleiche Drehrichtung gesetzt.

forwardGoal(int direction)
Es werden zwei Variablen deklariert, zum einen *counterEnd* und *timeEnd*. Die Variable *counterEnd* dient dazu eine bestimmte Zeit zu warten bis alle drei vorderen Sensoren die schwarze Linie gefunden haben. Ist die Linie so erstmal gefunden wird der Stoßarm mittels des Servomotors entsprechend *direction,* also der Startposition, so ausgerichtet, dass die Bälle beim Vorbeifahren umgestoßen werden können. Es wird der Linie mittels *followLine(int)* gefolgt bis eine bestimmte Zeit abgelaufen ist, diese wird mittels *timeEnd* bestimmt, und dann mittels *stop(void)* gestoppt.

forward(void)
Die Motoren werden auf Maximalgeschwindigkeit gesetzt und dann eine sehr kurze Zeit ‚blind' geradeaus gefahren.

dance(int direction)
Lässt den Roboter durch eine Vielzahl von ‚blinden' Bewegungsabläufen ‚tanzen'. Diese Funktion ist eher ein Gimmick und dient nicht dem erreichen des Ziels sondern nur zur Unterhaltung der Anwender wodurch die Funktion nicht näher erläutert wird.

Es wurde versucht die einzelnen Funktionen flexibel anwendbar zu machen, was bedeutet, dass die gleichen Funktionen für beide Startpositionen benutzt werden können und durch einen jeweiligen *direction* Parameter, welcher die Identifikationsnummern der betroffenen Sensoren widerspiegelt, individuell handeln.

4. Quellenangabe

Fachbereich Informatik und Medien: *Nutzerhandbuch AKSEN-Board*, Version 1.04 zur AKSEN-Bibliothek 0.965, Fachhochschule Brandenburg, 2005

5. Anhang

Vollständiger Quellcode

```
#include <stdio.h>
#include <regc515c.h>
#include <stub.h>

/*
      ********************************************

      Dancing Queen - where frustration and entertainment can't be closer
      together

      ********************************************

      analog0 = startsensor

      analog2 = linie folgen rechts
      analog3 = linie folgen mitte
      analog4 = linie folgen links

      analog7 = oktokoppler rechts
      analog8 = oktokoppler links

      motor0 = rechts
      motor1 = servo
      motor2 = links

      ---SERVO---
      Start von Links: 55° einstellen, rote Nase oben (rot)
      118° Schwenk

      Start von Rechts: 115° einstellen, rote Nase oben (lila)
      51° Schwenk

      ********************************************
*/

void followLine(int speed) {
      if (analog(2) < 100) {
            motor_pwm(2,speed);
      } else {
            motor_pwm(2,7);
      }

      if (analog(4) < 100) {
            motor_pwm(0,speed);
      } else {
            motor_pwm(0,7);
      }
}

void stop(void) {
      motor_pwm(2,0);
      motor_pwm(0,0);
      while(1);
}
```

```
//direction: 0 = rechts ; 2 = links
void turn(int direction) {
	motor_richtung(direction,1);
	motor_pwm(0,8);
	motor_pwm(2,8);

	while(analog(3) > 40) { // solange mitte schwarz

	}

	sleep(20); //mitte sieht weiß
	while(analog(3) < 180 || analog(direction + 2) < 180) { //solange
	Sensoren Mitte und in Abbiegerichtung nicht schwarz sehen

	}
	motor_richtung(direction,0);

}

void lastTurn(int direction) {
	int offSite;

	motor_richtung(direction,1);
	motor_pwm(0,7);
	motor_pwm(2,7);

	if (direction) {
		offSite = 8;
	} else {
		offSite = 7;
	}

	//drehen
	sleep(200);
	while(analog(0) < 180) {

	}
	motor_richtung(direction,0);
	motor_pwm(0,8);
	motor_pwm(2,8);
	sleep(200);

	while(analog(offSite) < 180) {

	}

	lcd_cls();
	lcd_puts("erste Linie");
	sleep(100);
	//erste Linie überquert
	while(analog(offSite) < 180) {

	}
	lcd_cls();
	lcd_puts("zweite Linie");
	sleep(100);

	//zweite Linie erkannt
	while(analog(offSite) < 180) {
```

```
	}
	lcd_cls();
	lcd_puts("dritte Linie");

	motor_richtung(direction,1);
	motor_pwm(0,6);
	motor_pwm(2,6);
	while(analog(3) > 40) { // solange mitte schwarz

	}
	sleep(20); //mitte sieht weiß
	while(analog(3) < 180 || analog(direction + 2) < 180) { //solange
	Sensoren Mitte und in Abbiegerichtung nicht schwarz sehen

	}

	motor_richtung(direction,0);
}

void forwardGoal(int direction) {
	int counterEnd = 0;
	long int timeEnd = akt_time() + 5000;

	motor_pwm(0,8);
	motor_pwm(2,8);

	while (counterEnd < 300) {
		followLine(1);

		if (analog(2) > 40 && analog(3) > 40 && analog(4) > 40) {
			counterEnd++;
		}
	}

	//Arm in Stellung bringen
	if (direction) {
		servo_arc(1,51);
	} else {
		servo_arc(1,118);
	}

	//Moleküle spalten
	while(akt_time() < timeEnd) { //nach 5 Sekunden stoppen
		followLine(3);
	}
	stop();
}

void forward(void) {
	motor_pwm(0,10);
	motor_pwm(2,10);
	sleep(100);
}

void dance(int direction) {
	//zur Mitte fahren
	motor_pwm(2,8);
	motor_pwm(0,8);
	sleep(800);

	//um eigene Achse drehen
	motor_pwm(0,0);
```

```
sleep(1000);

//stop
motor_pwm(2,0);
motor_pwm(0,0);
sleep(300);

// und zurück drehen
motor_pwm(2,7);
motor_richtung(2,1);
sleep(1000);
motor_richtung(2,0);

//stoppen, Arm schwenken
motor_pwm(2,0);
if (!direction) {
      servo_arc(1,70);
      sleep(400);
      servo_arc(1,160);
      sleep(400);
      servo_arc(1,115);
      sleep(400);
} else {
      servo_arc(1,100);
      sleep(400);
      servo_arc(1,10);
      sleep(400);
      servo_arc(1,55);
      sleep(400);
}

//um eigene Achse und zurück drehen, andere Richtung
motor_pwm(0,7);
motor_pwm(2,0);
sleep(800);
motor_richtung(0,1);
sleep(800);
motor_richtung(0,0);

//auf Stelle drehen
motor_richtung(2,1);
motor_pwm(0,7);
motor_pwm(2,7);
sleep(1500);
motor_richtung(2,0);

//stop
motor_pwm(2,0);
motor_pwm(0,0);
sleep(300);
//motor_pwm(2,7);

//leichte Kurve und zurück fahren
motor_pwm(2,7);
motor_pwm(0,4);
sleep(1500);
motor_richtung(0,1);
motor_richtung(2,1);
sleep(1500);
motor_richtung(0,0);
motor_richtung(2,0);
```

```
	//stop
	motor_pwm(2,0);
	motor_pwm(0,0);
	sleep(300);

	//nochmal, andere Seite
	motor_pwm(0,7);
	motor_pwm(2,4);
	sleep(1500);
	motor_richtung(0,1);
	motor_richtung(2,1);
	motor_pwm(2,7);
	motor_pwm(0,4);
	sleep(1500);
	motor_richtung(0,0);
	motor_richtung(2,0);

	//stoppen, Arm schwenken
	motor_pwm(2,0);
	if (!direction) {
		servo_arc(1,70);
		sleep(400);
		servo_arc(1,160);
		sleep(400);
		servo_arc(1,115);
		sleep(400);
	} else {
		servo_arc(1,100);
		sleep(400);
		servo_arc(1,10);
		sleep(400);
		servo_arc(1,55);
		sleep(400);
	}

	//auf Stelle drehen
	motor_richtung(2,1);
	motor_pwm(0,10);
	motor_pwm(2,10);
	sleep(3000);

	stop();
}

void AksenMain(void) {

	int counterLine = 0;
	int turns[12] = {2, 1, 2, 0, 1, 0, 1, 2, 1, 2, 1, 3}; //initialisiert
	fuer linke Startposition
	int nextTurn;
	int startPos = dip_pin(0); //Startpunkt, 0 = rechts, 1 = links
	long int timeTurn = 0;
	long int timeEnd = 0;

	//Arm mittig ausrichten
	if (dip_pin(3) == 1) {
		if (!startPos) {
			servo_arc(1, 115);
		} else {
			servo_arc(1, 55);
		}
```

```
            stop();
      }

      motor_richtung(2,0);
      motor_richtung(0,0);

      //Fahrtweg festlegen, 0 = rechts, 1 = gerade aus, 2 = links, 3 =
      letzte links
      if (!startPos) {
            //turns = {0, 1, 0, 2, 1, 2, 1, 0, 1, 0, 1, 3};
            turns[0] = 0;
            turns[2] = 0;
            turns[3] = 2;
            turns[5] = 2;
            turns[7] = 0;
            turns[9] = 0;
            turns[11] = 5;
      }

      //nicht starten solange Licht aus ist
      while(analog(0) > 100) {

      }
      timeEnd = akt_time() + 120000;

      if (!dip_pin(1)) {
            dance(startPos);
      }

      while(timeEnd > akt_time()) { //nach 120 Sekunden stoppen
            followLine(3);

            if ((analog(7) > 180 || analog(8) > 180) && timeTurn <
                  akt_time()) {
                  nextTurn = turns[counterLine];

                  if (nextTurn == 0 || nextTurn == 2) {
                        turn(nextTurn);
                  } else if (nextTurn == 1) {
                        forward();
                  } else if (nextTurn == 3 || nextTurn == 5) {
                        lastTurn(nextTurn - 3);
                        forwardGoal(nextTurn - 3);
                  }

                  timeTurn = akt_time() + 200;
                  counterLine++;
            }
      }

      //Roboter stoppen
      stop();
}
```