Fachhochschule Brandenburg
University of Applied Sciences

Autonome Mobile Systeme - Projekt

MASDAR CITY - Personal Rapid Transit

Dokumentation:

Der Bussomat

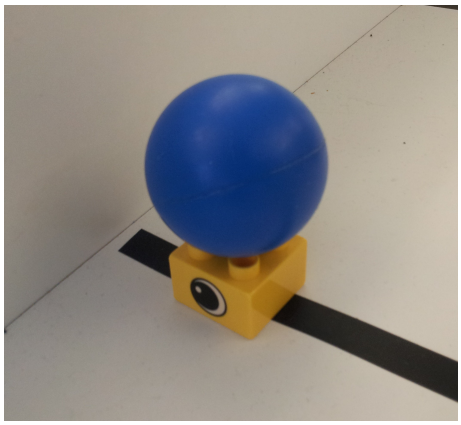
Wintersemester 2013/2014

Benker, Christian 20110027
Paleit, Dario 20110055

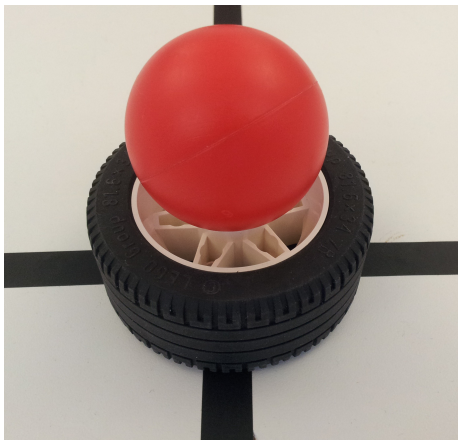
Aufgabe.....3
Lösungsweg.....4
Vorstellung.....5
Hardware.....6
Software.....7
Fazit.....9
Quellcode.....10

Aufgabe

Im Rahmen des Projektes sollte selbstständig ein Roboter entwickelt werden, welcher auf einer gitternetzartigen Straßenkarte entlangfahren und „Personen“ aufsammeln kann. Dazu liest der Roboter zuerst die aktuelle Karte ein und berechnet einen Weg zu den „Personen“. Gesperrte Kreuzungen dürfen dabei nicht befahren werden. Danach werden die „Personen“ selbstständig vom Roboter aufgelesen und zum Zielort befördert.
Um die Aufgabe lösen zu können, standen jedem Team Lego-Bausteine verschiedenster Art, ein AKSEN-Board, Motoren, ein Akkupack und verschiedene Sensoren zur Verfügung.



Die blauen Kugeln stellen „Personen“ dar.
Der gelbe Stein hält die Kugel an ihrem Platz.



So sieht eine gesperrte Kreuzung aus.
Diese darf vom Roboter nicht befahren werden.

Lösungsweg

Der Roboter sollte einfach und funktional aufgebaut sein, da es sehr wahrscheinlich war, dass im laufe des Projekts Änderungen am Roboter vorzunehmen sein würden. Deswegen wurden die Bestandteile des Roboters einzeln entwickelt und angebracht, ohne vorher einen Gesamtplan erstellt zu haben. Durch diese flexible Bauweise ist es z.B. möglich, die Sensoren, die für die Spurhaltung notwendig sind, problemlos etwas weiter vor bzw. zurück zu setzen.

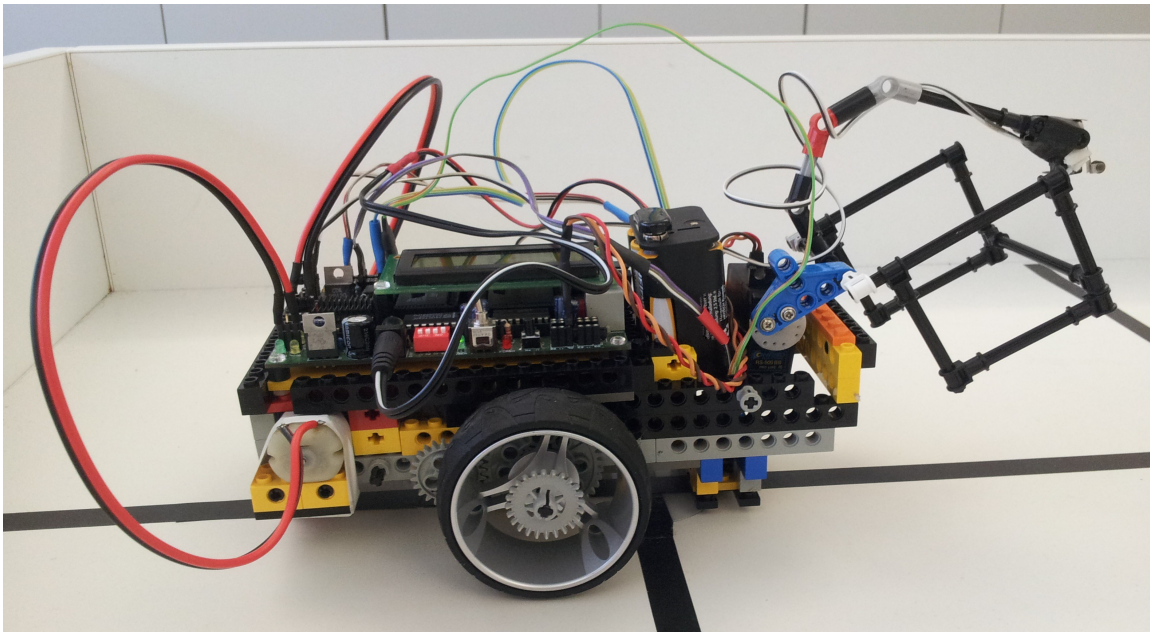
Den Weg zu einer „Person“ zu berechnen, stellte eine besondere Herausforderung dar. Wir haben uns dazu entschieden, die Karte zu „fluten“. Das heißt, alle erreichbaren Kreuzungen, ausgehend von der Startposition des Roboters, mit Zahlen zu versehen. Diese Zahlen geben dabei die minimale Anzahl an zu überfahrenen Kreuzungen an, bis man sich wieder an der Startposition befindet. Mit dieser Methode lässt sich erstens feststellen, ob eine „Person“ überhaupt erreichbar ist und zweitens ein optimaler Weg bestimmen.

Ursprünglich sollte unser Roboter nur mit den 2 Optokopplern, die am vorderen Bereich angebracht sind, fahren können. Bei einer Kurvenfahrt sollte dazu eines der Räder stehen bleiben, bis ein Optokoppler wieder eine Spur erkennt. Das funktionierte zwar, war aber äußerst fehleranfällig bei zu stark oder zu schwach geladenem Akku. Außerdem gestaltete sich das aufnehmen und absetzen von „Personen“ besonders schwierig. Darum wurden auf Höhe der Radachse 2 weiter Optokoppler angebracht. Mit deren Hilfe konnte sich der Roboter nun auf der Stelle drehen. Das funktionierte so gut, dass es für alle Drehungen übernommen wurde.

Das Aufnehmen einer „Person“ zu realisieren ist keine leichte Aufgabe gewesen. Wir haben dazu über Lösungswege nachgedacht, wie z.B. mit Hilfe von Infrarotsensoren eine Lichtschranke zu bauen oder die Entfernung des Roboters zur „Person“ zu ermitteln. Im Laufe unserer Überlegungen bemerkten wir, dass jede „Person“ immer direkt vor einer Wand steht. Das brachte uns auf eine Lösung, die unserem Anspruch auf eine simple Bauweise gerecht wurde. Ein Art Käfig am vorderen Teil des Roboters, der durch einen Servomotor gehoben und gesenkt werden kann, soll die „Person“ aufnehmen. Dabei befindet sich der Käfig in gesenktem Zustand tief genug, um den Ball auch während der Fahrt festzuhalten und hoch genug, um den Stein, auf dem der Ball liegt, nicht mitzuschleifen. An der Spitze befindet sich ein einfacher Drucksensor, der betätigt wird, wenn der Roboter mit gehobenen Käfig gegen die Wand fährt.

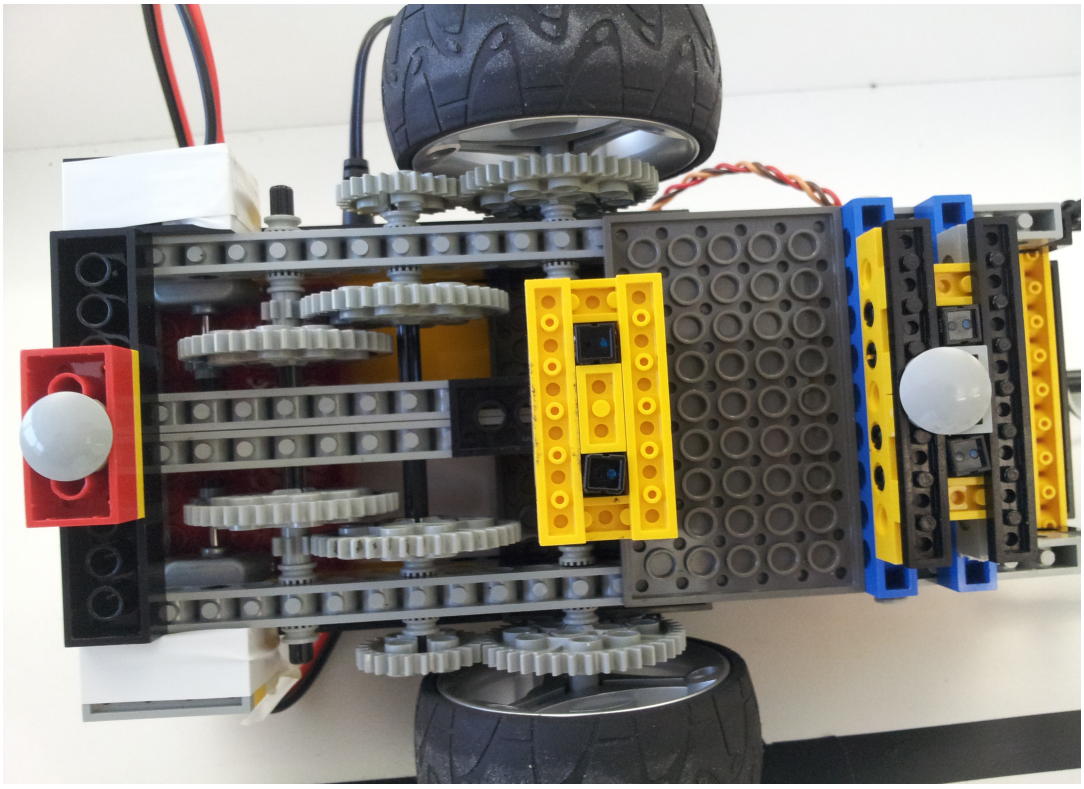
Vorstellung

Der Bussomat

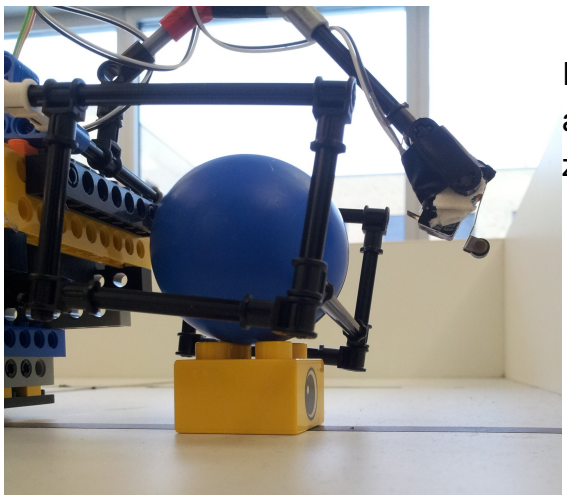


Kompliziert ist nicht gleich besser. Die Greifvorrichtung des Bussomaten ist ein gutes Beispiel dafür, wie mit einfachen Mitteln eine effektive Lösung gefunden werden kann. Dieser Gedanke stand während der gesamten Entwicklung des Roboters im Vordergrund. Auch wenn er etwas wüst aussieht, so hat er den Vorteil, dass er sehr leicht und schnell in seine Einzelbestandteile zerlegt werden kann, wenn z.B. ein Sensor falsche Werte liefert, weil er beschädigt worden ist oder die Übersetzung des Getriebes geändert werden muss. Der Bussomat lässt sich leicht aus- und umbauen. Er sollte mit möglichst wenigen Sensoren auskommen und trotzdem gute Ergebnisse liefern. Weniger Sensoren bedeutet aber leider auch weniger Sicherheit bei der Streckenverfolgung. So kann ein zu stark geladener Akku leider schon mal zum Verlust der Orientierung führen, weil Drehungen zu schnell ausgeführt werden und der Bussomat über das eigentliche Ziel hinausschießt.

Hardware



Wie auf dem Bild oben zu sehen ist, wird unser Roboter von zwei Elektromotoren angetrieben, die jeweils für eines der beiden Räder zuständig sind. Das Getriebe sorgt dafür, dass die für den Roboter viel zu hohe Drehzahl der Motoren auf eine angemessene Geschwindigkeit herabgesetzt wird. Durch diese Antriebsart ist der Roboter sehr wendig. Er kann auf der Stelle drehen. Es ist wichtig, darauf zu achten, dass die Motoren richtig herum am AKSEN-Board angeschlossen sind, da sich die Räder sonst nicht in die gewünschte Richtung drehen.
Des weiteren sind auf dem Bild noch die vier verbauten Optokoppler zu sehen, mit deren Hilfe der Roboter sieht.



Im Bild links sieht man den Käfig zum aufnehmen des Balls und den Druckschalter zum ertasten einer Wand.

Software

Um den Weg zu berechnen, werden in der Main-Methode die drei Methoden
fillArray()
erkenneZiele()
berechneWeg()
nacheinander ausgeführt.

Die Methode fillArray() liest zuerst den DIP-Schalter aus und ermittelt so, ob sich der Roboter auf Startposition A oder B befindet. Die aktuelle Karte ist in dem Array _fa[] abgespeichert und enthält nur die Zahlen: 46 = '.' , 70 = 'F' und 120 = 'x'.
Beim fluten einer Karte dieser Größe wird die Zahl 46 nicht erreicht, darum wird dieses Array direkt verwendet und geflutet, um später daraus alle erreichbaren „Personen“ sowie die kürzesten Wege zu ihnen zu ermitteln.

Die Methode erkenneZiele() geht nun alle Ziele durch, also alle Kreuzungen, die im Array _fa[] den Wert 70 = 'F' haben. Hat eine Nachbarkreuzung eines Ziels einen Wert kleiner als 46, so ist dieses Ziel für den Roboter erreichbar. Alle erreichbaren Ziele werden in das Array ziele[] geschrieben.

Die Methode berechneWeg() berechnet zuerst den Weg, den der Roboter zurücklegt, nachdem er gerade eine „Person“ aufgenommen hat. Er befindet sich zu diesem Zeitpunkt an der letzten Kreuzung vor dem Ziel mit Blickrichtung zum Ziel. Nachdem also dieser Rückweg berechnet wurde, wird aus ihm der Hinweg errechnet. Der Hinweg wird dann gefolgt vom Rückweg in das Array route[] eingetragen. Dies geschieht für jede der erreichbaren „Personen“.

Die Route besteht nun an sich aus einzelnen Zeichen, diese werden dann in der Main-Methode nacheinander zu echten Anweisungen decodiert.

Wenn der Roboter geradeaus fährt, nutzt er die vorderen zwei Optokoppler, um auf der Spur zu bleiben. Wenn nur einer der beiden Optokoppler zu einem Zeitpunkt schwarz erkennt, dann wird der Motor, der sich auf dieser Seite befindet, verlangsamt, bis keiner der beiden vorderen Optokoppler mehr schwarz erkennt und der Motor wieder beschleunigen kann. Oder aber beide Optokoppler erkennen schwarz, dann nimmt der Roboter an, eine Kreuzung erkennen zu können. Die Methode gerade() endet mit dem Erkennen einer Kreuzung ohne die Motoren zu stoppen. So bleibt der Roboter in Fahrt, wenn er mehrmals nacheinander geradeaus fahren muss.

Will der Roboter eine Kurve fahren, so fährt er erst mit etwas langsamerer Geschwindigkeit weiter geradeaus, bis die Optokoppler auf Höhe der Radachse eine Kreuzung erkennen. Befindet sich der Roboter schon auf einer Kreuzung, so erkennen dies die Sensoren und das geradeaus Fahren wird direkt übersprungen. Zum Drehen bewegen sich die beiden Räder mit gleicher Geschwindigkeit in entgegengesetzter Richtung. Der Roboter dreht also auf der Stelle, bis einer der vorderen Optokoppler schwarz wahrnimmt, er sich also auf gewünschter Spur befindet. Bei der 180 Grad Drehung wird dies zweimal nacheinander getan.

Fazit

Das Projekt war sehr interessant und hat uns viel Spaß bereitet.
Den Roboter zu entwickeln und Lösungen für die mitunter immer wieder neu auftauchenden Probleme zu suchen, war anspruchsvoll und lehrreich.

Am Ende war der Bussomat doch noch etwas zu sehr vom Ladestand des Akkus beeinflusst, was bei zu stark oder zu schwach geladenem Akku schnell zu einem Fehlverhalten führte.
Aber er hat gezeigt, dass er in der Lage ist, seine Aufgabe schnell und sicher zu erledigen, worauf wir als Entwickler sehr stolz sind.

Quellcode

```
/*bussomat.h*/
//Autor: Paleit Dario - Benker Christian

//Weg berechnen
void fillArray();
void fill(char, char);
void erkenneZiele();
void zieleLinks();
void zieleRechts();
char erreichbar(char);

void berechneWeg();
void berechneNaechsteKreuzung();
void rWegEintragen();
void beschreibeRoute();

//fahren
void gerade();
void ranfahren();
void d180();
void links();
void rechts();
void greifen();
void ablegen();
```

```c
/*bussomat.c*/
//Autor: Paleit Dario - Benker Christian

//Diese Include-Datei macht alle Funktionen der
//AkSen-Bibliothek bekannt.
//Unbedingt einbinden!
#include <stub.h>

// ****************************
// Einlesen eines Fahrauftrages:
// Ändern des #define auf den gewünschten Fahrauftrag, bspw. '#define FA4'
//oder '#define FA12'
// -> der Fahrauftrag befindet sich in der Variablen _fa vom Typ 'unsigned char[71]'
// -> die Nummer des Fahrauftrags, bspw. 4 oder 12, befindet sich in der Variablen
//_fa_nr vom Typ 'unsigned char'
// und sollte zur Kontrolle angezeigt werden - Ausgabe bspw. mit 'lcd_ubyte(_fa_nr);'
#define FA23
#include "fa.h"
#include "bussomat.h"
// ****************************

//Variablen zum Fahren
#define BLACK 50
#define SLOW 3
#define FAST 8
#define DREH 3

//Laufvariablen
int x = 0;
int y = 0;
int z = 0;

//weitere Variablen und Arrays
char startpunkt = 64;
char kreuzung = 0;
char fertig = 0;

char ziele[4];//erreichbare Ziele
char rWeg[50];//Rückweg von Ziel- bis Startposition
char route[400];//komplette Route, die gefahren werden muss

char aktRichtung;
char aktKreuzung;
char naechsteRichtung;
char naechsteKreuzung;



void fillArray(){

	//lese DIP-Schalter aus und lege Startposition fest
	if(dip_pin(0) == 0){
		startpunkt = 68;
	}

	//Startpunkt im Array mit 0 belegen
	_fa[startpunkt] = 0;

	//Karte fluten
	//x - gibt die aktuelle Position im Array an
	//y - wird benutzt, um das Array zu fluten beginnend mit 0
	//fertig - wenn diese Variable nicht auf 1 gesetzt wird, ist Flutung beendet
	do{
		fertig = 0;
```

```
		for(x = 0; x < 70; x++){
			//durchsuche Array nach Kreuzungen mit der aktuellen Flutungszahl
			if(_fa[x] == y){
				fill(x, y);//gehe alle Nachbarkreuzungen durch und versuche
						//sie zu fluten
				fertig = 1;
			}
		}
		y++;
	}while(fertig == 1);
}

//prüfe zuerst, ob eine Nachbarkreuzung existiert (Rand der Karte)
//46 ist eine passierbare, noch nicht geflutete Kreuzung
void fill(char x, char y){
	y++;
	//north
	if(x - 7 >= 0){
		if(_fa[x - 7] == 46){
			_fa[x - 7] = y;
		}
	}
	//south
	if(x + 7 < 70){
		if(_fa[x + 7] == 46){
			_fa[x + 7] = y;
		}
	}
	//west
	if(x % 7 > 0){
		if(_fa[x - 1] == 46){
			_fa[x - 1] = y;
		}
	}
	//east
	if(x % 7 < 6){
		if(_fa[x + 1] == 46){
			_fa[x + 1] = y;
		}
	}
}

//schreibe erreichbare Zielpunkte in Array
//x - Position im Array _fa[]
//y - Position im Array ziele[]
//70 ist der Wert eines Ziels im Array _fa[]
void erkenneZiele(){

	y = 0;
	for(x = 1; x < 6; x++){
		if(_fa[x] == 70){
			if(erreichbar(x)){
				ziele[y] = x;
				y++;
			}
		}
	}
	//Ziele werden später in umgekehrter Reihenfolge angefahren, mit der sie im
	//Array ziele[] stehen
	//hier wird bestimmt, welche Ziele zuerst angefahren werden
	if(startpunkt == 64){
		zieleRechts();
		zieleLinks();
	}else{
```

```
                zieleLinks();
                zieleRechts();
        }
}

void zieleLinks(){
        for(x = 7; x < 64; x += 7){
                if(_fa[x] == 70){
                        if(erreichbar(x)){
                                ziele[y] = x;
                                y++;
                        }
                }
        }
}

void zieleRechts(){
        for(x = 13; x < 70; x += 7){
                if(_fa[x] == 70){
                        if(erreichbar(x)){
                                ziele[y] = x;
                                y++;
                        }
                }
        }
}

//prüfe zuerst, ob eine Nachbarkreuzung existiert (Rand der Karte)
//hat eine Nachbarkreuzung einen Wert < 46 ist das Ziel für den Roboter erreichbar
char erreichbar(char x){
        char erreichb = 0

        //north
        if(x - 7 >= 0){
                if(_fa[x - 7] < 46){
                        erreichb = 1;
                }
        }
        //east
        if(x % 7 < 6){
                if(_fa[x + 1] < 46){
                        erreichb = 1;
                }
        }
        //south
        if(x + 7 < 70){
                if(_fa[x + 7] < 46){
                        erreichb = 1;
                }
        }
        //west
        if(x % 7 > 0){
                if(_fa[x - 1] < 46){
                        erreichb = 1;
                }
        }

        return erreichb;
}

void berechneWeg(){

        for(x = 3; x >= 0; x--){//gehe alle Ziele durch
```

```
if(ziele[x] > 0){//0 -> kein Ziel, andere Zahlen sind eingetragene
                 //Zielkreuzungen

        //aktuelle Richtung des Ziels bestimmen
        //befindet sich das Ziel an der nördlichen, östlichen oder
        //westlichen Wand
        //denn in diese Richtung schaut auch der Roboter wenn er den
        //Rückweg antritt
        //zuerst wird der Komplette Rückweg berechnet und dann daraus der
        //Hinweg
        if(ziele[x] < 7){
               aktRichtung = 'n';
        }else if(ziele[x] % 7 == 0){
               aktRichtung = 'w';
        }else {
               aktRichtung = 'e';
        }

        //auf dieser Kreuzung befindet sich der Roboter wenn er den
        //Rückweg antritt
        if(aktRichtung == 'n'){
               aktKreuzung = ziele[x] + 7;
        }else if(aktRichtung == 'w'){
               aktKreuzung = ziele[x] + 1;
        }else{
               aktKreuzung = ziele[x] - 1;
        }
        //nun ist die Position, an der der Roboter den Rückweg vom Ziel
        //zum Start antritt, gefunden


        //alten Rückweg löschen und y Null setzen
        //y - Position im Array rWeg[]
        for(y = 49; y > 0; y--){
               rWeg[y] = 0;
        }

        //hier wird der komplette Rückweg berechnet
        //bis auf die erste Position und die letzte Aktion
        while(_fa[aktKreuzung] > 0){//solange Rückweg noch nicht bis
                                    //Startposition berechnet

               berechneNaechsteKreuzung();//naechsteRichtung() und
                                          //naechsteKreuzung() berechnen

               rWegEintragen();//aus aktueller und nächster Richtung den
                               //zu fahrenden Weg bestimmen

               aktRichtung = naechsteRichtung;
               aktKreuzung = naechsteKreuzung;
        }

        //letzte Aktion, damit der Rückweg auch korrekt eingetragen ist
        //so drehen bzw. an Kreuzung heranfahren, dass der Ball abgelegt
        //werden kann
        if(aktRichtung == 'e'){
               rWeg[y] = 'r';
               y++;
        }else if(aktRichtung == 'w'){
               rWeg[y] = 'l';
               y++;
        }else{
               rWeg[y] = 'k'; //an Kreuzung heranfahren
               y++;
```

```
                        }


                        beschreibeRoute();
                }
        }
}

//suche die Nachbarkreuzung mit dem geringsten Wert
void berechneNaechsteKreuzung(){
        int weg = 100;
        //north
        if(aktKreuzung - 7 >= 0){
                if(weg > _fa[aktKreuzung - 7]){
                        weg = _fa[aktKreuzung - 7];
                        naechsteKreuzung = aktKreuzung - 7;
                        naechsteRichtung = 'n';
                }
        }
        //east
        if(aktKreuzung % 7 < 6){
                if(weg > _fa[aktKreuzung + 1]){
                        weg = _fa[aktKreuzung + 1];
                        naechsteKreuzung = aktKreuzung + 1;
                        naechsteRichtung = 'e';
                }
        }
        //west
        if(aktKreuzung % 7 > 0){
                if(weg > _fa[aktKreuzung - 1]){
                        weg = _fa[aktKreuzung - 1];
                        naechsteKreuzung = aktKreuzung - 1;
                        naechsteRichtung = 'w';
                }
        }
        //south
        if(aktKreuzung + 7 < 70){
                if(weg > _fa[aktKreuzung + 7]){
                        weg = _fa[aktKreuzung + 7];
                        naechsteKreuzung = aktKreuzung + 7;
                        naechsteRichtung = 's';
                }
        }
}

//aus aktueller und nächster Richtung den zu fahrenden Weg bestimmen
void rWegEintragen(){

        if(aktRichtung == 'n'){
                switch(naechsteRichtung){
                        case 'n':
                        {
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
                        case 'e':
                        {
                                rWeg[y] = 'r';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
```

```
                case 's':
                {
                        rWeg[y] = 'd';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
                break;
                case 'w':
                {
                        rWeg[y] = 'l';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
        }
}
if(aktRichtung == 's'){
         switch(naechsteRichtung){
                case 'n':
                {
                        rWeg[y] = 'd';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
                break;
                case 'e':
                {
                        rWeg[y] = 'l';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
                break;
                case 's':
                {
                        rWeg[y] = 'g';
                        y++;
                }
                break;
                case 'w':
                {
                        rWeg[y] = 'r';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
        }
}
if(aktRichtung == 'e'){
        switch(naechsteRichtung){
                case 'n':
                {
                        rWeg[y] = 'l';
                        y++;
                        rWeg[y] = 'g';
                        y++;
                }
                break;
                case 'e':
                {
                        rWeg[y] = 'g';
                        y++;
```

```
                        }
                        break;
                        case 's':
                        {
                                rWeg[y] = 'r';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
                        case 'w':
                        {
                                rWeg[y] = 'd';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                }
        }
        if(aktRichtung == 'w'){
                switch(naechsteRichtung){
                        case 'n':
                        {
                                rWeg[y] = 'r';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
                        case 'e':
                        {
                                rWeg[y] = 'd';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
                        case 's':
                        {
                                rWeg[y] = 'l';
                                y++;
                                rWeg[y] = 'g';
                                y++;
                        }
                        break;
                        case 'w':
                        {
                                rWeg[y] = 'g';
                                y++;
                        }
                }
        }
}

//y - Position im Array rWeg[]
//z - Position im Array route[]
void beschreibeRoute(){

        //wenn schon etwas in route[] eingetragen wurde, befindet sich der Roboter
        //nicht wie erwartet auf Startposition mit Richtung nach Norden
        //deswegen Richtung anpassen
        if(z > 0){
                y--;
                if(rWeg[y] == 'r'){
```

```
                route[z] = 'r';
                z++;
            }else if(rWeg[y] == 'l'){
                route[z] = 'l';
                z++;
            }else{
                route[z] = 'd';
                z++;
            }
        }
        //hier wird der komplette Rückweg von hinten nach vorn durchgegangen
        //und daraus der entsprechende Hinweg errechnet und in die Route eingefügt
        y--;
        for(y; y > 0; y--){
            if(rWeg[y] == 'r'){
                route[z] = 'l';
                z++;
            }else if(rWeg[y] == 'l'){
                route[z] = 'r';
                z++;
            }else if(rWeg[y] == 'g'){
                route[z] = 'g';
                z++;
            }//wenn 'k' also „an Kreuzung“ ranfahren() -> nichts dazuschreiben
        }

        //Drehung in Richtung Ziel zum aufnehmen
        if(rWeg[y] == 'r'){
            route[z] = 'r';
            z++;
        }else if(rWeg[y] == 'l'){
            route[z] = 'l';
            z++;
        }//wenn 'g' nichts dazuschreiben

        route[z] = 'e'; //einsammeln
            z++;

        //kompletten Rückweg anfügen
        while(rWeg[y] != 0){
            route[z] = rWeg[y];
            z++;
            y++;
        }

        route[z] = 'a'; //ablegen
            z++;
}

void gerade(){

        motor_richtung(0, 0);
        motor_richtung(3, 0);
        motor_pwm(0, FAST);
        motor_pwm(3, FAST);
        //erst Kreuzung überfahren, dann weitermachen
        //ohne sleep() würde direkt wieder eine Kreuzung erkannt werden
        //und die Methode wäre beendet
        sleep(150);

        //Weg folgen bis Kreuzung
        kreuzung = 0;
        while(kreuzung == 0){
```

```
                if(analog(0) > BLACK){
                        if(analog(2) > BLACK){
                                kreuzung = 1;
                        }else{
                                motor_pwm(0, SLOW);
                                while(analog(0) > BLACK && analog(2) <= BLACK);//warte bis
                                                                                //Weg wiedergefunden
                                motor_pwm(0, FAST);
                        }
                }
                if(analog(2) > BLACK){
                        if(analog(0) > BLACK){
                                kreuzung = 1;
                        }else{
                                motor_pwm(3, SLOW);
                                while(analog(2) > BLACK && analog(0) <= BLACK);//warte bis
                                                                                //Weg wiedergefunden
                                motor_pwm(3, FAST);
                        }
                }
        }
}

void ranfahren(){
        //wenn er gerade vor der Kreuzung steht, dann heranfahren
        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
        while(analog(5) <= BLACK || analog(7) <= BLACK);
        motor_pwm(0, 0);
        motor_pwm(3, 0);

}

void greifen(){

        motor_richtung(0, 0);
        motor_richtung(3, 0);
        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
        sleep(150);

        while(digital_in(0) == 1){

                if(analog(0) > BLACK){
                        motor_pwm(0, 2);
                }else if(analog(2) > BLACK){
                        motor_pwm(3, 2);
                }else{
                        motor_pwm(0, SLOW);
                        motor_pwm(3, SLOW);
                }
        }

        motor_richtung(0, 1);
        motor_richtung(3, 1);
        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
        sleep(100);
        motor_pwm(0, 0);
        motor_pwm(3, 0);
        servo_arc(0, 80);//greifen

        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
```

```
        while(analog(5) <= BLACK || analog(7) <= BLACK);//fahre zurück bis Kreuzung
        motor_pwm(0, 0);
        motor_pwm(3, 0);
        motor_richtung(0, 0);
        motor_richtung(3, 0);
}

void ablegen(){

        sleep(300);//ohne sleep() wird Ball weggeschleudert
        servo_arc(0, 55);//ablegen

}

void rechts(){

        motor_richtung(0, 0);
        motor_richtung(3, 0);

        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
        while(analog(5) <= BLACK || analog(7) <= BLACK);
        motor_pwm(0, 0);
        motor_pwm(3, 0);

        motor_richtung(0, 1);//Drehung
        motor_pwm(0, DREH);
        motor_pwm(3, DREH);
        sleep(200);
        while(analog(0) <= BLACK);
        sleep(50);
        motor_pwm(0, 0);
        motor_pwm(3, 0);
        motor_richtung(0, 0);

}

void links(){

        motor_richtung(0, 0);
        motor_richtung(3, 0);

        motor_pwm(0, SLOW);
        motor_pwm(3, SLOW);
        while(analog(5) <= BLACK || analog(7) <= BLACK);
        motor_pwm(0, 0);
        motor_pwm(3, 0);

        motor_richtung(3, 1);//Drehung
        motor_pwm(0, DREH);
        motor_pwm(3, DREH);
        sleep(200);
        while(analog(2) <= BLACK);
        sleep(50);
        motor_pwm(0, 0);
        motor_pwm(3, 0);
        motor_richtung(3, 0);

}

void d180(){

        motor_richtung(0, 1);//Drehung
        motor_richtung(3, 0);
```

```
	motor_pwm(0, DREH);
	motor_pwm(3, DREH);
	sleep(300);
	while(analog(0) <= BLACK);
	sleep(300);
	while(analog(0) <= BLACK);
	motor_pwm(0, 0);
	motor_pwm(3, 0);
	motor_richtung(0, 0);

}

//Hauptprogrammroutine
void AksenMain(void){

	//Initialisierung für Motoren und Optokoppler
	motor_richtung(0, 0);
	motor_richtung(3, 0);
	led(2, 1);
	led(3, 1);
	servo_arc(0, 55);

	while(digital_in(0) == 1){//warte auf Druck (zum richtigen Hinstellen)

		lcd_cls();
		lcd_ubyte(analog(0));
		lcd_puts("-");
		lcd_ubyte(analog(2));
		lcd_puts("_");
		lcd_ubyte(analog(5));
		lcd_puts("-");
		lcd_ubyte(analog(7));
		sleep(100);

	}

	lcd_cls();
	lcd_puts("berechne Weg");

	fillArray();
	erkenneZiele();
	berechneWeg();

	lcd_cls();
	lcd_puts("BEREIT");

	while(analog(5) > 6);//warte auf Startlicht

	lcd_cls();
	fertig = 0;
	x = 0;

	//arbeite Route ab
	while(fertig == 0){

		switch(route[x]){
			case 0:
			{
				fertig = 1;
			}
			break;
			case 'g':
			{
				gerade();
```

```
                        }
                        break;
                        case 'l':
                        {
                                links();
                        }
                        break;
                        case 'r':
                        {
                                rechts();
                        }
                        break;
                        case 'd':
                        {
                                d180();
                        }
                        break;
                        case 'k':
                        {
                                ranfahren();
                        }
                        break;
                        case 'e':
                        {
                                greifen();
                        }
                        break;
                        case 'a':
                        {
                                ablegen();
                        }
                        break;
                }
                x++;
        }

        motor_pwm(0, 0);
        motor_pwm(3, 0);

        while(1);

}
```