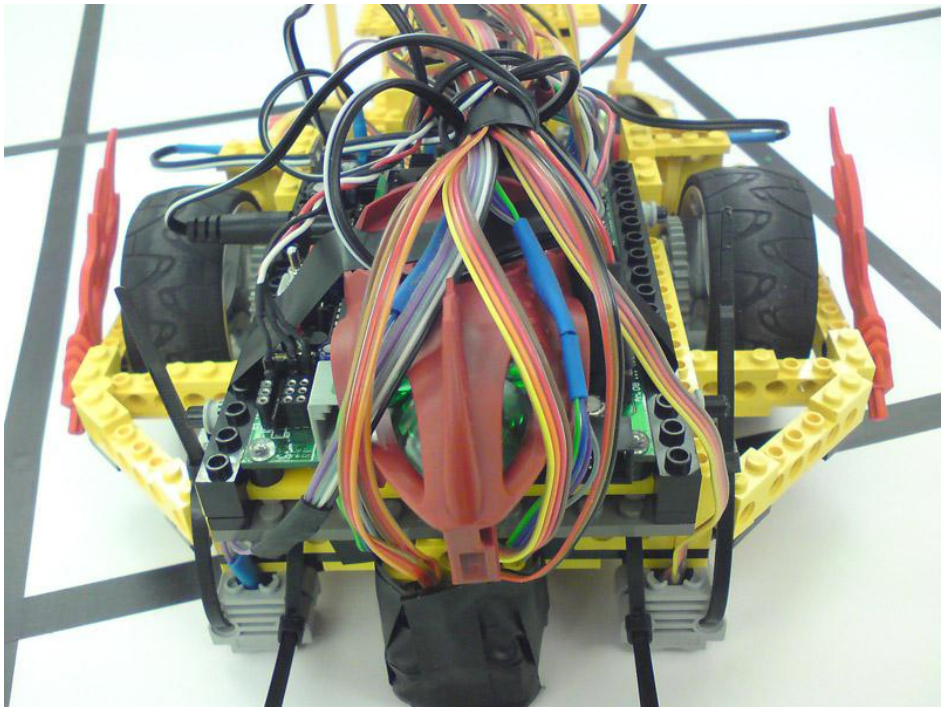
# AMS-Projekt WS11/12

**Von Josef Mögelin und Mathias Seetge**



***Flaming Okto-Optokoppler***

# 1. Aufgabenstellung

Die gestellte Aufgabe ist es, einen autonomen Roboter aus Lego und mit Hilfe eines Aksenboards zu bauen. Dieser solle durch ein vorgegebenes Labyrinth fahren (siehe Abb. 1). Am Ende des Labyrinths sollen weiterhin drei blaue Kugeln von den roten getrennt werden. Die roten Kugeln sollen ihre Position dabei nicht ändern. Der Zeitrahmen für diese Aufgabe beträgt zwei Minuten. Um das Ganze etwas spannender zu gestalten, soll am Ende des Projekts ein Wettkampf zwischen allen Robotern stattfinden, um den zu küren, der diese Aufgabe am besten gemeistert hat.

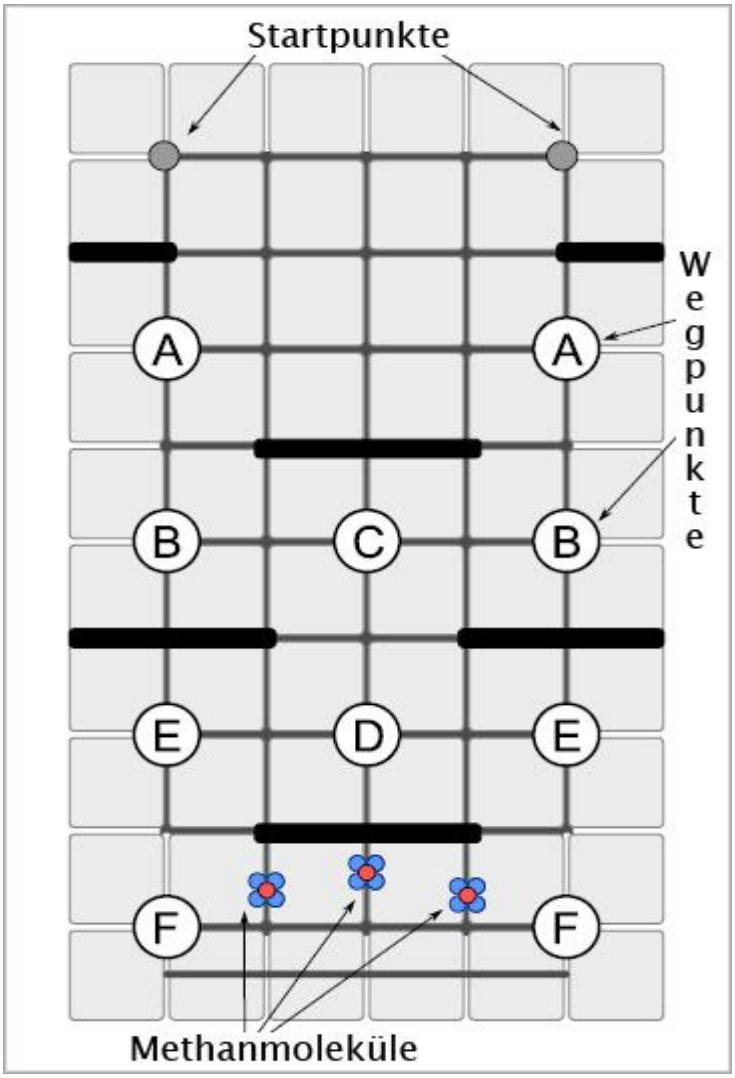



Abbildung 1: Labyrinth

## 2. Besondere Wettkampfregeln

Das Labyrinth bleibt statisch und wird nicht verändert. Die Startposition wird zunächst per Münzwurf durch die Jury bestimmt. Startsignal ist das Aufleuchten der beiden Startpunkte. Die Dauer einer Runde beträgt 120 Sekunden. Spätestens dann müssen beide Roboter anhalten.

Es erfolgt eine Disqualifikation bei:

- Verlust von Bauelementen
- Menschlichem Eingreifen
- Angriff auf andere Roboter
- Frühstart bzw. Missachtung der Zeitvorgabe

Nach folgenden Regeln erfolgt die Punktevergabe:

- Lebenspunkt: 1 Punkt fürs Losfahren bei dem Lichtsignal (Startpunkte)
- Wegpunkte: Einmalig je 3 Punkte für jede erreichte Position(A, B, ...)
- Kværner: 10 Punkte für jede getrennte blaue Kugel von den roten (siehe Abbildung 2)
    - jedoch zählen diese Punkte nicht, falls auch nur eine rote Kugel ihren Platz verlässt

- Ein disqualifizierter Roboter erhält keine Punkte für diese Runde

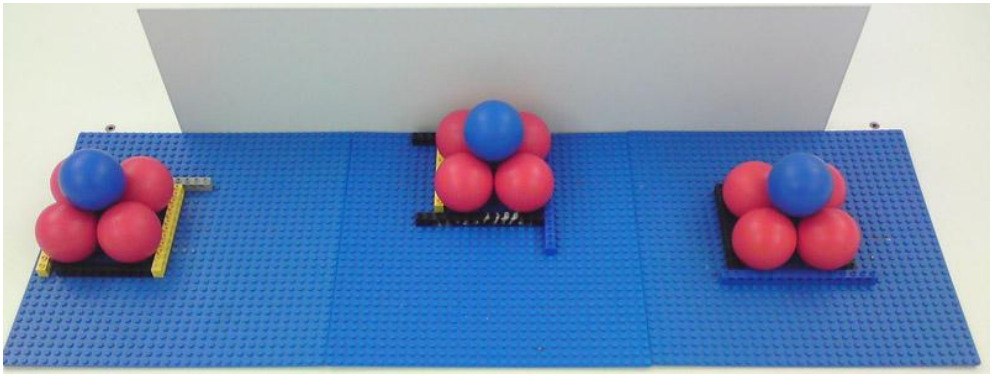


Abbildung 2: Kværner-Molekühlkugeln, ungetrennt

# 3.Lösungsansätze zur Durchquerung des Labyrinths

## 3.1. Rechte-Hand-Regel

### 3.1.1 Konzept

Als erste Idee zur Überwindung des Labyrinths kam uns natürlich die berühmte Rechte-Hand-Regel in den Sinn, bei der man, um ein Labyrinth zu durchqueren, einfach nur die Wand mit seiner rechten Hand berühren und dieser solange folgen muss, bis man das Ziel erreicht hat. Voraussetzung hierbei ist das Start- und Zielpunkt mit der Außenwand verbunden sind, was bei dieser Aufgabe gegeben war.

### 3.1.2 Vor- und Nachteile

Ein großer Vorteil dieser Variante ist natürlich seine theoretische Einfachheit. Da das Labyrinth statisch ist, gibt es einen unveränderlichen Orientierungspunkt, den man lediglich entlangfahren muss. Daraus ergibt sich eine hohe Zuverlässigkeit, denn einmal korrekt umgesetzt, müsste der Ablauf ohne große Abweichungen beliebig oft wiederholbar sein.

Zu bedenken gibt es jedoch, dass dieser Weg durch das Labyrinth zum einen nicht der kürzeste sein wird, trotz dessen aber die Möglichkeit besteht, nicht alle Wegpunkte abfahren zu können, was quasi einen doppelten Nachteil darstellt.

### 3.1.3 Umsetzung

Als mögliche Hardwareumsetzung dachten wir an Infrarot-Sensoren, die die Wand solange abtasten, bis eine signifikante Änderung auftritt und der Roboter darauf reagieren muss (Durchgang/Kurve). Ansonsten soll einfach nur dem Verlauf der Wand bis zum Ziel gefolgt werden.

### 3.1.4 Probleme

Hauptproblem bei der Umsetzung dieser Idee waren die Infrarot-Sensoren. Trotz mehrerer Testreihen erhielten wir keine konstanten Messergebnisse, um eine ausreichend genaue Reaktion des Roboters ableiten zu können. Diese waren maximal dazu geeignet, eine nahe Wand wahr zu nehmen, jedoch keinesfalls sich daran auszurichten. Besonders problematisch waren die engen Kurven, die durch die dünnen Hindernisholzwände entstehen. Diese machen ein “an der Wand entlangfahren” für solche Sensoren fast unmöglich. Zur Überwindung dieses gravierenden Problems wären weitere Mechanismen nötig gewesen, die mit einem hohen Aufwand verbunden gewesen wären, weshalb wir die Idee schlussendlich verwarfen.

## 3.2. Kürzester Weg

### 3.2.1 Konzept

Bei dieser Idee geht es darum mit dem Roboter den kürzesten Weg zum Ziel zu wählen. Es gibt unterschiedliche Möglichkeiten dieses Konzept umzusetzen:

1. “blind”, d.h. ohne Sensorik, dafür mit fest vorgegebene Bewegungsabläufe
2. “sehend”, d.h. mit Sensoren, die die Umgebung erfassen, und daraus abgeleiteten Bewegungsabläufen

### 3.2.2 Vor- und Nachteile

Dieses Konzept erlaubt es bei einem Zweikampf (idealerweise) schneller als der Gegner im Zielbereich zu sein und so die meisten Punkte zu sichern. Dies stellt jedoch auch ein gewisses Risiko dar, da die Wegpunkte außer acht gelassen werden müssen und so bei einem Fehlschlag ein völliger Punkteverlust droht. Gerade bei “blindem” fahren stellt sich zudem die Frage der Zuverlässigkeit und Wiederholbarkeit.

## 3.2.3 Umsetzung

#### 3.2.3.1 Blind

Bei der Umsetzung dieser Variante handelt es sich um eine Folge von statischen Befehlen mit Zeitschaltung durch sleep()-Befehle. Der Roboter wird anfangs ausgerichtet und fährt dann die einprogrammierte Strecke Stück für Stück ab.

#### 3.2.3.2 Sehend

Hierbei handelt es sich um eine verbesserte Version des blinden Fahrens. Statt sich auf zeitliche Intervalle allein zu verlassen werden durch Optokoppler Bodenmarkierungen wahrgenommen und damit Bewegungsänderungen initialisiert.

### 3.2.4 Probleme

#### 3.2.4.1 Blind

Dieses Konzept lässt sich nur dann ausführen, wenn die Hardware akkurat funktioniert und somit ein deterministischer Ablauf gewährleistet werden kann. Jedoch musste dieser Gedanke aufgrund von Ungenauigkeiten bei der Hardware (insbesondere dem Getriebe) wieder schnell verworfen werden. Die zur Verfügung stehenden, unterschiedlichen Motoren konnten von uns nicht in ihrer Geschwindigkeit ausreichend parallelisiert werden, um gerade Strecken zu fahren. Zudem beeinflusste die Akkuleistung mitunter die Geschwindigkeit des Roboters und damit das richtige Timing für Aktionen wie in eine Kurve einlenken. Diese endeten dann oftmals recht schnell mit der Kollision an einer Wand. Aufgrund dieser Probleme kam diese Methode zumindest für das gesamte Labyrinth für uns nicht in Frage.

#### 3.2.4.2 Sehend

Auch hier spielt die Genauigkeit des Roboters eine wichtige Rolle. Durch Verzerrungen im Getriebe kam der Roboter schnell von der Strecke ab und fuhr gegen die Wände, da er diese relativ eng passieren wollte. Weiterhin gab es Probleme bei der Zählung/Messung der überfahrenen Linien und somit zur Fehlinterpretation bei der Orientierung, da der Roboter zu schnell fuhr. Eine starke Reduktion der Geschwindigkeit wäre zwar möglich gewesen, allerdings hätte dies den einzigen Vorteil dieser Methode stark vermindert. Aufgrund von Kollisionen mit den Wänden wurde außerdem der Roboter mehrmals beschädigt, was zu einem späteren Redesign der Karosserie (ursprüngliches Design, siehe Abbildung 3) veranlasste. Dies brachte zwar mehr Stabilität und Widerstandsfähigkeit, allerdings ging dies auf Kosten von Geschwindigkeit und Akkuleistung.

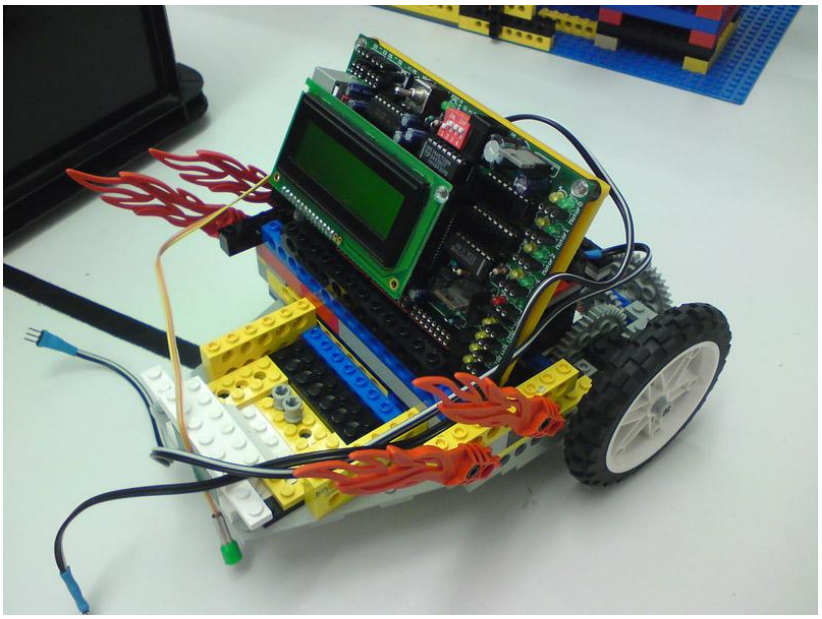


Abbildung 3: ursprüngliches Design

## 3.3. Linien folgen (ausgewählt und ausgeführtes Konzept)

### 3.3.1 Konzept

Idee ist es, den Linien des Labyrinths zu folgen und somit einen sicheren Weg zu finden. Dies kann jedoch nicht auf der gesamten Strecke angewendet werden, da die Linien sich nicht vollständig durch den Parkour ziehen. Für diesen kurzen Teil müssen andere Konzepte verwendet werden.

### 3.3.2 Vor- und Nachteile

Diese Variante birgt den wichtigsten Vorteil: Zuverlässigkeit. Des weiteren kann im Idealfall die Ungenauigkeit der Hardware durch ständige Neukorrektur ausgeglichen werden. Dem gegenüber steht die längste Fahrtzeit aller Konzepte, welche sich zum Teil aus der Strecke, als auch aus der Geschwindigkeit zusammensetzt. Aufgrund ständiger Neuausrichtungen ist auch die Geschwindigkeit bei weitem geringer als bei den anderen Konzepten. Der Akkuverbrauch ist ein weiterer, nicht vernachlässigbarer Faktor. Durch Zweckentfremdung der Hardware zur Trennung der Kugeln konnte jedoch ein geringer Startvorsprung erreicht werden (siehe Abbildung 4). Dieser sollte die Nachteile etwas kompensieren.

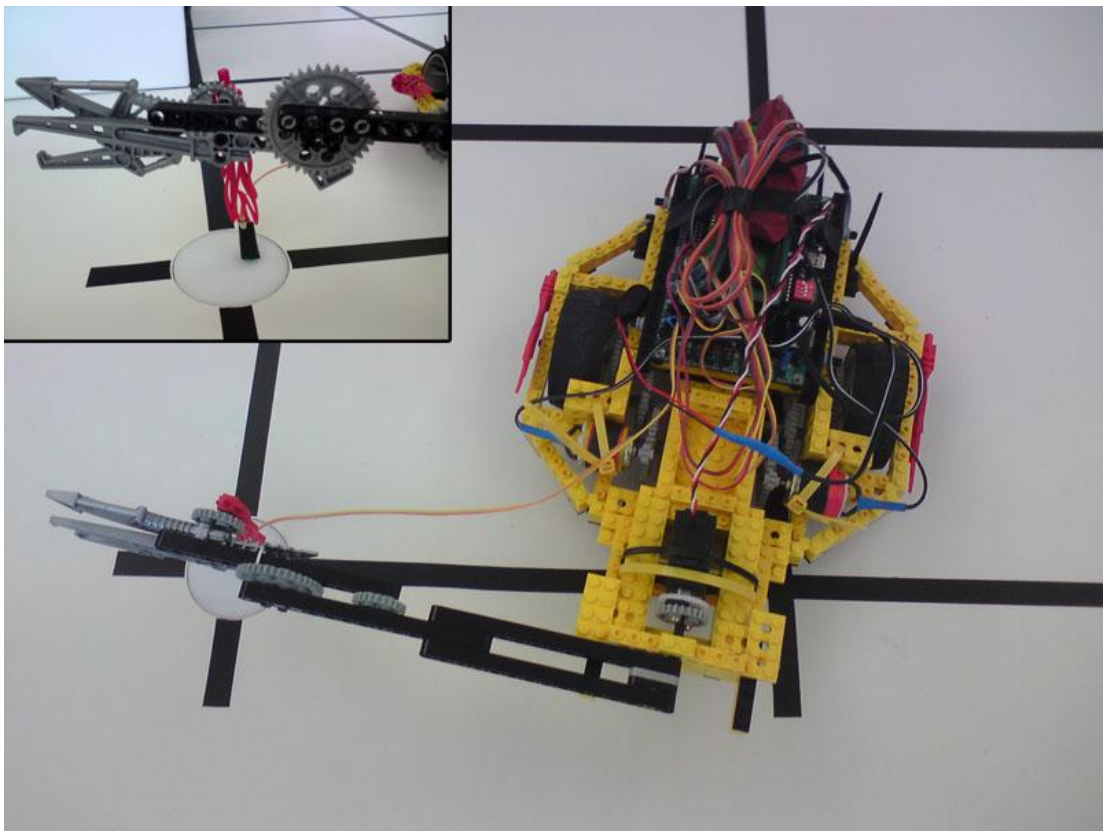


Abbildung 4: Startposition

### 3.3.3 Umsetzung

Folgende Aktoren und Sensoren haben wir verbaut:

- 1x Fotosensor
- 8x Optokoppler
- 2x Elektromotor
- 1x Servomotor

Der Fotosensor ist für den Start zuständig. Wird ein höherer Wert der Helligkeit gemessen als der angegeben Schwellwert, wird der Start des Roboters eingeleitet. Der Fotosensor ist an dem Stab befestigt, der zur Abteilung der Molekülkugeln genutzt wird. Dieser wiederum ist mit dem Servomotor verbunden um den Stab, während der Fahrt, in die Höhe auszurichten und somit eine Kollision mit den Wänden zu vermeiden. Dadurch konnte ein Wegvorsprung von einer gesamten Ecke herausrausgearbeitet werden. An der Vorderseite des Roboters befinden sich 8 Optokoppler (siehe Abbildung 5). Diese sind nah zum Boden ausgerichtet und messen die Stärke der Reflexion ihres ausgesendeten Signals. Sie dienen zur Navigation über das Liniennetz. Die zwei Elektromotoren setzen den Roboter über das Getriebe in Bewegung.

Am Ende des Labyrinths ist jedoch das Liniennetz unterbrochen und somit gibt es keine Orientierungsmöglichkeiten mehr auf dem Boden. Aus diesem Grund fährt der Roboter in diesem Abschnitt “blind”. Das besondere dabei ist, dass er nicht wie die anderen Modelle vorwärts, sondern rückwärts fährt. Dies liegt an der Position der befestigten Optokoppler. Sie sind zu weit vorne verbaut, als dass es möglich wäre eine “gerade” Ausgangsposition zu gewährleisten. Daher dreht der Roboter sich in die entgegengesetzte Richtung und findet die Linie auf der gegenüberliegenden Seite der letzten Kreuzung. Von hier aus fährt er rückwärts. Nach einer festgelegten Zeit geht der Roboter dann in einen rotierenden Suchmodus bis er eine schwarze Linie erkennt. Dieser folgt er dann bis zum Ende.

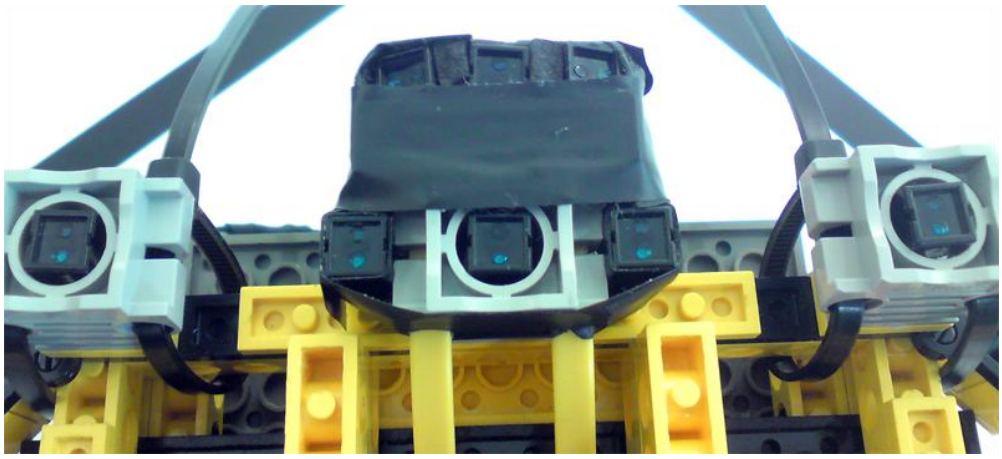


Abbildung 5: Optokoppler

### 3.3.4 Probleme

Die Optokoppler waren anscheinend nicht gut genug befestigt und sind so im Laufe der Zeit stark durch Kollisionen von ihrer Ursprungsausrichtung und Position abgekommen, was das Einlesen von Informationen u.U. beeinträchtigt hat. Weiterhin hat sich das Getriebe (siehe Abbildung 6) nach etwa 60 Testfahrten stark verzogen, sodass ein Linksdrall eintrat. Dieser war so schwerwiegend, dass der Roboter Schwierigkeiten hatte auf seiner Linie zu bleiben. Auch der Austausch einiger Bestandteile brachte nur geringe Linderung für dieses Problem.

Zudem haben die ersten drei verbauten Optokoppler aufgrund von Designentscheidungen einen zu großen Abstand, welcher das exakte Folgen der Linie unmöglich machte. Aus diesem Grund mussten weitere Sensoren verbaut werden. Die bereits befestigten Optokoppler konnten wegen robuster Implementierung in die Karosserie jedoch nicht mehr entfernt werden und blieben somit erhalten. Die erhöhte Anzahl der Optokoppler hat aber trotzdem keine nennenswerten Vorteile erbracht. Vielmehr wurde dadurch noch mehr Strom verbraucht, was zu einem schnelleren Erliegen des Akkus führte.

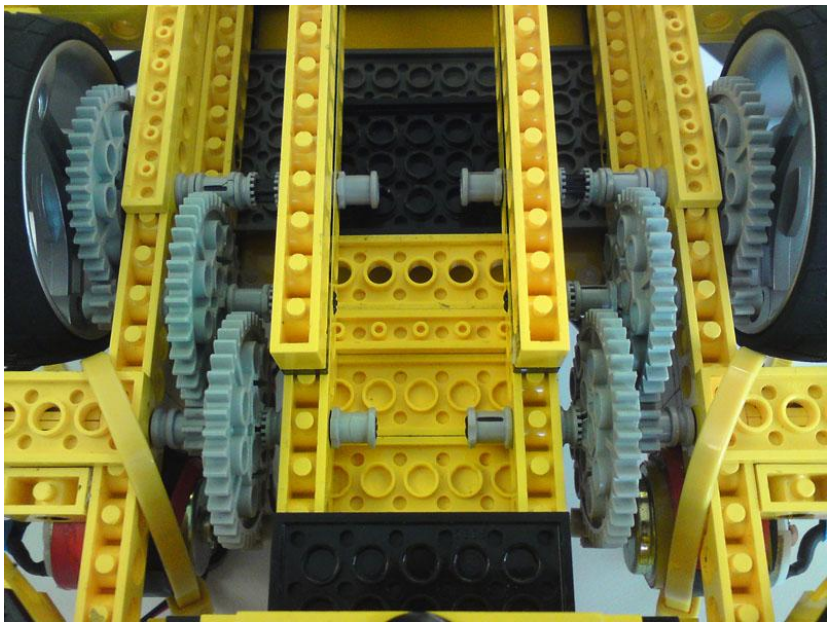


Abbildung 6: Getriebe

# 4. Lösungsansätze zur Kugeltrennung

## 4.1. Über- Wand-Greifer

### 4.1.1 Konzept

Bei diesem Ansatz geht es darum mit der letzten Wand des Labyrinth, die den Weg zum Zielbereich versperrt, zu kollidieren und mit einer Gerüstvorrichtung darüber hinweg zu greifen, um die blauen Kugeln herunterzustoßen.

### 4.1.2 Vor- und Nachteile

Vorteilhaft bei dieser Variante ist zum einen der enorme Zeitgewinn, da etwa ein Fünftel der Strecke eingespart wird, zum anderen handelt es sich beim eingesparten Weg auch noch um den kompliziertesten der Route, was eine eventuell aufwändige Realisierung der Durchquerung erspart. Dem entgegen steht natürlich die Umsetzung eines ebenso komplexen Kugeltrenn-Mechanismus mit zugehöriger Gerüstkonstruktion.

### 4.1.3 Umsetzung

Zum wandübergreifenden Trennen der Kugeln wird ein hohes, gabelstaplerfront-ähnliches Gerüst benötigt, welches mit einem Servomotor auf und ab bewegt werden kann. Zudem muss motorisch eine Stange von dort vor und zurück bewegt werden können, um die blauen von den roten kugeln zu trennen.

### 4.1.4 Probleme

Einfach technisch zu aufwändig.

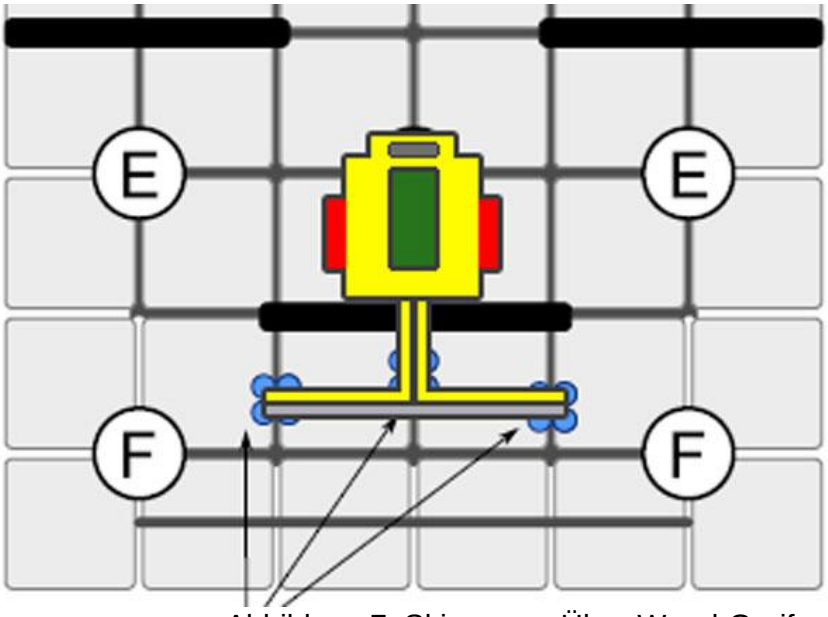



Abbildung 7: Skizze von Über-Wand-Greifer

## 4.2. Der horizontale Stab

### 4.2.1 Konzept

Der klassische Ansatz, wie er letztendlich auch umgesetzt wurde. Hierbei wird eine Stange zu gegebener Zeit motorisch zur Seite ausgerichtet, um die blauen Kugeln herunterzustoßen.

### 4.2.2 Vor- und Nachteile

Einfache Lösung, die allerdings ansonsten keinerlei Vorteile mit sich bringt. Nachteil ist höchstens, dass sie von der Konkurrenz ebenfalls verwendet wird und daher das Ergebnis in einem Zweikampf nicht positiv beeinflusst.

### 4.3.3 Umsetzung

Zur Realisierung wird einfach eine längliche Stange vertikal an die Welle eines Servomotors montiert, der dann im Zielgebiet angesteuert wird und die Stange auf den nötigen Winkel kippt (siehe Abbildung 8).

### 4.4.4 Probleme

Keine, abgesehen von der exakten Ermittlung der benötigten Höhe der Stange zum Trennen der Kugeln.

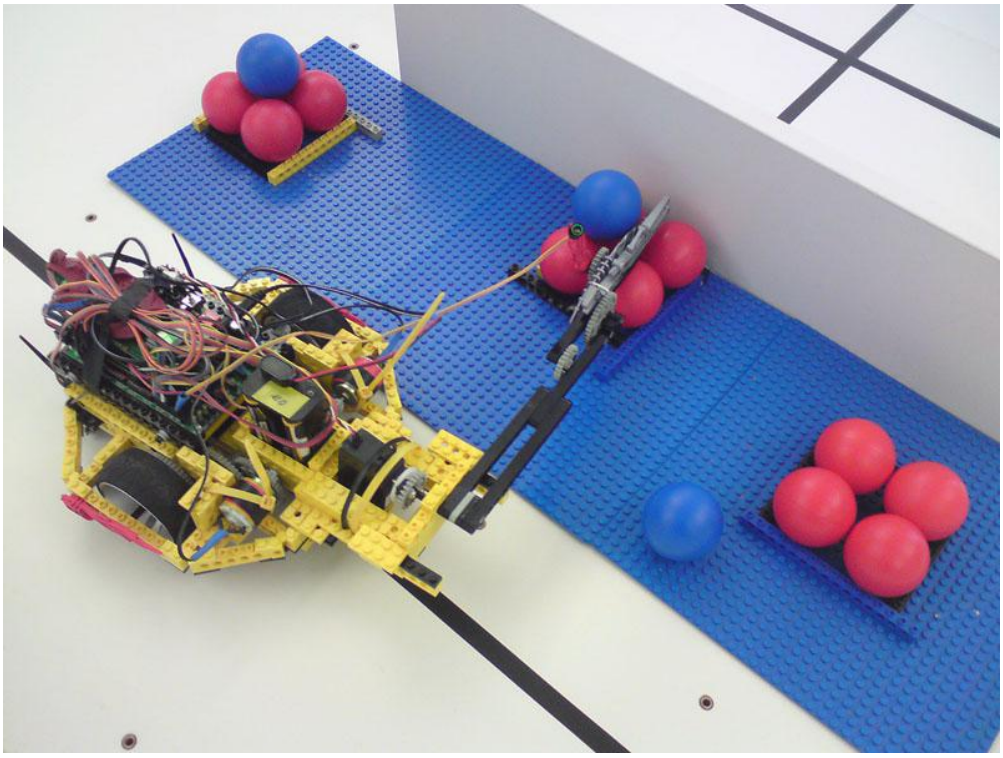


Abbildung 8: Stab beim Trennen der Kugeln

## 4.3. Destructionmodus

### 4.3.1 Konzept

Um im Wettkampf gegen schnellere/bessere Roboter zu gewinnen, gibt es einen Modus in dem versucht wird, dass Maß der eventuell verlorenen Punkte möglichst geringzuhalten. Sollte die Situation eintreten, das der konstruierte Roboter verlieren wird, ist es das Ziel mindestens eine rote Kugel aus der Zielfläche zu befördern. In diesem Fall zählen die 30 geholten Punkte des Gegners nicht mehr.

### 4.3.2 Vor- und Nachteile

Klarer Vorteil ist die Minimierung der Punktunterschiede mit den anderen Teams. Wenn ein gegnerisches Team sich nur auf die “Zielpunkte” konzentriert und die Wegpunkte außer Acht lässt könnten wir sogar im direkt Vergleich gewinnen. Der Nachteil an dieser Strategie ist der Zeitpunkt der Entscheidung. Es muss vor dem Rennen der Sieger bekannt sein, ansonsten werden möglicherweise wichtige Punkte durch eine Fehlstrategie verschenkt.

### 4.3.3 Umsetzung

Es gibt drei Möglichkeiten zur Umsetzung:

1. Hardwareveränderung am Roboter
2. Modus in der Software vorgesehen
3. Anfahren der Kugeln

Bei der Veränderung der Hardware des Roboters wird ein zusätzliches Bauteil am Ende des Stabes montiert. Dieses hängt tief genug um nicht nur blaue, sondern auch rote Kugeln aus der Zielposition zu befördern. Diese Methode wird im gebauten Roboter verwendet.

Für Variante Zwei wird der Stab mit dem Servomotor so tief verankert, dass auch rote Kugeln berührt werden können. Dies benötigt einen weitere Einstellungsmöglichkeit per DIP-Panel.

Die letzte Variante wird durch eine geänderte Fahrtroute umgesetzt. Hierbei folgt der Roboter nicht mehr der schwarzen Linien, sondern fährt direkt auf die Zielpunkte zu.

### 4.3.4 Probleme

Damit diese Strategie umgesetzt werden kann, muss der Roboter auch bis zum Ende durchfahren. Dies konnte allerdings im Wettkampf nicht gewährleistet werden und somit erfolgte keine Umsetzung der geplanten “Sabotage”. Aufgrund der Bauweise konnte der Servomotor sich nicht weit genug drehen, so dass der Stab immer zu hoch für die roten Kugeln angesetzt war.

## 5. Zeitstrahl

In den ersten vier bis fünf Wochen ging es hauptsächlich nur um die Konstruktion des Roboters. Hierfür wurden zuerst ältere Roboter analysiert, um deren Stärken und Schwächen einzuschätzen. In den nächsten zwei Wochen entstand das Grundgerüst des Quellcodes. Damit war es möglich Linien zu erkennen, zählen und zu folgen. In den letzten Wochen wurde versucht der Quellcode zu optimieren. Ein geeignetes Setting zu finden hatte oberste Priorität. Aufgrund der unterschiedlichen Leistung der Akkus hat dies sehr viel Zeit in Anspruch genommen und war bis zum Tag des Wettbewerbs recht variabel.

## 6. Gründe der Designwahl

Nach reichlichen Überlegungen und unterschiedlichen Versuchansätzen fiel die Wahl auf das Prinzip der Linienfolgung. Diese Variante schien die sicherste und zu gleich einfachste Möglichkeit das gewünschte Ziel zu erreichen. Daraus abgeleitet konnten wir, da der Roboter den Boden mit Sensoren abtasten muss, nicht auf Geschwindigkeit setzen und realisierten deshalb ein stärkeres Getriebe (1:125). Jede weitere Designentscheidung diente allein der besseren Optik.

## 7. Probleme im Wettbewerb

Leider konnte die Zuverlässigkeit des Roboters nicht gewährleistet werden. Aufgrund von starken Abhängigkeiten zwischen Akku und Getriebe war es nicht möglich einen konsistenten Quellcode zu schreiben. Werte mussten oft je Akkustand angepasst werden. Während des Wettbewerbs war die Betriebsfähigkeit jedoch auf ein absolutes Minimum zusammengebrochen. Sowohl bei den letzten Testläufen, als auch nach dem Abschlussevent lief er allerdings wieder relativ konstant. Eine weitere technische Panne war ein defekter Fotosensor am Wettkampftag, der kurz vor dem Start ausgetauscht werden musste.

Ein weiteres Problem war die Einstellung der unterschiedlichen Modi. Bezeichnungen wie die der Schalter auf dem Aksenboard, beispielweise On - 1, 2, 3, 4 (siehe Abbildung 9) waren einfach irreführend gewählt und führten im entscheidenden Moment zu menschlichem Versagen.

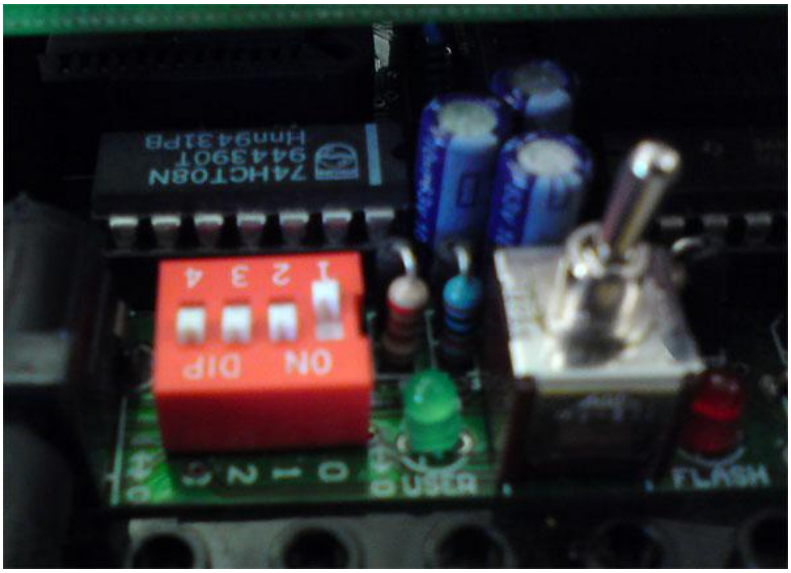


Abbildung 9: Modi-Schalter auf Aksenboard

## 8. Vorschläge

Alles in allem hat uns das Projekt viel Spaß gemacht und wir haben einiges über die Robotik gelernt. Dennoch könnte man noch ein paar Kleinigkeiten verbessern. Zum einen würde es viel Mühe und Zeit ersparen, wenn die Legobauelemente sortiert wären. Weiterhin wäre es schön, wenn alle Gruppen die gleichen Voraussetzungen hätten bzw. die gleichen Bauteile zur Verfügung stehen würden. So hatten einige z.B. sehr gute und speziell gleiche Motoren und andere eben nicht. Auch ist es sicher veranschaulichend, wenn Roboter aus älteren Jahrgängen noch existieren, dennoch fehlen dann den neuen Gruppen wichtige Bauteile. Entweder sollte man dann neue Teile zur Verfügung stellen oder die alten Roboter teilweise auseinander bauen dürfen.

```
/*
Programm: FOOK (Flaming Okto-Optokoppler)
*/

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

// Motoren
#define MOTOR_LEFT 0
#define MOTOR_RIGHT 2
#define MOTOR_RIGHT_MAX_SPEED 9
#define MOTOR_LEFT_MAX_SPEED 9

// Sensoren
#define LL 0
#define ML 1
#define MM 2
#define MR 3
#define RR 4

#define FL 8
#define FM 10
#define FR 12

// Direction
#define TURN_TURN 0
#define TURN_LITE_LEFT 1
#define TURN_STFU 2
#define TURN_LITE_RIGHT 3

#define TURN_MID_LEFT 4
#define TURN_MID_RIGHT 5

// Servo-Winkel
#define SERVO_PORT 0
#define SERVO_UP 102
#define SERVO_LEFT 148
#define SERVO_RIGHT 40

#define SERVO_LEFT_D 155
#define SERVO_RIGHT_D 33



void armUp();
void armLeftDown();
void armRightDown();
void turnLeft();

void turnLeft();
void turnRight();
void turnLeftSpecial();
void turnRightSpecial();
void followLine(unsigned char);
void driveLeftBackwards();
void driveRightBackwards();
```

```
void stop();

void leftCourse();
void rightCourse();

char action;
int blindCount;

void AksenMain(void)
{
    blindCount = 0;
    action = TURN_STFU;

    // auf lichtsognal warten
    while(analog(7)>30);

    if(dip_pin(0)) {
       leftCourse();
    } else if(dip_pin(3)) {
       rightCourse();
    }


    while(1);
}


// Start linke Seite
void leftCourse() {

    action = TURN_MID_LEFT;
    armUp();

    //followLine(0);
    //turnLeft();

    followLine(0);
    sleep(300);
    followLine(0);

    turnLeft();
    followLine(0);
    turnRight();

    followLine(0);
    sleep(300);
    followLine(0);

    turnRight();

    followLine(0);
    sleep(300);
    followLine(0);

    turnLeft();

    followLine(0);
```

```
    sleep(300);


    followLine(0);
    turnLeft();

    followLine(0);
    sleep(300);
    followLine(0);

    turnLeft();
    driveLeftBackwards();
    turnLeftSpecial();

    armRightDown();
    followLine(0);
    action = TURN_MID_RIGHT;
    followLine(0);
    armUp();

    stop();

}

// Start rechte Seite
void rightCourse() {

    action = TURN_MID_RIGHT;
    armUp();

    //followLine(0);
    //turnRight();

    followLine(0);
    sleep(300);
    followLine(0);

    turnRight();
    followLine(0);
    turnLeft();

    followLine(0);
    sleep(300);
    followLine(0);

    turnLeft();

    followLine(0);
    sleep(300);
    followLine(0);

    turnRight();

    followLine(0);
    sleep(300);
```

```
    followLine(0);
    turnRight();

    followLine(0);
    sleep(300);
    followLine(0);

    turnRight();
    driveRightBackwards();
    turnRightSpecial();

    armLeftDown();
    followLine(0);
    action = TURN_MID_LEFT;
    followLine(0);
    armUp();

    stop();

}


// Funktion zur Entscheidung der naechsten Aktion beim Linienfolgen
unsigned char guessNextTurn(unsigned char *sensor, unsigned char followMode) {

    if (followMode!=0) {

        if(blindCount>400) {
            action = TURN_TURN;
            return TURN_TURN;
         }

    } else {

        if ((sensor[LL] > 120 || sensor[RR]>120) && (sensor[ML] > 120 || sensor[MM] > 120 ||
sensor[MR] > 120)) {
            action = TURN_TURN;
            return TURN_TURN;
        }
    }


    if (sensor[FM] > 150) {
        action = TURN_STFU;
        return TURN_STFU;
    }


    if (sensor[FL] > 120 && sensor[FR] < 150) {
        action = TURN_LITE_LEFT;
        return TURN_LITE_LEFT;
    }

    if (sensor[FR] > 120 && sensor[FL] < 150){
        action = TURN_LITE_RIGHT;
        return TURN_LITE_RIGHT;
    }
```

```
    if (sensor[ML] > 120) {
        action = TURN_MID_LEFT;
        return TURN_MID_LEFT;
    }

    if (sensor[MR] > 120) {
        action = TURN_MID_RIGHT;
        return TURN_MID_RIGHT;
    }

    if (followMode!=0)
        blindCount++;

    return action;

}

// Motordrehrichtung festlegen
void setDirection(unsigned char leftDirection, unsigned char rightDirection) {
    motor_richtung(MOTOR_LEFT, leftDirection);
    motor_richtung(MOTOR_RIGHT, rightDirection);
}

// Geschwindigkeit setzen
void setSpeed(unsigned char leftVel, unsigned char rightVel) {
    motor_pwm(MOTOR_LEFT, leftVel);
    motor_pwm(MOTOR_RIGHT, rightVel);
}

//Geschwindigkeit auf Miximalwerte setzen
void setMaxSpeed() {
    motor_pwm(MOTOR_LEFT, MOTOR_LEFT_MAX_SPEED);
    motor_pwm(MOTOR_RIGHT, MOTOR_RIGHT_MAX_SPEED);
}

// Motoren stoppen
void stop() {
    motor_pwm(MOTOR_LEFT, 0);
    motor_pwm(MOTOR_RIGHT, 0);
}

// Leicht nach links fahren
void turnLiteLeft() {
    motor_pwm(MOTOR_LEFT, 5);
    motor_pwm(MOTOR_RIGHT, 8);
}
// Leicht nach rechts fahren
void turnLiteRight() {
    motor_pwm(MOTOR_LEFT, 8);
    motor_pwm(MOTOR_RIGHT, 5);
}

// Maessig nach links fahren
void turnMidLeft() {
    motor_pwm(MOTOR_LEFT, 1);
```

```
    motor_pwm(MOTOR_RIGHT, 8);
}

// Maessig nach rechts fahren
void turnMidRight() {
    motor_pwm(MOTOR_LEFT, 8);
    motor_pwm(MOTOR_RIGHT, 1);
}

// Stark nach links fahren
void turnHardLeft() {
    setDirection(1,0);
    motor_pwm(MOTOR_LEFT, 5);
    motor_pwm(MOTOR_RIGHT, 5);
}

// Stark nach rechts fahren
void turnHardRight() {
    setDirection(0,1);
    motor_pwm(MOTOR_LEFT, 5);
    motor_pwm(MOTOR_RIGHT, 5);
}

// Nach links drehen
void turnLeft() {
    unsigned char i;
    unsigned char sensor[13];

    turnHardLeft();

    // nicht zu fruehes lesen nach kreuzung
    sleep(250);

    do {
        for (i = LL; i <= RR; i++)
            sensor[i] = analog(i);

        sensor[FL] = analog(FL);

    } while(!(sensor[LL]<120 && sensor[FL]>150 && sensor[RR]<120));

}

// Nach rechts drehen
void turnRight() {
    unsigned char i;
    unsigned char sensor[13];

    turnHardRight();

       // nicht zu fruehes lesen nach kreuzung
    sleep(250);

    do {
        for (i = LL; i <= RR; i++)
            sensor[i] = analog(i);
```

```
        sensor[FR] = analog(FR);

    } while(!(sensor[LL]<120 && sensor[FR]>150 && sensor[RR]<120));

}

// Nach rechts drehen - spezieller Suchmodus
void turnRightSpecial() {
    unsigned char i;
    unsigned char sensor[13];

    setDirection(0,1);
    motor_pwm(MOTOR_LEFT, 3);
    motor_pwm(MOTOR_RIGHT, 3);

    do {
        for (i = LL; i <= RR; i++)
            sensor[i] = analog(i);

        sensor[FM] = analog(FM);

    } while(!(sensor[FM]>150));
}

// Nach links drehen - spezieller Suchmodus
void turnLeftSpecial() {
    unsigned char i;
    unsigned char sensor[13];

    setDirection(1,0);
    motor_pwm(MOTOR_LEFT, 3);
    motor_pwm(MOTOR_RIGHT, 3);

    do {
        for (i = LL; i <= RR; i++)
            sensor[i] = analog(i);

        sensor[FM] = analog(FM);

    } while(!(sensor[FM]>150));
}

// Rueckwaerts fahren auf rechter Strecke (mit Linksdrall)
void driveRightBackwards() {
    setDirection(1,1);

    motor_pwm(MOTOR_RIGHT, 9);

    if(dip_pin(2)) {
       motor_pwm(MOTOR_LEFT, 7);
       sleep(3000);

    } else {
       motor_pwm(MOTOR_LEFT, 8);
       sleep(2800);   // je akku anders :T
    }
```

```
}

// Rueckwaerts fahren auf linker Strecke (mit Rechtsdrall)
void driveLeftBackwards() {
    setDirection(1,1);
    motor_pwm(MOTOR_LEFT, 9);

    if(dip_pin(1)) {
       motor_pwm(MOTOR_RIGHT, 7);
       sleep(3000);

    } else {
       motor_pwm(MOTOR_RIGHT, 8);
       sleep(2800);   // je akku anders :T
    }

}

// Stange in vertikale position bringen
void armUp() {

    servo_arc(SERVO_PORT, SERVO_UP);
}

// Stange nach links senken
void armLeftDown() {

    if(dip_pin(1)) {
       servo_arc(SERVO_PORT, SERVO_LEFT_D - 20);
       sleep(250);
       servo_arc(SERVO_PORT, SERVO_LEFT_D);
    } else {
       servo_arc(SERVO_PORT, SERVO_LEFT - 20);
       sleep(250);
       servo_arc(SERVO_PORT, SERVO_LEFT);
    }

}

// Stange nach rechts senken
void armRightDown() {

    if(dip_pin(2)) {
       servo_arc(SERVO_PORT, SERVO_RIGHT_D + 20);
       sleep(250);
       servo_arc(SERVO_PORT, SERVO_RIGHT_D);
    } else {
       servo_arc(SERVO_PORT, SERVO_RIGHT + 20);
       sleep(250);
       servo_arc(SERVO_PORT, SERVO_RIGHT);
    }

}

// Linie folgen
void followLine(unsigned char followMode) {
    unsigned char i, nextTurn;
```

```
    unsigned char sensor[13];

    unsigned char vCount;

    setMaxSpeed();
    vCount = 0;


    do {
        setDirection(0,0);

        for (i = LL; i <= RR; i++)
            sensor[i] = analog(i);

        sensor[FL] = analog(FL);
        sensor[FM] = analog(FM);
        sensor[FR] = analog(FR);

        if(followMode!=0)
            nextTurn = guessNextTurn(sensor, 1);
        else
            nextTurn = guessNextTurn(sensor, 0);


        if (nextTurn == TURN_LITE_LEFT)
            turnLiteLeft();

        else if (nextTurn == TURN_LITE_RIGHT)
            turnLiteRight();

        else if (nextTurn == TURN_MID_LEFT)
            turnMidLeft();

        else if (nextTurn == TURN_MID_RIGHT)
            turnMidRight();

        else if (nextTurn == TURN_STFU)
            setMaxSpeed();

        }

    } while (!(nextTurn==TURN_TURN));

    blindCount=0;
    // ein stueck ueber-kreuzung-fahren
    sleep(300);
}
```