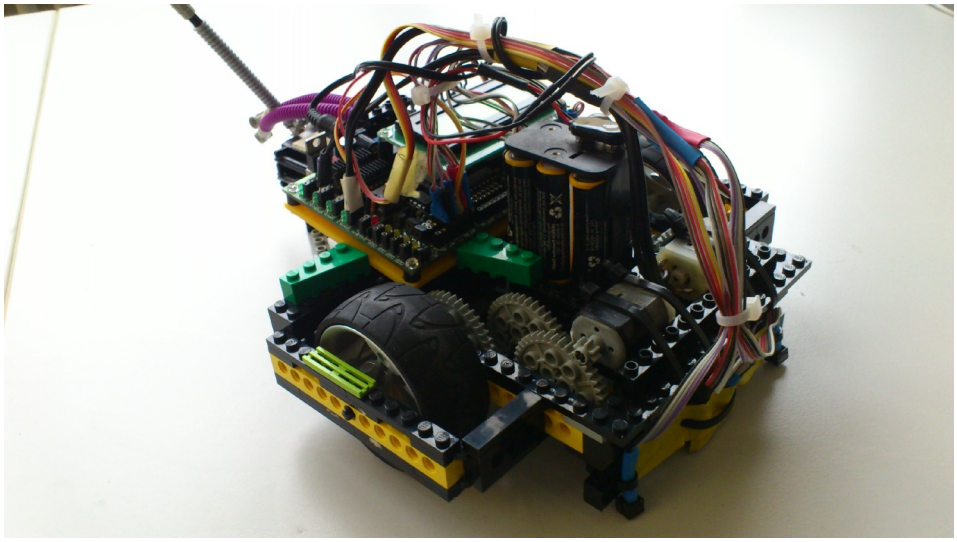
Lego-Projekt

Wintersemester 2011/2012

Dokumentation des LEGO- Roboters

„Scared Lizard“



von
Daniel Schmidt
und
Benjamin Hoffmann

Abgabe: 20.01.2012

Betreuer: Ingo Boersch

Inhaltsverzeichnis

1 Aufgabenstellung

Es soll ein Lego-Roboter mit Hilfe eines Aksen-Boards entwickelt werden. Ziel ist das Herunterstoßen dreier Kugeln am Ende eines statischen Labyrinths. Das Bewegen der darunter liegenden Kugeln führt zu Punkteabzug[1] in der Wertung. Gestartet wird der Roboter mittels eines Lichtsignals an den Startpunkten. Der Roboter hat 120 Sekunden Zeit die Aufgabe zu erfüllen.

Das korrekte Starten beim Lichtsignal bringt einem einen Punkt ein. Für jeden abgefahrenen Wegpunkt werden einmalig drei Punkte gutgeschrieben. Für jede abgetrennte Kugel gibt es zehn Punkte.

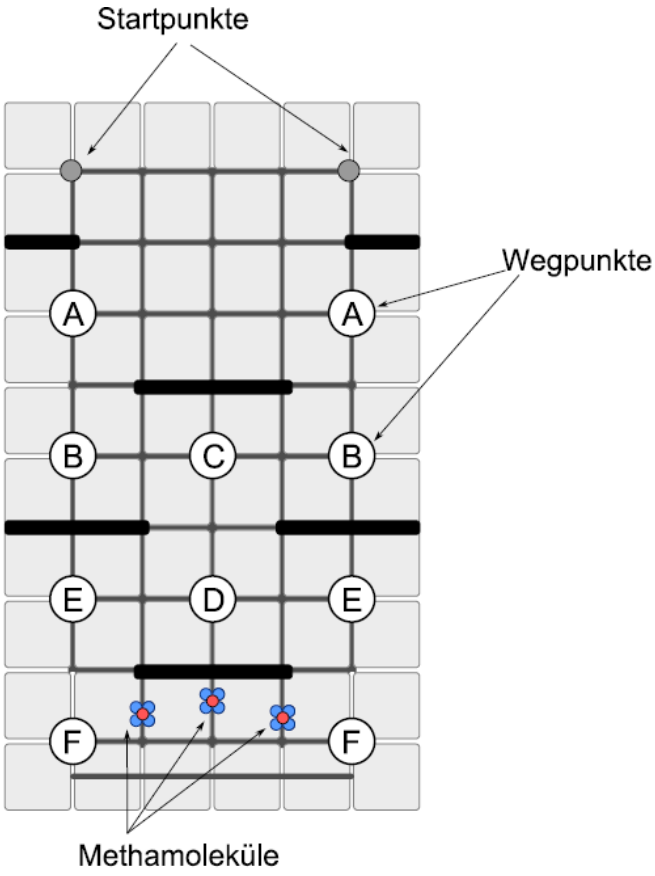


Abbildung 1: Spielfeld mit Beschriftung

1 Sobald auch nur eine blaue Kugel (siehe Abbildung 1) sich nicht mehr in ihrer Ausgangslage befindet, bekommt keiner der Roboter die Punkte für das Abtrennen der Kugeln.

2 Zur Verfügung stehende Mittel

Zur Bewältigung unserer Aufgabe wurden folgende Bauteile bereitgestellt:

- AKSEN-Board
- Lego-Bausteine in allen Farben und Formen
- Sensoren (Optokoppler, Infrarotsender/-empfänger, Schalter, Lichtsensor)
- Aktoren (Motor, Servomotor)

Für die Programmierung des AKSEN-Boards wird die Programmiersprache „C” eingesetzt. Der Zugriff auf die Sensoren und die Ansteuerung der Aktoren erfolgt mittels Aufruf von Bibliotheksfunktionen.

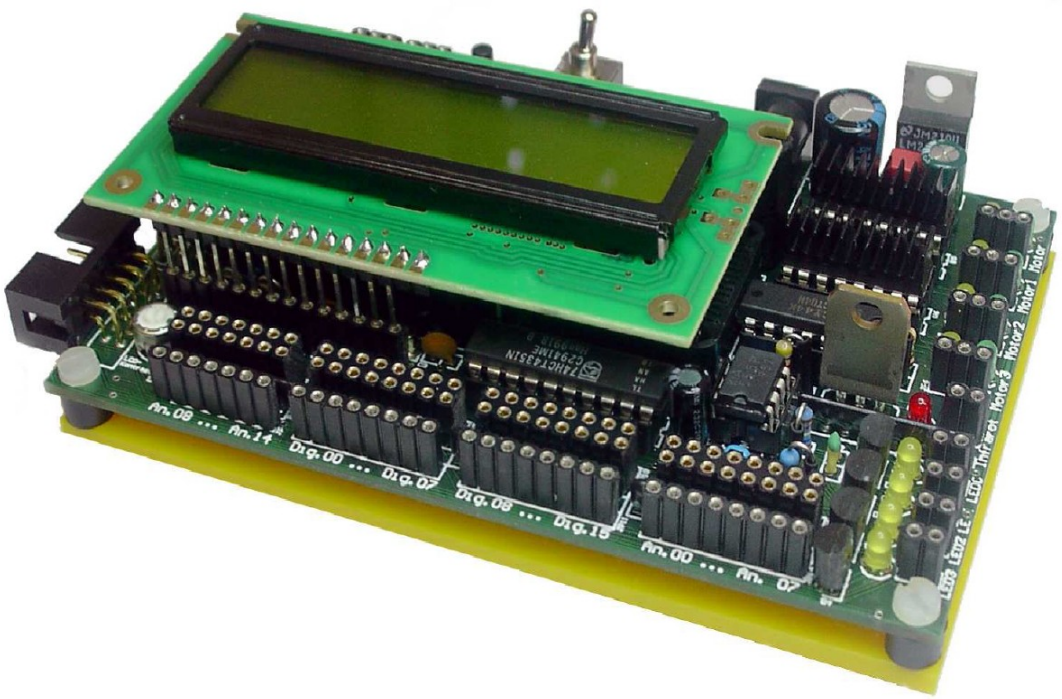


Abbildung 2: AKSEN-Board

3 Ideen

a Roboter mit Odometrie[2]

Der Roboter besitzt keine Sensoren, welche die Umwelt wahrnehmen, sondern misst nur die Länge des zurückgelegten Weges.

Vorteil

- **Schnelligkeit**: Der Roboter ist an keine Markierungen gebunden und kann somit den kürzesten Weg abfahren. Unnötiges Fahren von Kurven kann vermieden werden.

Nachteil

- **Unzuverlässigkeit**: Der Erfolg hängt sehr stark von der Ausrichtung am Start ab. Beim Zusammenstoß mit Hindernissen ist das Erreichen des Ziels unwahrscheinlich, da sich die Richtung des Roboters ändert. Auch ohne einen Zusammenstoß führen kleine Abweichungen am Anfang zu großen Veränderungen am Ende.

b Linienfolger

Der Roboter besitzt Sensoren, welche die Navigation mittels der Linien am Boden ermöglichen. Er folgt ihnen und erreicht somit das Ziel.

Vorteile

- **Zuverlässigkeit**: Der Roboter misst seine Position relativ zur Linie und kann somit seine Richtung anpassen. Der Erfolg ist nicht so abhängig von der Positionierung an der Startposition wie bei der Nutzung von Odometrie – kleinere Abweichungen werden schnell ausgemerzt.
- **Einfachheit**: Es handelt sich um ein einfaches Prinzip, welches in kurzer Zeit und mit geringem Testaufwand umgesetzt werden kann. Variable Routen können durch Aufteilung in Streckenabschnitte ohne deutlichen Mehraufwand implementiert werden.

Nachteile

- **Langsamkeit**: Der Roboter kann nicht den direkten Weg einschlagen sondern fährt nur auf Linien, was einen deutlich längeren Weg zur Folge hat.
- **Blindheit**: Kurz vor dem Ziel befindet sich eine Strecke ohne Wegmarkierungen. Diese muss „blind” gefahren werden, wobei Ungenauigkeiten unvermeidbar sind.

2 Im Vorjahr benutzte der Roboter „Zicke” diesen Ansatz, dieses Jahr beruhte der Roboter „Flying Knipser” auf diesem Prinzip.

c Wandfolger

Der Roboter misst den Abstand zur Spielfeldbegrenzung/Wand und folgt ihr bis zum Ziel.[3]

Vorteil

- **Schnelligkeit**: Der Roboter hängt nicht von den Markierungen am Boden ab – er fährt nicht blind und kann somit auftretende Abweichungen korrigieren und muss nicht die Umwege, die das Folgen der Linie zur Folge hätten, in Kauf nehmen.

Nachteil

- **Ungenauigkeit**: Die Sensordaten sind nicht akkurat.

4 Konstruktion (Hardware)

a Auswahl einer Idee

Wir haben uns entschieden, einen simplen Linienfolger zu entwickeln. Gründe dafür waren die Robustheit, Fehlertoleranz und das erprobte Konzept[4].

b Unsere ersten Schritte

Zunächst einmal wurde das Getriebe entworfen. Wir wählten am Anfang eine Übersetzung von 1:81, mussten aber schnell feststellen, dass dieser Roboter zum einen zu wenig Kraft besaß und dass der Geschwindigkeitsvorteil nicht sinnvoll genutzt werden konnte. Der Roboter besaß auch eine sehr blockartige[5] Bauweise, welche die Erweiterung erschwerte und unnötiges Gewicht bedeutete. Dies und das nicht ganz reibungslos funktionierende Getriebe veranlassten uns, den Roboter neu zu bauen.[6]

c Neubau

Als Vorbild für die Neuauflage des Roboters dient der „StangInator”. Anstelle einer Blockbauweise wird eine Rahmenbauweise bevorzugt, die es ermöglicht, Platz für die größeren Zahnräder des Getriebes zu gewinnen. Das neue Getriebe besitzt eine Übersetzung von 1:125 und verfügt somit über mehr Kraft.

3 Ein ähnlichen Ansatz wird beim Roboter „Thunder” eingesetzt.
4 Im letzen Jahr gewann der „StangInator” den Wettbewerb, welcher auch Linien folgte.
5 Das Aussehen ähnelte dem eines Ziegelsteins.
6 Einen großen Teil dieses Neubaus erledigte Daniel, welcher insgesamt eher für den Bau unseres Roboters verantwortlich war, wohingegen Benjamin einen Großteil der Programmierung übernahm.

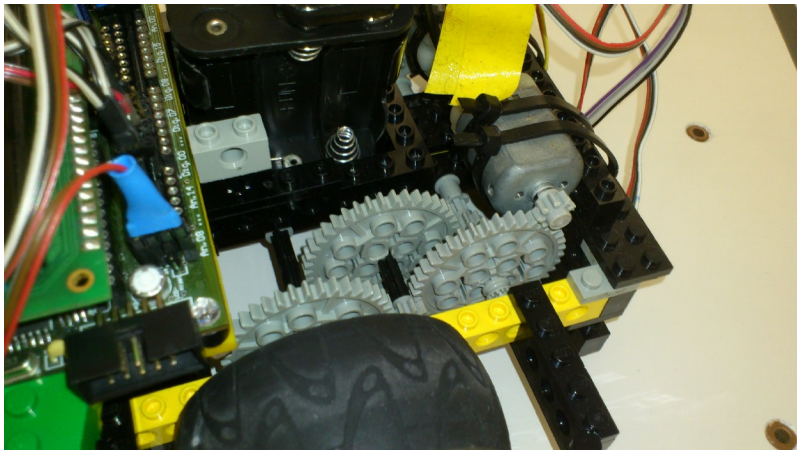


Abbildung 3: Das neue Getriebe mit Motor

Angetrieben werden die zwei Räder durch jeweils einen Motor, der am vorderen Teil des Roboters befestigt ist. Durch die Verwendung von zwei Motoren werden die Räder separat gesteuert. Eine andere Lösungsidee – die Umsetzung eines Differentialgetriebes und die Lenkung durch einen Servomotor – wurde verworfen[7].

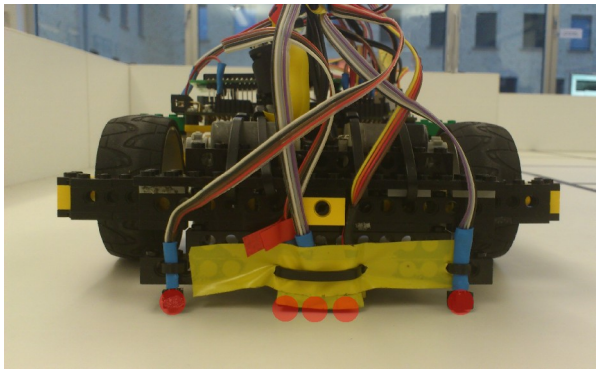


Abbildung 4: Front des Roboters - die fünf Optokoppler sind rot markiert

Fünf Optokoppler wurden zunächst einmal zur Erkennung der Linien verbaut: Drei vorne in der Mitte und jeweils einen links und rechts davon. Am Anfang wurden die Sensoren hinten angebracht, welches jedoch das Folgen einer Linie unmöglich machte. Später musste noch ein Sensor am Ende hinzugefügt werden, damit der Roboter bei der letzten Kurve erfährt, wann er aufhören muss zu drehen[8]. Zu beachten ist dabei, dass die drei Sensoren in der Mitte nicht zu dicht, aber auch nicht zu weit entfernt sind, um eine optimale Linienerkennung zu ermöglichen.

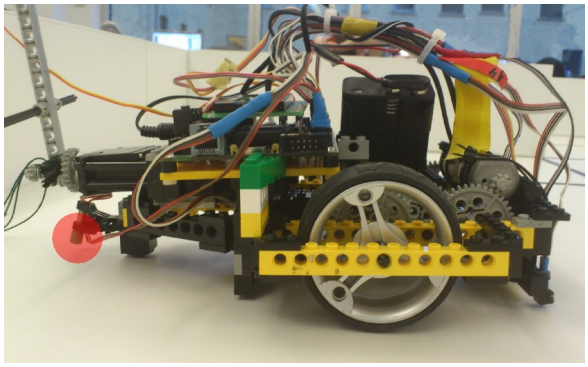


Abbildung 5: Seitenansicht; der Lichtsensor ist rot markiert

7 Der Wenderadius ist zu groß und es ist kein Drehen auf der Stelle möglich. Insgesamt ist ein solcher Roboter nicht so wendig.

8 Das letzte Stück der Strecke ist unmarkiert und beim Drehen auf dieses Stück ist diese Markierung für die vorderen Sensoren nicht mehr sichtbar. Somit wüsste der Roboter nicht, wann die 90° Drehung beendet sei.

Das Lichtsignal, welches den Roboter startet, wird durch einen Lichtsensor am hinteren Ende des Roboterrumpfes erkannt.

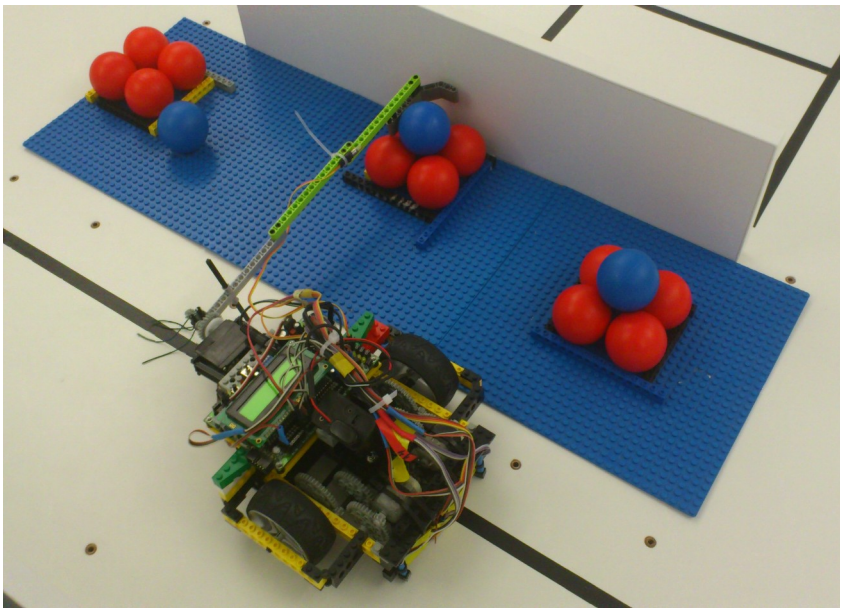


Abbildung 6: Roboter mit dem ersten (zu schweren) Arm

Das Herunterstoßen der Kugeln erfolgt bei uns durch einen etwa 25 cm langen Arm. Betätigt wird dieser mittels eines Servomotors. Im Ruhezustand befindet sich der Arm senkrecht in der Luft und wird je nach Wahl der Route entweder nach links oder rechts heruntergelassen. Der erste Arm, der verwendet wurde, war leider zu schwer und es wird vermutet, dass dieser auch für den besonders hohen Stromverbrauch verantwortlich war. Der zweite Arm ist wesentlich leichter und flexibler. Durch eine Vorrichtung aus Gummi an der unteren Befestigung des Arms zum Servomotor wird ein Ausschlagen des Arms während der Fahrt verhindert.

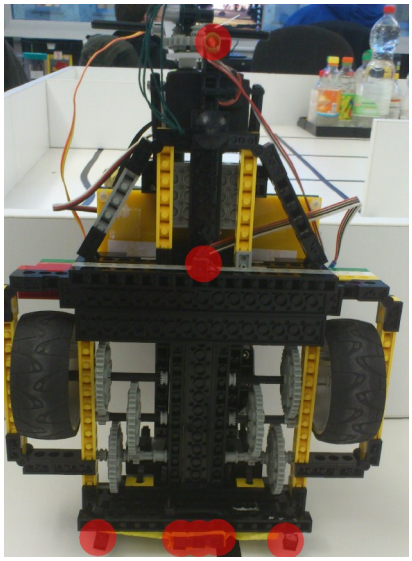


Abbildung 7: Ansicht des Roboters von unten mit markierten Sensoren

5 Programmierung

Das Programm besitzt mehrere Methoden, welche klar abgegrenzte Aufgaben besitzen. Am Anfang wurde eine Methode implementiert, welche das Folgen einer Linie umsetzt. Dabei werden die fünf Sensordaten am vorderen Teil des Roboters ausgewertet und je nach Position der Linie bewegt sich der Roboter geradeaus, nach links oder nach

rechts. Das Besondere an dieser Methode[9] ist, dass sie als Argument die Anzahl der Abschnitte zwischen zwei Kreuzungen besitzt, welche befahren werden. Ein Aufruf mit dem Argument „2” bedeutet, dass er eine Kreuzung überfährt und somit zwei Linien abfährt.

Die Kurvenfahrt wurden auf ähnliche Art und Weise programmiert. Dabei werden die Räder in unterschiedliche Richtungen gedreht. Der Roboter dreht solange, bis er auf eine Linie stößt.

Das Drehen in der letzten Kurve vor der Einfahrt in die „Zielgerade” stellte eine Herausforderung dar. Die Daten von den vorderen Sensoren reichen nicht aus, um zu bestimmen, ab wann der Roboter gerade ausgerichtet ist. Ein zusätzlicher Optokoppler am Ende des Roboters schafft Abhilfe. Die Drehgeschwindigkeit muss aber verringert werden, damit die Drehung zuverlässig funktioniert.

Probleme traten bei der Einfahrt in die „Zielgerade” ein. Da der Roboter im letzten Abschnitt blind fährt, ist die Ankunftsposition von vielen Faktoren abhängig und variiert stark. Insbesondere bei unterschiedlichen Ladeständen der Akkus gab es Schwierigkeiten.

Die Route wird intern als Array von nicht negativen Werten betrachtet. Das Array wird vom Anfang bis zum Ende durchgearbeitet und je nach Wert wird die passende Funktion aufgerufen.

Die Route von der rechten Startposition zum Ziel über den rechten Weg lautet zum Beispiel:

```
char rightWay [23] = {STRAIGHT, RIGHT, STRAIGHT,
STRAIGHT, RIGHT, STRAIGHT, LEFT, STRAIGHT, STRAIGHT,
LEFT, STRAIGHT, STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
RIGHT, STRAIGHT, STRAIGHT, TURN_GOAL_RIGHT, BLIND_FOR-
WARD_RIGHT, LEFT, ARM_LEFT,
STRAIGHT_IGNORE_CROSSINGS };
```

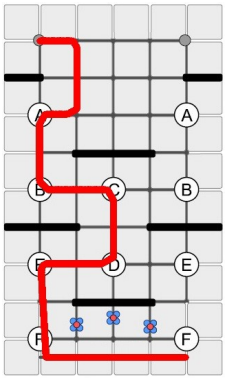


Abbildung 8: Von der rechten Startposition (oben) bis zum Ziel (unten)

Mehrere aufeinanderfolgende Aufrufe von STRAIGHT werden zu einem Aufruf von followLine(..) zusammengefasst.

Die Armbewegung ist ein Teil der Route und wird kurz nach der Einfahrt auf die Zielgerade durchgeführt.

In der Zielgerade gab es noch das Problem, dass neben der Linie angebrachte Schrauben durch die Sensoren als Linie erkannt wurden. Behoben wurde dies, indem bei der letzten Li-

9 Methodenname: „void followLine(int crossings)”

nie die Auswertung der äußeren Optokoppler nicht stattfindet und der Roboter solange fährt, bis er das Ende der Linie erreicht hat.

6 Besonderheiten

Durch die Repräsentation von Routen durch Arrays, die sequentiell abgearbeitet werden, wird es möglich, mehrere Routen auf verschiedene Belegungen der Schalter[10] abzulegen.

In den Roboter eingebaut ist außerdem ein Modus, welcher es erlaubt, die unteren Kugeln aus ihrer Position zu bewegen. Somit wurde es möglich, auch gegen schnellere Roboter ein Unentschieden oder sogar Gewinn zu erzielen. Der Modus wird vor dem Start durch Umlegen eines Schalters am AKSEN-Boards aktiviert und verändert während der Ausführung den Winkel des Arms[11]. Da die Regeln besagen, dass der gegnerische Roboter keine Punkte für das Abräumen erhält, wenn auch die unteren Kugeln aus ihrer ursprünglichen Position entfernt wurden, hätte im Idealfall kein Roboter gegen uns gewinnen können[12].

Es ist uns also möglich, für jeden Gegner eine passende Strategie auszuwählen.

Der Name unseres Roboters „Scared Lizard”[13] entstammt übrigens der Zeit, in der es häufig vorkam, dass der bewegliche Arm abfiel[14].

7 Wettbewerb und Probleme

Der Wettbewerb verlief für uns sehr gut. Zunächst einmal muss angemerkt werden, dass verschiedene Modi zur Auswahl standen, jedoch vergessen wurde, einen Modus einzubauen, welcher von der rechten Startposition über den linken Weg zum Ziel gelangt.

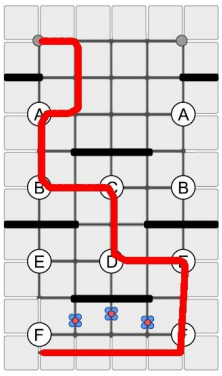


Abbildung 9: Nicht implementierter Modus

Wenn wir gegen einen schnelleren Roboter antraten, erreichte dieser glücklicherweise entweder das Ziel nicht oder wir starteten von der linken Position und konnten somit zwischen den zwei Modi auswählen.

Die Akkumulatoren, die in vorherigen Versuchen immer Probleme bereiteten, hielten durch. Bei der ersten Fahrt ohne gegnerischen Roboter wurde aus taktischen Gründen ein schwacher Akkumulator verwendet, weil es

10 Gemeint sind die Schalter auf dem AKSEN-Board, welche mit dip_pin($i \in 0,1,2,3$) abgefragt werden.
11 Wir nannten den Modus am Anfang „Destruction-Mode”, später dann „Panic-Mode” in Anlehnung an unseren Robotern amen.
12 Bei hundertprozentiger Zuverlässigkeit und falls keine Zusammenstöße im Labyrinth auftreten.
13 Zu deutsch „Ängstliche Eidechse”
14 Bedrohte Eidechsen können in größter Not den Schwanz abstoßen (Autotomie), siehe http://www.zeit.de/2005/39/Stimmts_39 (14.01.2012)

nicht auf Geschwindigkeit ankam und der Roboter auch größtenteils mit schlechten Akkus getestet wurde und damit am zuverlässigsten lief. Während des weiteren Verlaufs des Wettbewerbs nutzten wir einen neuen, aufgeladenen Akku und dieser musste nicht ausgetauscht werden.

Während des Wettbewerbs fuhr der Roboter immer erfolgreich durch das Labyrinth, blieb aber zweimal kurz nach dem Erreichen der Zielgerade stehen und drehte nicht weit genug. Dies wird wahrscheinlich durch die Methode

```
void turn(int left)
```

verursacht, die nicht für das orthogonale Auftreffen auf eine Linie und das Einlenken auf selbige ausgelegt ist. Abhilfe könnte die Verwendung einer separaten Methode schaffen.

Ein weiteres Problem, welches nicht im Wettbewerb auftrat, ergibt sich bei unterschiedlicher Aufladung des Akkumulators. In einigen Fällen verfehlt der Roboter dann während der blinden Fahrt die Zielgerade und bleibt an der Wand stehen.

In seltenen Fällen trat während der Proben ein Problem mit zu stark aufgeladenen Akkumulatoren auf. Der Roboter bewegte sich zu hektisch und erreichte in manchen Durchläufen nicht einmal die Zielgerade.

8 Vorschläge

Es steht außer Frage, dass der Roboter in seiner jetzigen Form nicht perfekt ist. Es mangelt an Robustheit. Die Sensoren könnten besser platziert werden, welches beim aktuellen Modell leider nicht möglich ist. Idealerweise befinden sich die Optokoppler etwas näher an der Achse[15]. Der schwenkbare Arm ist nicht gut befestigt. Eventuell wäre es besser, einen Arm zu entwerfen, welcher durch das Auslösen des Servomotors herunterfällt.

Der Übergang vom Labyrinth zu der Zielregion erfolgt blind. Der Abstand zwischen Roboter und Wand könnte helfen, den Roboter zuverlässig und ohne Abhängigkeit vom Zustand des Akkus zur Zielregion zu navigieren.

15 Somit könnte der sechste Optokopplers am Ende wegfallen und es müsste nicht eine spezielle Methode verwendet werden.

9 Zusammenfassung

Verbaut wurden im Roboter folgende Teile:

- Sechs Optokoppler (Fünf am vorderen Teil, einen hinten)
- Einen Lichtsensor (am hinteren Teil des Roboters)
- Zwei Motoren (links und rechts vorne, treiben jeweils ein Rad an)
- Einen Servomotor mit angebrachtem beweglichen Arm
- Zwei Räder
- Gerüst aus Lego

Das Programm weist folgende Eigenschaften auf:

- Eine Quellcodedatei (reines C)
- 429 Zeilen
- 18 Methoden
- 29 Makros (Konstanten, mit welchen das Verhalten des Roboters angepasst werden kann, z.B. Geschwindigkeit oder Ports der Sensoren/Aktoren)

Während des Wettbewerbs standen uns folgende Modi zur Verfügung[16]:

1. Der Roboter wartet auf ein Lichtsignal, bevor er mit der Abarbeitung des restlichen Programmes beginnt[17][18]
2. Der Roboter startet von links und nimmt den linken Weg zum Ziel.
3. Der Roboter startet von links und nimmt den rechten Weg zum Ziel.
4. Der Roboter gerät in Panik und stößt am Ende alle Kugeln aus ihrer Position, sodass auch der Gegner keine Punkte bekommen[19]

16 Die Auswahl erfolgte mit Hilfe der vier Schalter auf dem AKSEN-Board.
17 Dies ist eigentlich unnötig. Besser wäre eine Route, die von der rechten Startposition über die linke Seite zum Ziel führt.
18 Wurde im Wettkampf immer eingesetzt. Eigentlich nur fürs Testen gedacht.
19 Kann in Kombination mit den anderen Modi eingesetzt werden.

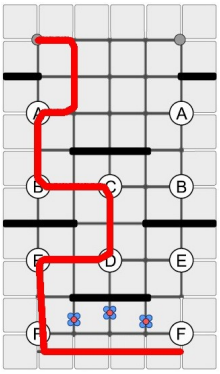


Abbildung 10: Route, falls Pin 2 und 3 nicht gesetzt

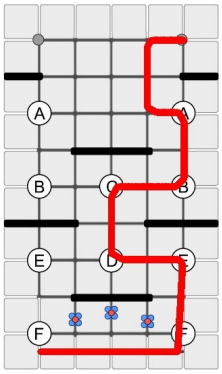


Abbildung 11: Route, falls Pin 2 gesetzt

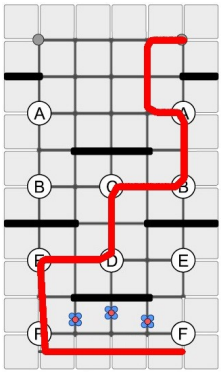


Abbildung 12: Route, falls Pin 3 gesetzt

Abschließend lässt sich sagen, dass es uns mit „Scared Lizard” gelungen ist, einen sehr konservativen und minimalistischen „Linienfolger”-Roboter zu bauen. Er ist vergleichsweise klein, relativ wendig und einigermaßen robust. An der Zuverlässigkeit lässt sich noch arbeiten.

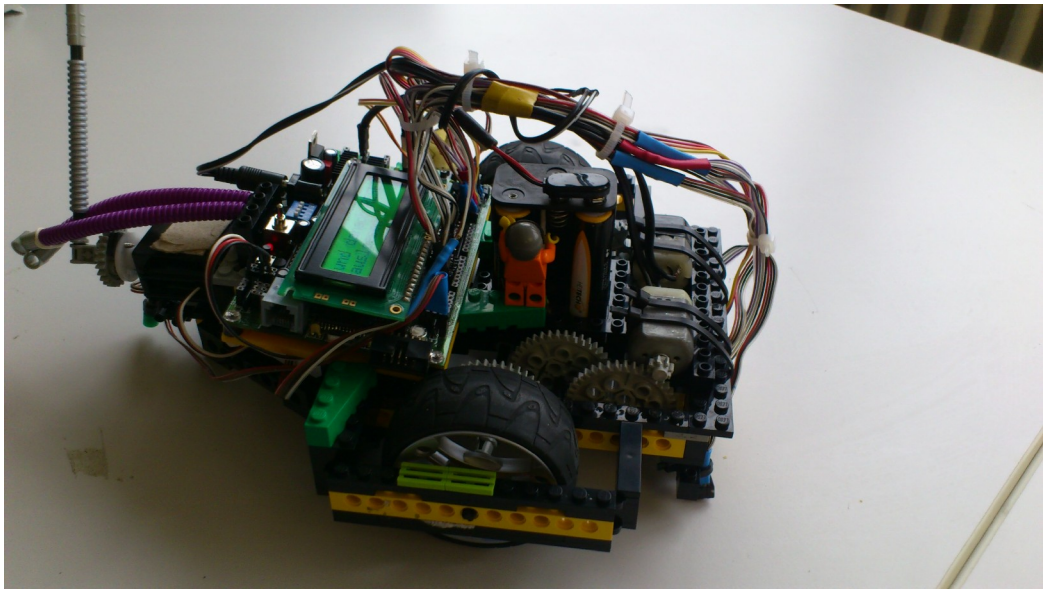


Abbildung 13: Finale Version

10 Anhang

```
//Autoren: Daniel Schmidt, Benjamin Hoffmann

//Standard-Include-Files
#include <stdio.h>

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

// engine ports
#define ENGINE_PORT_LEFT 3
#define ENGINE_PORT_RIGHT 1

// count of line detectors (starting @port 0)
#define SENSOR_COUNT 5

// light sensor port (only used to start roboter)
#define LIGHT_SENSOR_PORT 8

#define ACTIVATE_LED 1

// speed for slight turns
#define SPEED_MAX 10
#define SPEED_MIN 0

// speed for actual turns
#define SPEED_TURN_MIN 5
#define SPEED_TURN_MAX 10

// speed for turn right before the atoms
#define SPEED_TURN_GOAL_MIN 3
#define SPEED_TURN_GOAL_MAX 6

#define RIGHT 0
#define LEFT 1
#define STRAIGHT 2
#define TURN_GOAL_RIGHT 3
#define TURN_GOAL_LEFT 4
#define BLIND_FORWARD_RIGHT 5
#define BLIND_FORWARD_LEFT 6
#define ARM_LEFT 7
#define ARM_RIGHT 8
#define STRAIGHT_IGNORE_CROSSINGS 9
```

```
#define SERVO_PORT 0
// angles for arm
#define SERVO_UP 90
#define SERVO_LEFT_DESTRUCTION 140
#define SERVO_LEFT 130
#define SERVO_RIGHT_DESTRUCTION 5
#define SERVO_RIGHT 13

#define SLEEP_TIME_TURN 0 /* applied before turning process*/
#define SLEEP_TIME_BLINDFORWARD_AFTER 50
/* the worse the battery the higher the value must be*/

void followWay(char *a, int n);
void waitForStart();
void followLine(int crossings);
void turn(int left);
void showSensorInfo();
void stop();
void turnTowardsGoal(int left);
void blindForward(int left);

void armUp();
void armLeft();
void armRight();

//Hauptprogrammroutine
void AksenMain(void)
{
     char rightWay [23]= {STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
RIGHT, STRAIGHT, LEFT, STRAIGHT, STRAIGHT, LEFT, STRAIGHT, STRAIGHT,
RIGHT, STRAIGHT, STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
TURN_GOAL_RIGHT, BLIND_FORWARD_RIGHT, LEFT, ARM_LEFT, STRAIGHT_IGNO-
RE_CROSSINGS };

     char leftWay [23]= {STRAIGHT, LEFT, STRAIGHT, STRAIGHT, LEFT,
STRAIGHT, RIGHT, STRAIGHT, STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
LEFT, STRAIGHT, STRAIGHT, LEFT, STRAIGHT, STRAIGHT, TURN_GOAL_LEFT,
BLIND_FORWARD_LEFT, RIGHT, ARM_RIGHT, STRAIGHT_IGNORE_CROSSINGS };

     char leftWay2 [23]= {STRAIGHT, LEFT, STRAIGHT, STRAIGHT, LEFT,
STRAIGHT, RIGHT, STRAIGHT, STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
LEFT, STRAIGHT, STRAIGHT, RIGHT, STRAIGHT, STRAIGHT,
TURN_GOAL_RIGHT, BLIND_FORWARD_RIGHT, LEFT, ARM_LEFT, STRAIGHT_IGNO-
RE_CROSSINGS };



     led(0,1);

     if (dip_pin(0))
          waitForStart();
```

```
	/*if(dip_pin(3)) {
		showSensorInfo();
	}*/

	// START:

	armUp();

	if (!dip_pin(1) && !dip_pin(2))
		followWay(rightWay, 23);
	else if (!dip_pin(2))
		followWay(leftWay, 23);
	else
		followWay(leftWay2, 23);

	stop();
	armUp();

	// Die folgende Endlosschleife "erzeugt" das Programmende
	while(1);
}

// Folgt einem Weg (erwartet einen Char-Array und die Anzahl der ab-
zuarbeiteten Wegpunkte
void followWay(char *a, int n)
{
	int countCrossings;
	while(n > 0)
	{
		switch (*a)
		{
			case LEFT: turn(LEFT); a++; n--; break;
			case RIGHT: turn(RIGHT); a++; n--; break;
			case STRAIGHT:
				countCrossings = 0;
				while(n > 0 && *a == STRAIGHT) {
					a++;
					n--;
					countCrossings++;
				}
				followLine(countCrossings);
				break;
			case TURN_GOAL_RIGHT: turnTowardsGoal(RIGHT);
					a++; n--; break;
			case TURN_GOAL_LEFT: turnTowardsGoal(LEFT);
					a++; n--; break;
			case BLIND_FORWARD_RIGHT: blindForward(RIGHT);
					a++; n--; break;
			case BLIND_FORWARD_LEFT: blindForward(LEFT);
					a++; n--; break;
			case ARM_LEFT: armLeft(); a++; n--; break;
```

```
                case ARM_RIGHT: armRight(); a++; n--; break;
                case STRAIGHT_IGNORE_CROSSINGS: followLine(-1);
                                                a++; n--; break;
            }
        }
}

// waits for a light signal
void waitForStart()
{
      while(analog(LIGHT_SENSOR_PORT) > 30);
}

// Gibt Sensorinformationen aus
void showSensorInfo()
{
      unsigned char i;

      while(1)
      {
            for (i = 0; i < SENSOR_COUNT; ++i)
            {
                  lcd_ubyte(analog(i));
                  lcd_puts(" ");
            }

            sleep (40);
            lcd_cls();
      }
}
```

```
// Some methods for controlling the robot's engines.
// 0 - forward, 1 - backwards
void setEngineDirection(unsigned char left, unsigned char right)
{
     motor_richtung(ENGINE_PORT_LEFT,left);
     motor_richtung(ENGINE_PORT_RIGHT,right);
}

// speed between 0 (slow) and 10 (fastest)
void setSpeed(unsigned char left, unsigned char right) {
     motor_pwm(ENGINE_PORT_LEFT, left);
     motor_pwm(ENGINE_PORT_RIGHT, right);
}

// sets speed for both engines to 10 (fastest) without changing the
direction
void setMaxSpeed()
{
     if (ACTIVATE_LED)
     {
          led(0,1);
          led(1,1);
     }

     motor_pwm(ENGINE_PORT_LEFT, SPEED_MAX);
     motor_pwm(ENGINE_PORT_RIGHT, SPEED_MAX);
}


// engines halt
void stop()
{
     if (ACTIVATE_LED)
     {
          led(0,0);
          led(1,0);
     }

     motor_pwm(ENGINE_PORT_LEFT, 0);
     motor_pwm(ENGINE_PORT_RIGHT, 0);
}
```

```
// turns slightly towards left
void turnSlightlyLeft()
{
	if (ACTIVATE_LED)
	{
		led(0,1);
		led(1,0);
	}
	motor_pwm(ENGINE_PORT_LEFT, SPEED_MIN);
	motor_pwm(ENGINE_PORT_RIGHT, SPEED_MAX);
}

// turns slightly towards right
void turnSlightlyRight()
{
	if (ACTIVATE_LED)
	{
		led(0,0);
		led(1,1);
	}
	motor_pwm(ENGINE_PORT_LEFT, SPEED_MAX);
	motor_pwm(ENGINE_PORT_RIGHT, SPEED_MIN);
}

/****************
Returns direction, used for follwing a line:
0 - TURN SLIGHTLY LEFT
1 - GO STRAIGHT FORWARD
2 - TURN SLIGHTLY RIGHT
3 - CROSSING
255 - WHITE SPACE - nothing to follow =(
*****************/
unsigned char returnDirection(unsigned char *s)
{
	if (s[3] > 100 || s[4] > 100)
		return 3;
	else if (s[0] < 100 && s[1] > 160 && s[2] < 100)
		return 1;
	else if (s[0] <  50 && s[1] <  50 && s[2] <  50)
		return 255;
	else
		return s[0] > s[2] ? 2 : 0;
}
```

```
/****************
Robot follows a line for a certain amount of crossings
****************/
void followLine(int crossings)
{
      unsigned char i, direction = 1, lastDirection;
      unsigned char lichtSensor[SENSOR_COUNT];

      setEngineDirection(0,0);
      setMaxSpeed();

      do
      {
            for (i = 0; i < SENSOR_COUNT; ++i)
                  lichtSensor[i] = analog(i);

            lastDirection = direction;
            direction = returnDirection(lichtSensor);

            if (lastDirection > 2 && direction <= 2) {
                  /* 3 means crossing */
                  crossings--;
            }

            if (direction == 0)
            {
                  turnSlightlyLeft();
            } else if (direction == 2)
            {
                  turnSlightlyRight();
            } else {
                  setMaxSpeed();
            }
      } while (crossings > 0 && direction != 255 ||
                  (crossings < 0 && direction != 255));
}


// turns left or right depending on parameter
void turn(int left)
{
      unsigned char i;
      unsigned char lichtSensor[SENSOR_COUNT];

      int total, hasStarted = 0;

      // because of sensor positions one has to be in a certain  //
distance from the crossing so go on for a bit farther
      sleep(SLEEP_TIME_TURN);

      if (left == LEFT) {
            setEngineDirection(1,0);
```

```
            setSpeed(SPEED_TURN_MIN, SPEED_TURN_MAX);
      } else {
            setEngineDirection(0,1);
            setSpeed(SPEED_TURN_MAX, SPEED_TURN_MIN);
      }

      do
      {
            for (i = 0; i < SENSOR_COUNT; ++i)
                  lichtSensor[i] = analog(i);

            total =  lichtSensor[0] + lichtSensor[1]
                        + lichtSensor[2];

            if (total < 50) {
                  led(0,0);
                  hasStarted = 1;
            }

      } while (!hasStarted || total < 250);

      led(0,1);
}

// turns toward goal (there is no line ahead, so get
// sensor information from behind!)
void turnTowardsGoal(int left) {
      unsigned char i;
      unsigned char lichtSensor[SENSOR_COUNT], stopLichtSensor;

      int total, hasStarted = 0;

      if (left == RIGHT) {
            setEngineDirection(1,0);
            setSpeed(SPEED_TURN_GOAL_MIN, SPEED_TURN_GOAL_MAX);
      }
      else {
            setEngineDirection(0,1);
            setSpeed(SPEED_TURN_GOAL_MAX, SPEED_TURN_GOAL_MIN);
      }

      do
      {
            for (i = 0; i < SENSOR_COUNT; ++i)
                  lichtSensor[i] = analog(i);

            total = lichtSensor[0] + lichtSensor[1]
                        + lichtSensor[2];

            stopLichtSensor = analog(5);

            if (total < 50 && stopLichtSensor < 30) {
```

```
                led(0,0);
                hasStarted = 1;
           }

     } while (!hasStarted || stopLichtSensor < 100);
     led(0,1);
     stop();
}

// blindly follow an imaginery line until you can see an
// existing line
void blindForward(int left)
{
     unsigned char i;
     int total;

     setEngineDirection(0,0);

     // TODO: tuning
     if (left == RIGHT)
          setSpeed(10,10);
     else
          setSpeed(9,10);

     do
     {
          total = 0;

          for (i = 0; i < SENSOR_COUNT; ++i)
               total += analog(i);
     } while (total < 250);

     sleep(SLEEP_TIME_BLINDFORWARD_AFTER);
     stop();
}

// arm goes up
void armUp() {
     servo_arc(SERVO_PORT, SERVO_UP);
}

// arm towards left
void armLeft() {
     if (dip_pin(3)) // DESTRUCTION MODE!
          servo_arc(SERVO_PORT, SERVO_LEFT_DESTRUCTION);
     else
          servo_arc(SERVO_PORT, SERVO_LEFT);

}

// arm towards right
void armRight() {
```

```
	if (dip_pin(3)) // DESTRUCTION MODE!
		servo_arc(SERVO_PORT, SERVO_RIGHT_DESTRUCTION);
	else
		servo_arc(SERVO_PORT, SERVO_RIGHT);
}
```