HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Studienarbeit

Künstliche Intelligenz

mit Lego Mindstorms Robotern

Beat Wieland
bwieland@hsr.ch

Daniel Bögli
dboegli@hsr.ch

Michael Remund
mremund@hsr.ch

Urs Heimann
uheimann@hsr.ch

6. Februar 2003

Betreuer: Prof. Dr. J. Joller

Letzte Änderung am 6. Februar 2003.
Dieses Dokument wurde mit LaTeX gesetzt.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Künstliche Intelligenz, Robotik und Swarm Intelligence sind Schlagwörter die man in letzter Zeit sehr oft hört. Es sind aktuelle Themen, die sich im Moment sehr schnell entwickeln. Sich damit zu beschäftigen, die Entwicklung zu verfolgen und zu verstehen macht Spass!

Diese Semesterarbeit gibt uns die Möglichkeit, in unserem Informatikstudium einmal etwas mit Hardware zu machen und die Lego Mindstorms Produkte auf Herz und Nieren zu testen.

1.2. Anforderungen

Die Projektanforderungen sind relativ offen formuliert. Die gesamte Studienarbeit ist in mehrere Unterprojekte gegliedert.

Das erste Projekt befasst sich mit den *Vehikeln von Braitenberg*. Zu jedem dieser Vehikel soll ein konkretes Beispiel implementiert werden.

Der zweite Teil besteht aus einer Implementierung eines Beispiels zum Thema *Jini*. Jini ist eine auf RMI basierende Technologie von Sun.

Der letzte Teil der Studienarbeit befasst sich mit *Swarm Intelligence*. Dazu soll ein Beispiel eines einfachen Algorithmus implementiert und dokumentiert werden.

Die *Dokumentation* dieser Arbeit soll Interessierten zeigen, wie die Theorien konkret umgesetzt werden können.

1.3. Ziele

Wir wollen uns mit dieser Studienarbeit in das komplexe Thema der Robotik einarbeiten. Wir möchten uns dabei einen Überblick über diese Thematik verschaffen und vor allem etwas Praktisches realisieren. Es ist also nicht das Ziel, Theorien und Formeln herzuleiten.

Auch soll diese Dokumentation dem Leser und der Leserin einen Einblick in die künstliche Intelligenz und die Robotik ermöglichen.

1.4. Dokumentaufbau, Gliederung

Dieses Dokument ist wie folgt gegliedert:

- In Kapitel 2 wird auf die verwendete Hard- und Software eingegangen.
- Das Kapitel 3 ist den Braitenberg Vehikeln gewidmet. Hier werden konkrete Implementationen der verschiedenen Vehikel vorgestellt.
- In Kapitel 4 ist ein kleines Projekt mit Jini zu finden.
- Unser Projekt zum Thema Swarm Intelligence wird im Kapitel 5 vorgestellt.
- Vor dem Anhang befinden sich der Zeitplan und die persönlichen Berichte der Teammitglieder.
- Im Anhang befindet sich der Source Code der Projekte, ein Glossar mit Erklärungen zu Abkürzungen und speziellen Begriffen und je ein Verzeichnis für die verwendeten Hilfsmittel und die Literatur.

2. Evaluation

Für diese Studienarbeit werden die Mindstorms Produkte von Lego verwendet. In diesem Kapitel wird die zur Verfügung stehende Hardware auf ihre Praxistauglichket getestet, und es werden einige wichtige Eckdaten zusammengefasst. Danach folgt eine Beschreibung der Software, die mit der Hardware zusammen verwendet werden kann.

2.1. Hardware

Weiterführende Informationen zur Hardware der Lego Mindstorms Produkte finden sich auf den Homepages Lego Mindstorms Internals[6] und RCX Sensor Input Page[7].

2.1.1. RCX

Der RCX ist das Herz eines jeden Lego Roboters. Die technischen Daten des RCX sind:

- Der RCX hat drei Eingänge für Sensoren und drei Ausgänge. Es ist grundsätzlich möglich mehrere Sensoren an einem Eingang zu verwenden. Dazu mehr im Teil über Sensoren in Kapitel 2.1.4.
- Als Prozessor verwendet er einen Hitachi-Mikrokontroller (HD6433292), der mit 5 MHz getaktet ist und 5V Betriebsspannung benötigt.
- Der Prozessor hat 16kB ROM und 512B RAM. Dazu kommen noch 32kB externes RAM. Wieviel Speicher man für eigene Programme verwenden kann, hängt davon ab, was für eine Firmware verwendet wird. Dazu mehr im Abschnitt 2.2.
- Weiter verfügt der RCX über Timer, Counter, einen Speaker zur Tonausgabe und eine Infrarotschnittstelle zur Kommunikation mit dem IR-Tower oder einem andern RCX.

2.1.2. IR-Tower

Um Daten, z. B. die auf dem PC geschriebenen Programme, auf den RCX zu übertragen, wird eine Infrarot-Verbindung benötigt. Inzwischen gibt es zwei verschiedene IR-Tower, die für diesen Zweck verwendet werden können. Der eine wird an die serielle Schnittstelle des Computers angeschlossen, der andere an den USB Port.

Der IR-Tower muss mit 9V versorgt werden. Die serielle Variante verwendet eine 9V Batterie, der USB-Tower benötigt keine separate Batterie, da er über die USB Schnittstelle gespiesen wird. Die Reichweite beträgt cirka 0.1 bis 2.5m.

2.1.3. Fernbedienung

Im Ultimate Accessory Set befindet sich auch ein Fernbedienung, mit der man dem RCX Befehle erteilen kann. Ist ein nettes Spielzeug, wir haben sie aber nicht produktiv eingesetzt.

2.1.4. Sensoren

Lego bietet verschiedene Sensoren an:

- Drucksensor
- Lichtsensor
- Rotationssensor
- Temperatursensor

Die Sensoren liefern einen Wert zwischen 0 und 5 Volt. Dieser Wert wird in einen sogenannten RAW Wert umgerechnet. 0V entspricht 0 und 5V entsprechen 1023. Dieser RAW Wert wird in den für den betroffenen Eingang gewählten Sensor-Typ umgerechnet. (folgende Tabellen sind auf RCX Sensor Input Page[7] zu finden)

Behrührungssensor	$RAW < 450 \Rightarrow touch = 1$ $RAW > 565 \Rightarrow touch = 0$	Taster gedrückt Taster nicht gedrückt
Lichtsensor	$light = (1022 - RAW)/7$	für 0 (dunkel) bis 100 (hell)
Temperatursensor	$temp = (785 - RAW)/8$	für -20 bis $+70$ Grad Celsius

Die folgende Tabelle zeigt einige Beispielwerte für die Licht, Temperatur und Berührungs-Sensoren.

Volt	RAW	*light*	*temp*	*touch*
0.0	0	-	-	1
1.1	255	-	70	1
1.6	322	100	57.9	1
2.2	450	82	41.9	1
2.8	565	65	27.5	0
3.8	785	34	0.0	0
4.6	945	11	-20	0
5.0	1023	0	-	0

Der *Rotationssensor* liefert pro Umdrehung 16 Impulse. Das heisst alle 22.5 Grad einen Impuls. Es wird zwischen vor und zurückdrehen unterschieden. Der Zählbereich ist -32767 bis $+32767$. Damit der Sensor richtig funktioniert, darf die Drehzahl höchstens 500 Umdrehungen pro Minute betragen.

Im Internet gibt es sehr viele Anleitungen zum Eigenbau von Sensoren. Diese sind meist viel günstiger als die originalen Sensoren von Lego. Man findet auf diesen Seiten meist auch einige Tipps wie man mehrere Sensoren an einem Eingang verwenden kann. Auf der RCX Sensor Input Page[7] hat es eine umfangreiche Sammlung mit Vorschlägen.

2.1.5. Aktoren

Als Aktoren stehen zur Verfügung:

- Motor
- Lampe

2.1.6. Lego

Das Prinzip von Lego eignet sich sehr gut für den Bau von Robotern, da diese sehr einfach zusammengebaut werden können und keine mechanische Werkstatt benötigt wird.

Mit den Bausteinen von Lego Technik ist es relativ einfach, stabile Konstruktionen zu erstellen. Ausführliche Erklärungen und Tipps zum Bau von Robotern mit Lego gibt das Buch Building Robots [1]. Dort werden unter anderem die geometrischen Grundlagen der Legobausteine, Übersetzungen mit Zahnrädern, Verstrebungstechniken, aber auch Themen wie Differentialgetriebe und Gangschaltung behandelt. Es ist ein sehr empfehlenswertes Buch für alle, die öfters mit Lego Technik zu tun haben.

2.2. Software

Lego liefert mit dem RCX auch Software zur Programmierung mit. Es ist aber möglich andere Programmiersprachen zu verwenden, da sich die Firmware des RCX im RAM befindet und somit ausgewechselt werden kann.

Im Folgenden werden die verschiedenen erhältlichen Firmwares und Programmiersprachen kurz vorgestellt und kurz deren Vor- und Nachteile erläutert.

Im Anhang unter dem Punkt Hilfsmittel sind die verwendeten Programme, mit einem Hinweis wo die aktuellste Version bezogen werden kann, nochmals aufgeführt.

2.2.1. RIS Software

Die von Lego mitgelieferte Entwicklungsumgebung ist sehr einfach zu bedienen. Man kann hier in einer graphischen Umgebung programmieren, indem man vorgefertigte Blöcke wie Legosteine aneinander klebt. Es gibt Blöcke für Schlaufen (while), für Entscheidungen (if), aber auch solche um Motoren einzuschalten oder Sensorwerte einzulesen. So kann man den Programmablauf in einer Art Flussdiagramm darstellen.

Als Vorteil könnte noch erwähnt werden, dass der Download (PC zu RCX) der selbst geschriebenen Programme viel schneller als bei leJOS von statten geht.

Wie schon erwähnt, lassen sich mit dieser Entwicklungsumgebung schnell und einfach kleine Programme schreiben; man hat damit aber auch nur sehr beschränkte Möglichkeiten. Daher eigenet sich diese Software nicht für unsere Projekte.

2.2.2. NQC

NQC verwendet dieselbe Firmware wie die von Lego mitgelieferte Software. Für eigene Programme stehen also 6kB Speicherplatz zur Verfügung. NQC ist eine Abkürzung für

"Not Quite C". Wie der Name sagt, ist diese Programmiersprache ein vereinfachte Version von C. Es lassen sich damit schon komplexere Programme schreiben. Der Download der Programme nimmt auch hier ziemlich wenig Zeit in Anspruch. Allerdings hat man noch nicht soviele Möglichkeiten wie mit brickOS oder leJOS.

2.2.3. brickOS, legOS

Mit brickOS (füher legOS) kann man den RCX mit C oder C++ programmieren. Der Code wird auf dem PC geschrieben und mit einem Crosscompiler für den Hitachiprozessor des RCX kompiliert. Nach dem Kompiliervorgang muss man den lauffähigen Code auf den RCX laden.

2.2.4. leJOS

Wie sein Vorgänger (TinyVM), ist leJOS eine kleine virtuelle Maschine. Mit LeJOS kann der RCX mit einigen Einschränkungen mit Java programmiert werden. Für Hardware-Komponennten, wie Motoren oder Sensoren, werden Klassen zur Verfügung gestellt, die das Programmieren erleichtern. Die VM benötig 16kB Speicherplatz. Für Programme können die restlichen 16kB gebraucht werden. Die ältere TinyVM benötigt nur 10kB Speicher. Als einzig wirklichen Nachteil von leJOS kann der ziemlich langwierige Downloadprozess der Programme vom PC zum RCX aufgeführt werden.

2.2.5. Entscheid

Für unsere Projekte kamen eigentlich nur brickOS und leJOS in Frage. Da leJOS viel einfacher zu handhaben und mehr Dokumentation dazu vorhanden ist, haben wir uns für die Java-Variante (leJOS Version 2.0.0) entschieden.

Als Entwicklungsumgebung benutzen wir *Eclipse* in der Version 2.0.2.

3. Braitenberg Vehikel

In diesem Kapitel werden die Implementationen der Braitenberg Vehikel 1-5 vorgestellt. Die theoretischen Grundlagen sind aus Braitenbergs Buch Vehicles [3]. Die vollständigen Source Codes der jeweiligen Klassen sind im Anhang zu finden.

3.1. Einführung

Braitenberg beschreibt Maschinen mit einer sehr simplen inneren Struktur. Interessant wird es jedoch, wenn man diese Maschinen oder Vehikel betrachtet als wären es Tiere in einer natürlichen Umgebung. Das heisst, wenn man nichts von der Programmierung dieser Vehikel weiss, erscheint es so, als handeln sie intelligent, als würden sie von ihrer Umgebung beeinflusst oder als ob sie bestimmte Ziele zu erreichen versuchten.

3.2. Vehikel 1 (Herumfahren)

3.2.1. Definition

Das Braitenberg Vehikel 1 ist ein relativ einfacher Roboter, der auf Umwelteinflüsse reagiert. Er besitzt lediglich einen Sensor und einen Motor. Je stärker der Sensor stimuliert wird, desto schneller dreht sich der Motor.

Weiss man nicht wie dieses erste Vehikel aufgebaut ist, hat man das Gefühl, dass es helle Orte nicht mag und immer beschleunigt um von einem solchen wegzukommen. Es scheint zwar ein relativ dummes Tier zu sein, aber es lebt definitiv, da etwas totes sich kaum so verhalten würde!

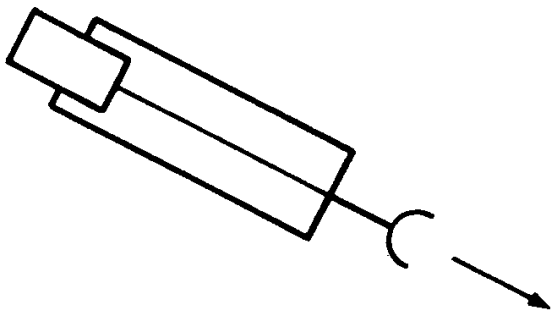

Abbildung 3.1.: Originalbild von Braitenberg

3.2.2. Hardware-Modell

Als Sensor dient ein nach oben gerichteter Hellikeitssensor. Der Antriebsmotor wird in direkter Abhängigkeit des Sensormesswertes angesteuert. Je stärker die Lichtintensität am Ort des Fahrzeugs ist, desto schneller bewegt sich dieses vorwärts. Erreicht es auf seinem Weg einen Ort der genügend dunkel ist, bleibt es stehen.

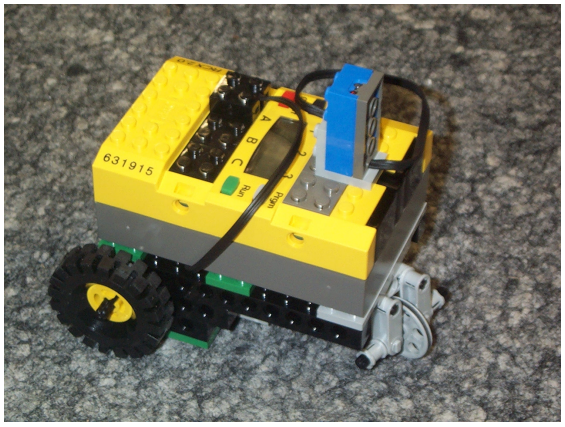

Abbildung 3.2.: Braitenberg Vehikel 1

3.2.3. Implementation

Beschreibung

Die Klasse Vehicle01 implementiert das Verhalten. Die Klasse PWM stellt eine pulsweitenmodulierte Motoransteuerung zur Verfügung. Die Ansteuerung der Motoren mit PWM ist notwenig, da mit den von der Virtual Machine zur Verfügung gestellten Methoden die Motoren nicht langsam gedreht werden können. Die Motorgeschwindigkeit wird mit einer linearen Funktion berechnet. Die Funktion hat die folgende Form:

$$MotorGeschwindikeit : Helligkeit \rightarrow (Helligkeit - DunkelWert) \cdot Steilheit$$

Im Labor mit Beleuchtung haben sich die Werte 22 für DunkelWert und 4 für die Steilheit bewährt.

Methode hideFromLight der Klasse Vehicle01

```
public void hideFromLight()
{
    int value;
    leftMotor.start();
    rightMotor.start();
    leftMotor.startMotor();
    rightMotor.startMotor();
```

```
    while (Button.readButtons() == 0)
    {
       // Sensorwert einlesen
       value = lightSensor.readValue();
       Thread.yield();
       printValue(value);

       // Motorgeschwindigkeit setzten
       leftMotor.setSpeed((value - DARKNESS) * GRADIENT);
       rightMotor.setSpeed((value - DARKNESS) * GRADIENT);
    }

    // Programmende
    lightSensor.passivate();
    leftMotor.endThread();
    rightMotor.endThread();
}
```

3.3. Vehikel 2 (Angst und Aggression)

3.3.1. Definition

Mit zwei Sensoren und zwei Motoren ausgerüstet, reagiert das Braitenberg Vehikel 2 bereits sehr flexibel auf Umwelteinflüsse. Das Prinzip ist dasselbe wie bei Vehikel 1: Bei stärkerer Stimulation des Sensors wird der zugehörige Motor schneller betrieben. Im Prinzip ist Vehikel 2 nichts anderes als zwei aneinander gewachsene Vehikel des Typs 1.

Je nach Verbindung der Sensoren mit den Motoren steuert das Fahrzeug auf die Quelle zu (Abbilung 3.3 links) oder von ihr weg (Abbilung 3.3 rechts).

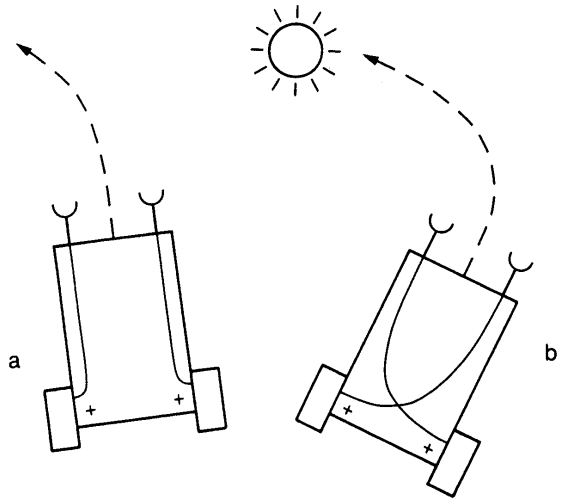

Abbildung 3.3.: Originalbild von Braitenberg

3.3.2. Hardware-Modell

Das erste Fahrzeug wird mit einem weiteren Hellikeitssensor ergänzt. Jeder Motor wird nun von einem der nach vorne gerichteten Sensoren gesteuert. Der linke Sensor steuert den rechten Motor und umgekehrt, wie beim ersten Vehikel. Das Fahrzeug steuert nun immer auf die Lichtquelle zu und verfolgt sie, was der Test mit der Taschenlampe bestätigt.

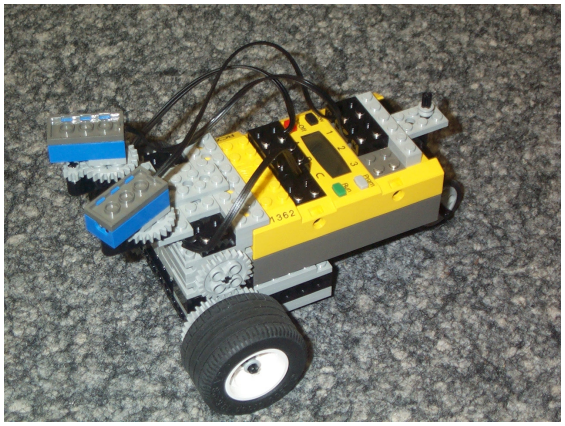

Abbildung 3.4.: Braitenberg Vehikel 2

3.3.3. Implementation

Beschreibung

Da das Vehikel 2 eigentlich zwei aneinander gebaute Vehikel 1 ist, muss der Source Code fast nicht verändert werden. Anstelle eines Sensors und eines Motors, werden jetzt

einfach zwei Sensoren und zwei Motoren verwendet, die genau gleich verbunden sind. Der linke Sensor steuert den linken Motor, der rechte Sensor den rechten Motor. Die Motorgeschwindigkeit wird mit der selben Funktion wie beim Vehikel 1 berechnet.

Methode steer der Klasse Vehicle02

```
public void steer()
{
    int sensorValueLeft;
    int sensorValueRight;
    leftMotor.start();
    rightMotor.start();
    leftMotor.startMotor();
    rightMotor.startMotor();

    while (Button.readButtons() == 0)
    {
       // Sensorwerte einlesen
       sensorValueLeft = lightSensorLeft.readValue();
       sensorValueRight = lightSensorRight.readValue();

       // Motorgeschwindigkeiten setzten
       leftMotor.setSpeed((sensorValueLeft - DARKNESS) * GRADIENT);
       rightMotor.setSpeed((sensorValueRight - DARKNESS) * GRADIENT);
    }

    lightSensorLeft.passivate();
    lightSensorRight.passivate();
    leftMotor.endThread();
    rightMotor.endThread();
 }
```

3.4. Vehikel 3 (Liebe)

3.4.1. Definition

Das dritte Vehikel ist vom Aufbau her identisch mit dem zweiten. Wieder steuert je ein Sensor einen Motor an, allerdings nicht mehr positiv sondern negativ. Das heisst, je stärker der Sensor stimuliert wird desto langsamer wird der zugehörige Motor.

Das Verhalten ist ähnlich wie beim Vehikel 2. Die Variante mit überkreuzter Ansteuerung wendet sich von der Quelle ab (Abbildung 3.5 links). Das Fahrzeug hat aber seine Agressivität verloren. Die andere Variante (Abbildung 3.5 rechts) folgt zwar der Lichtquelle, versucht aber nicht mehr sie zu rammen, sondern verlangsamt die Fahrt je näher die Lichtquelle ist.

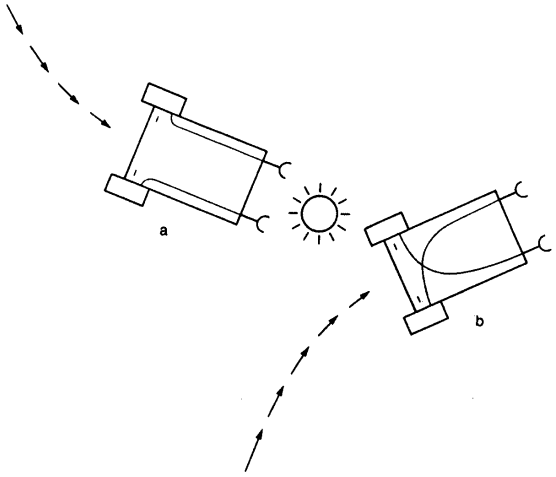

Abbildung 3.5.: Originalbild von Braitenberg

3.4.2. Hardware-Modell

Der Umbau des Fahrzeugs entfällt, lediglich die Motoren-Ansteuerung wird in der Software angepasst.

3.4.3. Implementation

Beschreibung

Anstelle einer steigenden Funktion wie beim Vehikel 2 wird hier eine fallende Funktion verwendet.

$$MotorGeschwindikeit : Helligkeit \rightarrow (Helligkeit - DunkelWert) \cdot Steilheit \ + 100$$

Im Labor mit Beleuchtung haben sich die Werte 30 für DunkelWert und -6 für Steilheit bewährt.

Methode steer der Klasse Vehicle03

```
public void steer()
{
   int sensorValueLeft;
   int sensorValueRight;
   leftMotor.start();
   rightMotor.start();
   leftMotor.startMotor();
   rightMotor.startMotor();

   while (Button.readButtons() == 0)
   {
```

```
        // Sensorwerte einlesen
        sensorValueLeft = lightSensorLeft.readValue();
        sensorValueRight = lightSensorRight.readValue();

        // Motorgeschwindigkeiten setzten
        leftMotor.setSpeed((sensorValueLeft - DARKNESS) * GRADIENT + 100);
        rightMotor.setSpeed((sensorValueRight - DARKNESS) * GRADIENT + 100);
    }

    // Programmende
    lightSensorLeft.passivate();
    lightSensorRight.passivate();
    leftMotor.endThread();
    rightMotor.endThread();
}
```

3.5. Vehikel 4 (Werte und spezielle Geschmäcker)

3.5.1. Definition

Das vierte Vehikel ist vom Aufbau her identisch mit dem zweiten und dem dritten. Wieder steuert je ein Sensor einen Motor an. Die Funktion ist aber nicht mehr linear sondern hat eine Glockenform. Das heisst, bis zu einem bestimmten Stimulus des Sensors dreht sich der dazugehörige Motor immer schneller. Wird dieser Wert überschritten, dreht sich der Motor wieder langsamer.

Das Vehikel steuert zuerst auf die Lichtquelle zu. Unterschreitet das Fahrzeug eine gewisse Distanz wendet es sich ab. Im Ideallfall schwenkt das Fahrzeug wie in Abbildung 3.6 in eine Kreisbahn ein.

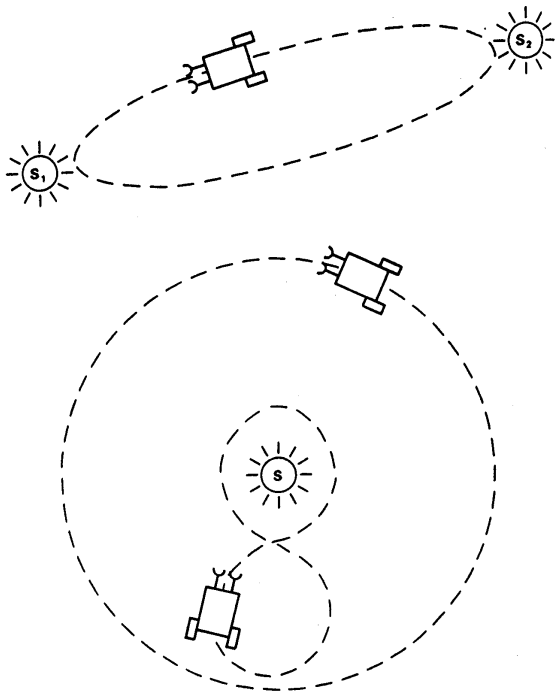

Abbildung 3.6.: Originalbild von Braitenberg

3.5.2. Hardware-Modell

Der Umbau des Fahrzeugs entfällt, lediglich die Motorenansteuerung wird in der Software angepasst.

3.5.3. Implementation

Beschreibung

Die einzige Änderung gegenüber dem dritten Vehikel ist, dass nun nicht mehr eine lineare Funktion verwendet wird. Als Funktion wurde die Gaussche Verteilungsfunktion gewählt. Bei der Implementation musste getrickst werden, da das Math Package (besonders `pow()`) mit leJOS nicht sauber funktioniert.

$$Motorgeschwindigkeit : Helligkeit \rightarrow \frac{Korrekturfaktor \cdot e^{(-1/2 \frac{(Helligkeit - Erwartungswert)^2}{Standardabweichung^2})}}{Standardabweichung \cdot \sqrt{2\,\pi}}$$

Mit einer Standardabweichung von 20, einem Erwartungswert von 40 und einem Korrekturfaktor von 5014 ergibt dies die folgende Kurve:

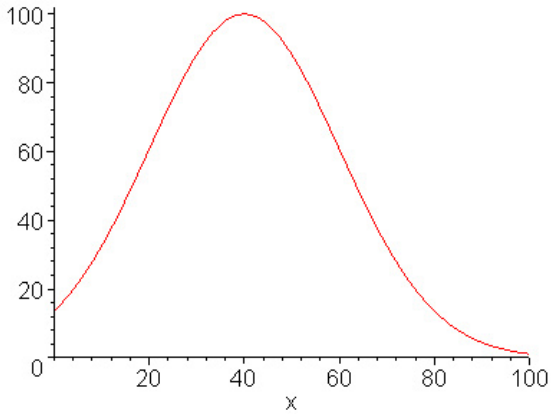

Abbildung 3.7.: Motoransteuerungsfunktion

Methode steer der Klasse Vehicle04

```
public void steer()
{
   int speed;

   leftMotor.start();
   rightMotor.start();
   leftMotor.startMotor();
   rightMotor.startMotor();

   while (Button.readButtons() == 0)
   {
      // Motorgeschwindigkeiten setzten
      speed = calcSpeed2(lightSensorLeft);

      DebugRCX.printValue(speed);

      leftMotor.setSpeed(speed);

      rightMotor.setSpeed(calcSpeed2(lightSensorRight));
   }

   // Programmende

   leftMotor.endThread();
   rightMotor.endThread();
}
```

Methode calcSpeed der Klasse Vehicle04

```
private int calcSpeed(BlackAndWhite lightSensor)
{
   int helligkeit = lightSensor.getValue();
   int erwartungswert = 40;
   int standardabweichung = 20;
   int korrekturfaktor = 5014;

   double exponent = -((helligkeit-erwartungswert) * (helligkeit-erwartungswert)
                       (2 * standardabweichung * standardabweichung);
  double zaehler = korrekturfaktor * Math.exp(exponent);
  double nenner = standardabweichung * Math.sqrt(2 * Math.PI );

  return (int) (zaehler/nenner);
}
```

3.6. Vehikel 5 (Logik)

3.6.1. Definition

Das fünfte Vehikel unterscheidet sich grundlegend von seinen Vorgängern. Die Verknüpfungen zwischen Sensoren und Motoren waren bisher direkte Verbindungen, verbunden mit einer mehr oder weniger komplizierten Übergangsfunktion.

Nun werden die Verbindungen mit Threshold-Devices (TD, Schwellwert-Einheiten) realisiert. Diese TDs sind mit logischen Toren aus der Digitaltechnik vergleichbar. Somit lassen sich, durch geschicktes Verknüpfen der TDs, theoretisch beliebig komplexe Schaltungen aufbauen. Es ist möglich Speicher und Zählwerke zu realisieren, die alles können was ein heutiger Computer kann. Vehikel 5 ist somit nicht klar definiert, sondern lässt viel Spielraum für Kreativität beim Bauen und Programmieren des Roboters.

3.6.2. Hardware-Modell

Auch wenn nun theoretisch Fahrzeuge mit beliebig komplexem Verhalten gebaut werden könnten, stossen wir mit unserem Lego-Rechner bald an Grenzen. Einerseits können wir nur drei Sensoren und drei Motoren ansteuern, andererseits steht für den Programmcode nur ca. 16kB zur Verfügung.

Je nach Implementation hat der Roboter eine andere Anzahl und Art von Sensoren und Motoren. Als Beispiele haben wir zwei unterschiedlich komplexe Roboter gebaut, die dem Vehikel 5 entsprechen:

Vehikel 5a hat lediglich einen nach oben gerichteten Helligkeitssensor und einen Motor.

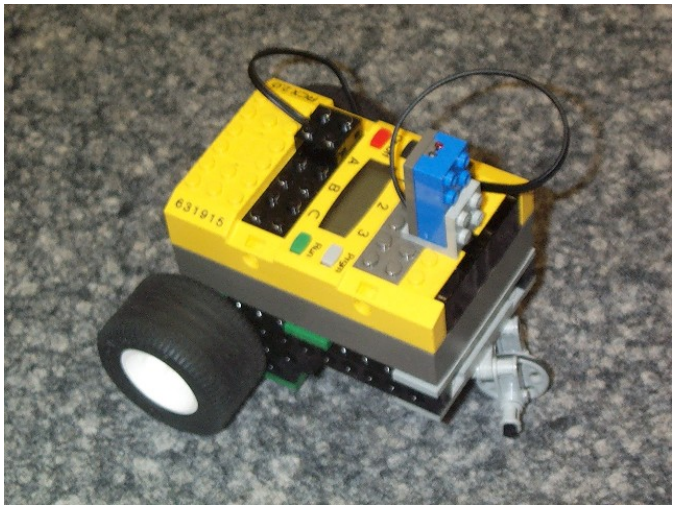

Abbildung 3.8.: Braitenberg Vehikel 5a

Vehikel 5b hat zwei nach vorne gerichtete Helligkeitssensoren und zwei Motoren die je ein Rad ansteuern, ähnlich wie Vehikel 2 bis 4. Zusätzlich sind vorne Tastsensoren eingebaut, die der Kollisionserkennung dienen. Sie können zusammen auf einen Sensoreingang des RCX geführt werden, da es nicht relevant ist welcher Sensor betätigt wird. Zur Anzeige dass der Roboter mit einem Hindernis kollidiert ist, wird eine Lampe verwendet, die wie ein Motor angesteuert werden kann.

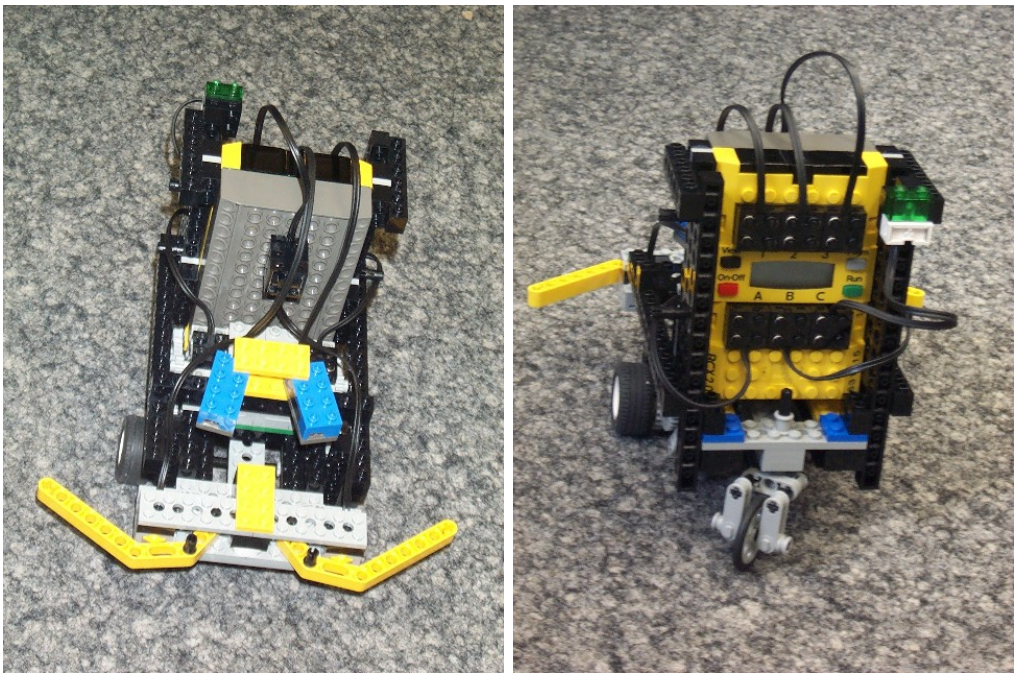

Abbildung 3.9.: Braitenberg Vehikel 5b

3.6.3. Implementation

Beschreibung

Die Klasse Vehicle05 beinhaltet zwei verschiedene Implementationen (5a und 5b), die sich in der Verknüpfung der TD unterscheiden. Die TD ist als separate Klasse ThresholdDevice im Package Tools implementiert. Die Funktionalität einer TD ist eigentlich simpel. Jede TD hat einen individuellen Schwellwert, der beim Erzeugen angegeben wird. Übersteigt nun die Summe aller an den Eingängen erscheinenden Werte diesen Schwellwert wird am Ausgang der Wert 100, ansonsten der Wert 0 ausgegeben. Ändert sich der Zustand einer TD benachrichtigt sie automatisch alle an ihr angehängten TDs und fordert sie auf sich zu aktualisieren. Bei einer Verbindung kann zusätzlich angegeben werden, ob der Wert invertiert werden soll. So lassen sich Subtraktionen realisieren (Aus 100 wird -100).

Um dem Vehikel die gewünschte Funktionalität zu geben, müssen nur mehrere TDs miteinander verbunden, und die Sensorwerte eingespiesen werden. Mit Rückkopplungen können primitive Speicher realisiert werden. Bei ungünstigen Konstellationen kann es dabei aber auch zu Endlosschlaufen führen die einen Stack-Overflow zur Folge haben. Es ist also Vorsicht geboten.

Damit das Vehikel auf die Ausgabewerte reagieren kann, musste noch eine spezialisierte TD erstellt werden. Die Klasse ThresholdMotor, ebenfalls im Packege Tools, hat die gleiche Funktionalität wie eine normale TD, mit der Erweiterung, dass sie zusätzlich einen Motor ansteuern kann (0: Motor aus, 100: Motor an). Die Implementation der TDs entspricht weitgehendst der Beschreibung von Braitenberg in seinem Buch Vehicles [3].

Vehikel 5a bildet die Funktionalität eines Schmitt-Triggers, wie aus der Elektrotechnik bekannt, nach. Es hat einen Helligkeitssensor und einen Motor. Steigt der Helligkeitswert über 70 an, fährt der Roboter los und stoppt erst wieder wenn der Helligkeitswert unter 30 gesunken ist.

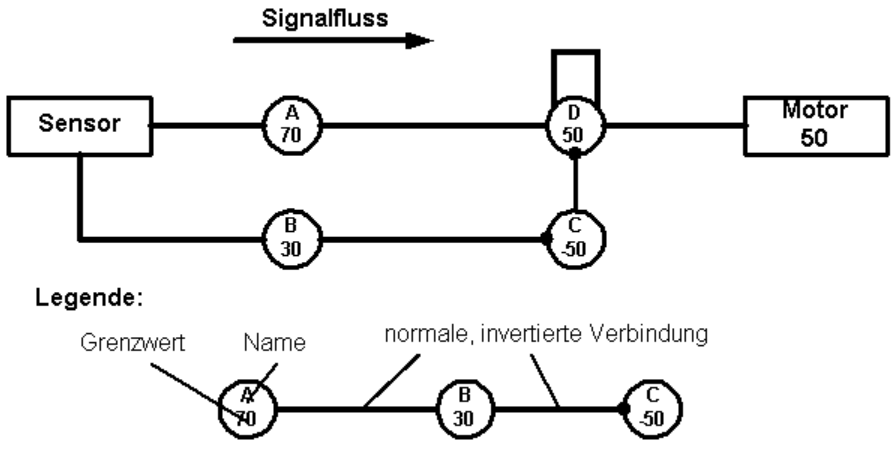

Abbildung 3.10.: Schema des Vehikels 5a nach der Notation von Braitenberg.

Vehikel 5b hat ein komplexeres Innenleben. Es vergleicht die Werte der beiden Helligkeitssensoren und steuert in Richtung Lichtquelle. Kommt der Roboter der Quelle nahe, dass beide Sensoren den Wert 100 melden, bleibt er stehen. Wird die Lichtquelle bewegt, beginnt er von neuem mit der Suche. Stösst der Roboter mit einem Hinderniss zusammen, beendet er die Suche und die Warnlampe geht an.

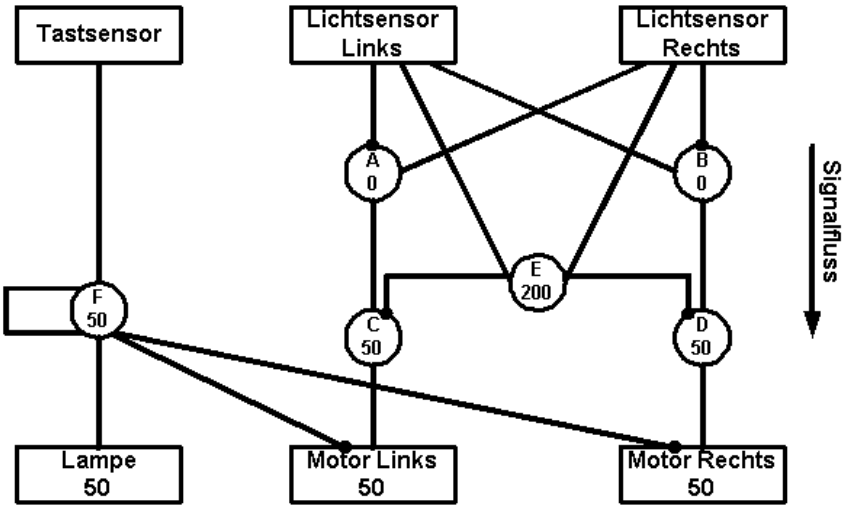

Abbildung 3.11.: Schema des Vehikels 5b nach der Notation von Braitenberg.

Methode init5a der Klasse Vehicle05

```
public void init5a()
{
   lightSensorLeft = Sensor.S1;
   lightSensorLeft.setTypeAndMode(
      SensorConstants.SENSOR_TYPE_LIGHT,
      SensorConstants.SENSOR_MODE_PCT);
   lightSensorLeft.activate();

   tA = new ThresholdDevice(70);
   tB = new ThresholdDevice(30);
   tC = new ThresholdDevice(-50);
   tD = new ThresholdDevice(50);

   leftMotor = new ThresholdMotor(Motor.A, 50);

   tA.attachInputDevice(lightSensorLeft);
   tB.attachInputDevice(lightSensorLeft);

   tD.attachInputDevice(tA);
```

```
    tA.attachOutputDevice(tD, false);

    tC.attachInputDevice(tB);
    tB.attachOutputDevice(tC, true);

    tD.attachInputDevice(tC);
    tC.attachOutputDevice(tD, true);

    tD.attachInputDevice(tD);
    tD.attachOutputDevice(tD, false);

    leftMotor.attachInputDevice(tD);
    tD.attachOutputDevice(leftMotor, false);
}
```

Methode run5a der Klasse Vehicle05

```
public void run5a()
{
    int sensorValue;
   while (Button.readButtons() == 0)
    {
        // Sensorwerte einlesen
        sensorValue = lightSensorLeft.readValue();

        // Sensorverte in Schwellwerteinheiten einspeisen
        tA.setInput(lightSensorLeft, sensorValue);
        tB.setInput(lightSensorLeft, sensorValue);

        // Gewünschten Statuswert anzeigen
        //LCD.showNumber(theMotor.getThreshold());
        //LCD.showNumber(theMotor.getOutput());
        LCD.showNumber(sensorValue);

        LCD.refresh();
    }

    // Programmende
    lightSensorLeft.passivate();
    leftMotor.killMotor();
}
```

3.7. Weitere Vehikel

Vehikel 6 beschreibt die Selektion nach Darwin, das siebte erstellt eine Art Assoziationen zwischen Sensorwerten. Es lernt so, und hat Erinnerungen, das heisst, es assoziert Neues mit schon Erlebtem.

Braitenberg definiert so 14 verschiedene Vehikel, die er alle in Vehicles [3] vorstellt und ausführlich beschreibt.

Da diese Vehikel die Möglichkeiten der Mindstorms übersteigen oder reine Gedankenexperimente sind, können wir sie leider nicht implementieren.

3.8. Schlussfolgerungen

Die Vehikel so zu betrachten, als wüsste man nichts von ihrem inneren Aufbau, ist ein sehr interessanter Ansatz. Man erhält so eine neue Vorstellung davon was Intelligenz ist. Selbst wenn man weiss, wie sich die Vehikel verhalten sollten, ist es immer wieder überraschend, was passiert wenn der Run-Knopf gedrückt wird. Da die Struktur dieser Vehikel sehr einfach ist, ist es möglich diese Theorie mit den Produkten von Lego zu realisieren und die Ideen von Braitenberg einer breiten Masse zugänglich zu machen.

In einem weiteren Schritt könnten Java Beans entwickelt werden, die die Ansteuerung von Motoren und Sensoren übernehmen. In einer grafischen Oberfläche könnte dann das Verhalten der Braitenberg Vehikel mit Drag and Drop definiert werden.

4. Nachahmende Roboter mit Jini

4.1. Einführung

Jini ist der Name einer auf RMI basierenden Technologie von Sun, welche ein Framework für verteilte Applikationen darstellt. Es ermöglicht ein "Plug and Play" Verhalten, das heisst, es ermöglicht den Geräten sich gegenseitig zu finden.

Jini ist keine Client/Server Architektur. Es ist viel mehr eine Geräte/Geräte Architektur, wo jedes Gerät Dienste im Netzwerk zur Verfügung stellen und jedes Gerät diese Dienste als Client nutzen kann. Um dies zu ermöglichen ist jedoch ein *lookup service* im Netzwerk nötig. Der *lookup service* ist eine Art zentrale Verwaltung, die für das Registrieren von Diensten zuständig ist. Wie in Abbildung 4.1 aus dem Buch von Ferrari [2] ersichtlich, ist neben dem *lookup service* (reggie) auch ein RMI Activation Deamon (rmid) und ein HTTP Server nötig. Der RMI Activation Deamon aktiviert bei Bedarf den *lookup service* und der HTTP Server stellt .class bzw. .jar Files zum herunterladen zur Verfügung.

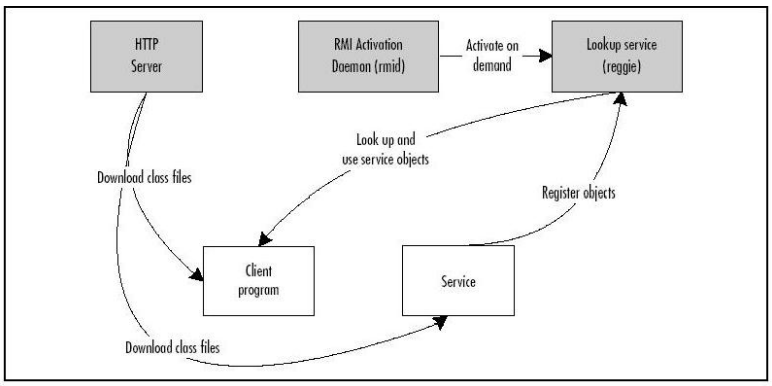

Abbildung 4.1.: Jini Komponenten

4.2. Implementaton eines Jini Beispieles

Um Jini etwas kennen zu lernen, haben wir ein Beispiel aus dem Buch von Ferrari [2] implementiert. Dabei sollen sich mehrere Roboter über einen Proxy PC registrieren. Ein Roboter kann nun einen Schritt (z.B. eine Linksdrehung) ausüben; sobald er fertig ist, wird dem nächsten Roboter dieser Schritt mitgeteilt, welcher dann diesen ausführt. Danach kann der nächste Roboter einen zufälligen Schritt wählen.

Da der RCX zu wenig leistungsfähig ist um Jini zu nutzen, ist ein PC als Proxy nötig.

Der Roboter (mit der original Firmware von Lego) wird dabei vom Proxy-Computer per Direct Control gesteuert. Leider ist es mit diesem Verfahren nicht möglich die Roboter zu adressieren, da die Firmware dies nicht unterstützt. Daher muss jeder Roboter von einem eigenen Tower (mit optischer Abgrenzung) gesteuert werden.

4.3. Implementation

4.3.1. Interface Mindstorm

Interface, welches von der Klasse MindstormProxy implementiert wird.

4.3.2. Klasse MindstormProxy

Objekte dieser Klasse werden beim *lookup service* registriert und sind fähig, mit einem Remote-Client zu kommunizieren. Dies wird durch RMI bewerkstelligt, was bedingt, dass von dieser Klasse eine _stub Klasse gebildet werden muss.

Die _stub Klasse kann mit dem RMI-Compiler einfach gebildet werden:

```
rmic -v1.2 -keep mindstorm.MindstormProxy
```

4.3.3. Klasse MindstormService

Diese Klasse bildet einen Roboter ab, erzeugt ein MindstormProxy Objekt und registriert dieses mit dem Jini *loctor service*.

4.3.4. Klasse Client

Die Klasse sucht mittels Multicast vorhandene *lookup services*, überprüft ob die registrierten Objekte das Interface Mindstorm implementieren und nimmt diese, falls nicht schon vorhanden, in einen Vector auf. Diese Klasse dient als Controller für die aufgenommen Mindstorm Objekte.

Die Klasse Client besitzt eine Innere Klasse EventListener. Von dieser Klasse muss ebenfalls eine _stub Klasse gebildet werden:

```
rmic -v1.2 -keep mindstormClient.Client$EventListener
```

4.3.5. jar-Files

Damit der HTTP Server die .class Files der _stub Klassen zur Verfügung stellen kann, müssen noch entsprechende .jar Files erzeugt werden:

```
jar cvf mindstormClient.jar -C mindstormClient/ Client$EventListener_stub.class
jar cvf mindstorm.jar -C mindstorm/ MindstormProxy_stub.class
```

4.4. Batch-Files

4.4.1. File starthttp.bat

Dieses Skript startet einen HTTP Server, der die Stubfiles im mit -dir angegebenen Verzeichnis zur Verfügung stellt. Das Verzeichnis muss auch das File *reggie-dl.jar* enthalten. Der Server arbeitet auf Port 8081. Damit alle Dienste starten können, dürfen keine Logfiles im angegebenen temporären Verzeichnis sein. Daher wird mit den ersten beiden Zeilen das Verzeichnis gelöscht und wieder erstellt. Nach erfolgreichem Starten des Servers wird eine Liste mit den angebotenen Files auf die Konsole geschrieben.

```
rd /s /q L:\lego_source_code\Jini\temp
md L:\lego_source_code\Jini\temp

java -jar C:\Programme\jini1_2_1_001\lib\tools.jar -port 8081
  -dir L:\lego_source_code\Jini\httproot -trees -verbose
```

4.4.2. File startrmid.bat

Dieses Skript startet den RMI Activation Deamon, ohne Sicherheits-Check, und speichert seine Logfiles im angegebenen Verzeichnis.

```
rmid -J-Dsun.rmi.activation.execPolicy=none
     -log L:\lego_source_code\Jini\temp\rmidlog
```

4.4.3. File startreggie.bat

Dieses Skript startet den Reggie-Dienst. Als ersten Paramater wird die URL des reggie-dl.jar Files angegeben. In diesem Fall zur Verfügung gestellt, vom HTTP Server auf dem Host “axar” und Port 8081. Ausserdem ist ein Security Policy File nötig, hier ein standard File im Jini Installations-Verzeichnis. Als letzter Parameter wird das Verzeichnis für die Logfiles mitgegeben. Nach einer Weile wird dieses Skript beendet, da reggie sich mit RMI selber registriert. Von jetzt an wird RMI reggie aktivieren.

```
java -jar C:\Programme\jini1_2_1_001\lib\reggie.jar
  http://axar:8081/reggie-dl.jar
  c:\Programme\jini1_2_1_001\policy\policy.all
  L:\lego_source_code\Jini\temp\reggielog
```

4.4.4. File startmindstorm.bat

Dieses Skript startet das Programm MindstormService. Für jeden Roboter wird dieses File ausgeführt. Zuerst muss der Classpath richtig gesetzt werden. Die virtuelle Maschine muss wissen, dass es das MindstormService.jar File vom Server auf dem Rechner ”axar” herunter laden kann. In diesem Fall wird ein Roboter namens Barney mit am COM1 angeschlossenem Tower gestartet.

```
set classpath=.;c:\programme\jini1_2_1_001\lib\jini-core.jar;
  C:\Programme\jini1_2_1_001\lib\jini-ext.jar;
  C:\Programme\LEGO\lejos\RcxJavaAPI\rcxjava\rcx.jar;
  C:\Programme\j2sdk1.4.0\jre\lib\ext\comm.jar

java -Djava.rmi.server.codebase=http://axar:8081/mindstormClient.jar
  mindstorm.MindstormService Barney COM1
```

4.4.5. File startmindstormClient.bat

Dieses Skript startet das Programm MindstormClient welches als Controller dient. Zuerst wird der Classpath richtig gesetzt. Die virtuelle Maschine muss wissen, dass es das mindstorm.jar File vom Server auf dem Rechner "axar" herunter laden kann. Es ist aber darauf zu achten, dass der Client nur registrierte Mindstorms findet, wenn sie sich in der gleichen Broadcast Domain befinden.

```
set classpath=.;c:\programme\jini1_2_1_001\lib\jini-core.jar;
  C:\Programme\jini1_2_1_001\lib\jini-ext.jar;
  C:\Programme\LEGO\lejos\RcxJavaAPI\rcxjava\rcx.jar;
  C:\Programme\j2sdk1.4.0\jre\lib\ext\comm.jar

java -Djava.rmi.server.codebase=http://axar:8081/mindstorm.jar
  mindstormClient.Client
```

4.5. Schlussfolgerungen

Jini sieht auf den ersten Blick sehr interessant aus, ist aber leider ziemlich kompliziert. So muss man sich mit dem Security Manager abgeben, mit Kommandozeilen-Tools Stub- und jar Files generieren und mehrere Dienste zur Verfügung stellen (http-Server, Reggie, RMI Activation Deamon).

Obschon wir den Hauptteil des Codes zur Verfügung hatten, war es sehr zeitaufwändig und mühsam das Beispiel zu implementieren. Besonders aus den seitenlangen, kryptischen Fehlermeldungen schlau zu werden war nicht immer einfach.

Aus diesen Gründen und da es mit Jini nicht möglich ist mehrere Roboter einzeln von *einem* Tower aus zu steuern, unterliessen wir es, uns weiter mit dieser Technologie zu befassen und als Grundlage für weitere Aufgaben zu nutzen.

5. Swarm Intelligence

5.1. Einführung

Das Weg-Suche Verhalten von Ameisen basiert auf Pheromonspuren, denen andere Ameisen folgen. Jede Ameise legt eine Pheromonspur. Eine Ameise wählt mit einer Wahrscheinlichkeit, die proportional zur Pheromonmenge ist, einen Pfad. Ziel dieses Projektes ist es, mittels diesem Alogrithums den kürzesten Weg in einem Graphen zu finden.

Die möglichen Pfade[1] zwischen einem Start und Endpunkt werden mit schwarzem Klebeband auf weissen Untergrund geklebt. Ein Pfad kann sich in zwei weitere Pfade verzweigen. Alle Wege[2] führen zum Ziel, das heisst es gibt keine Sackgassen.

Die Lego Ameise fährt selbstständig vom Start- zum Zielpunkt. Die Entscheidung welche Richtung sie bei einer Verzweigung wählt, ist ihr selbst überlassen. Ihre Entscheidung kann sie anhand ihrer internen Pheromonkarte fällen. Der Steuerungsalgorithmus hat jedoch keinen Zugriff auf die Struktur der Karte, sondern nur auf die Pheromonwerte der jeweils zur Wahl stehenden Pfade.

Die Lego Ameise fährt selbständig und autonom. Wenn ein PC vorhanden ist, so wird nach jeder Runde die Pheromonkarte zum PC übertragen. Der PC hat aber keinen Einfluss auf das Verhalten der Lego Ameise.

Die Software ist in drei Teile unterteilt. Roboter, PC und gemeinsam genutzte Komponenten.

5.2. Versuchsaufbau

Die verschiedenen Wege wurden mit schwarzem Isolierklebeband aufgeklebt. Beginn und Ende einer Edge sind jeweils mit einem grünen Papierstreifen markiert. Auf den Start (Zielpunkt) ist ein IR-Tower ausgerichtet. Am Ende einer Runde stoppt die Ameise und kann über den IR-Tower Daten zum PC übermitteln. Die Vereinigungen sind so entworfen, dass es nicht möglich ist, einen Weg versehentlich zurück zu fahren. Damit die Markierungen zuverlässig erkannt werden können, muss der Aufbau gleichmässig ausgeleuchtet werden.

[1] Kanten, Edges
[2] Aneinanderreihung von Pfaden

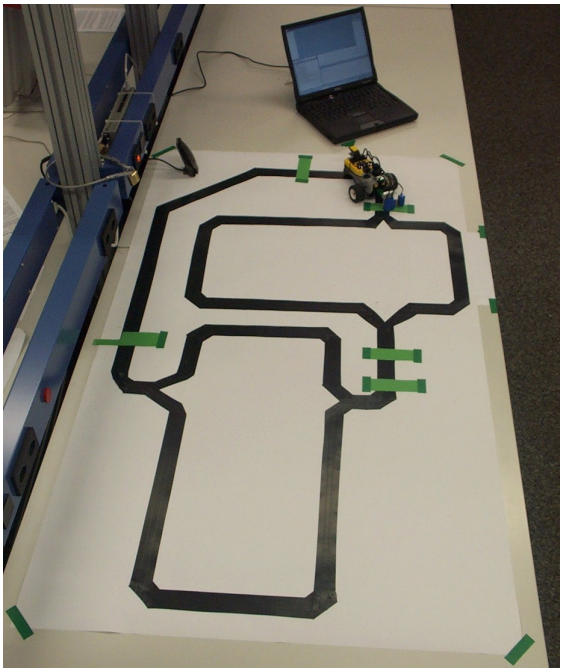

Abbildung 5.1.: Versuchsaufbau

5.3. Hardware des Roboters

Der Roboter ist vom Typ Turtle. Das heisst, er besitzt zwei einzeln angetriebene Räder, und ein frei drehendes Stützrad. Der Robotor kann somit fast auf der Stelle drehen. Um den Pfaden zu folgen werden zwei optische Sensoren verwendet. Ein zusätzlicher dritter Sensor wird benötigt, um Verzweigungen zuverlässig zu detektieren.

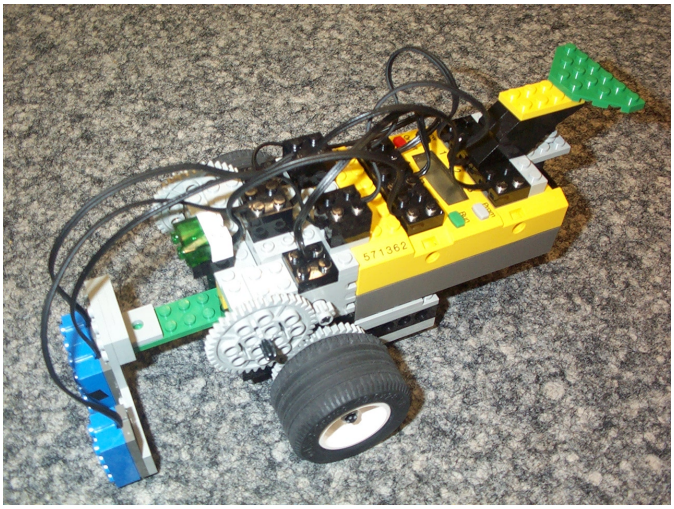

Abbildung 5.2.: Robo Ameise Seitenansicht

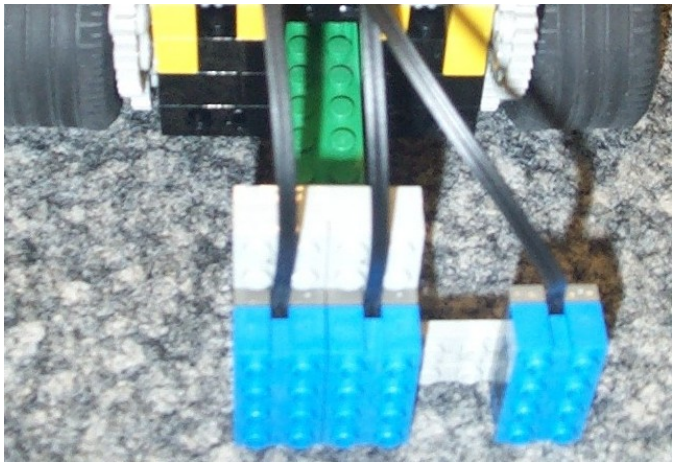

Abbildung 5.3.: Robo Ameise Sensoren

5.4. Software des Roboters

Die Software des Roboters besteht aus der Hauptklasse `RoboAnt` und der Klasse `Colours`. Ferner werden die Klassen des `map`- und `communication`-Packages verwendet.

Die Klasse `RoboAnt` implementiert die drei Haupt-Verhaltensweisen des Roboters: Linefollower, Wahl der nächsten Edge und Verteilung Pheromon.

5.4.1. Ablauf einer Runde

Eine Runde läuft wie folgt ab: Der Roboter folgt der Linie (Kapitel 5.4.2), bis alle drei optischen Sensoren gleichzeitig den Wert für Grün melden. Dies bedeutet, dass der Roboter am Ende einer Edge angelangt ist und er (wenn es eine Verzweigung ist) die nächste Edge wählen muss. Mit der Methode `calculateProbability()` wird die Wahrscheinlichkeit anhand der internen Pheromonkarte (Kapitel 5.6.1) für den linken Edge berechnet (Kapitel 5.4.3). Hat der Roboter sich für eine Edge entschieden, folgt er dieser bis er wieder ans Ende der Edge gelangt. Ist er schliesslich am Ziel (auch auf seiner internen Karte gespeichert) angelangt, verteilt er abhängig von der benötigten Zeit, Pheromon auf die Edges, die er in dieser Runde befahren hat (Kapitel 5.4.4). Schliesslich schickt er die aktuelle Karte an einen PC (Kapitel 5.6.2), der diese graphisch darstellt (Kapitel 5.5).

Einen groben Überblick über den Ablauf einer Runde gibt auch Abbildung 5.4.

In diesem Projekt wird ein leicht abgeänderter Algorithmus aus dem Buch Swarm Intelligence [4] angewandt. Die Änderung bezieht sich vor allem auf die Verteilung des Pheromons, da im Gegensatz zum Beispiel im Buch jeweils nur eine Ameise unterwegs ist.

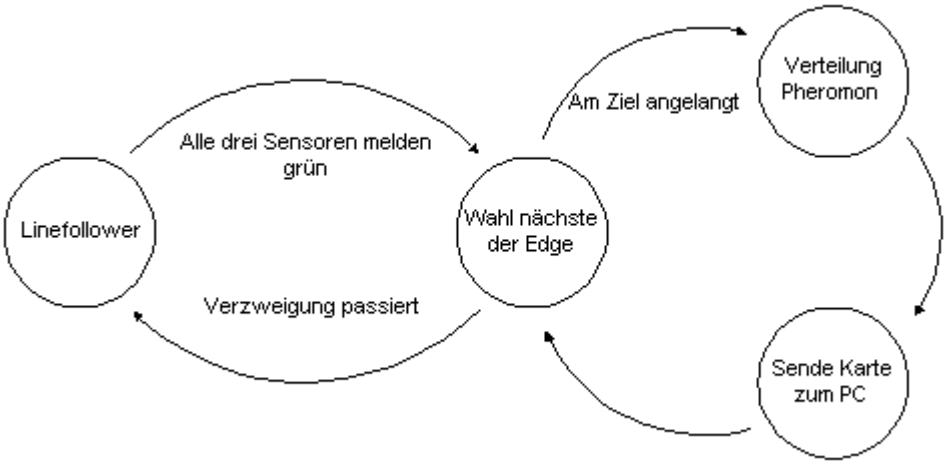

Abbildung 5.4.: Zustandsdiagramm Roboter Software

5.4.2. Linefollower

Der Roboter muss einer Linie folgen können. Er muss aber auch Verzweigungen und Vereinigungen von Edges erkennen können.

Das Linienfolgen wird mit zwei Sensoren realisiert. Die abzufahrende Strecke ist relativ breit, mit schwarzem Klebeband, auf einen weissen Hintergrund geklebt. Die zwei nahe beieinanderstehenden Lichtsensoren aus Abbildung 5.3 müssen im schwarzen Bereich sein. Wird mit dem linken Sensor ein Wert gelesen, der grösser oder gleich dem Wert für Weiss ist, dreht der rechte Motor rückwärts. Dasselbe gilt für den rechten Sensor und den linken Motor. So wird sicher gestellt, dass der Roboter sobald er zu weit in eine Richtung fährt, dies sofort korrigiert. Auch wenn er einmal zu schnell fährt und die schwarze Linie mit beiden Sensoren ganz verlässt bevor er korrigieren kann, ist dies kein Problem. Weil dann beide Sensoren im weissen Bereich sind, drehen beide Motoren rückwärts und der Roboter fährt zurück, bis er die schwarze Linie wieder gefunden hat. Dies ist ein sehr einfacher Algorithmus und entspricht etwa einem vereinfachten Braitenberg Vehikel vom Typ 2.

5.4.3. Wahl der nächsten Edge

Am Anfang liegt auf allen Edges gleich viel Pheromon, so dass bei einer Verzweigung die Wahrscheinlichkeit für beide Edges gleich gross ist. Da überall gleich viel Pheromon liegt, hängt die Wahl also nur vom Zufallsgenerator ab. Die Ameise misst die Zeit für einen Durchlauf und verteilt anhand dieser Zeit verschieden viel Pheromon auf die Edges die sie benutzt hat.

In der zweiten Runde wird die Ameise bei einer Verzweigung den Edge bevorzugen, den sie in der ersten per Zufall gewählt hat, da auf diesem nun mehr Pheromon liegt. Hat die erste Runde allerdings viel Zeit in Anspruch genommen, wird die Wahrscheinlichkeit,

dass sie denselben Weg nochmals geht nur minim grösser sein. Wurde die erste Runde dagegen schnell abgefahren, ist die Wahrscheinlichkeit für diesen Edge um einiges grösser.

Die Wahrscheinlichkeit p_l für den linken Edge berechnet sich aus

$$p_l = \frac{(k+\rho_l)^n}{(k+\rho_r)^n + (k+\rho_l)^n},$$

für den rechten Edge gilt

$$p_r = 1 - p_l.$$

ρ_l stellt die Menge Pheromon auf dem linken und ρ_r die Menge auf dem rechten Edge dar. Der Parameter n verkörpert den Nichtlinearitätsgrad der Funktion: ist n gross, wird ein Edge der nur wenig mehr Pheromon hat, mit höherer Wahrscheinlichkeit gewählt, als wenn n klein ist. Der Parameter k ist der Anziehungsgrad eines Edges ohne Pheromon. Je grösser k ist, desto grösser ist die Menge Pheromon, die benötigt wird, um die Wahl des Edges nicht zufällig zu machen. In der Abbildung 5.5 ist die Wahrscheinlichkeit für die Wahl einer Edge abhängig von der Pheromonmenge, mit n=2 und k=100 dargestellt.

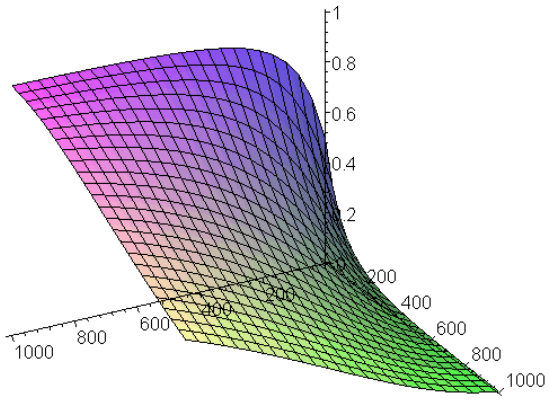

Abbildung 5.5.: Wahrscheinlichkeit für eine Edge mit n=2, k = 100

5.4.4. Verteilung Pheromon

Wählen viele Ameisen denselben Weg, so steigt die Pheromonkonzentration auf diesem an. Wird ein Weg nicht mehr begangen, so verdunstet das Pheromon. Je kürzer ein Weg ist, desto mehr Ameisen können ihn pro Zeit passieren. Da wir nur jeweils eine Ameise pro Runde benutzten können, muss diese, um den natürlichen Algorithmus nachzubilden, einen Schwarm repräsentieren. Je kürzer der Weg ist, dem die Roboter Ameise gefolgt ist, desto mehr Pheromon muss auf diesem liegen. Aus dieser Beziehung lässt sich folgende Formel für die Pheromonverteilung ableiten:

$$Pheromonkonzentration := x \rightarrow a\,(x-t)^n$$

x: Zeit für eine Runde
t: Zeit einer Runde, für die kein Pheromon verteilt wird
a: Korrekturfaktor, damit die maximale Pheromonmenge begrenzt werden kann
n: n muss Gerade sein. Je kürzer die Zeitunterschiede der Wege, desto grösser muss n gewählt werden

Mit $a = 3$, $t = 40$ und $n = 2$ ergibt das folgenden Graphen:

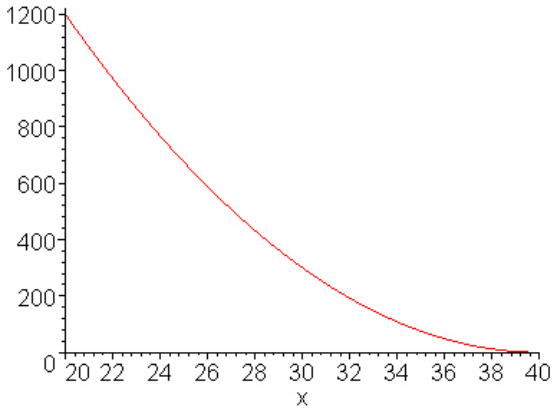

Abbildung 5.6.: Pheromonmenge pro Zeit

Am Ende jeder Runde wird das Pheromon „verdunstet“. Je mehr Pheromon auf einem Weg liegt, desto mehr verdunstet pro Zeiteinheit. Da die Laufzeit schon in der Pheromonverteilung vorkommt, muss sie hier nicht mehr unbedingt berücksichtig werden.

$$Pheromonverdunstung := y \rightarrow \frac{y}{d}$$

y: Pheromonmenge
d: Verdunstungs-Divisor

5.5. Software PC

5.5.1. PC-Software Anforderungen

Die PC-Software visualisiert die Pheromondaten. Die Lego Ameise meldet sich nach einem erfolgreichem Durchlauf automatisch beim PC. Auf dem PC kann der dynamische Verlauf der Wegsuche beobachtet werden.

5.5.2. PC-Software Bedienung

Starten des Programms

Das Programm kann mit dem Batchfile *LAntPC.bat* gestartet werden. Damit die Kommunikation zum RCX funktioniert, muss der IR-Tower richtig installiert werden, falls

dies nicht schon im Rahmen vorangegangener Arbeiten mit dem RCX geschehen ist. (siehe Kapitel: 2.1.2).

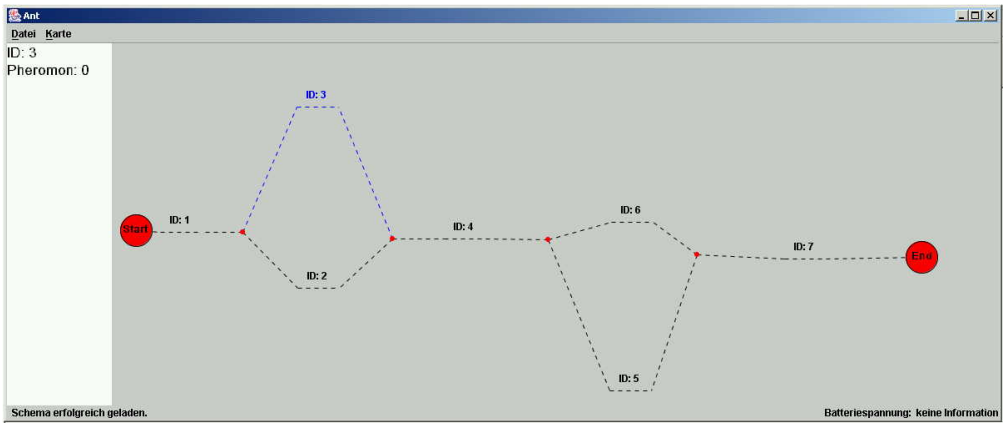

Abbildung 5.7.: PC-Software Screenshot

Laden der Karte

Im Menu *Karte* gibt es zwei Möglichkeiten die Karte neu zu laden. In beiden Fällen ist es nicht möglich den Lesevorgang abzubrechen wenn er einmal gestartet wurde. Das Programm wartet dann, bis es eine Verbindung mit dem RCX aufgebaut und die gewünschte Datenmenge gelesen hat.
Reload Once versucht die Karte ein einziges Mal vom RCX zu lesen.
Start Auto Reload veranlasst das Programm die Karte ohne weitere Aufforderung regelmässig zu aktualisieren. Nach einem erfolgreichen Lesevorgang wartet das Programm 15 Sekunden, bis es erneut zu lesen versucht. In dieser Zeit kann über denselben Menueintrag das automatische Auslesen wieder gestoppt werden.

Erstellen eines Darstellungs-Schemas

Die eingelesene Karte wird im Normalfall nicht vernünftig dargestellt. Die einzelnen Kanten und Knoten können mit Drag&Drop im Fenster verschoben werden, bis die Karte übersichtlich dargestellt wird. Das jeweils aktuelle Element wird mit blauer Farbe hervorgehoben, da sonst nicht immer klar ersichtlich ist, welches verschoben werden soll.

Damit nicht nach jedem Aktualisieren der Karte die Elemente neu angeordnet werden müssen, können die Positionen in einem Schema gespeichert werden. Dazu muss die Karte einmal nach den eigenen Wünschen angeordnet werden. Im Menu *Datei* kann die Darstellung mit *Schema speichern* gesichert werden. Von nun an wird die Karte immer nach diesem Schema dargestellt wenn sie neu geladen wird. Mit *Schema öffnen* kann zum Beispiel nach einem Neustart des Programms ein zuvor gespeichertes Schema wieder geladen werden.

5.5.3. PC-Software Design

Als Programmiersprache drängte sich Java auf, da die Software für den Roboter in Java geschrieben wurde und bereits Klassen für die Kommunikation zwischen RCX und PC vorhanden waren. Das Programm ist für unseren Versuchsaufbau optimiert, für andere Karten und Roboter sind eventuell geringfügige Anpassungen notwendig. Ein Klassendiagramm ist dieser Dokumentation im Anhang beigelegt.

Weitere Details über die Implementation können der generierten JavaDoc-Dokumentation oder dem Sourcecode entnommen werden.

Package: logic

`logic` beinhaltet Klassen, die für das Aufbereiten der Daten zuständig sind. Da der grösste Teil dieser Arbeit bereits auf dem RCX oder in der Kommunikation geschieht, enthält es nur zwei Klassen:

MapReader ist für das Laden der Karte vom RCX zuständig. Damit beim laden und vor allem beim Warten auf eine Verbindung nicht die ganze Applikation blockiert wird, ist sie von der Klasse `Thread` abgeleitet.

SchemaLoader speichert und lädt die Darstellungs-Schemata vom Typ `MapSchema`. Sie serialisiert das `MapSchema` und legt es auf der Festplatte ab.

Package: gui

`gui` beinhaltet alle Klassen, die direkt mit der Benutzeroberfläche zu tun haben:

MainWindow ist von `JFrame` abgeleitet und ist der Einstiegspunkt der Applikation. Sie initialisiert die anderen Teile der Applikation.

RoutePanel ist von `JPanel` abgeleitet und zeigt die Karte auf dem Bildschirm an. Sie enthält mehrere `VisualEdge`s und `Node`s und ist beim Laden einer neuen Karte für das korrekte Zusammenfügen und Anordnen dieser verantwortlich. Ist ein Schema geladen, wird dieses bei jedem aktualisieren der Karte verwendet.

EdgeInfo ist von `JTextArea` abgeleitet und zeigt Information zu einer bestimmten Kante an.

AntMenu ist von `JMenuBar` abgeleitet und enthält die Menueintäge.

StatusLabel ist von `JPanel` abgeleitet und zeigt Status- und Fehlermeldungen an. Damit Meldungen von einem beliebigen Ort in der Applikation angezeigt werden können, ist diese Klasse als Singleton realisiert. Fehlermeldungen werden durch rote Farbe hervorgehoben. Beim Auslesen der Karte wird zusätzlich die aktuelle Batteriespannung aus dem RCX gelesen und ebenfalls hier angezeigt. Sinkt die Spannung auf einen Wert unter

7.5 Volt, wird sie zur Warnung in roter Farbe dargestellt.

VisualEdge ist von `JLabel` abgeleitet und stellt eine Kante der Karte im `RoutePanel` dar. Ein Teil der angezeigten Kante wird immer horizontal dargestellt und lässt sich im `RoutePanel` verschieben. Dort wird auch die ID der Kante angezeigt. Aus den Positionsdaten der angehängten Knoten wird die Grösse und Position berechnet und die Verbindung zu den Knoten gezeichnet. Die Pheromonkonzentration auf der Kante wird durch die Strichdicke visualisiert. Der genaue Pheromonwert wird im `EdgeInfo` angezeigt, wenn auf die Kante geklickt wird. Um die Drag&Drop funtkionalität zu realisieren wurde ein spezieller Mouselistener namens `EdgeMouseListener` implementiert.

Node ist von `JLabel` abgeleitet und dient zur Verbindung der Kanten im `RoutePanel`. Auch sie können durch Drag&Drop positioniert werden. Dazu sind die beiden Hilfsklassen `NodeMouseListener` und `NodeMouseMotionListener` notwendig. Der Start- und Endpunkt der Karte wird mit einem roten Punkt dargestellt.

MapSchema enthält die Darstellungsinformationen der im `RoutePanel` angezeigten Karte. Sie ist serialisierbar und kann somit mit dem `SchemaLoader` auf der Festplatte abgelegt werden.

LAntPC.bat

Die Batchdatei ruft das Hauptfenster der Applikation auf. Die benötigten Klassen sind in den beigefügten Archiven zusammengefasst.

```
java -cp LAnt_pc.jar;pcrcxcomm.jar gui.MainWindow
```

LAnt_PC.jar: Die Selbstgeschriebenen Klassen.
pcrcxcomm.jar: Klassen zur Kommunikation zwischen RCX und PC.

5.6. Von PC und Roboter genutzte Packages

5.6.1. Pheromon Karte

Das `map` Package besteht aus den Klassen `MapManager`, `Map`, `Edge` und `Direction`. Die Klassen sind so programmiert, dass sie mit Standard Java *und* mit LeJOS verwendet werden können.

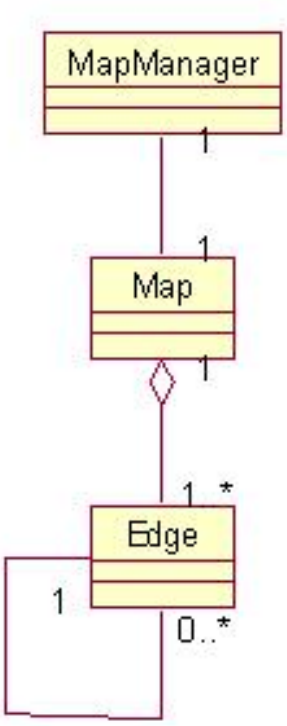

Abbildung 5.8.: UML Diagramm des Packages **map**

MapManager

Diese Klasse liefert die Pheromondaten für die Wegsuche. Die Struktur der gesamten Karte ist nicht ersichtlich. Die einzigen Strukturinformationen, die abgefragt werden können sind, ob es sich um eine Verzweigung handelt, oder ob das Ziel erreicht worden ist. Der jeweils gewählte Weg wird gespeichert. Ist eine Runde fertig gefahren, können die Pheromonwerte der Edge Objekte aktualisiert werden.

Map

Verwaltet die Edge Objekte. Kann sich selber serialisieren bzw. deserialisieren. Die Serialisierung von Referenzen beschränkt sich auf die Serialisierung von Edge Referenzen. Die Serialisierung wird benötigt, um die Map über die IR Schnittstelle übertragen zu können. Wie in Abbildung 5.1 ersichtlich, wird zuerst die Anzahl zu übertragende Edges und dann die Edges selber in ein Byte Array geschrieben. m_x ist die Anzahl Bytes für die entsprechende Edge.

Bezeichnung	Bytes
Anzahl Edges	1
Edge 1	m_1
Edge 2	m_2
...	...
Edge n	m_n

Tabelle 5.1.: Serialisierung Map

Edge

Repräsentiert einen Streckenabschnitt. Speichert den jeweiligen Pheromonwert und besitzt Referenzen auf die nachfolgenden Edge Objekte. Ein Edge Objekt kann sich wie in Abbildung 5.2 dargestellt serialisieren.

Bezeichnung	Bytes
EdgeID	1
Pheromonwert	4
Anzahl nächste Edges	1
ID nächste Edge 1	1
ID nächste Edge 2	1
...	...
ID nächste Edge n	1

Tabelle 5.2.: Serialisierung Edge

Direction

Richtungs-Konstanten für Methodenaufrufe in `MapManager` und `Map`.

5.6.2. Kommunikation zwischen PC und RCX

Communication

Die Klasse `Communication` ist für die Kommunikation zwischen PC und RCX verantwortlich. Sie kann sowohl für den PC, wie auch für den RCX gebraucht werden, indem die Library `rcxrcxcomm.jar` oder `pcrcxcomm.jar` eingebunden wird.

Die Kommunikation ermöglicht das Übertragen und Lesen einer Map und des Batteriezustandes des RCX. Das interne Byte Array ist aus Speichergründen genau so gross gewählt wie nötig, um die Map zu serialisieren. Wichtig ist, dass beide Seiten (PC/RCX) die gleich grosse Arraygrösse haben, da die `int read(byte[] b)` Funktion der Klasse `InputStream` so lange wartet, bis das Array voll gefüllt wurde. Dieses Verhalten entspricht eigenartigerweise nicht der Java Spezifikation, welche ein vorzeitiges Return mit

Anzahl gelesenen Bytes als Rückgabewert vorsieht.

5.7. Probleme

Bei der Entwicklung des Swarm Intelligence Algorithmus traten zwei grössere Probleme auf. Dies ist zum Ersten das Erkennen der Verzweigungen und Vereinigungen. Als einfachste Variante erwogen wir das Anbringen von grünen Stellen am Beginn und Ende einer Kante. Leider sind die Lichtsensoren von LEGO qualitativ nicht sehr gut. Die Folge war, dass am schwarzen Rand der Strcke manchmal beide Sensoren Grün angaben. Hinzu kam, dass die Lichtsensoren unterschiedliche Werte meldeten, obschon sie direkt nebeneinander platziert sind. Das Problem hätte mit vier Lichtsensoren einfach gelöst werden können, indem man die grüne Markierung quer zur Bahn auf beide Seiten etwa 5cm hinausragen und zwei Sensoren, je links und rechts über den RCX hinaus montiert hätte. Die Anschlüsse beider Sensoren hätte man auf den dritten, freien Eingang des RCX nehmen können, welcher dann Grün angibt, wenn beide Sensoren über einer Markierung sind.

Da wir aber mit drei Sensoren auskommen mussten, haben wir nur einen Sensor linksaussen montiert. Wenn jetzt alle drei Sensoren Grün angeben, ist die Wahrscheinlichkeit sehr gross, dass die Sensoren tatsächlich über einer grünen Stelle sind. Diese Variante hat sich dann zum Glück auch sehr gut bewährt.

Das zweite Problem war der mangelnde Speicher auf dem RCX. Dieses Problem bemerkten wir erst, als wir alle Module integrieren wollten. Bis jetzt haben wir nie daran gedacht, dass der Speicher von 32KB nicht ausreichen würde. Wir haben dann begonnen, den Sourcecode zu optimieren, indem wir z. B. aus `int bytes` machten und auf verschiedene Klassen verzichteten. Dies genügte jedoch noch nicht. Mit dem lejos-Emulator fanden wir dann den grossen "Recourssenfresser", nämlich die pow Funktion aus der `Math` Klasse. Auf diese Funktion und die gesamte Klasse konnten wir jedoch relativ einfach verzichten und so 2KB RAM einsparen, was bis jetzt genügte.

5.8. Testen

Die einzelnen Komponenten des Projekts wurden einzeln getestet. Durch dieses Vorgehen konnte gewährleistet werden, dass die Zeit für die Fehlersuche auf dem RCX minimal blieb.

5.8.1. Map

Das `map` Package verwendet keine leJOS Klassen. Deshalb konnte es mit einem Testprogramm auf einem PC getestet werden.

5.8.2. Kommunikation

Die Kommunikation wurde separat auf einem RCX mit einer Testmap getestet.

5.8.3. Linie Folgen

Der Roboter wurde auf dem Testfeld ausgesetzt und beobachtet. Die nötigen Kalibrationen wurden vorgenommen, bis der Roboter der Linie zuverlässig folgen konnte.

5.8.4. PC Software

Mit einem Map-Generator kann unabhängig vom Roboter eine Karte generiert werden. Mit diesen Testkarten wurde die PC Software ausgiebig getestet.

5.8.5. Systemtest

Nach dem Komponententest wurde das gesamte System getestet. Der Roboter fand erfolgreich den kürzesten Weg auf der Teststrecke.

5.9. Schlussfolgerungen

Die Schnelligkeit, mit der der Roboter den kürzesten Weg findet, ist vom Zufall abhängig. Es kann durchaus sein, das *eine* Runde genügt, um den kürzesten Weg zu finden. Es kann aber auch ohne weiteres 20 Runden dauern. Der kürzeste Weg wird aber in jedem Fall gefunden. Vergrössert man die Wahrscheinlichkeit, dass ein Weg mit weniger Pheromon gewählt wird, so tendiert das System zum Schwingen. Ein kürzerer Weg kann zugunsten eines längeren Weges verloren gehen. Das Resultat ist bei beiden Konfigurationen stark vom Zufall abhängig. In der Natur wird dieser Faktor durch eine grosse Anzahl an Individuen kompensiert. Die benötigte grosse Anzahl von Individuen schränkt die effiziente, praktische Anwendung dieses Algorithmus stark ein.

6. Zeitplan

In der *Evaluationsphase* (Kapitel 2) haben wir uns damit auseinandergesetzt, was sich mit den Mindstorms überhaupt alles realisieren lässt. Da wir die verschiedenen Programmiersprachen testen mussten, nahm diese erste Phase ca. drei Wochen in Anspruch.

Parallel zur Evaluation, schrieben wir zur *Einarbeitung* einige Testprogramme, und analysierten die Hardware auf die Tauglichkeit für unser Projekt.

Nachdem wir uns mit Hard- und Software vertraut gemacht hatten, bearbeiteten wir die *Braitenberg Vehikel* (Kapitel 3). In dieser Phase haben wir uns mit dem Buch von Braitenberg [3] vertraut gemacht und zu jedem der ersten fünf Vehikel ein Beispiel realisiert.

Neben den Braitenberg Vehikeln beschäftigten wir uns auch mit *Jini* (Kapitel 4). Dieses Projekt dauerte etwa zwei Wochen.

In das letzte Projekt, *Swarm Intelligence* (Kapitel 5), haben wir am meisten Zeit investiert. Wir haben uns mit dem Buch Swarm Intelligence [4] in das Thema eingearbeitet und einen relativ einfachen Algorithmus mit den Mindstorms realisiert.

Während der ganzen Studienarbeit haben wir die *Dokumentation* auf dem aktuellen Stand gehalten. Zu Beginn mussten wir uns auch mit dem Textsatzsystem LaTeX vertraut machen.

In Abbildung 6.1 ist ersichtlich, welche Tätigkeiten von Woche 44 bis Woche 6 ausgeübt wurden.

Abbildung 6.1.: Projekt-Zeitplan

7. Persönliche Berichte

7.1. Beat Wieland

Ich habe dieses Projekt gewählt, weil es mir die Möglichkeit bot, viel neues Wissen zu erlernen. In vielen Projekten musste eine Applikation entwickelt werden. Dies ist zwar interessant, aber für mich war der Reiz grösser, wieder einmal etwas mit Hardware zu machen. Ich hatte vorher noch nie etwas von den Vehikeln von Braitenberg gehört und auch von Swarm Intelligence hatte ich nur eine vage Vorstellung.

Im Verlaufe des Projekts gab es hin und wieder einige Überraschungen. Die Sensoren waren beispielsweise nicht so genau, wie ich mir das gewünscht hätte. Bei den optischen Sensoren liegen die Werte für Weiss und Schwarz relativ nahe beieinander. Auch lieferten unsere drei Sensoren alle leicht unterschiedliche Werte für dieselbe Farbe. Mit einigem Feintuning ist es uns aber gelungen diese Probleme zu lösen.

Auch der beschränkte Speicherplatz für Benutzerprogramme hat uns im letzten, etwas grösseren Projekt, zu schaffen gemacht. Dieses Problem konnten wir lösen, indem wir auf Funktionen von speicherintensiven Klassen verzichteten.

Wir hatten uns zu Beginn des Projektes entschieden, die Dokumentation mit LaTeX zu schreiben. Dies führte entgegen unserer Befürchtungen zu keinen Problemen.

Insgesamt bin ich mit dem Verlauf des Projektes sehr zufrieden. Ich würde sicherlich wieder einmal ein Projekt durchführen, dass etwas mit Robotik oder künstlicher Intelligenz zu tun hat!

7.2. Daniel Bögli

Wieder einmal etwas mit Hardware machen, dass war mein Wunsch bei der Auswahl der Studienarbeit. Leider war in der Auswahlliste nichts dergleichen zu finden. Schnell war die Idee geboren “etwas” mit den Lego Mindstorms zu machen. Mit Herrn Joller fanden wir einen Betreuer, der uns bei der Verwirklichung dieser Idee half.

Das Programmieren des RCX mit Java war für mich in zweierlei Hinsicht etwas Neues. Ich hatte vorher noch nie mit Java programmiert und die Lego RCX-Umgebung war mir unbekannt. Da die leJOS API gut dokumentiert ist, ging die Einarbeitung dennoch zügig vonstatten.

Die meisten Probleme wurden von der Hardware verursacht. Besonders die COM-Port Tower schienen ein Eigenleben zu besitzen. Helles Licht von Leuchtstoffröhren kann die Übertragung über einen COM-Port Tower empfindlich stören. Dementsprechend war der einzige USB-Tower begehrt. Der begrenzte Speicherplatz machte sich gegen Ende des Swarm Intelligence Projektes bemerkbar. Speicheroptimierung war angesagt. Klassen

die nicht unbedingt benötigt wurden, mussten eingespart werden. Dadurch war kein Ausgabe mehr von Strings über das LCD Display möglich. Das Debuggen des RCX gestaltete sich noch schwieriger als bisher.

Die leJOS API ist allen zu empfehlen, die Lego Mindstorms Objekt Orientiert programmieren wollen. Ausser der Klasse Math funktioniert die Virtual Machine sehr zuverlässig. Als Entwicklungsumgebung ist Eclipse zu empfehlen. Die automatische Syntax Überprüfung spart viel Zeit. Wenn man die Wahl hat, empfehle ich unbedingt den USB Tower zu verwenden. Der sendestärkere USB Tower funktioniert viel zuverlässiger als sein COM-Port Pendant. Die Programmierung des RCX mit NQC ist nur bedingt zu empfehlen. Zu eingeschränkt ist der Sprachumfang.

Die Programmierung der Mindstorms machte Spass und hatte auch die eine oder andere Knacknuss zu bieten. Das Erfolgserlebnis ist höher wenn sich die Software “bewegt”, als wenn nur ein paar Grafiken oder Zeichen auf einem Monitor auftauchen.

7.3. Michael Remund

Ich habe mich für diese Studienarbeit entschieden, weil ich wieder einmal etwas programmieren wollte, was nicht nur auf dem PC-Bildschirm zu betrachteten ist. Die Lego Mindstorm Roboter eignen sich für diese Arbeit sehr gut, da man mit Lego Technics innert kürzester Zeit unterschiedliche Vehikel bauen kann. Rasch merkten wir, dass Java die umfangreichste und geeignetste Programmiersprache ist, um den RCX zu programmieren.

Unter dem ausgeschriebenen Titel “LEGO Robotik - Swarm Intelligence” habe ich mir vorgestellt, dass mehrere Roboter zusammen eine Aufgabe lösen müssen, indem sie miteinander kommunizieren. Wir stellten jedoch fest, dass die Infrarot Towers (vor allem die mit der COM Schnittstelle) eine beschränkte Übertragungs-Sicherheit und -Geschwindigkeit bieten. Daher ist es nicht möglich, dass mehrere Roboter in vernünftiger Zeit miteinander kommunizieren. Für die Implementation unseres Swarm Algorithmus ist dies auch nicht nötig.

Das Anwenden der Jini Technologie bestätigte mir ebenfalls, dass die Kommunikation zwischen den Robotern nicht trivial ist.

Im Verlaufe der Arbeit stiessen wir natürlich auch auf Probleme. So hatten wir zu wenig RAM auf dem RCX und vermeintlich zu wenig Sensoren um eine grüne Markierung zu erkennen. Ich stellte fest, dass wenn vier Leute versuchen ein Problem zu lösen, es sehr rasch viele Lösungsvorschläge gibt. Beide Probleme konnten so in vernünftiger Zeit gelöst werden und die “Ameise” erfolgreich auf ihre Wegsuche gelassen werden.

Da ich ein Natur-Mensch bin, machte es mir viel Spass, die Informatik mit der Biologie zu kombinieren. Diese Studienarbeit war für mich eine ideale Arbeit!

7.4. Urs Heimann

Swarm Intelligence faszinierte mich schon lange. Wie schaffen es kleine Insekten sich in grossen Verbänden zu Organisieren? Und das offensichtlich oft besser als wir Menschen

mit enorm viel mehr Hirnmasse. Leider hatte ich bisher nie die Gelegenheit, mich tiefer mit dieser Materie zu befassen, und somit kam mir diese Studienarbeit gerade recht.

Unglücklicherweise musste ich zu Beginn des Projekts für zwei Wochen ins Militär, wodurch ich die ganze Evaluationsphase verpasste. Ich brauchte danach einige Zeit bis ich mich eingearbeitet und den Wissensrückstand wieder aufgeholt hatte.

Die Lego Roboter erwiesen sich als Ideal für unsere Zwecke. Sie lassen sich einfach und schnell anpassen und optimieren, womit wir uns auf die Programmierung konzentrieren konnten. Allerdings sind wir an die Grenzen gestossen was die Anzahl und Genauigkeit der Sensoren und den vorhandenen Speicherplatz betrifft. Um einen noch intelligenteren Roboter zu Programmieren, hätten wir wohl die Java Umgebung verlassen müssen.

Die Arbeit gab mir Gelegenheit die frisch erworbenen Java Kenntisse zu vertiefen und zu erweitern. Die Eclipse Entwicklungsumgebung war für mich neu, doch sie hat mich überzeugt und ich werde sie bestimmt weiterverwenden. LaTeX kannte ich bereits von der letzten Arbeit und bin mehr und mehr ein Fan davon. Vor allem wenn mehrere Leute gleichzeitig an der Dokumentation arbeiten, bringt es nur Vorteile gegenüber Word und Ähnlichem.

Schlussendlich hat mir die Arbeit gut gefallen. Auch die Zusammenarbeit im Team hat ohne grössere Reibereien geklappt und die einzige Krisensitzung hat ihr Ziel erreicht und das Problem (Speichermangel), war zehn Minuten später behoben.

A. Source Code der Braitenberg Vehikel

A.1. Vehikel 1

```
package vehicle01;

import josx.platform.rcx.*;
import tools.PWM;

public class Vehicle01
{
   private Sensor lightSensor;
   private PWM rightMotor;
   private PWM leftMotor;

   private final int DARKNESS = 22; // Bei diesem Wert wird gestoppt
   private final int GRADIENT = 4;  // Steilheit der Funktion

   public Vehicle01()
   {
      // Lichtsensor definieren
      lightSensor = Sensor.S1;
      lightSensor.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensor.activate();

      // Motoren zuweisen und starten
      leftMotor = new PWM(Motor.A);
      rightMotor = new PWM(Motor.B);
      leftMotor.forward();
      rightMotor.forward();
   }

   public void hideFromLight()
   {
      int value;
      leftMotor.start();
      rightMotor.start();
```

```
      leftMotor.startMotor();
      rightMotor.startMotor();

      while (Button.readButtons() == 0)
      {
         // Sensorwert einlesen
         value = lightSensor.readValue();
         Thread.yield();
         printValue(value);

         // Motorgeschwindigkeit setzten
         leftMotor.setSpeed((value - DARKNESS) * GRADIENT);
         rightMotor.setSpeed((value - DARKNESS) * GRADIENT);
      }

      // Programmende
      lightSensor.passivate();
      leftMotor.endThread();
      rightMotor.endThread();
   }

   private void printValue(int value)
   {
      LCD.showNumber(value);
      LCD.refresh();
   }

   public static void main(String[] args)
   {
      Vehicle01 theTimid = new Vehicle01();
      theTimid.hideFromLight();
   }
}
```

A.2. Vehikel 2

```
package vehicle02;

import josx.platform.rcx.*;
import tools.PWM;

public class Vehicle02
{
   private Sensor lightSensorRight;
   private Sensor lightSensorLeft;
   private PWM rightMotor;
   private PWM leftMotor;

   private final int DARKNESS = 10; // Bei diesem Wert wird gestoppt
   private final int GRADIENT = 4;  // Steilheit der Funktion

   public Vehicle02()
   {
      // Lichtsensoren definieren
      lightSensorLeft = Sensor.S1;
      lightSensorRight = Sensor.S2;

      lightSensorLeft.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensorLeft.activate();

      lightSensorRight.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensorRight.activate();

      // Motoren zuweisen und starten
      leftMotor = new PWM(Motor.A);
      rightMotor = new PWM(Motor.B);
      leftMotor.forward();
      rightMotor.forward();
   }

   public void steer()
   {
      int sensorValueLeft;
      int sensorValueRight;
```

```
      leftMotor.start();
      rightMotor.start();
      leftMotor.startMotor();
      rightMotor.startMotor();

      while (Button.readButtons() == 0)
      {
         // Sensorwerte einlesen
         sensorValueLeft = lightSensorLeft.readValue();
         sensorValueRight = lightSensorRight.readValue();

         // Motorgeschwindigkeiten setzten
         leftMotor.setSpeed((sensorValueLeft - DARKNESS) * GRADIENT);
         rightMotor.setSpeed((sensorValueRight - DARKNESS) * GRADIENT);
      }

      lightSensorLeft.passivate();
      lightSensorRight.passivate();
      leftMotor.endThread();
      rightMotor.endThread();
   }

   private void printValue(int value)
   {
      LCD.showNumber(value);
      LCD.refresh();
   }

   public static void main(String[] args)
   {
      Vehicle02 theTimid = new Vehicle02();
      theTimid.steer();
   }
}
```

A.3. Vehikel 3

```
package vehicle03;

import josx.platform.rcx.*;
import tools.PWM;

public class Vehicle03
{
   private Sensor lightSensorRight;
   private Sensor lightSensorLeft;
   private PWM rightMotor;
   private PWM leftMotor;

   private final int DARKNESS = 30; // ab diesem Wert max. Geschw.
   private final int GRADIENT = -6; // Steilheit der Funktion

   public Vehicle03()
   {
      // Lichtsensoren definieren
      lightSensorLeft = Sensor.S1;
      lightSensorRight = Sensor.S2;

      lightSensorLeft.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensorLeft.activate();

      lightSensorRight.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensorRight.activate();

      // Motoren zuweisen und starten
      leftMotor = new PWM(Motor.A);
      rightMotor = new PWM(Motor.B);
      leftMotor.forward();
      rightMotor.forward();
   }

   public void steer()
   {
      int sensorValueLeft;
      int sensorValueRight;
```

```
      leftMotor.start();
      rightMotor.start();
      leftMotor.startMotor();
      rightMotor.startMotor();

      while (Button.readButtons() == 0)
      {
         // Sensorwerte einlesen
         sensorValueLeft = lightSensorLeft.readValue();
         sensorValueRight = lightSensorRight.readValue();

         // Motorgeschwindigkeiten setzten
         leftMotor.setSpeed((sensorValueLeft - DARKNESS) * GRADIENT + 100);
         rightMotor.setSpeed((sensorValueRight - DARKNESS) * GRADIENT + 100);
      }

      // Programmende
      lightSensorLeft.passivate();
      lightSensorRight.passivate();
      leftMotor.endThread();
      rightMotor.endThread();
   }

   public static void main(String[] args)
   {
      Vehicle03 theTimid = new Vehicle03();
      theTimid.steer();
   }
}
```

A.4. Vehikel 4

```
package vehicle04;

import josx.platform.rcx.*;
import tools.*;

public class Vehicle04
{
   private PWM rightMotor;
   private PWM leftMotor;

   private BlackAndWhite lightSensorRight;
   private BlackAndWhite lightSensorLeft;

   public Vehicle04()
   {
      // Lichtsensoren definieren
      lightSensorRight = new BlackAndWhite(Sensor.S1);
      lightSensorLeft = new BlackAndWhite(Sensor.S2);

      // Motoren zuweisen und starten
      rightMotor = new PWM(Motor.A);
      leftMotor = new PWM(Motor.B);
      leftMotor.forward();
      rightMotor.forward();

   }

   public void steer()
   {
      int speed;

      leftMotor.start();
      rightMotor.start();
      leftMotor.startMotor();
      rightMotor.startMotor();

      while (Button.readButtons() == 0)
      {
         // Motorgeschwindigkeiten setzten
         speed = calcSpeed(lightSensorLeft);

         DebugRCX.printValue(speed);
```

```
        leftMotor.setSpeed(speed);

        rightMotor.setSpeed(calcSpeed(lightSensorRight));
    }

    // Programmende

    leftMotor.endThread();
    rightMotor.endThread();
}

private int calcSpeed_linear(BlackAndWhite x)
{
    int value = x.getValue();
    int min = x.getBlack();
    int max = x.getWhite();
    int avg = 35;
    int gradient1 = 100 / (avg - min);
    int gradient2 = -100 / (max - avg);

    if ((value >= min) && (value <= avg))
    {
        return (int) ((value - min) * gradient1);
    }
    else
    {
        if ((value > avg) && (value <= max))
        {
            return (int) (gradient2 * (value - avg) + 100);
        }
        else
        {
            return 0;
        }
    }
}

private int calcSpeed(BlackAndWhite lightSensor)
{
    int helligkeit = lightSensor.getValue();
    int erwartungswert = 40;
    int standardabweichung = 20;
    int korrekturfaktor = 5014;
```

```
      //DebugRCX.breakPoint("x",helligkeit);

      double exponent = -((helligkeit-erwartungswert) * (helligkeit-erwartungswert)) /
                          (2 * standardabweichung * standardabweichung);

      //DebugRCX.breakPoint("exp",(int) exponent);

      double zaehler = korrekturfaktor * Math.exp(exponent);

      //DebugRCX.breakPoint("z",(int) zaehler);

      double nenner = standardabweichung * Math.sqrt(2 * Math.PI );

      //DebugRCX.breakPoint("n",(int) abweichung);

      return (int) (zaehler/nenner);
   }

    public static void main(String[] args)
   {
      Vehicle04 theTimid = new Vehicle04();
      theTimid.steer();
   }
}
```

A.5. Vehikel 5

```
package vehicle05;

import josx.platform.rcx.*;
import tools.PWM;
import tools.ThresholdDevice;
import tools.ThresholdMotor;

public class Vehicle05
{

   private ThresholdDevice tA, tB, tC, tD, tE, tF, tG, tH, tI;
   private ThresholdMotor leftMotor, rightMotor, motorThree;
   private Sensor lightSensorLeft, lightSensorRight, touchSensor;

   public Vehicle05()
   {

   }

   public void init5a()
   {
      lightSensorLeft = Sensor.S1;
      lightSensorLeft.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
         SensorConstants.SENSOR_MODE_PCT);
      lightSensorLeft.activate();

      tA = new ThresholdDevice(70);
      tB = new ThresholdDevice(30);
      tC = new ThresholdDevice(-50);
      tD = new ThresholdDevice(50);

      leftMotor = new ThresholdMotor(Motor.A, 50);

      tA.attachInputDevice(lightSensorLeft);
      tB.attachInputDevice(lightSensorLeft);

      tD.attachInputDevice(tA);
      tA.attachOutputDevice(tD, false);

      tC.attachInputDevice(tB);
      tB.attachOutputDevice(tC, true);
```

```
      tD.attachInputDevice(tC);
      tC.attachOutputDevice(tD, true);

      tD.attachInputDevice(tD);
      tD.attachOutputDevice(tD, false);

      leftMotor.attachInputDevice(tD);
      tD.attachOutputDevice(leftMotor, false);

   }

   public void run5a()
   {
      int sensorValue;
     while (Button.readButtons() == 0)
      {
         // Sensorwerte einlesen
         sensorValue = lightSensorLeft.readValue();

         // Sensorverte in Schwellwerteinheiten einspeisen
         tA.setInput(lightSensorLeft, sensorValue);
         tB.setInput(lightSensorLeft, sensorValue);

         // Gewünschten Statuswert anzeigen
         //LCD.showNumber(theMotor.getThreshold());
         //LCD.showNumber(theMotor.getOutput());
         LCD.showNumber(sensorValue);

         LCD.refresh();
      }

      // Programmende
      lightSensorLeft.passivate();
      leftMotor.killMotor();

   }


   public void init5b()
   {
      lightSensorLeft = Sensor.S1;
      lightSensorLeft.setTypeAndMode(
         SensorConstants.SENSOR_TYPE_LIGHT,
```

```
   SensorConstants.SENSOR_MODE_PCT);
lightSensorLeft.activate();

lightSensorRight = Sensor.S2;
lightSensorRight.setTypeAndMode(
   SensorConstants.SENSOR_TYPE_LIGHT,
   SensorConstants.SENSOR_MODE_PCT);
lightSensorRight.activate();

touchSensor = Sensor.S3;
touchSensor.setTypeAndMode(
   SensorConstants.SENSOR_TYPE_TOUCH,
   SensorConstants.SENSOR_MODE_PCT);
touchSensor.activate();

tA = new ThresholdDevice(0);
tB = new ThresholdDevice(0);
tC = new ThresholdDevice(50);
tD = new ThresholdDevice(50);
tE = new ThresholdDevice(200);
tF = new ThresholdDevice(50);

leftMotor  = new ThresholdMotor(Motor.A, 50);
rightMotor = new ThresholdMotor(Motor.B, 50);
motorThree = new ThresholdMotor(Motor.C, 50);

tA.attachInputDevice(lightSensorLeft);
tA.attachInputDevice(lightSensorRight);

tB.attachInputDevice(lightSensorLeft);
tB.attachInputDevice(lightSensorRight);

tE.attachInputDevice(lightSensorLeft);
tE.attachInputDevice(lightSensorRight);

tF.attachInputDevice(touchSensor);

tC.attachInputDevice(tA);
tA.attachOutputDevice(tC, false);

tD.attachInputDevice(tB);
tB.attachOutputDevice(tD, false);

tC.attachInputDevice(tE);
```

```
    tE.attachOutputDevice(tC, true);

    tD.attachInputDevice(tE);
    tE.attachOutputDevice(tD, true);

    leftMotor.attachInputDevice(tC);
    tC.attachOutputDevice(leftMotor, false);

    rightMotor.attachInputDevice(tD);
    tD.attachOutputDevice(rightMotor, false);

    leftMotor.attachInputDevice(tF);
    tF.attachOutputDevice(leftMotor, true);

    rightMotor.attachInputDevice(tF);
    tF.attachOutputDevice(rightMotor, true);

    tF.attachInputDevice(tF);
    tF.attachOutputDevice(tF, false);

    motorThree.attachInputDevice(tF);
    tF.attachOutputDevice(motorThree, false);
}

public void run5b()
{
    int lightLeft, lightRight, touch;
  while (Button.readButtons() == 0)
    {
       // Sensorwerte einlesen
       lightLeft  = lightSensorLeft.readValue();
       lightRight = lightSensorRight.readValue();
       touch      = touchSensor.readValue();

       // Sensorverte in Schwellwerteinheiten einspeisen
       tA.setInput(lightSensorLeft, lightLeft*-1);
       tB.setInput(lightSensorLeft, lightLeft);
       tE.setInput(lightSensorLeft, lightLeft);

       tA.setInput(lightSensorRight, lightRight);
       tB.setInput(lightSensorRight, lightRight*-1);
       tE.setInput(lightSensorRight, lightRight);

       tF.setInput(touchSensor, touch);
```

```
            // Gewünschten Statuswert anzeigen
            //LCD.showNumber(theMotor.getThreshold());
            //LCD.showNumber(theMotor.getOutput());
            //LCD.showNumber(sensorValue);

            LCD.refresh();
         }

         // Programmende
         lightSensorLeft.passivate();
         lightSensorRight.passivate();
         touchSensor.passivate();
         leftMotor.killMotor();
         rightMotor.killMotor();
         motorThree.killMotor();

      }

      public static void main(String[] args)
      {
         Vehicle05 logic = new Vehicle05();
         logic.init5b();
         logic.run5b();
      }

}
```

A.6. Tools

A.6.1. PWM

```
package tools;

import josx.platform.rcx.*;
import java.lang.Thread;

/**
 * Ansteuerung eines Motors mit Puls Weiten Modulation.
 */
public class PWM extends Thread
{
   private Motor theMotor;
   private boolean running;
   private boolean forward;
   private boolean endPWMThread;
   private int runTime;
   private int stopTime;

   public PWM(Motor aMotor)
   {
      theMotor = aMotor;
      running = false;
      runTime = 20;
      stopTime = 5;
      forward = true;
      endPWMThread = false;
      theMotor.setPower(1);
   }

   /**
    * Setzten der Geschwindigkeit in %.
    * @param Geschwindigkeit 0-100.
    */
   public void setSpeed(int percent)
   {
      if (percent > 100)
      {
         percent = 100;
         theMotor.setPower(7);
      }
      else
```

```
      {
         theMotor.setPower(1);
      }

      if (percent < 0)
      {
         percent = 0;
      }

      runTime = percent;
      stopTime = 100 - percent;
   }

   private void PWMModulation()
   {
      if (runTime > 0)  // Wenn die Motorlaufzeit <= 0 ist,
      {                 // Motor nicht einschalten.
         if (forward)
         {
            theMotor.forward();
         }
         else
         {
            theMotor.backward();
         }

         try
         {
            Thread.sleep(runTime);
         }
         catch (InterruptedException e)
         {
         }
      }

      if (stopTime > 0) // Wenn die Motorstoppzeit <= 0 ist,
      {                 // Motor nicht stoppen.
         theMotor.stop();
         try
         {
            Thread.sleep(stopTime);
         }
         catch (InterruptedException e)
         {
```

```
      }
   }
}

/**
 * Forwärts fahren.
 */
public void forward()
{
   forward = true;
}

/**
 * Rückwärts fahren.
 */
public void backward()
{
   forward = false;
}

/**
 * Startet die PWM. Läuft in einem Thread.
 * @see java.lang.Runnable#run()
 */
public void run()
{
   while (!endPWMThread)
   {
      if (running)
      {
         PWMModulation();
      }
   }
}

/**
 * Startet den Motor.
 */
public void startMotor()
{
   running = true;
}

/**
```

```
    * Stoppt den Motor.
    */
   public void stopMotor()
   {
      running = false;
   }

   /**
    * Stoppt die PWM, und den Thread.
    */
   public void endThread()
   {
      endPWMThread = true;
   }
}
```

A.6.2. Threshold Device

```
/*
 * Class:  ThresholdDevice
 * -----------------------
 * Autor:  Urs Heimann
 * Datum:  9.12.2002
 * Beschreibung:
 * Implementation der Schwellwert-Einheiten wie sie Braitenberg in
 * seinem Buch beim Vehikel 5 beschreibt. Jede Device hat einen eigenen
 * Schwellwert und kann mit anderen Devices verknüpft werden.
 * Wenn die sumierten Eingangswerte grösser sind als der Schwellwert,
 * wird am Ausgang der Wert 100 ausgegeben, sonst der Wert 0.
 */

package tools;

import java.util.*;


public class ThresholdDevice
{
   protected int threshold;        // Schwellwert
   protected int output;           // Ausgangswert
   private Vector inputDevices;    // Eingangseinheiten
   private Vector inputValues;     // Eingabewerte
   private Vector outputDevices;   // Ausgabe Einheiten
   private Vector outputInvert;    // Ausgabe Invertierung
   private int inCounter;          // Anzahl Eingangseinheiten
   private int outCounter;         // Anzahl Ausgangseinheiten


   // Constructors
   public ThresholdDevice()
   {
      threshold  = 0;
      outCounter = 0;
      inCounter  = 0;
      inputDevices  = new Vector(10);
      inputValues   = new Vector(10);
      outputDevices = new Vector(10);
      outputInvert  = new Vector(10);
   }
```

```
public ThresholdDevice(int initThreshold)
{
   threshold  = initThreshold;
   outCounter = 0;
   inCounter  = 0;
   inputDevices  = new Vector(10);
   inputValues   = new Vector(10);
   outputDevices = new Vector(10);
   outputInvert  = new Vector(10);
}


// Schwellwert setzen
public void setThreshold(int newThreshold)
{
   threshold = newThreshold;
}

// Eingangseinheit verbinden
public void attachInputDevice(Object anObject)
{
   inputDevices.insertElementAt(anObject, inCounter);
   inputValues.insertElementAt(new IntWrapper(0), inCounter);
   inCounter++;
}

// Ausgangseinheit verbinden
public void attachOutputDevice(Object anObject, boolean inverted)
{
   outputDevices.insertElementAt(anObject, outCounter);
   if( inverted )
   {
      outputInvert.insertElementAt(new IntWrapper(-1), outCounter);
   }
   else
   {
      outputInvert.insertElementAt(new IntWrapper(1), outCounter);
   }
   outCounter++;
}

// Eingangswert setzen
public void setInput(Object inDevice, int newValue)
{
```

```
      int oldOutput = output;

      setInputValue(inDevice, newValue);
      updateOutput();

      if( oldOutput != output)
      {
         notifyOutputDevices();
      }
   }

   protected void setInputValue(Object inDevice, int newValue)
   {
      Object tempElement;
      for (int i=0; i < inCounter; i++)
      {
         if(inputDevices.elementAt(i) == inDevice)
         {
            ((IntWrapper)inputValues.elementAt(i)).setValue(newValue);
         }
      }
   }

   protected void updateOutput()
   {
      int sum = 0;
      Object tempElement;

      for (int i=0; i < inCounter; i++)
      {
        tempElement = inputValues.elementAt(i);
         if( tempElement != null )
         {
          {
               sum = sum + ((IntWrapper)tempElement).getValue();
          }
         }
      }

      if( sum >= threshold )
      {
         output = 100;
      }
      else
```

```
      {
         output = 0;
      }

   }

   protected void notifyOutputDevices()
   {
      ThresholdDevice out;
      int invert = 1;

      for (int i=0; i < outCounter; i++)
      {
        out = (ThresholdDevice)outputDevices.elementAt(i);
        if( out != null )
        {
         invert = ((IntWrapper)outputInvert.elementAt(i)).getValue();
         out.setInput(this, (output * invert));
        }
      }
   }


   public int getOutput()
   {
      return output;
   }

   public int getThreshold()
   {
      return threshold;
   }

}
```

A.6.3. Threshold Motor

```
/*
 * Class:  ThresholdMotor
 * ------------------------
 * Autor:  Urs Heimann
 * Datum:  9.12.2002
 * Beschreibung:
 * Spezialisierung der ThresholdDevice.
 * Hat die gleiche Funktion, aber gibt zusätzlich den
 * Ausgabewert an einen Motor aus.
 */

package tools;


import josx.platform.rcx.*;



public class ThresholdMotor extends ThresholdDevice
{

   private PWM theMotor;


   public ThresholdMotor(Motor aMotor)
   {
      super();
      theMotor = new PWM(aMotor);
      theMotor.forward();
      theMotor.start();
      theMotor.startMotor();
      theMotor.setSpeed(0);

   }

   public ThresholdMotor(Motor aMotor, int threshold)
   {
    super(threshold);
      theMotor = new PWM(aMotor);
      theMotor.forward();
      theMotor.start();
      theMotor.startMotor();
```

```
      theMotor.setSpeed(0);

   }

   public void setInput(Object inDevice, int newValue)
   {
      int oldOutput = output;

      super.setInputValue(inDevice, newValue);
      super.updateOutput();

      if( oldOutput == output)
      {
         return;
      }

      super.notifyOutputDevices();

      if( output >= threshold )
      {
         theMotor.setSpeed(101);
      }
      else
      {
         theMotor.setSpeed(0);
      }
   }

   public void killMotor()
   {
      theMotor.endThread();
   }
}
```

A.6.4. IntWrapper

```
/* Class:  IntWrapper
 * Author: Urs Heimann
 * Datum:  9.12.2002
 */

package tools;


public class IntWrapper
{

   private int value;

   public IntWrapper()
   {
      value = 0;
   }

   public IntWrapper(int initValue)
   {
      value = initValue;
   }

   public int getValue()
   {
      return value;
   }

   public void setValue(int newValue)
   {
      value = newValue;
   }


}
```

A.6.5. BlackAndWhite

```
package tools;
import josx.platform.rcx.*;

/**
 * Speicher die hellsten und dunkelsten Werte eines Sensors.
 * Kannn abgefragt werden, ob sich der Sensor über einer schwarzen
 * oder weissen Stelle befindet
 */

public class BlackAndWhite
{
   private int white;
   private int black;
   private int tolerance;
   private Sensor theSensor;

   public BlackAndWhite(Sensor theSensor)
   {
      this.theSensor = theSensor;
      theSensor.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
                               SensorConstants.SENSOR_MODE_PCT);

      theSensor.activate();
      white = 45;
      black = 28;
      tolerance = 7;
   }

   /**
    * Abfrage ob der Sensor sich über schwarzem Untergrund befindet.
    * @return true wenn Schwarz.
    */
   public boolean isBlack()
   {
      int measure;

      measure = theSensor.readValue();
      checkMeasure(measure);

      if (measure < (black + tolerance))
       {
          return true;
```

```
        }

      return false;
   }


   /**
    * Gibt den Mittelwert der gemessenen Werte zurück.
    * @return Mittelwert.
    */
   public int median()
   {
      return (white + black) / 2;
   }


   /**
    * Gibt den Sensor Wert zurück. Speichert extremum Werte.
    * @return Sensor Wert.
    */
   public int getValue()
   {
      int measure;

      measure = theSensor.readValue();
      checkMeasure(measure);
      return measure;
   }

   private void checkMeasure(int measure)
   {
      if (measure < black)
      {
         black = measure;
      }

      if (measure > white)
      {
         white = measure;
      }
   }
```

```
    /**
     * Gibt die Toleranz für Schwarz und Weiss zurück.
     * @return Toleranz
     */
    public int getTolerance()
    {
       return tolerance;
    }

    /**
     * Setzt die Toleranz für Schwarz und Weiss.
     * @param tolerance Toleranz.
     */
    public void setTolerance(int tolerance)
    {
       this.tolerance = tolerance;
    }

    /**
     * Gibt den dunkelsten gemessenen Wert zurück.
     * @return Dunkelster Wert
     */
    public int getBlack()
    {
       return black;
    }

    /**
     * Gibt den hellsten gemessenen Wert zurück.
     * @return Hellster Wert.
     */
    public int getWhite()
    {
       return white;
    }

}
```

B. Source Code des Jini Beispiels

B.1. mindstorm

B.1.1. MindstromProxy

```
/**
 * MindstormProxy.java
 *
 */

package mindstorm;

import java.rmi.server.UnicastRemoteObject;
import net.jini.core.event.UnknownEventException;
import net.jini.core.event.RemoteEvent;
import net.jini.core.event.RemoteEventListener;
import shared.Mindstorm;
import java.io.Serializable;
import java.rmi.RemoteException;
import rcx.RCXListener;
import rcx.RCXPort;
import rcx.comm.Win32USBPort;

public class MindstormProxy
extends UnicastRemoteObject
implements Mindstorm, Serializable, RCXListener
{

// Make sure to generate a stub class using the
// RMI compiler since this class extends
// UnicastRemoteObject.

protected static final byte FORWARDS = (byte) 0x80;
protected static final byte BACKWARDS = (byte) 0x00;
protected static final byte ON = (byte) 0x80;
protected static final byte OFF = (byte) 0x40;
protected static final byte MOTOR_A = (byte) 0x01;
protected static final byte MOTOR_C = (byte) 0x04;
```

```
protected static final byte MOTOR_DIR = (byte) 0xe1;
protected static final byte MOTOR_ON_OFF = (byte) 0x21;

// Store our unique ID so that the client
// can differentitate between robots.
protected String rcxId = null;

protected RCXPort rcxPort = null;
// protected Win32USBPort rcxUSBPort = null;

protected RemoteEventListener remoteListener = null;

// Store any message received from the RCX
// as a raw byte array for retrieval by the client.
// Note that any subsequent message will overwrite
// the existing one and if messages are retireved
// in quick succession data may be lost. A more
// robust implementation could be emplyed for a
// production system...
protected byte[] lastRCXEvent = null;

// Similarly for the most recent dance step performed...
protected int lastDanceStep = 0;

protected long seqNo = 0;

public MindstormProxy() throws RemoteException
{
}

public String getRcxId()
{
return rcxId;
}

protected void setRcxId(String id)
{
rcxId = id;
}

protected void openRcxPort(String port)
{
// Open a specific serial port
rcxPort = new RCXPort(port);
```

```
//rcxUSBPort = new Win32USBPort();
// Register as a listener with the RCX
rcxPort.addRCXListener(this);

}

protected void executeMovement(int movementId)
{
// Execute one of our robot's 8
// spectacular dance steps.
System.out.println("Executing step: " + movementId);
switch (movementId)
{
case 0 :
// Directly forward
setMotorDir(FORWARDS, (byte) (MOTOR_A | MOTOR_C));
motorOn((byte) (MOTOR_A | MOTOR_C));
break;
case 1 :
// Directly back
setMotorDir(BACKWARDS, (byte) (MOTOR_A | MOTOR_C));
motorOn((byte) (MOTOR_A | MOTOR_C));
break;
case 2 :
// Rotate right
setMotorDir(FORWARDS, MOTOR_A);
setMotorDir(BACKWARDS, MOTOR_C);
motorOn((byte) (MOTOR_A | MOTOR_C));
break;
case 3 :
// Rotate left
setMotorDir(BACKWARDS, MOTOR_A);
setMotorDir(FORWARDS, MOTOR_C);
motorOn((byte) (MOTOR_A | MOTOR_C));
break;
case 4 :
// Forward right
setMotorDir(FORWARDS, MOTOR_A);
motorOn(MOTOR_A);
break;
case 5 :
// Forward left
setMotorDir(FORWARDS, MOTOR_C);
motorOn(MOTOR_C);
```

```
break;
case 6 :
// Reverse right
setMotorDir(BACKWARDS, MOTOR_A);
motorOn(MOTOR_A);
break;
case 7 :
// Reverse left
setMotorDir(BACKWARDS, MOTOR_C);
motorOn(MOTOR_C);
break;
}
// Each dance step is of a 0.3 second duration
try
{
Thread.sleep(300);
}
catch (InterruptedException e)
{
//
}
motorsOff();
}

protected void motorOn(byte motors)
{
// Turn one or both motors on
byte[] msg = new byte[] {
};
sendToRcx(new byte[] { MOTOR_ON_OFF, (byte) (ON | motors)});
}

protected void motorsOff()
{
// Turn both motors off
sendToRcx(new byte[] { MOTOR_ON_OFF, (byte) (OFF | MOTOR_A | MOTOR_C)});
}

protected void setMotorDir(byte dir, byte motors)
{
// Set direction for one or both motors
sendToRcx(new byte[] { MOTOR_DIR, (byte) (dir | motors)});
}
```

```
protected void sendToRcx(byte[] msg)
{

System.out.println("Sending to port: " + byteArrayToString(msg));

if (!rcxPort.write(msg))
{
System.err.println("Error writing to port");
}
}

public void imitateThis(int danceStep) throws RemoteException
{
// This should be the dance step that the other
// robot has just performed. We will attempt to
// imitate it. Handle this from a new thread.
new ImitationThread(danceStep, this).start();
}

public void receivedMessage(byte[] msg)
{
// Receive messages from the RCX
if (null != msg)
{
System.out.println("RCX message: " + byteArrayToString(msg));

if (null != remoteListener)
{
// Store the event contents
// for retrieval by the listener
lastRCXEvent = msg;

// Notify the listener of the event.
// Listener will have to call back to
// obtain the details.
RemoteEvent evt =
new RemoteEvent(this, EVENT_RCX_MSG, seqNo++, null);
try
{
remoteListener.notify(evt);
}
catch (UnknownEventException e)
{
System.err.println("Event exception");
```

```
}
catch (java.rmi.RemoteException e)
{
System.err.println("Remote exception");
}

}
}
}

protected String byteArrayToString(byte[] msg)
{

// Convert an array of bytes to a human-readable
// string

StringBuffer sBuf = new StringBuffer();

for (int ix = 0; ix < msg.length; ix++)
{
int dm = msg[ix] >= 0 ? (int) msg[ix] : ((int) msg[ix]) + 256;
sBuf.append(Integer.toHexString(dm) + " ");
}

return sBuf.toString();
}

public void receivedError(String err)
{
// Receive errors from the RCX
System.err.println("RCX error: " + err);
}

public byte[] GetLastRCXMessage()
{
// This will be called by the client
// to retrieve details of the last message
// after we have sent a notification.
return lastRCXEvent;
}

public int GetLastStepPerformed()
{
// This will be called by the client
```

```
// to retrieve details of the last step
// after we have sent a notification.
return lastDanceStep;
}

public void RegisterRemoteListener(RemoteEventListener listener)
{
// Called by the client to register its
// listener object (which should also extend
// UnicastRemoteObject so that we can notify it
// remotely).
remoteListener = listener;
}

// Declare this thread class as an inner class
class ImitationThread extends Thread
{

protected int step;
protected MindstormProxy proxy = null;

public ImitationThread(int danceStep, MindstormProxy proxy)
{

// Store the reference to the
// object that created us as well
// as the dance step to execute.
this.proxy = proxy;
step = danceStep;
}

public void run()
{

// Firstly execute the move in
// imitation of the other robot.
executeMovement(step);

// Then wait 10 seconds
try
{
Thread.sleep(10000);
}
catch (InterruptedException e)
```

```
{
// Do nothing
}

// Now randomly pick a new movement
// (between 0 and 7 inclusive).
int newMovement;
newMovement = (int) (8 * (Math.random()));

// Perform our randomly-selected movement.
executeMovement(newMovement);

// Now let the remote client
// know they we just performed it.
//
// The client will have to call back
// to get the details.
lastDanceStep = newMovement;
if (null != remoteListener)
{
RemoteEvent evt =
new RemoteEvent(proxy, EVENT_DANCE_STEP, seqNo++, null);
try
{
remoteListener.notify(evt);
}
catch (UnknownEventException e)
{
System.err.println("Event exception");
}
catch (java.rmi.RemoteException e)
{
System.err.println(e.toString());
}
}
}
}
}
```

B.1.2. MindstromProxy_Stub

```
// Stub class generated by rmic, do not edit.
// Contents subject to change without notice.

package mindstorm;

public final class MindstormProxy_Stub
extends java.rmi.server.RemoteStub
implements shared.Mindstorm, java.rmi.Remote
{
private static final long serialVersionUID = 2;

private static java.lang.reflect.Method $method_GetLastRCXMessage_0;
private static java.lang.reflect.Method $method_GetLastStepPerformed_1;
private static java.lang.reflect.Method $method_RegisterRemoteListener_2;
private static java.lang.reflect.Method $method_getRcxId_3;
private static java.lang.reflect.Method $method_imitateThis_4;

static {
try
{
$method_GetLastRCXMessage_0 =
shared
.Mindstorm
.class
.getMethod("GetLastRCXMessage", new java.lang.Class[] {
});
$method_GetLastStepPerformed_1 =
shared
.Mindstorm
.class
.getMethod("GetLastStepPerformed", new java.lang.Class[] {
});
$method_RegisterRemoteListener_2 =
shared.Mindstorm.class.getMethod(
"RegisterRemoteListener",
new java.lang.Class[] {
 net.jini.core.event.RemoteEventListener.class });
$method_getRcxId_3 =
shared
.Mindstorm
.class
.getMethod("getRcxId", new java.lang.Class[] {
```

```
});
$method_imitateThis_4 =
shared.Mindstorm.class.getMethod(
"imitateThis",
new java.lang.Class[] { int.class });
}
catch (java.lang.NoSuchMethodException e)
{
throw new java.lang.NoSuchMethodError(
"stub class initialization failed");
}
}

// constructors
public MindstormProxy_Stub(java.rmi.server.RemoteRef ref)
{
super(ref);
}

// methods from remote interfaces

// implementation of GetLastRCXMessage()
public byte[] GetLastRCXMessage() throws java.rmi.RemoteException
{
try
{
Object $result =
ref.invoke(
this,
$method_GetLastRCXMessage_0,
null,
-272164347401682904L);
return ((byte[]) $result);
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (java.lang.Exception e)
{
```

```
throw new java.rmi.UnexpectedException(
"undeclared checked exception",
e);
}
}

// implementation of GetLastStepPerformed()
public int GetLastStepPerformed() throws java.rmi.RemoteException
{
try
{
Object $result =
ref.invoke(
this,
$method_GetLastStepPerformed_1,
null,
-6002936083996362040L);
return ((java.lang.Integer) $result).intValue();
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (java.lang.Exception e)
{
throw new java.rmi.UnexpectedException(
"undeclared checked exception",
e);
}
}

// implementation of RegisterRemoteListener(RemoteEventListener)
public void RegisterRemoteListener(
net.jini.core.event.RemoteEventListener $param_RemoteEventListener_1)
throws java.rmi.RemoteException
{
try
{
ref.invoke(
this,
```

```
$method_RegisterRemoteListener_2,
new java.lang.Object[] { $param_RemoteEventListener_1 },
4719547772000210964L);
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (java.lang.Exception e)
{
throw new java.rmi.UnexpectedException(
"undeclared checked exception",
e);
}
}

// implementation of getRcxId()
public java.lang.String getRcxId() throws java.rmi.RemoteException
{
try
{
Object $result =
ref.invoke(
this,
$method_getRcxId_3,
null,
3809454315623997426L);
return ((java.lang.String) $result);
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (java.lang.Exception e)
{
throw new java.rmi.UnexpectedException(
```

```
"undeclared checked exception",
e);
}
}

// implementation of imitateThis(int)
public void imitateThis(int $param_int_1) throws java.rmi.RemoteException
{
try
{
ref.invoke(
this,
$method_imitateThis_4,
new java.lang.Object[] { new java.lang.Integer($param_int_1)},
-5878800822475910876L);
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (java.lang.Exception e)
{
throw new java.rmi.UnexpectedException(
"undeclared checked exception",
e);
}
}
}
```

B.1.3. MindstromService

```
/**
 * MindstormService.java
 *
 */

package mindstorm;

import net.jini.discovery.LookupDiscovery;
import net.jini.discovery.DiscoveryListener;
import net.jini.discovery.DiscoveryEvent;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.lookup.ServiceItem;
import net.jini.core.lookup.ServiceRegistration;
import net.jini.core.lease.Lease;
import net.jini.lease.LeaseRenewalManager;
import net.jini.lease.LeaseRenewalEvent;
import net.jini.lease.LeaseListener;
import java.io.Serializable;
import java.io.IOException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;

public class MindstormService
implements Serializable, LeaseListener, DiscoveryListener
{
// This service will be responsible for creating
// an instance of the proxy and registering it
// with the Jini locator service.
protected MindstormProxy proxy = null;

// The LeaseRenewalManager will ensure that the
// lease with the locator serivce is regulalry
// renewed.
protected LeaseRenewalManager leaseManager = new LeaseRenewalManager();

// Store our own ID so we can be differentiated from
// any other Mindstorms in the federation.
static protected String rcxId = null;

// The name of the serial port.
static protected String portId = null;
```

```
static public void main(String args[])
{

// Since reggie is running with a security policy,
// we will have to as well. This assumes that
// the policy file is located at
// c:\jini1_2\policy\policy.all
// Adjust this for specific installations.
System.setProperty(
"java.security.policy",
"C:\\Programme\\jini1_2_1_001\\policy\\policy.all");
System.setSecurityManager(new SecurityManager());

// Check that the correct arguments were passed in
if (args.length < 2)
{
System.err.println("Usage: java MindstormService RcxId Port");
System.exit(1);
}

rcxId = args[0];
portId = args[1];

new MindstormService();

// Ensure service runs indefinitely so
// we can keep renewing the lease.
Object keepAlive = new Object();
synchronized (keepAlive)
{
try
{
keepAlive.wait();
}
catch (java.lang.InterruptedException e)
{
// do nothing
}
}
}

public MindstormService()
{
```

```
try
{
// Lookup using Multicast (i.e. look for any lookup
// services on the network). We will receive a
// callback to the discovered() method with a
// list of all lookup services found.
LookupDiscovery lookupDiscovery =
new LookupDiscovery(LookupDiscovery.ALL_GROUPS);
lookupDiscovery.addDiscoveryListener(this);
System.out.println("ok1");

}
catch (IOException e)
{
System.err.println(e.toString());
System.exit(1);
}
}

public void discarded(DiscoveryEvent event)
{
// Must be implemented from the
// DiscoveryListener interface.
}

public void discovered(DiscoveryEvent event)
{
System.out.println("ok2");

//try {
// Must be implemented from the
// DiscoveryListener interface.

// This method will be called with a list
// of all lookup services found
// (actually their registrar proxies).
ServiceRegistrar[] regArray = event.getRegistrars();

try
{
proxy = new MindstormProxy();
}
catch (RemoteException e)
{
```

```
System.err.println(e.toString());
System.exit(1);
}

proxy.setRcxId(rcxId);

// Turn off security management while
// opening the comm port. This requires
// holding a reference to the existing
// security manager so it can be
// restored again.
SecurityManager sm = System.getSecurityManager();
System.setSecurityManager(null);

// Request that the serial port
// be opened.
proxy.openRcxPort(portId);

// Turn security management back on again
System.setSecurityManager(sm);

// Iterate through the array of
// lookup services that were found
// on the network.
for (int ix = 0; ix < regArray.length; ix++)
{
ServiceRegistrar svcRegistrar = regArray[ix];

// register ourselves as a service

ServiceItem serviceItem = new ServiceItem(null, proxy, null);

// Request a 10 second lease duration. This
// means that the lease will require renewing
// at least once every 10 seconds.
ServiceRegistration serviceRegistration = null;

try
{
serviceRegistration =
svcRegistrar.register(serviceItem, Lease.FOREVER);
}
catch (RemoteException e)
{
```

```
// If the service registration
// fails, we can still try with
// any other lookup services
// on the network.
System.err.println(e.toString());
continue;
}

// Request the Lease Renewal Manager
// to perform regular renewals of the
// lease indefinitely.
leaseManager.renewUntil(
serviceRegistration.getLease(),
Lease.FOREVER,
Lease.ANY,
this);
System.out.println(
"Successful - Service ID: " + svcRegistrar.getServiceID());

}
}

public void notify(LeaseRenewalEvent evt)
{
// Will receive events concerning abnormal
// lease behaviour. Ignored in this
// example.
}
}
```

B.2. mindstormclient

B.2.1. Client$EventListener_Stub

```
// Stub class generated by rmic, do not edit.
// Contents subject to change without notice.

package mindstormClient;

public final class Client$EventListener_Stub
extends java.rmi.server.RemoteStub
implements net.jini.core.event.RemoteEventListener, java.rmi.Remote
{
private static final long serialVersionUID = 2;

private static java.lang.reflect.Method $method_notify_0;

static {
try
{
$method_notify_0 =
net.jini.core.event.RemoteEventListener.class.getMethod(
"notify",
new java.lang.Class[] {
 net.jini.core.event.RemoteEvent.class });
}
catch (java.lang.NoSuchMethodException e)
{
throw new java.lang.NoSuchMethodError(
"stub class initialization failed");
}
}

// constructors
public Client$EventListener_Stub(java.rmi.server.RemoteRef ref)
{
super(ref);
}

// methods from remote interfaces

// implementation of notify(RemoteEvent)
public void notify(net.jini.core.event.RemoteEvent $param_RemoteEvent_1)
throws java.rmi.RemoteException, net.jini.core.event.UnknownEventException
```

```
{
try
{
ref.invoke(
this,
$method_notify_0,
new java.lang.Object[] { $param_RemoteEvent_1 },
1042791172444620860L);
}
catch (java.lang.RuntimeException e)
{
throw e;
}
catch (java.rmi.RemoteException e)
{
throw e;
}
catch (net.jini.core.event.UnknownEventException e)
{
throw e;
}
catch (java.lang.Exception e)
{
throw new java.rmi.UnexpectedException(
"undeclared checked exception",
e);
}
}
}
```

B.2.2. Client

```
/**
 * Client.java
 *
 */

package mindstormClient;

import shared.Mindstorm;
import net.jini.discovery.LookupDiscovery;
import net.jini.discovery.DiscoveryListener;
import net.jini.discovery.DiscoveryEvent;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.core.lookup.ServiceMatches;
import net.jini.core.event.RemoteEventListener;
import net.jini.core.event.RemoteEvent;
import net.jini.core.event.UnknownEventException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RMISecurityManager;
import java.util.*;
import java.io.Serializable;
import java.io.IOException;

public class Client implements DiscoveryListener
{

// This client will control multiple Mindstorms. Use
// the Vector object as a flexible container for
// holding a variable number of objects.
protected Vector mindstorms = new Vector();

static public void main(String args[])
{

// Since reggie is running with a security policy,
// we will have to as well. This assumes that
// the policy file is located at
// c:\jini1_2\policy\policy.all
// Adjust this for specific installations.
System.setProperty(
"java.security.policy",
```

```
"C:\\Programme\\jini1_2_1_001\\policy\\policy.all");
System.setSecurityManager(new SecurityManager());

new Client();

// Ensure client runs indefinitely.
Object keepAlive = new Object();
synchronized (keepAlive)
{
try
{
keepAlive.wait();
}
catch (java.lang.InterruptedException e)
{
// Do nothing
}
}
}

public Client()
{

// Lookup using Multicast (i.e. look for any lookup
// services on the network).
LookupDiscovery lookupDiscovery = null;

try
{
lookupDiscovery = new LookupDiscovery(LookupDiscovery.ALL_GROUPS);
}
catch (IOException e)
{
System.err.println(e.toString());
System.exit(1);
}
lookupDiscovery.addDiscoveryListener(this);

}

public void discarded(DiscoveryEvent event)
{
// Must be implemented from the
// DiscoveryListener interface.
```

```
}

public void discovered(DiscoveryEvent event)
{

// Must be implemented from the
// DiscoveryListener interface.

// This method will be called with a list
// of all lookup services found
// (actually their registrar proxies).
ServiceRegistrar[] regArray = event.getRegistrars();
Class[] classes = new Class[] { Mindstorm.class };
ServiceTemplate svcTemplate = new ServiceTemplate(null, classes, null);

for (int ix = 0; ix < regArray.length; ix++)
{
ServiceRegistrar svcRegistrar = regArray[ix];

// For each lookup service there could be
// 0 or many registered matching objects.
// Request a maximum of 10.
ServiceMatches matches;
try
{
matches = svcRegistrar.lookup(svcTemplate, 10);
}
catch (RemoteException e)
{
System.err.println("Remote exception in lookup");
continue;
}
for (int iy = 0; iy < matches.totalMatches; iy++)
{
Mindstorm matched = (Mindstorm) (matches.items[iy].service);

if (null != matched)
{
// Call ProcessMatch() to either
// add it to the collection
// or ignore it.
try
{
ProcessMatched(matched);
```

```
}
catch (RemoteException e)
{
// This will likely be old
// services that are no longer
// available.
continue;
}
}
}
}

System.out.println("Finished list");
if (mindstorms.isEmpty())
{
// Client cannot operate if no Mindstorm
// services have been found.
System.err.println("Could not find any Mindstorms");
System.exit(1);
}
else
{
// All OK; we have at least one Mindstorm
// service.
// Start it off by getting the first
// Mindstorm to perform the first dance step.
try
{
((Mindstorm) mindstorms.firstElement()).imitateThis(0);
}
catch (RemoteException e)
{
// If we can't invoke the first
// Mindstorm then no point
// continuing.
System.err.println(e.toString());
System.exit(1);
}
}
}

protected void ProcessMatched(Mindstorm matched) throws RemoteException
{
```

```
// matched represents a Mindstorm object
// that may or may not be a duplicate instance
// of one we already have.

String foundId = matched.getRcxId();

// Only add the Mindstorm to the
// collection if we don't
// already have it.
if (IsUnique(foundId))
{

mindstorms.add(matched);
System.out.println("Found ID: " + foundId);

// Create a new listener
// object to receive events
// from this Mindstorm.
matched.RegisterRemoteListener(new EventListener(matched, foundId));
}
else
{
System.out.println("Ignoring ID: " + foundId);
}

}

protected boolean IsUnique(String id)
{

// Iterate through the list of Mindstorms and
// determine whether we already have this one.

for (Iterator iter = mindstorms.iterator(); iter.hasNext();)
{
String existingId = null;
try
{
existingId = ((Mindstorm) iter.next()).getRcxId();
if (id.equals(existingId))
{
return false;
}
}
```

```
catch (RemoteException e)
{
System.err.println("Caught: " + e.toString());
}
}
return true;
}

protected void NotifyNextMindstorm(String id, int step)
{

// One of the Mindstorms has notified us of a
// step performed; notify the next one.

// Firstly loop through the Vector of
// Mindstorms until we find the match
for (Iterator iter = mindstorms.iterator(); iter.hasNext();)
{
String currId = null;
try
{
currId = ((Mindstorm) iter.next()).getRcxId();
if (id.equals(currId))
{
// We have the match. Either get the
// next from the list or the first
// in the list.
// If there's only one, it will just
// notify itself...
if (iter.hasNext())
{
((Mindstorm) iter.next()).imitateThis(step);
}
else
{
((Mindstorm) mindstorms.firstElement()).imitateThis(
step);
}
}
}
catch (RemoteException e)
{
System.err.println("Caught: " + e.toString());
}
```

```
}
}

class EventListener
extends UnicastRemoteObject
implements RemoteEventListener, Serializable
{

// This class must extend UnicastRemoteObject
// because it needs to receive calls back
// from a remote JVM. Remember to generate a
// stub class for it using the RMI compiler!

protected String rcxId = null;
protected Mindstorm remoteMindstorm = null;

public EventListener(Mindstorm mindstorm, String rcxId)
throws RemoteException
{

// Call the default constructor for
// the parent class.
super();

// Keep a reference to the remote object that
// we're listening to.
this.remoteMindstorm = mindstorm;

// Keep a copy of the remote ID so we don't have
// to make round trips to look it up.
this.rcxId = rcxId;
}

public void notify(RemoteEvent evt)
throws UnknownEventException, RemoteException
{

// Will receive notification messages from
// the service that it is associated with.
switch ((int) evt.getID())
{

case Mindstorm.EVENT_RCX_MSG :
// We have received some form
```

```
// of message from the remote RCX.
byte[] msg;
msg = remoteMindstorm.GetLastRCXMessage();

// We could process the message here
// in some way.
break;

case Mindstorm.EVENT_DANCE_STEP :
// This is a notification from a remote
// Mindstorm that it has just completed
// a dance step. We must query the
// service to find out what the
// step was.
int step;
step = remoteMindstorm.GetLastStepPerformed();
System.out.println(
"Received step: " + step + " from: " + rcxId);

// Pass the dance step along
// the list of robots.
NotifyNextMindstorm(rcxId, step);
break;
}
}
}
}
```

B.3. shared

B.3.1. Mindstorm

```
/**
 * Mindstorm.java
 *
 */

package shared;

import java.rmi.Remote;
import java.rmi.RemoteException;
import net.jini.core.event.RemoteEventListener;

public interface Mindstorm
    extends Remote {

    // Define the two possible message types:
    // EVENT_RCX_MSG is a message from a remote RCX.
    // EVENT_DANCE_STEP is a notification that a Mindstorm
    // has completed a dance step.
    public final int EVENT_RCX_MSG = 1;
    public final int EVENT_DANCE_STEP = 2;

    // Method to retrieve the ID of a Mindstorm.
    public String getRcxId() throws RemoteException;

    // Tell a Mindstorm to imitate a dance step.
    public void imitateThis(int danceStep) throws RemoteException;

    // Retrieve the most recent RCX Message received. Should
    // be called in response to receiving an EVENT_RCX_MSG.
    public byte[] GetLastRCXMessage() throws RemoteException;

    // Retrieve the most recent dance step performed
    // by an RCX. Should be called in response to
    // receiving an EVENT_DANCE_STEP.
    public int GetLastStepPerformed() throws RemoteException;

    // Allow the client to register a listener
    // to receive events from the service.
    public void RegisterRemoteListener(RemoteEventListener listener)
        throws RemoteException;
```

```
}
```

C. Source Code des Swarm Intelligence Projektes

C.1. PC

C.1.1. Package gui

MainWindow

```
package gui;

import javax.swing.JFrame;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import logic.*;


/**
 * Das Hauptfenster der Ant-Anwendung. <br>
 * Die enthaltenen Elemente sind: <br>
 * <code>{@link gui.EdgeInfo    EdgeInfo}   </code>
 * - Zeigt Informationen &uuml;ber die Karte. <br>
 * <code>{@link gui.AntMenu     AntMenu}    </code>
 * - Die Menuzeile zum steuern des Programms. <br>
 * <code>{@link gui.StatusLabel StatusLabel}</code>
 * - Zeigt Informationen &uuml;ber den Programmstatus. <br>
 * <code>{@link gui.RoutePanel  RoutePanel} </code>
 * - Zeigt eine graphische Darstellung der Karte. <br>
 *
 * @author Urs Heimann
 */
public class MainWindow extends JFrame
{
  /** Statische Referenz auf das Hauptfenster. */
  private static MainWindow main = null;

  /**
```

```
 * liefert das Hauptfenster, falls vorhanden.
 * @return das Hauptfenster.
 */
public static MainWindow getMainWindow()
{
  return( main );
}


/** Referenz auf einen {@link logic.MapReader MapReader}. */
private MapReader mapReader;
/**Referenz auf das {@link gui.RoutePanel RoutePanel} das die Karte anzeigt.*/
private RoutePanel routePanel;


/**
 * Erzeugt das Hauptfenster und initialisiert die einzelnen Komponenten.
 *
 * @param mr Der Kartenleser.
 */
public MainWindow(MapReader mr)
{
  super("Ant");
  main = this;
  mapReader = mr;
  this.setName("Hauptfenster");

  this.addWindowListener(
    new WindowAdapter()
    {
       public void windowClosing(WindowEvent e)
         {
           System.exit(0);
         }
    }
   );
   this.setSize(1200,500);
   this.getContentPane().setLayout(new BorderLayout());


   EdgeInfo eInfo = new EdgeInfo();
   this.getContentPane().add(eInfo, BorderLayout.WEST);

   routePanel = new RoutePanel(eInfo);
```

```
    this.getContentPane().add(routePanel, BorderLayout.CENTER);


    this.setJMenuBar(new AntMenu(this));
    this.getContentPane().add(StatusLabel.getStatusLabel(),
                              BorderLayout.SOUTH);

    this.show();

}


/**
 * Liefert den Kartenleser.
 *
 * @return Der Kartenleser.
 */
public MapReader getMapReader()
{
  return( mapReader );
}


/**
 * Liefert das Anzeigefenster der Karte.
 *
 * @return Das Anzeigefenster.
 */
public RoutePanel getRoutePanel()
{
  return( routePanel );
}


/**
 * Destruktor, l&ouml;scht die statische Referenz bevor das Hauptfenster
 * zerst&ouml;rt wird.
 */
public void finalize() throws Throwable
{
  main = null;
 super.finalize();
}
```

```
  /**
   * Die Main-Prozedur des Ant-Programms.
   * Kommandozeilen-Parameter werden nicht beachtet.
   *
   * @param args Wird nicht beachtet.
   */
  public static void main(String[] args)
  {
   MapReader mr = new MapReader();
   MainWindow mw = new MainWindow(mr);
   mr.start();
  }

}  // class MainWindow
```

AntMenu

```
package gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import logic.*;


/**
 * Das Menu der Ant Anwendung.
 *
 * @author Urs Heimann
 */
public class AntMenu extends JMenuBar
{

  /** Referenz auf das Hauptfenster. */
  private MainWindow main;

  /**
   * Erstellt ein neues Menu. <br>
   * <q>Datei</q> - Untermenu zum Laden, Speicher, Schliessen von Dateien. <br>
   * <q>Map</q>   - Untermenu zum Laden der Karten.
   *
   * @param main Eine Referenz auf das {@link gui.MainWindow Hauptfenster}.
   */
  public AntMenu(MainWindow main)
  {
   this.main = main;
   createMenuDatei();
   createMenuMap();
  }


  /**
   * Erstellt das Untermenu mit dem Namen <q>Datei</q>. <br>
   * <q>Schema &ouml;ffnen</q> - l&auml;dt ein zuvor gespeichertes Schema.
   *                             <q>Ctrl + o</q><br>
   * <q>Schema speichern</q>   - Speichert das aktuelle Schema.
   *                             <q>Ctrl + s</q><br>
   * <q>Exit</q>               - beendet das Programm. <q>Ctrl + x</q><br>
```

```
 */
private void createMenuDatei()
{
  JMenu datei = new JMenu("Datei");
  datei.setMnemonic('D');

  // Load Schema
  JMenuItem loadSchema = new JMenuItem("Schema öffnen", 'ö');
  loadSchema.addActionListener(
    new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        main.getRoutePanel().setSchema(SchemaLoader.loadSchema());
      }
    }
  );
  loadSchema.setAccelerator(KeyStroke.getKeyStroke('O', Event.CTRL_MASK));

  // Save Schema
  JMenuItem saveSchema = new JMenuItem("Schema speichern", 's');
  saveSchema.addActionListener(
    new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
        {
         SchemaLoader.saveSchema(main.getRoutePanel().getSchema());
        }
     }
  );
  saveSchema.setAccelerator(KeyStroke.getKeyStroke('S', Event.CTRL_MASK));

  // Exit
  JMenuItem exit = new JMenuItem("Exit", 'x');
  exit.addActionListener(
        new ActionListener()
        {
         public void actionPerformed(ActionEvent e){System.exit(0);}
        }
  );
  exit.setAccelerator(KeyStroke.getKeyStroke('X', Event.CTRL_MASK));

  datei.add(loadSchema);
  datei.add(saveSchema);
```

```
    datei.addSeparator();
    datei.add(exit);

    this.add(datei);
  }


  /**
   * Erstellt das Untermenu mit dem Namen <q>Map</q>. <br>
   * <q>Reload Map</q> - Liest die Karte vom RCX ein. <q>Ctrl + r</q>
   */
  private void createMenuMap()
  {
    JMenu map = new JMenu("Karte");
    map.setMnemonic('K');

    // ReloadOnce
    JMenuItem reload = new JMenuItem("Reload Once", 'R');
    reload.addActionListener(new ReloadOnceButtonListener(main));
    reload.setAccelerator(KeyStroke.getKeyStroke('R', Event.CTRL_MASK));

    // ReloadMap
    JMenuItem autoreload = new JMenuItem("Start Auto Reload", 'A');
    autoreload.addActionListener(new ReloadButtonListener(main));
    autoreload.setAccelerator(KeyStroke.getKeyStroke('A', Event.CTRL_MASK));


    map.add(reload);
    map.add(autoreload);
    this.add(map);
  }
}  // class AntMenu



/**
 * L&ouml;st die Aktualisierung der Karte aus.
 *
 * @author Urs Heimann
 */
class ReloadButtonListener implements ActionListener
{

  /** Referenz auf das Hauptfenster. */
```

```
  private MainWindow main;

  private boolean reloadRunning;


  /** Erstellt einen neuen Listener.
   *
   * @param main Referenz auf das Hauptfenster.
   */
  public ReloadButtonListener(MainWindow main)
  {
   this.main = main;
   reloadRunning = false;
  }


  /**
   * L&ouml;st die aktualisierung der Karte aus, wenn auf den entsprechenden
   * Button gedr&uuml;kt wurde.
   *
   * @param e Der Ausl&ouml;sende Event.
   */
  public void actionPerformed(ActionEvent e)
  {
   if( reloadRunning )
   {
     reloadRunning = false;
     main.getMapReader().pause();
     ((JMenuItem)e.getSource()).setText("Start Auto Reload");
   }
   else
   {
     reloadRunning = true;
     main.getMapReader().endPause();
     ((JMenuItem)e.getSource()).setText("Stop Auto Reload");
   }
  }

}  // class ReloadButtonListener



/**
 * L&ouml;st die Aktualisierung der Karte aus.
```

```
 *
 * @author Urs Heimann
 */
class ReloadOnceButtonListener implements ActionListener
{

  /** Referenz auf das Hauptfenster. */
  private MainWindow main;


  /** Erstellt einen neuen Listener.
   *
   * @param main Referenz auf das Hauptfenster.
   */
  public ReloadOnceButtonListener(MainWindow main)
  {
   this.main = main;
  }


  /**
   * L&ouml;st die aktualisierung der Karte aus, wenn auf den entsprechenden
   * Button gedr&uuml;kt wurde.
   *
   * @param e Der Ausl&ouml;sende Event.
   */
  public void actionPerformed(ActionEvent e)
  {
   ((JMenuItem)e.getSource()).setEnabled(false);
   main.getMapReader().reloadOnce();
//    main.getMapReader().loadMap();
//    main.getRoutePanel().showMap(main.getMapReader());
   ((JMenuItem)e.getSource()).setEnabled(true);
  }

}  // class ReloadOnceButtonListener
```

MapSchema

```
package gui;

import java.io.Serializable;
import java.util.ArrayList;
import java.awt.geom.*;

/**
 * Speichert die Anordnung der Strecken und Knoten. <br>
 * Dieses Schema kann Serialisiert und auf der Festplatte gespeichert
 * werden. So kann die gleiche Anordnung zu einem sp&auml;teren Zeitpunkt
 * wieder hergestellt werden.
 *
 * @author Urs Heimann
 */
public class MapSchema implements Serializable
{

  /** X koordinaten der Knoten. */
  private ArrayList nodeX;
  /** Y koordinaten der Knoten. */
  private ArrayList nodeY;
  /** X koordinaten der Strecken. */
  private ArrayList edgeX;
  /** Y koordinaten der Strecken. */
  private ArrayList edgeY;
  /** Breite des Fensters. */
  private int windowWidth = 0;
  /** H&ouml;he des Fensters. */
  private int windowHeight = 0;


  /**
   * Erstellt ein neues, leeres Schema.
   */
  public MapSchema()
  {
    nodeX = new ArrayList();
    nodeY = new ArrayList();
    edgeX = new ArrayList();
    edgeY = new ArrayList();
  }
```

```
/**
 * Speichert die Koordinaten eines Knotens.
 *
 * @param id    Nummer des Knotens.
 * @param node  Der Knoten.
 */
public void setNode(int id, Node node)
{
 if(node == null)
 {
   nodeX.add(id, null);
   nodeY.add(id, null);
 }
 else
 {
   nodeX.add(id, new Integer((int)node.getNodePoint().getX()));
   nodeY.add(id, new Integer((int)node.getNodePoint().getY()));
 }
}


/**
 * Speichert die Koordinaten einer Strecke.
 *
 * @param id    Nummer der Strecke.
 * @param edge  Die Strecke.
 */
public void setEdge(int id, VisualEdge edge)
{
 if(edge == null)
 {
   edgeX.add(id, null);
   edgeY.add(id, null);
 }
 else
 {
   edgeX.add(id, new Integer(edge.getPosX()));
   edgeY.add(id, new Integer(edge.getPosY()));
 }
}


/**
```

```
 * Liefert die X Koordinate eines Knotens.
 *
 * @param id   Nummer des Knotens.
 * @return     Die X Koordinate wenn <code>id</code> g&uuml;ltig, sonst 0.
 */
public int getNodeX(int id)
{
 if( id >= nodeX.size() || id < 0) { return( 0 ); }
 if( nodeX.get(id) != null )
 {
    return( ((Integer)nodeX.get(id)).intValue() );
 }
 else
 {
   return( 0 );
 }
}


/**
 * Liefert die Y Koordinate eines Knotens.
 *
 * @param id   Nummer des Knotens.
 * @return     Die Y Koordinate wenn <code>id</code> g&uuml;ltig, sonst 0.
 */
public int getNodeY(int id)
{
 if( id >= nodeY.size() || id < 0) { return( 0 ); }
 if( nodeY.get(id) != null )
 {
    return( ((Integer)nodeY.get(id)).intValue() );
 }
 else
 {
   return( 0 );
 }
}


/**
 * Liefert die X Koordinate einer Strecke.
 *
 * @param id   Nummer der Strecke.
 * @return     Die X Koordinate wenn <code>id</code> g&uuml;ltig, sonst 0.
```

```
 */
public int getEdgeX(int id)
{
 if( id >= edgeX.size() || id < 0) { return( 0 ); }
 if( edgeX.get(id) != null )
 {
    return( ((Integer)edgeX.get(id)).intValue() );
 }
 else
 {
   return( 0 );
 }
}


/**
 * Liefert die Y Koordinate einer Strecke.
 *
 * @param id   Nummer der Strecke.
 * @return     Die Y Koordinate wenn <code>id</code> g&uuml;ltig, sonst 0.
 */
public int getEdgeY(int id)
{
 if( id >= edgeY.size() || id < 0) { return( 0 ); }
 if( edgeY.get(id) != null )
 {
    return( ((Integer)edgeY.get(id)).intValue() );
 }
 else
 {
   return( 0 );
 }
}



/**
 * Speichert die Fenstergr&ouml;sse.
 *
 * @param width   Fensterbreite.
 * @param height  Fensterh&ouml;he
 */
public void setWindowSize(int width, int height)
{
```

```
    windowWidth = width;
    windowHeight = height;
  }


  /**
   * Liefert die Fensterbreite.
   * @return  Fensterbreite
   */
  public int getWindowWidth()
  {
    return( windowWidth );
  }


  /**
   * Liefert die Fensterh&ouml;he.
   * @return  Fensterh&ouml;he
   */
  public int getWindowHeight()
  {
   return( windowHeight );
  }

}
```

EdgeInfo

```
package gui;

import javax.swing.JTextArea;
import java.awt.*;

/**
 * Zeigt die Informationen zu einer bestimmten Strecke der Karte an.
 *
 * @author Urs Heimann
 */
public class EdgeInfo extends JTextArea
{

/** Die Breite des Anzeigefensters */
  private static final int COLUMNS  = 9;
/** Die H&ouml;he des Anzeigefensters */
  private static final int ROWS     = 5;
/** Die Schriftgr&ouml;sse. */
  private static final int FONTSIZE = 16;


/**
 * Erstellt ein neues Infofenster.
 * F&uuml;r gr&ouml;sse und Schriftgr&ouml;sse werden Standardwerte verwendet.
 */
  public EdgeInfo()
  {
   super(ROWS, COLUMNS);
   this.setFont(new Font(null, Font.BOLD, FONTSIZE));
   this.setEditable(false);
  }

/**
 * Liest die Informationen aus Einer VisualEdge und zeigt sie im Fenster an.
 * Momentan werden lediglic die ID der Strecke und der Pheromonwert ausgelesen.
 *
 * @param ve  Die VisualEdge dessen Informationen angezeigt werden sollen.
 */
  public void showInfo(VisualEdge ve)
  {
   String info = "";
   info = info + "ID: " + ve.getID() + "\n";
```

```
   info = info + "Pheromon: " + ve.getPheromon() + "\n";

   this.setText(info);
   this.repaint();
  }


/**
 * Zeichnet den Inhalt des Feldes.
 * Verbessert die Darstellung durch gl&auml;tten der Schrift.
 *
 * @param g   Grafikobjekt
 */
  public void paintComponent(Graphics g)
  {
   ((Graphics2D)g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                    RenderingHints.VALUE_ANTIALIAS_ON);
   super.paintComponent(g);
  }
}
```

Node

```
package gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;



/**
 * Ein Knoten verbindet mehrere Strecken der Karte.
 *
 *
 * @author Urs Heimann
 */
public class Node extends JLabel {

  /** Breite (Durchmesser) der normalen Knoten. */
  private static final int WIDTH = 8;
  /** Breite (Durchmesser) der Start- und Endknoten. */
  private static final int STARTWIDTH = 40;
  /** Schriftgr&ouml;sse der Start- und Endknoten. */
  private static final int FONTSIZE = 12;
  /** Standardfarbe der Knoten. */
  private static final Color COLORNORMAL = Color.red;
  /** Farbe der Knoten wenn die Maus dar&uuml;ber bewegt wird. */
  private static final Color COLORHIGH = Color.blue;
  /** Farbe der Knoten wenn sie selektiert sind (Drag &amp; Drop). */
  private static final Color COLORSELECT = Color.cyan;

  /** Wenn <code>true</code> wird der Knoten als Startknoten dargestellt. */
  private boolean start = false;
  /** Wenn <code>true</code> wird der Knoten als Endknoten dargestellt. */
  private boolean end = false;
  /** Die Position des Knotens. */
  private Point2D.Double point;

  /** Wenn <code>true</code> wird der Knoten als Highlited dargestellt. */
  private boolean highlited = false;
  /** Wenn <code>true</code> wird der Knoten als Selected dargestellt. */
  private boolean selected = false;
```

```
/**
 * Erstellt einen neuen Knoten.
 */
public Node()
{
  point = new Point2D.Double(0, 0);
  this.setSize(WIDTH, WIDTH);
  this.setLocation(0 - WIDTH/2, 0 - WIDTH/2);
  NodeMouseListener nml = new NodeMouseListener(this);
  this.addMouseListener(nml);
  this.addMouseMotionListener(new NodeMouseMotionListener(this, nml));
}


/**
 * Erstellt einen neuen Knoten an der gegebenen Position.
 *
 * @param x   X Koordinate des neuen Knotens.
 * @param y   Y Koordinate des neuen Knotens.
 */
public Node(int x, int y)
{
  point = new Point2D.Double(x, y);
  this.setSize(WIDTH, WIDTH);
  this.setLocation(x - WIDTH/2, y - WIDTH/2);
  NodeMouseListener nml = new NodeMouseListener(this);
  this.addMouseListener(nml);
  this.addMouseMotionListener(new NodeMouseMotionListener(this, nml));
}


/**
 * Setzt den Knoten auf eine neue Position.
 *
 * @param x   X Koordinate der neuen Position.
 * @param y   Y Koordinate der neuen Position.
 */
public void setPos(int x, int y)
{
  point.setLocation(x, y);
  if( start )
  {
    setLocation(x - STARTWIDTH/2, y - STARTWIDTH/2);
```

```
    }
    else if( end )
    {
      setLocation(x - STARTWIDTH/2, y - STARTWIDTH/2);
    }
    else
    {
      setLocation(x - WIDTH/2, y - WIDTH/2);
    }
  }


  /**
   * Verschiebt den Knoten um den gew&uuml;nschten Offset.
   *
   * @param xOffset  negativ: Verschiebung nach links,
   *                  positiv: Verschiebung nach rechts
   * @param yOffset  negativ: Verschiebung nach oben,
   *                  positiv: Verschiebung nach unten
   */
  public void movePos(int xOffset, int yOffset)
  {
    point.setLocation(point.getX() + xOffset, point.getY() + yOffset);
    if( start )
    {
      setLocation((int)point.getX() - STARTWIDTH/2,
                  (int)point.getY() - STARTWIDTH/2);
    }
    else if( end )
    {
      setLocation((int)point.getX() - STARTWIDTH/2,
                  (int)point.getY() - STARTWIDTH/2);
    }
    else
    {
      setLocation((int)point.getX() - WIDTH/2,
                  (int)point.getY() - WIDTH/2);
    }
  }



  /**
   * Liefert die Koordinaten des Knotens.
```

```
 *
 * @return  Die Koordinaten des Knoten-Mittlepunktes.
 */
public Point2D.Double getNodePoint()
{
  return( point );
}


/**
 * Zeichnet den Knoten auf dem Bildschirm.
 * Diese Funktion wird automatisch aufgerufen wenn das Fenster neu gezeichnet
 * werden muss, oder die <code>repaint()</code> Methode aufgerufen wurde.
 *
 * @param g   Graphics Objekt des Systems.
 */
public void paintComponent(Graphics g)
{
 // Grafik initialisieren
  Graphics2D g2 = (Graphics2D)g;
  g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                      RenderingHints.VALUE_ANTIALIAS_ON);
  g2.setStroke(new BasicStroke(1));
  Color lineColor = Color.black;

  // Farbe auswählen
  Color fillColor = COLORNORMAL;
  if( highlited )
  {  fillColor = COLORHIGH; }
  if( selected )
  {  fillColor = COLORSELECT; }

  if( start )  // Startknoten zeichnen
  {
    g2.setColor(fillColor);
    g2.fillOval(1, 1, STARTWIDTH-2, STARTWIDTH-2);
    g2.setColor(lineColor);
    g2.drawOval(1, 1, STARTWIDTH-2, STARTWIDTH-2);
    Font font = new Font("Default", Font.BOLD, FONTSIZE);
    g2.setFont(font);
    g2.drawString("Start", 4, STARTWIDTH /2 + FONTSIZE /4);
  }
  else if ( end )  // Endknoten zeichnen
  {
```

```
    g2.setColor(fillColor);
    g2.fillOval(1, 1, STARTWIDTH-2, STARTWIDTH-2);
    g2.setColor(lineColor);
    g2.drawOval(1, 1, STARTWIDTH-2, STARTWIDTH-2);
    Font font = new Font("Default", Font.BOLD, FONTSIZE);
    g2.setFont(font);
    g2.drawString("End", 8 , STARTWIDTH /2 + FONTSIZE /4);
  }
  else   // normalen Knoten zeichnen
  {
    g2.setColor(fillColor);
    g2.fillOval(1, 1, WIDTH-2, WIDTH-2);
  }

}


/**
 * Kennzeichnet den Knoten als Startknoten.
 */
public void setStart()
{
  start = true;
  end = false;
  setSize(STARTWIDTH, STARTWIDTH);
  setLocation((int)point.getX() - STARTWIDTH/2,
              (int)point.getY() - STARTWIDTH/2);
}


/**
 * Kennzeichnet den Knoten als Endknoten.
 */
public void setEnd()
{
  end = true;
  start = false;
  setSize(STARTWIDTH, STARTWIDTH);
  setLocation((int)point.getX() - STARTWIDTH/2,
              (int)point.getY() - STARTWIDTH/2);
}


/**
```

```
   * Kenzeichnet den Knoten als Selektiert.
   * @param s  <code>true </code> - Knoten ist selektiert. <br>
   *            <code>false</code> - Knoten ist nicht selektiert. <br>
   */
  public void setSelected(boolean s)
  {
    selected = s;
  }


  /**
   * Kenzeichnet den Knoten als Highlited.
   *
   * @param h  <code>true </code> - Knoten ist gehighlited. <br>
   *            <code>false</code> - Knoten ist nicht gehighlited. <br>
   */
  public void setHighlited(boolean h)
  {
   highlited = h;
  }

}




/**
 * Wartet auf Mauseingaben auf den Knoten.
 *
 * @author Urs Heimann
 */
class NodeMouseListener implements MouseListener
{
  private Node node;
  private int startDragX;
  private int startDragY;

  public NodeMouseListener(Node n)
  {
    this.node = n;
  }


  public void mouseClicked(MouseEvent e)
```

```
{
}


/**
 * Wird die Maus &uuml;ber den Knoten bewegt wird er markiert.
 *
 * @param e   Ausl&ouml;sender Event.
 */
public void mouseEntered(MouseEvent e)
{
   node.setHighlited(true);
   node.repaint();
}


/**
 * Wird die Maus aus dem Knoten bewegt wird er wieder normal dargestellt.
 *
 * @param e   Ausl&ouml;sender Event.
 */
public void mouseExited(MouseEvent e)
{
   node.setHighlited(false);
   node.repaint();
}


/**
 * Wird eine Maustaste &uuml;ber dem Knoten gedr&uuml;ckt, wird er selektiert.
 *
 * @param e   Ausl&ouml;sender Event.
 */
public void mousePressed(MouseEvent e)
{
  node.setSelected(true);
  startDragX = e.getX();
  startDragY = e.getY();
}


/**
 * Wird die Maustaste wieder losgelassen, wird der Knoten wieder normal
 * dargestellt.
```

```
   *
   * @param e   Ausl&ouml;sender Event.
   */
  public void mouseReleased(MouseEvent e)
  {
    node.setSelected(false);
   node.movePos(e.getX() - startDragX, e.getY() - startDragY);
    node.repaint();
  }


  /**
   * Liefert den Startpunkt des Drags.
   *
   * @return  Die x Koordinate des Startpunktes.
   */
  public int getStartX()
  {
   return( startDragX );
  }



  /**
   * Liefert den Startpunkt des Drags.
   *
   * @return  Die y Koordinate des Startpunktes.
   */
  public int getStartY()
  {
   return( startDragY );
  }
}



/**
 * Wartet auf Mausbewegungen &uuml;ber dem Knoten.
 *
 * @author Urs Heimann
 */
class NodeMouseMotionListener implements MouseMotionListener
{
  private Node node;
```

```
  private NodeMouseListener mListener;


  public NodeMouseMotionListener(Node n, NodeMouseListener nml)
  {
    node = n;
    mListener = nml;
  }


  /**
   * Wird bei gedr&uuml;ckter Maustaste die Maus bewegt, wird der Knoten
   * verschoben (Drag &amp; Drop).
   *
   * @param e  Ausl&ouml;sender Event.
   */
  public void mouseDragged(MouseEvent e)
  {
   node.movePos(e.getX() - mListener.getStartX(),
                e.getY() - mListener.getStartY());
    node.repaint();
  }

  public void mouseMoved(MouseEvent e)
  {
  }
}
```

RoutePanel

```
package gui;

import javax.swing.JPanel;
import java.util.*;
import map.Edge;
import logic.MapReader;



/**
 * Graphische anzeige der Karte.
 *
 * @author Urs Heimann
 */
public class RoutePanel extends JPanel {

  /** Fenster zum anzeigen der Streckeninformationen. */
  private EdgeInfo edgeInfo;
  /** Liste der anzuzeigenden Strecken. */
  private ArrayList vEdges;
  /** Liste der anzuzeigenden Knoten. */
  private ArrayList nodes;
  /** Zur Darstellung verwendetes Schema. */
  private MapSchema schema;


  /**
   * Erstellt eine neue Anzeige.
   *
   * @param eInfo  Fenster zum anzeigen der Streckeninformationen.
   */
  public RoutePanel(EdgeInfo eInfo)
  {
    edgeInfo = eInfo;
    this.setSize(700, 500);
    vEdges = new ArrayList();
    nodes = new ArrayList();
   this.setName("RoutePanel");
  }


  /**
```

```
 * L&ouml;scht die zurzeit angezeigte Karte.
 */
public void clearMap()
{
 this.removeAll();
 this.vEdges.clear();
 this.nodes.clear();
 this.repaint();
}


/**
 * Erstellt die Knoten und Strecken aus einer neu geladenen Karte.
 * Die alte Karte wird verworfen, das Darstellungsschema bleibt jedoch
 * erhalten.
 *
 * @param reader  Der Kartenleser der die Karte aus dem RCX gelesen hat.
 */
public synchronized void showMap(MapReader reader)
{
 this.clearMap();


  int size = reader.getMapSize();

  // Knoten erstellen
  for( int i = 0; i < size; i++ )
  {

    nodes.add(i, new Node(i*120 +40, 200));
  }

  ((Node)nodes.get(0)).setStart();
  Node ne = new Node(900, 200);
  ne.setEnd();
  nodes.add(size, ne);

  Edge e;
  VisualEdge ve;

  // Strecken erstellen
  for( int i = 0; i < size; i++ )
  {
    e  = reader.getMap().getEdge((byte)i);
```

```
    ve = new VisualEdge(e, edgeInfo);
    ve.setStartPoint(((Node)nodes.get(i)).getNodePoint());

    vEdges.add(i, ve);
  }

  // Strecken mit Knoten verbinden.
  int numEdges;
  Node tempNode;
  for( int i = 0; i < size; i++ )
  {
    ve = (VisualEdge)vEdges.get(i);
    e = ve.getEdge();
    numEdges = e.getNumNextEdges();

    if( numEdges >= 1 )
    {
     tempNode = (Node)nodes.get(e.getEdge((byte)0).getIdNumber()-1);
      ve.setEndPoint(tempNode.getNodePoint());
     ve.setPos(i * 50, i*20 +20);
      for(int j = 1; j < numEdges; j++)
      {
        ve = (VisualEdge)vEdges.get(e.getEdge((byte)j).getIdNumber()-1);
        ve.setStartPoint(tempNode.getNodePoint());
        nodes.set(e.getEdge((byte)j).getIdNumber()-1, null);
      }
    }
    else
    {
      ve.setEndPoint(ne.getNodePoint());
    }
  }


  // Darstellen
  this.setLayout(null);

 this.add(ne);
  for( int i = 0; i < size; i++)
  {
    if( nodes.get(i) != null )
    {
      this.add((Node)nodes.get(i));
    }
```

```
    }
    for( int i = 0; i < size; i++)
    {
      if( vEdges.get(i) != null )
      {
        this.add((VisualEdge)vEdges.get(i));
      }
    }
    setSchema(schema);
  }


  /**
   * Stellt die Karte nach einem neuen Schema dar.
   *
   * @param schema  Das zur Darstellung verwendete Schema.
   */
  public void setSchema(MapSchema schema)
  {
   this.schema = schema;

   if(schema == null ){this.repaint(); return;}

   for( int i = 0; i < vEdges.size(); i++)
   {
     ((VisualEdge)vEdges.get(i)).setPos(schema.getEdgeX(i),
                                        schema.getEdgeY(i));
   }

   for( int i = 0; i < nodes.size(); i++)
   {
     if( nodes.get(i) != null )
     {
        ((Node)nodes.get(i)).setPos(schema.getNodeX(i), schema.getNodeY(i));
     }
   }

   MainWindow main = MainWindow.getMainWindow();
   if( main != null )
   {
     main.setSize(schema.getWindowWidth(), schema.getWindowHeight());
     main.doLayout();
   }
```

```
   this.repaint();
  }


  /**
   * Berechnet das Schema der aktuellen Darstellung.
   *
   * @return  Das neu berechnete Schema.
   */
  public MapSchema getSchema()
  {
   schema = new MapSchema();

   for( int i = 0; i < vEdges.size(); i++)
   {
     schema.setEdge(i, (VisualEdge)vEdges.get(i));
   }

   for( int i = 0; i < nodes.size(); i++)
   {
     {
       schema.setNode(i, (Node)nodes.get(i));
     }
   }

   MainWindow main = MainWindow.getMainWindow();
   if( main != null )
   {
      schema.setWindowSize(main.getSize().width, main.getSize().height);
   }
    this.schema = schema;
    return( schema );
  }

}
```

StatusLabel

```
package gui;

import javax.swing.*;
import java.awt.*;

/**
 * Zeigt den Programm-Status und eventuelle Fehlermeldungen in der
 * Fusszeile an. Damit Statusinformationen von allen m&ouml;glichen
 * Quellen angezeigt werden k&ouml;nnen ist es als <b>Singleton</b> realisiert.
 *
 * @author Urs Heimann
 */
public class StatusLabel extends JPanel {


  /** Textfarbe einer Statusmeldung. */
  private static final Color  STATUS_COLOR   = Color.BLACK;
  /** Textfarbe einer Fehlermeldung. */
  private static final Color  ERROR_COLOR    = new Color(200, 0, 0);
  /** Text der beim starten angezeigt wird. */
  private static final String WELCOME_STRING = "  Welcome to Ant";
  /** Text der beim starten als Batteriestatus angezeigt wird. */
  private static final
              String NO_BATTERIE_INFO = " Batteriespannung:  keine Information";
  /** Grenzspannung zum hervorheben der Spannungsanzeige. */
  private static final float LOW_VOLTAGE = 7.5f;

  /** Die Statuszeile. */
  private static StatusLabel dasLabel = null;

  /** Anzeige f&uuml;r Status- und Fehlermeldungen. */
  private JLabel status;
  /** Anzeige f&uuml;r den Batteriezustand des RCX. */
  private JLabel batterie;

  /**
   * Erstellt die Statuszeile.
   */
  private StatusLabel()
  {
    status = new JLabel(WELCOME_STRING);
    batterie = new JLabel(NO_BATTERIE_INFO, JLabel.RIGHT);
```

```
  this.setLayout(new BorderLayout());
  this.add(status, BorderLayout.CENTER);
  this.add(batterie, BorderLayout.EAST);
}

/**
 * Liefert das Status Objekt und erzeugt es bei bedarf.
 *
 * @return  das Status-Objekt
 */
public synchronized static StatusLabel getStatusLabel()
{
 if( dasLabel == null )
 {
   dasLabel = new StatusLabel();
 }

  return dasLabel;
}

/**
 * Schreibt eine Meldung in die Statuszeile.
 * Eine Fehlermeldung wird durch rote Farbe hervorgehoben.
 *
 * @param text    Die anzuzeigende Meldung.
 * @param error   <code>true</code> - falls es sich um eine Fehlermeldung
 *                 handelt. <br>
 *                 <code>false</code> - bei einer normalen Meldung.
 */
public synchronized void setStatus(String text, boolean error)
{
 if( error )
 {
   status.setForeground(ERROR_COLOR);
 }
 else
 {
   status.setForeground(STATUS_COLOR);
 }
  status.setText("  " + text);
  status.repaint();
}
```

```
  /**
   * Schreibt die Batteriespannung des RCX in die Statuszeile.
   * Ist die Batteriespannung niedrig, wird die Anzeige durch rote Farbe
   * hervorgehoben.
   *
   * @param voltage  Die Spannung der Batterie.
   */
  public synchronized void setBatterieStatus(float voltage)
  {
   if( voltage != 0f )
   {
     if( voltage < LOW_VOLTAGE )
     {
      batterie.setForeground(ERROR_COLOR);
     }
     else
     {
      batterie.setForeground(STATUS_COLOR);
     }
      batterie.setText("Batteriespannung: " + voltage);
   }
   else
   {
      batterie.setForeground(STATUS_COLOR);
      batterie.setText(NO_BATTERIE_INFO);
   }

  }

}
```

VisualEdge

```
package gui;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;

import map.Edge;



/**
 * Zeichnet eine Strecke der Karte auf den Bildschirm.
 * Die Strichdicke ist abh&auml;ngig vom Pheromonwert auf der Strecke.
 *
 * @author Urs Heimann
 */
public class VisualEdge extends JLabel
{
  /** Breite des festen Streckenabschnitts. */
  private static final int WIDTH = 50;
  /** H&ouml;he des festen Streckenabschnitts. */
  private static final int HEIGHT = 40;
  /** Skalierungsfaktor zwischen Pheromonwert und Liniendicke. */
  private static final float PHEROMON_SCALE = 100f;
  /** Normale Linienfarbe. */
  private static final Color COLORNORMAL = Color.red;
  /** Linienfarbe wenn Cursor &uuml;ber der Strecke ist. */
  private static final Color COLORHIGH = Color.blue;
  /** Linienfarbe wenn Strecke selektiert ist. */
  private static final Color COLORSELECT = Color.cyan;

  /** Strecke die Visualisiert wird. */
  private Edge edge;
  /** Streckennummer */
  private int id = 0;
  /** Pheromone auf der Strecke. */
  private int pheromon = 0;
  /** Startpunkt der Strecke (Koordinaten eines Knotens) */
  private Point2D.Double startPointAbs;
  /** Endpunkt der Strecke (Koordinaten eines Knotens) */
  private Point2D.Double endPointAbs;
```

```
  /** x Koordinate des festen Streckenabschnitts bezogen auf das RoutePanel. */
  private int xPosAbs;
  /** y Koordinate des festen Streckenabschnitts bezogen auf das RoutePanel. */
  private int yPosAbs;
  /** x Koordinate des Labels auf dem RoutePanel. */
  private int xRelOffset;
  /** y Koordinate des Labels auf dem RoutePanel. */
  private int yRelOffset;

  /** Wenn <code>true</code> wird der Knoten als Highlited dargestellt. */
  private boolean highlited = false;
  /** Wenn <code>true</code> wird der Knoten als Selected dargestellt. */
  private boolean selected  = false;



  /**
   * Erstellt eine neue Visuelle-Strecke aufgrund einer Strecke.
   *
   * @param e Die Strecke die Visualisiert werden soll.
   * @param eInfo Fenster das die weiteren Daten der Strecke anzeigt.
   */
  public VisualEdge(Edge e, EdgeInfo eInfo)
  {
   this.edge = e;
    this.id = edge.getIdNumber();
    this.pheromon = edge.getPheromone();

    startPointAbs = null;
    endPointAbs = null;
    xPosAbs = 0;
    yPosAbs = 0;
    EdgeMouseListener eml = new EdgeMouseListener(this, eInfo);
//    this.addMouseMotionListener(new EdgeMouseMotionListener(this, eml));
    this.addMouseListener(eml);
  }


  /**
   * Zeichnet die Strecke auf den Bildschirm.
   * Diese Funktion wird vom System aufgerufen.
   *
   * @param g  Grafikobjekt des Systems.
   * */
```

```
public void paintComponent(Graphics g)
{
  this.computePosition();
  Graphics2D g2 = (Graphics2D)g;
  g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                      RenderingHints.VALUE_ANTIALIAS_ON);

  if( highlited )
  {  g2.setColor(COLORHIGH); }
  if( selected )
  {  g2.setColor(COLORSELECT); }

  if( pheromon == 0 )
  {
    g2.setStroke(getStroke(PHEROMON_SCALE, true));
  }
  else
  {
    g2.setStroke(getStroke((float)pheromon, false));
  }
  g2.drawLine(xPosAbs - xRelOffset, yPosAbs - yRelOffset,
              xPosAbs - xRelOffset + WIDTH, yPosAbs - yRelOffset);

  g2.drawString("ID: " + Integer.toString(id),
                xPosAbs - xRelOffset + WIDTH / 4, yPosAbs - yRelOffset - 10);

  if( startPointAbs != null )
  {
    g2.drawLine((int)startPointAbs.getX() - xRelOffset,
                (int)startPointAbs.getY() - yRelOffset,
                xPosAbs - xRelOffset,
                yPosAbs - yRelOffset);

  }
  if( endPointAbs != null )
  {
    g2.drawLine((int)endPointAbs.getX() - xRelOffset,
                (int)endPointAbs.getY() - yRelOffset,
                xPosAbs - xRelOffset + WIDTH,
                yPosAbs - yRelOffset);
  }
}
```

```
/**
 * Liefert die Nummer der Strecke.
 *
 * @return  Nummer der Strecke.
 */
public int getID()
{
 return( id );
}


/**
 * Liefert die Strecke, die Visualisiert wird.
 *
 * @return  Die Strecke
 */
public Edge getEdge()
{
  return( edge );
}


/**
 * Setzt den Startpunkt der Strecke.
 *
 * @param p   Das point object eines Knotens.
 */
public void setStartPoint(Point2D.Double p)
{
  startPointAbs = p;
  computePosition();
}


/**
 * Setzt den Endpunkt der Strecke.
 *
 * @param p   Das point object eines Knotens.
 */
public void setEndPoint(Point2D.Double p)
{
  endPointAbs = p;
  computePosition();
}
```

```
/**
 * Liefert den Startpunkt der Strecke.
 *
 * @return   Das point object eines Knotens.
 */
public Point2D.Double getStartPoint()
{
  return(startPointAbs);
}



/**
 * Liefert den Endpunkt der Strecke.
 *
 * @return  Das point object eines Knotens.
 */
public Point2D.Double getEndPoint()
{
  return(endPointAbs);
}


/**
 * Setzt den Pheromonwert der Strecke.
 *
 * @param p  Neuer Pheromonwert.
 */
public void setPheromon(int p)
{
  pheromon = p;
}


/**
 * Liefert den Pheromonwert der Strecke.
 *
 * @return  Der Pheromonwert
 */
public int getPheromon()
{
  return(pheromon);
```

```
}


/**
 * Setzt den festen Streckenabschnitt auf die gew&uuml;nschten Koordinaten.
 *
 * @param x  x Koordinate
 * @param y  y Koordinate
 */
public void setPos(int x, int y)
{
  xPosAbs = x;
  yPosAbs = y;
  computePosition();
}


/**
 * Verschiebt den festen Streckenabschnitt um den gew&uuml;nschten Offset.
 *
 * @param xOffset   Horizontale verschiebung.
 * @param yOffset   Vertikale verschiebung.
 */
public void movePos(int xOffset, int yOffset)
{
  xPosAbs = xPosAbs + xOffset;
  yPosAbs = yPosAbs + yOffset;
  computePosition();
}


/**
 * Kenzeichnet die Strecke als Selektiert.
 *
 * @param s  <code>true </code> - Strecke ist selektiert. <br>
 *            <code>false</code> - Strecke ist nicht selektiert. <br>
 */
public void setSelected(boolean s)
{
  selected = s;
}


/**
```

```
 * Kenzeichnet die Strecke als Highlited.
 *
 * @param h  <code>true </code> - Strecke ist gehighlited. <br>
 *            <code>false</code> - Strecke ist nicht gehighlited. <br>
 */
public void setHighlited(boolean h)
{
 highlited = h;
}


/**
 * Liefert die Position des festen Streckenabschnittes.
 *
 * @return  x Koordinate.
 */
public int getPosX()
{
  return( xPosAbs );
}


/**
 * Liefert die Position des festen Streckenabschnittes.
 *
 * @return  yKoordinate.
 */
public int getPosY()
{
  return( yPosAbs );
}


/**
 * Berechnet die ben&ouml;tigte Gr&ouml;sse des Labels um
 * die Strecke darstellen zu k&ouml;nnen.
 */
private void computePosition()
{
  int xmin = xPosAbs;
  int ymin = yPosAbs - (HEIGHT + 5);
  int xmax = xPosAbs + WIDTH;
  int ymax = yPosAbs + 5;
```

```
    if( startPointAbs != null )
    {
      if( (int)startPointAbs.getX() < xmin )
      {  xmin = (int)startPointAbs.getX(); }
      if( (int)startPointAbs.getY() < ymin )
      {  ymin = (int)startPointAbs.getY(); }
      if( (int)startPointAbs.getX() > xmax )
      {  xmax = (int)startPointAbs.getX(); }
      if( (int)startPointAbs.getY() > ymax )
      {  ymax = (int)startPointAbs.getY(); }
    }
    if( endPointAbs != null )
    {
      if( (int)endPointAbs.getX() < xmin )
      {  xmin = (int)endPointAbs.getX(); }
      if( (int)endPointAbs.getY() < ymin )
      {  ymin = (int)endPointAbs.getY(); }
      if( (int)endPointAbs.getX() > xmax )
      {  xmax = (int)endPointAbs.getX(); }
      if( (int)endPointAbs.getY() > ymax )
      {  ymax = (int)endPointAbs.getY(); }
    }

    this.setSize(xmax - xmin +4, ymax - ymin +4);

    xRelOffset = xmin -2;
    yRelOffset = ymin -2;
    this.setLocation(xRelOffset, yRelOffset);

  }


  /**
   * Liefert
   */
  private BasicStroke getStroke(float width, boolean dashed)
  {
    if( dashed )
    {
      float dash1[] = {5.0f};
      return( new BasicStroke(width / PHEROMON_SCALE,
                              BasicStroke.CAP_BUTT,
                              BasicStroke.JOIN_MITER,
                              10.0f, dash1, 0.0f));
```

```
    }
    else
    {
      return( new BasicStroke(width / PHEROMON_SCALE,
                              BasicStroke.CAP_ROUND,
                              BasicStroke.JOIN_MITER));
    }
  }
}



// Listener
class EdgeMouseListener implements MouseListener
{
  private VisualEdge edge;
  private EdgeInfo edgeInfo;
  private int startDragX;
  private int startDragY;

  public EdgeMouseListener(VisualEdge e, EdgeInfo eInfo)
  {
    this.edge = e;
    this.edgeInfo = eInfo;
  }

  public void mouseClicked(MouseEvent e)
  {
   edgeInfo.showInfo(edge);
  }
  public void mouseEntered(MouseEvent e)
  {
    edge.setHighlited(true);
    edge.repaint();
  }
  public void mouseExited(MouseEvent e)
  {
    edge.setHighlited(false);
    edge.repaint();
  }
  public void mousePressed(MouseEvent e)
  {
   edge.setSelected(true);
    startDragX = e.getX();
```

```
    startDragY = e.getY();
  }
  public void mouseReleased(MouseEvent e)
  {
   edge.setSelected(false);
   edge.movePos(e.getX() - startDragX, e.getY() - startDragY);
    edge.repaint();
  }

  public int getStartX()
  {
   return( startDragX );
  }

  public int getStartY()
  {
   return( startDragY );
  }
}
```

C.1.2. Package logic

MapReader

```
package logic;

import map.Map;
import map.Edge;
import communication.Communication;
import java.io.*;
import gui.*;

/**
 * Der MapReader liest die Karte aus dem RCX und stellt sie f&uuml;r andere
 * Klassen bereit.
 *
 * @author Urs Heimann
 */
public class MapReader extends Thread {


  private static final int RELOAD_INTERVAL = 10;

  /** Die zuletzt gelesene Karte. */
  private Map map;

  /** <code>true</code> solange der Thread l&auml;uft. */
  private boolean running = true;
  /** <code>true</code> solange der Thread nicht versucht die Karte zu lesen. */
  private boolean sleeping = true;

  /** Wartezeit nach dem Auslesen der Karte bis wieder versucht wird zu lesen.
   * In Sekunden. */
  private int reloadInterval;

  /** Das Kommunikationsobjekt */
  private Communication comm;


  /**
   * Erstellt einen neuen Kartenleser.
   */
  public MapReader()
  {
```

```
   super("MapReader");
   reloadInterval = RELOAD_INTERVAL;
  }

  /**
   * L&auml;dt die Karte aus dem RCX und speichert sie.
   */
  public synchronized void loadMap()
  {
   if( comm == null )
   {
      StatusLabel.getStatusLabel().setStatus(
           "Kommunikation nicht verfügbar. Bitte Programm neu starten.", true);
     return;
   }

   StatusLabel.getStatusLabel().setStatus("Karte wird geladen...", false);
   float batt = 0f;
//    generateMap();

    try
{
      System.out.println("Lade Karte");
      comm.readMap(map);
      System.out.println("Lade Batteriespannung");
      batt = comm.readBatteryPower();
}
catch(IOException e)
{
  System.err.println(e);
      StatusLabel.getStatusLabel().setStatus("Fehler beim laden der Karte:  <<"
                                             + e.toString() + ">>", true);
  return;
}

    StatusLabel.getStatusLabel().setStatus("Karte geladen.", false);
    StatusLabel.getStatusLabel().setBatterieStatus(batt);
  }

  /**
   * Liefert die Karte.
   *
   * @return die zuletzt gelesene {@link map.Map Map}.
   */
```

```
  public Map getMap()
  {
    return (map);
  }


  /**
   * Liefert die gr&ouml;sse der Karte.
   *
   * @return Anzahl Strecken auf der Karte
   */
  public int getMapSize()
  {
    return map.numberOfEdges();
  }


  /**
   * Startmethode des MapReader Threads.
   */
  public void run()
  {
    map = new Map((byte)0);
    System.out.println("Mache Com");
    StatusLabel.getStatusLabel().setStatus("Initialisiere Kommunikation...",
                                           false);
try
{
      comm = new Communication();
}
catch(IOException e)
{
  System.err.println(e);
      StatusLabel.getStatusLabel().setStatus(
                            "Fehler beim initialisieren der Kommunikation:  <<"
                            + e.toString() + ">>", true);

      comm = null;
      return;
}
catch(UnsatisfiedLinkError e)
{
  System.err.println(e);
      StatusLabel.getStatusLabel().setStatus(
```

```
                                "Fehler beim initialisieren der Kommunikation:  <<"
                                + e.toString() + ">>", true);
      comm = null;
      return;
}
    StatusLabel.getStatusLabel().setStatus("Kommunikation initialisiert.",
                                           false);


   while( running )
   {
     if( !sleeping )
     {
       this.loadMap();
       MainWindow.getMainWindow().getRoutePanel().showMap(this);
        try
       {
         for(int i=0; i<reloadInterval; i++)
         {
            this.sleep(1000);
            if( sleeping ) { break; }
         }
        }
        catch(InterruptedException e)
        {}
     }
     else
     {
        try
       {
          this.sleep(100);
        }
        catch(InterruptedException e)
        {}
     }

   }
  }


  /**
   * Fordert den Thread auf sich zu beenden.
   */
  public void finish()
```

```
{
  running = false;

}

/**
 * Stopt das automatische auslesen der Karte.
 */
public void pause()
{
  sleeping = true;
}

/**
 * Startet das automatische auslesen der Karte.
 */
public void endPause()
{
  sleeping = false;
}


/**
 * Setzt die Wartezeit zwischen dem Auslesen der Karte.
 *
 * @param i Wartezeit in Sekunden
 */
public void setReloadInterval(int i)
{
  reloadInterval = i;
}


/**
 * Liefert die Wartezeit zwischen dem auslesen der Karte.
 *
 * @return Die aktuelle Wartezeit in Sekunden
 */
public int getReloadInterval()
{
 return( reloadInterval );
}
```

```
/**
 * L&auml;dt die Karte einmal neu.
 */
public void reloadOnce()
{
  if( sleeping )
  {
    sleeping = false;
    try
   {
      this.sleep(200);
    }
    catch(InterruptedException e){}
    sleeping = true;
  }
}


/**
 * Testfunktion.
 * Generiert eine Karte zu Testzwecken.
 */
private void generateMap()
{

        Edge edgeStart = new Edge((byte) 1);
        Edge edgeOneLeft = new Edge((byte) 2);
        Edge edgeOneRight = new Edge((byte) 3);
        Edge edgeMiddle = new Edge((byte) 4);
        Edge edgeTwoLeft = new Edge((byte) 5);
        Edge edgeTwoRight = new Edge((byte) 6);
        Edge edgeEnd = new Edge((byte) 7);

        edgeStart.attachEdge(edgeOneLeft);
        edgeStart.attachEdge(edgeOneRight);
        edgeOneLeft.attachEdge(edgeMiddle);
        edgeOneRight.attachEdge(edgeMiddle);
        edgeMiddle.attachEdge(edgeTwoLeft);
        edgeMiddle.attachEdge(edgeTwoRight);
        edgeTwoLeft.attachEdge(edgeEnd);
        edgeTwoRight.attachEdge(edgeEnd);

        map = new Map((byte) 7);
        map.addEdge(edgeStart);
```

```
            map.addEdge(edgeOneLeft);
            map.addEdge(edgeOneRight);
            map.addEdge(edgeMiddle);
            map.addEdge(edgeTwoLeft);
            map.addEdge(edgeTwoRight);
            map.addEdge(edgeEnd);
    }
}
```

SchemaLoader

```
package logic;

import gui.*;
import javax.swing.*;
import java.io.*;

/**
 * L&auml;dt und Speichert die Darstellung einer Karte. <br>
 * Die Darstellung ist in einem {@link gui.MapSchema MapSchema} abgelegt.
 * Das Schema wird serialisiert und in einer Datei abgespeichert.
 *
 * @author Urs Heimann
 */
public class SchemaLoader
{

/** Der Standardpfad zum abspeichern der Schemas. */
  private static final String HOME_PATH = "L:\\";


  public SchemaLoader()
  {
  }

/**
 * Liest das Schema aus einer Datei.
 *
 * @return das gelesene Schema wenn erfolgreich, sonst <code>null</code>.
 */
  public static MapSchema loadSchema()
  {
    JFileChooser chooser = new JFileChooser(HOME_PATH);
    int retval = chooser.showOpenDialog(null);
    if( retval == JFileChooser.APPROVE_OPTION )
    {
      MapSchema schema;
      try
      {
        FileInputStream f = new FileInputStream(
                                  chooser.getSelectedFile().getAbsolutePath());
        ObjectInputStream s = new ObjectInputStream(f);
        schema = (MapSchema)s.readObject();
```

```
        s.close();
        f.close();
      }
      catch(IOException e)
      {
       StatusLabel.getStatusLabel().setStatus(
                                          "Fehler beim lesen des Schemas:  <<"
                                          + e.toString() + ">>", true);
       return( null );
      }
      catch(ClassNotFoundException e)
      {
       StatusLabel.getStatusLabel().setStatus(
                                          "Fehler beim lesen des Schemas:  <<"
                                          + e.toString() + ">>", true);
       return( null );
      }

      StatusLabel.getStatusLabel().setStatus("Schema erfolgreich geladen.",
                                             false);
      return( schema );
    }
    return( null );
  }

/**
 * Schreibt das Schema in eine Datei.
 *
 * @param schema   das zu speichernde Schema.
 */

  public static void saveSchema(MapSchema schema)
  {
    JFileChooser chooser = new JFileChooser(HOME_PATH);
    int retval = chooser.showSaveDialog(null);
    if( retval == JFileChooser.APPROVE_OPTION )
    {

      try
      {
        FileOutputStream f = new FileOutputStream(
                                    chooser.getSelectedFile().getAbsolutePath());
        ObjectOutputStream s = new ObjectOutputStream(f);
        s.writeObject(schema);
```

```
        s.flush();
        f.flush();
        s.close();
        f.close();
      }
      catch(IOException e)
      {
       StatusLabel.getStatusLabel().setStatus(
                                    "Fehler beim schreiben des Schemas:  <<"
                                    + e.toString() + ">>", true);
       return;
      }

      StatusLabel.getStatusLabel().setStatus("Schema erfolgreich gespeichert.",
                                             false);

    }
  }
}
```

C.2. RCX

C.2.1. Package robot

RoboAnt

```
package robot;

import josx.platform.rcx.*;

import java.io.IOException;
import communication.Communication;
import map.*;

public class RoboAnt
{
   private final static int STARTPHEROMONE = 200;
   private final static double k = 100;         // attraction
   private final static byte n = 2;             // nonlinearity

   private Colours leftSensor;
   private Colours rightSensor;
   private Colours thirdSensor;

   private Motor leftMotor;
   private Motor rightMotor;
   private Motor lamp;

   private int startTime;

   private MapManager mapManager;
   private Map map;

   private Communication com;


   public static void main(String[] args)
   {
      RoboAnt ant = new RoboAnt();
   }


   public RoboAnt()
   {
      leftSensor = new Colours(Sensor.S2);      //Linker Sensor
```

```
leftSensor.setBlack(38); //38
leftSensor.setWhite(52);
leftSensor.setGreen(43);
//leftSensor.setTolerance(4);

rightSensor = new Colours(Sensor.S1);     //Rechter Sensor
rightSensor.setBlack(36); //37
rightSensor.setWhite(50);
rightSensor.setGreen(40); //41
//rightSensor.setTolerance(4);

thirdSensor = new Colours(Sensor.S3);     //Dritter Sensor (links aussen)
thirdSensor.setBlack(35);
thirdSensor.setWhite(48);
thirdSensor.setGreen(39);

leftMotor = Motor.B;                      //Motoren init
rightMotor = Motor.A;
lamp = Motor.C;

leftMotor.setPower(7);
rightMotor.setPower(7);
lamp.setPower(7);

                                          // Karte aufbauen
{
   Edge edgeStart = new Edge((byte) 1);
   Edge edgeOneLeft = new Edge((byte) 2);
   Edge edgeOneRight = new Edge((byte) 3);
   Edge edgeMiddle = new Edge((byte) 4);
   Edge edgeTwoLeft = new Edge((byte) 5);
   Edge edgeTwoRight = new Edge((byte) 6);
   Edge edgeEnd = new Edge((byte) 7);

   edgeStart.attachEdge(edgeOneLeft);
   edgeStart.attachEdge(edgeOneRight);
   edgeOneLeft.attachEdge(edgeMiddle);
   edgeOneRight.attachEdge(edgeMiddle);
   edgeMiddle.attachEdge(edgeTwoLeft);
   edgeMiddle.attachEdge(edgeTwoRight);
   edgeTwoLeft.attachEdge(edgeEnd);
   edgeTwoRight.attachEdge(edgeEnd);

   map = new Map((byte) 7);
```

```
        map.addEdge(edgeStart);
        map.addEdge(edgeOneLeft);
        map.addEdge(edgeOneRight);
        map.addEdge(edgeMiddle);
        map.addEdge(edgeTwoLeft);
        map.addEdge(edgeTwoRight);
        map.addEdge(edgeEnd);
    }

    for (byte i = 0;(i < 7); i++)             // Pheromonwerte init
    {
        map.getEdge(i).setPheromone(STARTPHEROMONE);
    }

    mapManager = new MapManager(map);

    try
    {
        com = new Communication();

        //Freier Speicher anzeigen
        LCD.showNumber((int) Runtime.getRuntime().freeMemory());

        writeToPC(map);                        // Daten zum PC senden
        live();                               // zum Leben erwecken
    }
    catch (IOException e)
    {}
}


/**
 * Startet den Main Loop. Ruft wenn beide Sensoren grün haben bothGreen auf.
 */
private void live()
{
    // Zeitstoppung starten und losfahren
    startTime = (int) System.currentTimeMillis();
    leftMotor.forward();
    rightMotor.forward();

    // Main Loop
    while (true)
    {
```

```
      if (leftSensor.isGreen()
         && rightSensor.isGreen()
         && thirdSensor.isGreen())
      {
         bothGreen();
      }

      followLine();
   }
}


/**
 * Entscheidet was zu tun ist, wenn ein grüner Streifen erreicht worden ist.
 * Möglichkeiten: branch(), followLine(), finishRound()
 */
private void bothGreen()
{
   lamp.forward();                     // Lampe anzünden

   if (mapManager.isFork())            // Verzweigung?
   {
      branch();
   }
   else                                //sonst: ein folgender edge? dann:
      if (!mapManager.isRoundFinished())
      {
         mapManager.go(Direction.CENTER);

         // Über grünen Streifen fahren.
         int waitTime = (int) System.currentTimeMillis();
         while (250 >= (int) (System.currentTimeMillis()) - waitTime)
         {
            followLine();
         }

      }
      else
      {
         finishRound();
      }

   lamp.stop(); // Lampe löschen
}
```

```
/**
 * Fährt bei einer Kreuzung nach links oder rechts, je nach Pheromonmenge.
 */
private void branch()
{
   byte direction = Direction.RIGHT;

   int random = RoboMath.random();     // Zufallszahl 0-127
   double p = calculateProbability();  // Wahrscheinlichkeit für Links 0-1

   int pl = (int) (p * RoboMath.MAXRANDOM);

   LCD.showNumber((int) random);    //Ausgabe der Wahrsch. in %

   if (random <= pl)                   //Wenn die Zufallszahl kleiner ist als
   {                                   //die Wahrsch., nach links.
      direction = Direction.LEFT;
   }
   else
   {
      direction = Direction.RIGHT;
   }

   mapManager.go(direction);

   // Fahre in die ermittelte Richtung
   if (direction == Direction.LEFT)
   {
      followLeft();
   }
   else if (direction == Direction.RIGHT)
   {
      followRight();
   }
}


/**
 * Stoppt die Zeitmessung und Motoren, sendet Daten zum PC.
 * Nach dem senden der Daten zum PC, wird die Map zurückgesetzt,
 * die Zeitmessung und Motoren gestartet.
 */
```

```
private void finishRound()
{
   // Zeitstoppen beenden
   int finishTime = (int) System.currentTimeMillis();
   int time = (int) finishTime - startTime;

   // Anhalten
   leftMotor.stop();
   rightMotor.stop();

   map.evaporatePheromone();
   mapManager.updatePheromone(time / 1000);

   // Daten zum PC senden
   writeToPC(map);

   mapManager.restart();

   // Zeitstoppung beginnen und losfahren
   startTime = (int) System.currentTimeMillis();
   leftMotor.forward();
   rightMotor.forward();

   int waitTime = (int) System.currentTimeMillis();
   while (250 >= (int) (System.currentTimeMillis()) - waitTime)
   {
     followLine();
   }
}


/**
 * Folgt der schwarzen Linie.
 */
private void followLine()
{
   if (leftSensor.isWhite())
   {
      rightMotor.backward();
   }
   else
   {
      rightMotor.forward();
   }
```

```
      if (rightSensor.isWhite())
      {
         leftMotor.backward();
      }
      else
      {
         leftMotor.forward();
      }
   }


   /**
    * Folgt dem linken Rand (links weiss, rechts schwarz) für 2 Sekunden
    */
   private void followLeft()
   {
      int branchTime = (int) System.currentTimeMillis();

      while (2000 >= (int) (System.currentTimeMillis()) - branchTime)
      {
         if (leftSensor.isBlack())
         {
            leftMotor.backward();
         }
         else
         {
            leftMotor.forward();
         }

         if (rightSensor.isWhite())
         {
            rightMotor.backward();
         }
         else
         {
            rightMotor.forward();
         }
      }
   }


   /**
    * Folgt dem rechten Rand in einer Rechtskurve
```

```
 */
private void followRight()
{
   int branchTime = (int) System.currentTimeMillis();

   while (2000 >= (int) (System.currentTimeMillis()) - branchTime)
   {
      //Sound.beep();
      //LCD.showNumber(leftSensor.getValue());

      if (leftSensor.isWhite())
      {
         leftMotor.backward();
      }
      else
      {
         leftMotor.forward();
      }

      if (rightSensor.isBlack())
      {
         rightMotor.backward();
      }
      else
      {
         rightMotor.forward();
      }

   }
}


/**
 * Berechnet die Wahrscheinlichkeit für das Linksabbiegen.
 * @return Wahrscheinlichkeit von 0-1
 */
private double calculateProbability()
{
   //Pa = (k+Ai)^n / ( (k+Ai)^n + (k+Bi)^n );
   //Pa = 1 - Pb;

   double pl;

   double l = mapManager.getPheromone(Direction.LEFT);
```

```
      double r = mapManager.getPheromone(Direction.RIGHT);

      pl =
         RoboMath.pow((k + l), n)
            / (RoboMath.pow((k + l), n) + RoboMath.pow((k + r), n));
      return pl;
   }


   /**
    * Sendet eine Map und Statusinformationen zum PC
    * @param Zu sendende Map
    */
   private void writeToPC(Map map)
   {
      try
      {
         //TextLCD.print("write Map");
         com.writeMap(map);
         com.writeBatteryPower(Battery.getVoltage());
      }
      catch (IOException e)
      {
         //TextLCD.print("Exc wri");
         //Sound.buzz();
         //System.exit(0);
      }

   }

}
```

Colours

```
package robot;

import josx.platform.rcx.*;


public class Colours
{
   private int white;
   private int black;
   private int green;

   private int tolerance;

   private Sensor theSensor;


   public Colours(Sensor theSensor)
   {
      this.theSensor = theSensor;
      theSensor.setTypeAndMode(SensorConstants.SENSOR_TYPE_LIGHT,
                               SensorConstants.SENSOR_MODE_PCT);

      theSensor.activate();

      white = 52;
      black = 36;
      green = 42;
      tolerance = 4;
   }


   public boolean isBlack()
   {
      int measure;

      measure = theSensor.readValue();
      //checkMeasure(measure);

      if (measure < (black + tolerance))
      {
         return (true);
      }
```

```
      else
      {
         return (false);
      }
   }


   public boolean isWhite()
   {
      int measure;

      measure = theSensor.readValue();
      //checkMeasure(measure);

      if (measure > (white - tolerance))
      {
         return (true);
      }
      else
      {
         return (false);
      }
   }


   public boolean isGreen()
   {
      int measure;

      measure = theSensor.readValue();

      if ( (measure <= green + 2) && (measure >= green -2) )
      //if ( (measure == green + tolerance/4) || (measure == green - tolerance/4) )
      {
         return (true);
      }
      else
      {
         return (false);
      }
   }
}
```

C.3. Shared

C.3.1. Package communication

Communication

```
package communication;

import java.io.*;
import josx.rcxcomm.*;
import map.*;

/**
 * <p>Title: Communication.java </p>
 * <p>Description: Diese Klasse ist für die Kommunikation zwischen
 * PC und RCX zust&auml;ndig </p>
 * @author Michael Remund, 30.01.2003
 */
public class Communication
{
private Map map;
private RCXPort port;
private InputStream is;
private OutputStream os;
byte[] byteMap;
/**
 * Buffergr&ouml;sse für ByteArray, muss bei ver&auml;nderter
     * Struktur der Map
 * eventuell angepasst werden
 */
public static final byte BUFFER_SIZE = 51;

public Communication() throws IOException
{
this.port = new RCXPort();

is = port.getInputStream();
os = port.getOutputStream();
byteMap = new byte[BUFFER_SIZE];
}

/**
 * @param map Map, welche &uuml;bermittelt werden soll
 */
public void writeMap(Map map) throws IOException
```

```
{
map.serialize(byteMap); //byteMap as returnvalue
os.write(byteMap);
}
/**
 * @param map Map, welche mit neuen Daten aufgebaut wird
 */
public void readMap(Map map) throws IOException
{
is.read(byteMap);
/*
//Kontrollausgabe auf PC
System.out.print("Alte Map Size: Read Data: ");
for (int i = 0; i < byteMap.length; i++)
{
    System.out.print((byteMap[i] & 0xff) + " ");
}
*/
map.deserialize(byteMap);
}

/**
 * @param f Batteriespannung, welche zum PC gesendet werden soll
 */
public void writeBatteryPower(float f) throws IOException
{
byte b = (byte) (f * 10); //Eine Komma Stelle
os.write(b);
}

/**
 * @return Batteriespannung des RCX
 */
public float readBatteryPower() throws IOException
{
return ((float) is.read()) / 10;
}

}
```

C.3.2. Package map

Edge

```
package map;
import java.util.Vector;

public class Edge
{
   private Vector nextEdges;
   private int pheromone = 0;
   byte idNumber = 0;
   private boolean end = false;

   public Edge()
   {
      nextEdges = new Vector();
   }

   public Edge(byte idNumber)
   {
      nextEdges = new Vector();
      this.idNumber = idNumber;
   }

   public void attachEdge(Edge edge)
   {
      nextEdges.addElement(edge);
   }

   public void detachAllEdges()
   {
      nextEdges.clear();
      nextEdges.setSize(0);
   }

   public Edge getEdge(byte index)
   {
      if (index < (byte)nextEdges.size())
      {
         return (Edge) (nextEdges.elementAt(index));
      }
      return null;
   }
```

```
public byte getNumNextEdges()
{
   return (byte) nextEdges.size();
}

public void setPheromone(int pheromone)
{
   this.pheromone = pheromone;
}

public int getPheromone()
{
   return pheromone;
}

public byte getIdNumber()
{
   return idNumber;
}

public void setIdNumber(byte idNumber)
{
   this.idNumber = idNumber;
}

public byte serialize(byte[] o, byte offset)
{
   o[offset++] = idNumber;
   o[offset++] = (byte) (pheromone & 0xFF);         //LLSB
   o[offset++] = (byte) ((pheromone >> 8) & 0xFF);  //MLSB
   o[offset++] = (byte) ((pheromone >> 16) & 0xFF); //LMSB
   o[offset++] = (byte) ((pheromone >> 24) & 0xFF); //MMSB
   o[offset++] = (byte) nextEdges.size();

   for (byte i = 0; i < nextEdges.size(); i++)
   {
      o[i + offset] = ((Edge) (nextEdges.elementAt(i))).getIdNumber();
   }

   offset += (byte)nextEdges.size();
   return offset;
}
```

```
   public boolean equals(Object edge)
   {
      if (this.idNumber == ((Edge) edge).idNumber)
      {
         return true;
      }
      return false;
   }

/* Auskommeniern sparte 3kb Speicherplatz auf RCX
 *
 *
   public String toString()
   {
      return "\nEdge Id Number: " + idNumber
          + "\nPheromone: " + pheromone
          + "\nNumber of next Edges: " + nextEdges.size()
          + "\nEnd: " + end;
   }
 */
}
```

Map

```
package map;
import java.util.Vector;
import java.io.*;

public class Map
{
   private static final byte DIVISOR = (byte)2;    //für evaporate Pheromone

   private Vector allEdges;          // Alle erzeugten Edge Instanzen
   private static Edge tmp;         // Speicherplatz sparen auf RCX.
                                      // Object wird immer wieder gebraucht

   public Map(byte numberOfEdges)
   {
      allEdges = new Vector(numberOfEdges);
      tmp = new Edge();
   }

   public void addEdge(Edge edge)
   {
      allEdges.addElement(edge);
   }

   public Edge getEdge(byte number)
   {
      return (Edge) allEdges.elementAt(number);
   }

   public int numberOfEdges()
   {
      return allEdges.size();
   }

   public void evaporatePheromone()
   {
      int pheromone;

      for (byte i = 0; i < allEdges.size(); i++)
      {
         pheromone = ((Edge) allEdges.elementAt(i)).getPheromone();
         if (pheromone > 0)
         {
```

```
            pheromone = pheromone / DIVISOR;
         }
         ((Edge) allEdges.elementAt(i)).setPheromone(pheromone);
      }
   }

   public byte serialize(byte[] buffer)
   {
      byte offset = 1;
      buffer[0] = (byte) allEdges.size();

      for (byte i = 0; i < allEdges.size(); i++)
      {
         offset = ((Edge) allEdges.elementAt(i)).serialize(buffer, offset);
      }
      return offset;
   }

   public void deserialize(byte[] byteMap)
   {
      int edgeOffset = 0;
      int numNextEdges;
      int numEdges;
      int pheromone;
      byte idEdge;

      Edge currentEdge;
      Edge newEdge;

      numEdges = byteMap[edgeOffset];              // Anzahl Edges in byteMap
      edgeOffset++;

      for (byte i = 0; i < numEdges; i++)
      {
         // Bereits vorhandene Edge wenn möglich überscheiben.
         currentEdge = findEdge(byteMap[edgeOffset]);
         if (currentEdge == null)
         {
            currentEdge = new Edge(byteMap[edgeOffset]);
            allEdges.addElement(currentEdge);
         }
         else
         {
           currentEdge.detachAllEdges();
```

```
        }

        edgeOffset++;

        // Pheromonwert auslesen
        pheromone = (byteMap[edgeOffset] & 0xff);// 0xff nötig, um den unsigned
                                                 // Wert des bytes zu erhalten
        edgeOffset++;
        pheromone += (byteMap[edgeOffset] & 0xff) << 8;

        edgeOffset++;
        pheromone += (byteMap[edgeOffset] & 0xff) << 16;

        edgeOffset++;
        pheromone += (byteMap[edgeOffset] & 0xff) << 24;

        currentEdge.setPheromone(pheromone);

        edgeOffset++;
        numNextEdges = byteMap[edgeOffset];

        edgeOffset++;

        // Nächste Edges an diesen Edge anhängen
        for (byte j = 0; j < numNextEdges; j++)
        {
           idEdge = byteMap[edgeOffset + j];
           newEdge = findEdge(idEdge);
           if (newEdge == null)
           {
              newEdge = new Edge(idEdge);
              allEdges.addElement(newEdge);
           }
           currentEdge.attachEdge(newEdge);
        }
        edgeOffset += numNextEdges;
     }

  }

  private Edge findEdge(byte id)
  {
     tmp.setIdNumber(id);
```

```
      int index = allEdges.indexOf(tmp);

      if (index >= 0)
      {
         return (Edge) allEdges.elementAt(index);
      }

      return null;
   }
}
```

MapManager

```
package map;

import java.util.Vector;

public class MapManager
{
   private Map theMap;
   private Edge currentEdge;
   private Vector drivenWay;

   private final static int MAXPHEROMONE = 1000;
   private final static byte MAXTIME = 40;
   private final static byte FACTOR = (byte) 3;

   public MapManager(byte[] byteMap, byte maxMapSize)
   {
      theMap = new Map(maxMapSize);
      theMap.deserialize(byteMap);
      init();
   }

   public MapManager(Map map)
   {
      theMap = map;
      init();
   }

   private void init()
   {
      currentEdge = theMap.getEdge((byte) 0);
      drivenWay = new Vector();
      drivenWay.addElement(currentEdge);
   }

   public Map getMap()
   {
      return theMap;
   }

   public void restart()
   {
      currentEdge = theMap.getEdge((byte) 0);
```

```
      drivenWay.clear();
      drivenWay.addElement(currentEdge);
   }

   public boolean isRoundFinished()
   {
      if (currentEdge.getNumNextEdges() == 0)
      {
         return true;
      }
      return false;
   }

   public boolean isFork()
   {
      if (currentEdge.getNumNextEdges() > 1)
      {
         return true;
      }
      return false;
   }

   public int getPheromone(byte direction)
   {
      if (currentEdge.getNumNextEdges() > direction)
      {
         return currentEdge.getEdge(direction).getPheromone();
      }
      return 0;
   }

   public void updatePheromone(int time)
   {
      int pheromone;

      if (time > MAXTIME)
      {
         time = MAXTIME;
      }

      for (byte i = 0; i < drivenWay.size(); i++)
      {
         pheromone = ((Edge) drivenWay.elementAt(i)).getPheromone();
```

```
            //pheromone += -22*time + 1040;
            pheromone += FACTOR * ((time - MAXTIME) * (time - MAXTIME));

            if (pheromone > MAXPHEROMONE)
            {
               pheromone = MAXPHEROMONE;
            }

            ((Edge) drivenWay.elementAt(i)).setPheromone(pheromone);
         }
      }

      public boolean go(byte direction)
      {
         if (currentEdge.getNumNextEdges() > direction)
         {
            currentEdge = currentEdge.getEdge(direction);
            drivenWay.addElement(currentEdge);
            return true;
         }
         return false;
      }

}
```

Direction

```
package map;
public class Direction
{
   public static final byte LEFT = 0;
   public static final byte RIGHT = 1;
   public static final byte CENTER = 0;
}
```

Glossar

Edge	Kante eines Graphen, Weg zwischen zwei Knoten.
Kante	Siehe Edge.
Knoten	Punkt an dem zwei (oder mehr) Edges zusammenkommen.
KI	Künstliche Intelligenz
leJOS	Die Java Virtual Machine für den RCX.
locator service	Dienst, welcher im Subnetz einen lookup service sucht.
lookup service	Dienst, welcher für die Registrieung von Objekten verantwortlich ist. z. B. Reggie.
NQC	Not Quite C, eine Programmiersprache.
Pheromone	Ein Markierungsstoff (Hormon) der langsam verdunstet.
RAW	Rohwert, den die Sensoren einlesen.
RCX	Robotics Command Explorer
reggie	Jini lookup service
RIS	Robotic Invention System. Das Mindstorms Hauptpaket von LEGO.
rmid	RMI Activaton Deamon, welcher bei Bedarf den lookup service aktiviert.
Runde	Eine Menge von Edges, die vom Start zum Ziel führt.
SO	self-organisation
TD	Threshhold Device
VM	Virtual Machine
Weg	siehe Runde.

Hilfsmittel

Hardware

- LEGO Robotics Invention System 2.0 (und Zubehör), http://www.lego.com
- Computer der HSR und persönliche Notebooks
- Digitalkamera Kodak DC240
- Taschenlampe

Software

- Eclipse 2.0.2, http://www.eclipse.org
- leJOS 2.0.0 und leJOS-Plugin für Eclipse, http://lejos.sourceforge.org
- Java 2 Standard Edition 1.4.1, http://java.sun.com
- Jini 1.2.1, http://java.sun.com/jini
- LaTeX: MiKTeX 2.2, http://www.miktex.org
- TeXnicCenter 1 Beta 6.01, http://www.toolscenter.org
- FreeVCS 2.2.1, http://www.freevcs.de
- OpenOffice 1.0.2, http://www.openoffice.org
- Maple 6
- Together 6.0.1

Literaturverzeichnis

[1] Mario Ferrari, Guilio Ferrari, Ralph Hempel, *Building Robots with LEGO Mindstorms*, (Syngress Publishing, 2002).

[2] Giulio Ferrari, Andy Gombos, Søren Hilmer, Jürgen Stuber, Mick Porter, Jamie Waldinger, Dario Laverde, *Programming LEGO Mindstorms with Java*, (Syngress Publishing, 2002).

[3] Valentino Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, (Eighth Printing, MIT Press, 2001).

[4] Eric Bonabeau, Marco Dorigo, Guy Theraulaz, *Swarm Intelligence from Natural to Artificial Systems*, (Oxford University Press, 1999).

[5] Stefano Nolfi, Dario Floreano, *Evolutionary Robotics: the Biology, Intelligence, and Technology of Self-Organizing Machines*, (Second Printing, Massachusetts Institute of Technology, 2001).

[6] *LEGO Mindstorms Internals*, http://www.crynwr.com/lego-robotics/

[7] Michael Gasperi, *MindStorms RCX Sensor Input Page*
http://www.plazaearth.com/usr/gasperi/lego.htm