Projekt künstliche Intelligenz
Teilprojekt Autonome mobile Systeme
Hasenjagd 2007

Poink

Projektdokumentation

von

Matthias Friese
Gregor Landmann
Martin Schwenke

20. August 2007

Betreuer:
Dipl.-Inform. Ingo Börsch
Prof. Dr.-Ing. Jochen Heinsohn

Fachhochschule Brandenburg
Fachbereich Informatik und Medien

Inhaltsverzeichnis

1 Zielsetzung

1.1 Wettbewerb

Die Hasenjagd besteht allgemein aus 2 autonomen Systemen auf dem Spielfeld. Ein autonomes System ist in der Rolle des Hasen das andere in der des Fuchses. Das Spielfeld ist ein n*n großes Feld mit oder ohne Hindernissen. Die zeitliche Begrenzung sind entweder 60 oder 120 Sekunden.[1]

1.2 grundlegende Design-Entscheidungen

Das Design von Poink wurde von Anfang an auf hohe Beschleunigung, hohe Geschwindigkeit und schnelle Richtungsänderungen hin ausgerichtet. Dies wurde durch einen kompakten Aufbau und ein geringes Gewicht ermöglicht. Letzteres schafft die Voraussetzung um eine höhere Übersetzung nutzen zu können. Die Sensoren zum empfangen des modulierten Signals aus dem Beacon des Gegners sind auf einem nach links und rechts drehbaren Kopf befestigt.

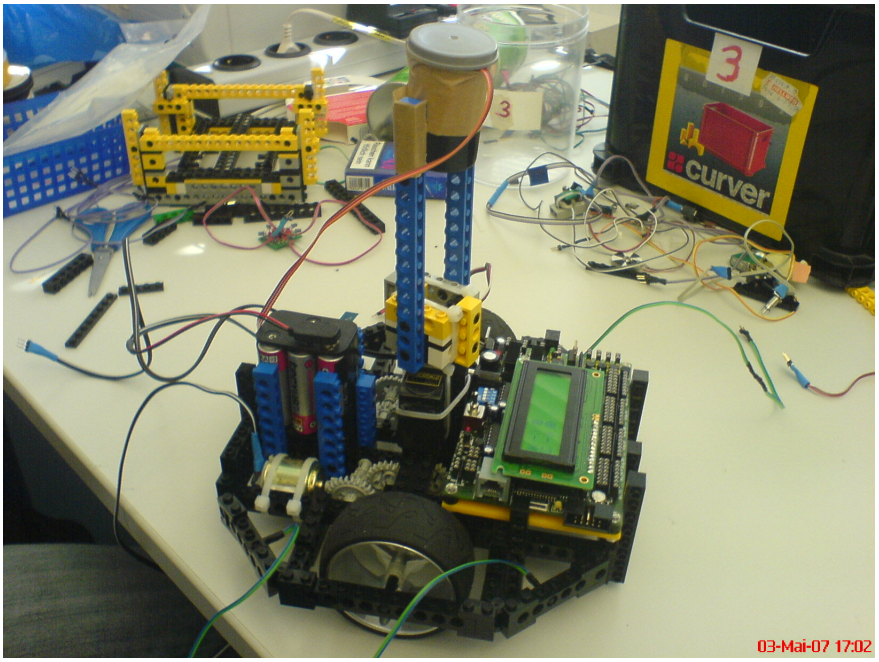

Abbildung 1: Poink Stand: 3. Mai 2007

[1] Die ausführliche Wettbewerbsdefinition ist unter `http://ots.fh-brandenburg.de/downloads/ki_ss07/Hasenjagd.pdf` zu finden.

1.3 Namenswahl

Die Namenswahl fand während des Projektes statt, als das Verzeichnis mit dem Sourcecode immer noch einen Bezeichner ähnlich „blablub“ hatte. Nach einigem überlegen wurde das Verzeichnis in Poink umbenannt. Der Ursprung des Wortes Poink geht auf die amerikanische Fernsehserie Animaniacs zurück. In dieser Serie gibt es die Figuren Pinky und der Brain[2], wobei Pinky neben „Narf“ unter anderem auch „Poink“ als Ausspruch verwendet. Poink ist als Referenz auf „Pinky und der Brain“ zu verstehen, die Gemeinsamkeiten mit der Hasenjagd sind weitreichend und können an dieser Stelle nur angerissen werden. Beispielsweise läuft beides in einem Labor ab, beides sind Ergebnisse eines experimentellen Projekts und bei beidem ist bei näherer Betrachtung unklar wer eigentlich der „Schlaue“ ist. Des Weiteren steht Poink für „Poink ohne irgendeine nennenswerte K.I.“. Diese rekursive Abkürzung beruht auf der scheinbar immer in der Zukunft liegenden Definition von K.I. bzw. Robotik das alles was bereits erreicht wurde nicht mehr als K.I.[3] bzw. Robotik[4] von der Allgemeinheit als solches erkannt wird. Leicht überspitzt könnte man also sagen das alles bereits existierende keine nennenswerte K.I. besitzt.

[2] 2 Labormäuse die jeden Abend versuchen die Weltherrschaft an sich zu reißen.
[3] prüfen von Schläuchen, automatisch öffnende Türen, Eliza
[4] Industrieroboter, Drohnen

2 Implementation

2.1 benutzte Hardware

- 1 AKSEN-Board[5][6]
- 8 Infrarot LED
- 8 Fotodioden
- 3 IR-Fernsteuerungsempfänger
- 1 Beacon
- 2 Elektromotoren
- 1 Servomotor
- 2 Stahlstangen als Stoßstangen
- 2 to N Tape
- 2 to N Kabelbinder
- 2 to N LEGO-Steine

[5] Beschreibungsseite vom ots: `http://ots.fh-brandenburg.de/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=20`

[6] Blog zum Board: `http://ots.fh-brandenburg.de/aksenblog/`

2.2 Besonderes

2.2.1 Geradeaus fahren

Mit diesem Feature begannen wir das Projekt. Das Problem mit dem geradeaus fahren ist, dass es Aufgrund von Ungenauigkeiten in der Konstruktion von Motor und Getriebe immer zu einer Abweichung kommt - der Roboter also eine Tendenz nach Links oder Rechts aufweist. Die Idee war nun die Motoren entsprechend zu regulieren um (genauer) geradeaus fahren zu können. Der Ansatz war nun die Umdrehungen an den Radachsen zu messen und entsprechend die Motoren langsamer oder schneller drehen zu lassen. Das Messen sollte mit einer kleinen „Schwarz/Weiß-Tortenstücken“-Scheibe und einer LED-Sensor-Kombination erfolgen. In der Praxis gibt es mit diesem Ansatz einige Probleme:

- bei einem Übergang von schwarz nach weiß wurden durch zittern im Motor mehrere Impulse gezählt
- die Anzahl der nötigen Abfragen war zu hoch um ausreichende Regelwirkung zu erzielen

Letztendlich wurde dieses Feature komplett gestrichen weil der Implementierungsaufwand in keinem Verhältnis zum praktischen Nutzen steht.

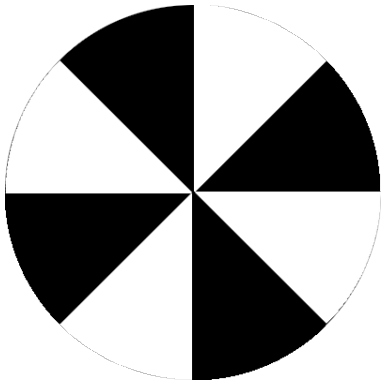

Abbildung 2: Schwarz-Weiß-Tortenstück-Scheibe

2.2.2 Beacon-Sensor-Kopf

Poink besitzt 3 IR-Sensoren, die am beweglichen Kopf montiert sind. Wenn nun ein Signal in einem der 3 Sensoren empfangen wurde, kann an Hand des Winkels den der Servo einnimmt die Position des Gegners bestimmt werden. Dabei wird der Sensor in der "Dose" (siehe Bild) mit einer höheren Wichtung im Programm verarbeitet als die Sensoren am Rand des Kopfes, dies ist nötig wenn der Beacon des Gegners mit mehreren Empfängern erkannt wird. Wenn nun der Gegner mit dem mittleren Sensor detektiert wurde, wird die Schrittweite des Servo-Motors von 3° auf 7° erhöht. Dies ist nötig um die Anzahl der Schwenks zu erhöhen. Wenn nun der Anschlag erreicht war wurde die Schrittweite wieder zurückgesetzt.

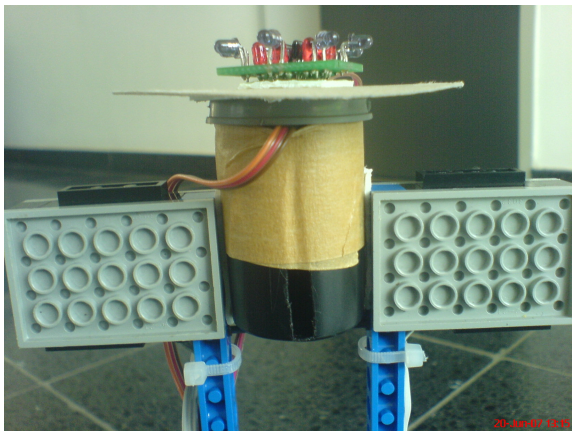

Abbildung 3: Kopf des Roboters

Listing 1: Änderung der Drehgeschwindigkeit

```
1  void MyServo(void)
2  {
3  [..]
4  if(IR0 == 1)
5  {
6          w_step =7;
7  }
8
9  if(winkel<=0)
10 {
11         richtung=0;
12         w_step = 3;
```

```
13 }
14
15 if(winkel>=90)
16 {
17         w_step = 3;
18 }
19 [..]
20 }
```

Der oben stehende Quellcode ist für folgendes gedacht. In der ersten if-Bedingung wird überprüft ob ein Gegner mit dem mittleren Sensor erkannt wird, dann wird die Schrittweite erhöht. Die beiden anderen Bedingungen sind für das zurück setzten der Schrittweite.

2.2.3 Vorranglogik-Beacon-Wand

Wenn sich der Roboter im „Fuchsmodus“ befindet und einen Gegner detektiert wird, wird die Wand-Detektion für die Dauer von 500 ms ausgeschaltet. Dies dient dazu den Gegner nicht als Wand zu detektieren. Es kann sonst passieren das die IR LED des Gegners in unseren IR Empfänger strahlt und des wegen der Roboter ausweicht an statt weiter in seine Richtung zu fahren.

Listing 2: Vorrang Beacon

```
1 void IR_PROC(void)
2 {
3         if(mod_ir0_status()>= IR_PROC_PERIODS_FOR_OK){IR0=1;}
4         if(mod_ir1_status()>= IR_PROC_PERIODS_FOR_OK){IR1=1;}
5         if(mod_ir2_status()>= IR_PROC_PERIODS_FOR_OK){IR2=1;}
6         if(mod_ir3_status()>= IR_PROC_PERIODS_FOR_OK){IR3=1;}
7
8         if(IR0==1)
9         {
10                WinkelGegner = winkel - 45;
11                IR0=0;
12                WinkelNeu = 1;
13                IgnoreWand = akt_time() + 500;
14        }
15 [..]
16 }
```

In diesem Teil des Quellcodes werden erst die einzelnen Empfänger am Kopf des Roboters abgefragt. Wenn der Sensor in der Mitte ein Signal zurück gibt, wird der Winkel des Gegners errechnet und die Wand-Detektion für die Dauer von 500 ms abgeschaltet.

2.2.4 Wand-Sensoren

Poink hat 8 Foto- und Leucht-Dioden als Wand-Sensoren, diese sind auf einem Außenring angeordnet(siehe Figur Wand-Sensoren-Skizze). Die Nummerierung der Sensoren entspricht dem jeweils benutzen Analog-Eingang am AKSEN-Board mit Ausnahme des 7. Sensors der an Analog8 angeschlossen ist, da Analog7 auf dem benutzten AKSEN-Board defekt ist. Der Außenring verringert durch seine rundliche Form außerdem ein Festsetzen an den im Wettkampf benutzten Hindernissen.

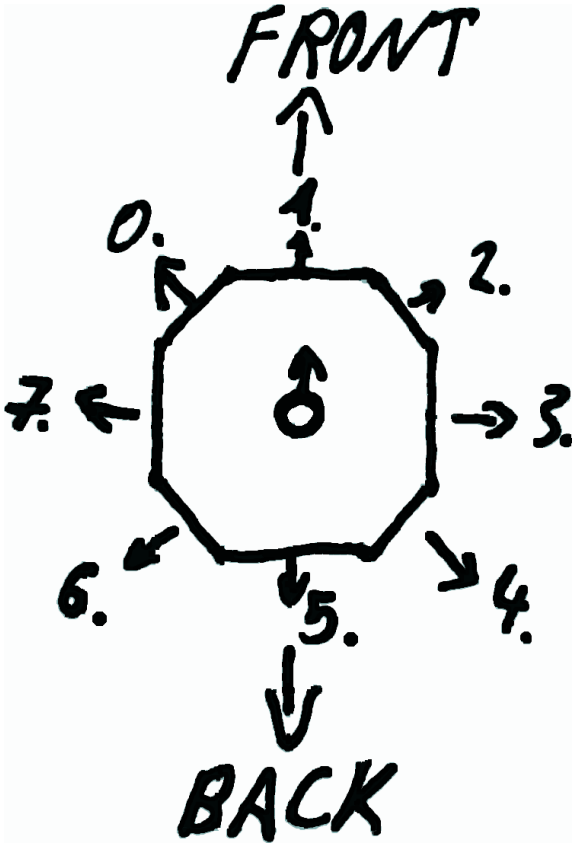

Abbildung 4: Wand-Sensoren-Skizze

2.2.5 Erhebung und Aufarbeitung der Wand-Sensorwerte

Ein Problem bei der Wand-Erkennung per „Lichtschranke“ ist, dass nicht erkannt werden kann von welcher Quelle das Licht emittiert wurde. Zumindest im Falle der Umgebungshelligkeit kann per Differenz-Verfahren[7] ausgeglichen werden. Wenn die Umgebungshelligkeit zu stark ist kann auch mit dem Differenz-Verfahren nicht ausgeglichen werden, in diesem Falle beträgt der erhobene Wert null. Die Realisierung wird anhand des 0. Wand-Sensors demonstriert.

Listing 3: 0. Wand-Sensor auslesen mit Vektorberechnung

```
1  void FindeWand(void)
2  {
3          int Umgebungswert;
4
5          //von allen IR-Sensoren Differenzwert ermitteln
6          WandX = WandY = 0;
7
8          //Umgebungshelligkeit messen
9          Umgebungswert = analog(0);
10         led(0,1);          //die ersten 4 led's ansteuern
11         led(1,1);          //die letzten 4 led's ansteuern
12         WandX += -1 * (Umgebungswert - analog(0)) * 0.7071067812;
13         WandY += 1 * (Umgebungswert - analog(0)) * 0.7071067812;
14         led(0,0);
15         led(1,0);
16 [..]
17 }
```

Zuerst wird die Umgebungshelligkeit gemessen und danach der eigentlich Wert mit eingeschalteten LEDs vom Umgebungswert subtrahiert. Das Ergebnis ist ein von der Umgebungshelligkeit bereinigter Wert. In den Zeilen 12 und 13 werden die Werte auch gleich als Vektor interpretiert. Das Ergebnis in den Variablen WandX und WandY ist ein Vektor der in Richtung der Wand zeigt sofern eine solche vorhanden ist. Dem Entsprechend wird dann eine Links bzw. Rechts-Kurve gefahren.

[7] beschrieben im Kapitel 3.2.4 LED-Ausgänge des AKSEN-Handbuches

2.2.6 Multi- kontra „Single“-Prozessimplementation

Da das Aksen-Board Multitasking, welches mit Zeitscheiben realisiert wird, unterstützt, war Anfangs auch geplant dieses Feature zu nutzen. Motivation hierfür war, die einzelnen Funktionsbereiche jeweils in einen Prozess zu packen. Dadurch erhofften wir uns eine funktionale Trennung und somit eine bessere Unabhängigkeit der einzelnen Programmmodule. Dabei sollten die Funktionsbereiche: Servo drehen, Wanderkennung, IR-Empfänger abfragen und Reaktionslogik jeweils einen eigenen Prozess bekommen. Bei der Umsetzung traten allerdings viele unerwünschte Effekte auf die nicht ohne Weiteres zu beheben waren. Mit Schuld daran war die notwendige Abfrage der IR-Sensoren. Bei deren Abfrage war zu beachten, das zu dieser Zeit kein Prozesswechsel stattfindet. Die meisten durch das Multitasking verursachten Probleme konnten beseitigt werden. Allerdings wurde dadurch das gesamte Laufzeitverhalten beeinträchtigt. Letztendlich wurde entschieden auf Multitasking komplett zu verzichten und zu einer Single-Prozess Lösung überzugehen. Dieser Schritt hat sich auch sehr bezahlt gemacht. Die Einzelnen Funktionsbereiche liefen ohne Probleme, das Laufzeitverhalten hat sich stark verbessert und die Programmabstürze waren verschwunden.

3 nice to have Features

In diesem Bereich sind Ideen aufgeführt die aus Gründen nicht mehr realisiert wurden.

3.1 Situationsabhängiges einschränken des Kopf-Drehbereiches

Die Idee ist, dass der Kopf mit den Beacon-Sensoren bei einer Erkennung des Gegners nicht mehr über den kompletten Bereich dreht. Stattdessen sollte der Dreh-Bereich auf ein paar Grad links und rechts vom erkannten Winkel benutzt werden. Dies ist aber nicht trivial umsetzbar, da sich beide Roboter bewegen müsste die Bewegung mit ein berechnet werden.

3.2 Hasen-Stop-Lauschenmodus

In Test-Verfolgungen hat sich gezeigt, dass Poink im Hasenmodus durch seine Schnelligkeit sehr nah an den Fuchs heran fährt. Um dies teilweise zu vermeiden gab es den Vorschlag, dass der Hase sich nicht vom Fleck bewegt solange er den Fuchs nicht „sieht“. Der Hase würde sich also nur noch drehen um nach dem Fuchs zu suchen und dann in eine andere Richtung davon zu fahren. Leider konnten wir dieses Verhalten aus zeitlichen Gründen nicht mehr implementieren, dies führte dazu das Poink in seiner ersten Hasenrunde dem Fuchs direkt rein gefahren ist.

Abbildungsverzeichnis

Listings