Sind kooperierende Roboter effizienter?

Jugend-forscht Arbeit 2010

Renke Schulte, David Ziegler und Tobias Sturm

Bitte beachten Sie die beiliegende DVD

Inhalt

Kurzfassung

Wir haben uns mit der Frage befasst, ob mehrere Roboter besser arbeiten als ein einzelner Roboter. Anhand einer selbst erdachten Aufgabenstellung für die Roboter haben wir den Erfolg des Schwarms mit dem des Einzelnen verglichen. Unsere Schwarmteilnehmer sind Asuro-Roboter (konstruiert vom Deutschen Zentrum für Luft- und Raumfahrt), die wir für unsere Aufgabe modifiziert haben. Parallel dazu haben wir eine Simulation entwickelt, mit der wir größere Roboterschwärme automatisiert untersuchen konnten.

Nach mehreren Versuchsreihen und Effektivitätsmessungen sind wir zu dem Ergebnis gekommen, dass unser Schwarm in absoluten Zahlen mehr Leistung erbringen kann, aber ein Roboter im Schwarm durchschnittlich nur 48% der Leistung eines Einzelnen erbringt.

1) Einleitung

Es gibt zur Zeit mehrere Forschungsprojekte die sich mit der Frage beschäftigen, wie Roboter zusammen arbeiten können. Dabei gibt es Ansätze, die Schwarmverhalten aus der Natur zum Vorbild nehmen.

In unserem zweijährigen Jugend-forscht-Projekt beschäftigen wir uns mit einem selbst gebauten Roboterschwarm. Wir wollten untersuchen, ob ein Roboter im Schwarm effizienter eine Aufgabe lösen kann als ein einzelner Roboter.

Dazu erdachten wir eine Aufgabe, die sowohl von einem einzelnen Roboter, als auch von einem Schwarm bewältigt werden konnte. Anschließend entwickelten wir eine Methode um die Effizienz (→ siehe Glossar) der Roboter bei der Lösung dieser Aufgabe zu messen.

Die Aufgabe bestand darin, auf einem Testgelände verteilte Flaschen zu einem nicht festgelegten Punkt auf einen Haufen zusammen zu schieben. Dabei sollten die Roboter untereinander nicht kommunizieren und möglichst einfach gebaut und programmiert sein.

Die Roboter fahren zufällig auf dem Testgelände herum und messen die Größe jedes Haufens auf den sie treffen. Bewertet ein Roboter einen Haufen als klein, trägt er diesen ab, bewertet der Roboter einen Haufen als groß, stellt er weitere Flaschen hinzu. Diesen Lösungsansatz haben wir uns in Anlehnung an die Art, wie die Termitengattung macrotermes Erdklümpchen zum Nestbau bildet, ausgedacht. Auch die Termiten verständigen sich darüber, an denselben Haufen zu arbeiten, nur indirekt über die Haufengröße. Die Termiten erkennen die Größe eines Haufens anhand der Pheromonkonzentration, während unsere Roboter die Größe des Haufens mithilfe eines Abstandssensors messen.

Bei den Robotern handelt es sich um Bausätze, die wir noch erweitern mussten. Die Lösung der Aufgabe haben wir den Robotern durch eigene Programmierung angeeignet. Parallel dazu haben wir eine Simulation entwickelt, mit der wir größere Roboterschwärme automatisiert untersuchen konnten.

In dieser Arbeit beschreiben wir die Konstruktion, Erweiterung und Programmierung der Roboter, sowie deren Simulation. Anschließend beschreiben wir die Methode zur Bewertung der Effizienz bei der Lösung der Aufgabe und beantworten die Frage, ob ein Roboter im Schwarm effizienter ist als alleine.

2) Material und Vorgehensweise

2.1) Hardware

Uns geht es in unserer Arbeit nicht darum, selbst einen Schwarmroboter zu entwickeln, sondern vor allem darum, mit vorhandener Technik einen Roboterschwarm zu untersuchen.

Wie oben beschrieben, sollen die einzelnen Roboter sehr einfach sein. Sie müssen also nur fahren, Objekte erkennen und verschieben können. Deshalb haben wir uns für den Asuro-Roboterbausatz des *Deutschen Luft- und Raumfahrtzentrums* entschieden, da er klein, einfach und nicht auf bestimmte Tätigkeiten spezialisiert ist. Des Weiteren ist er leicht zu bauen und zu programmieren (in C), was uns sehr entgegenkam, da wir zu Beginn des Projekts über keinerlei Robotikvorkenntnisse verfügten. Außerdem ist der Asuro gut modifizierbar. Diese Eigenschaft haben wir sehr zu schätzen gelernt, da wir nicht von Anfang an wussten, wie die Hardware letzten Endes aussehen würde und wir unser Konzept mehrmals über den Haufen werfen und uns neu orientieren mussten. Beispiel: Die Haufenerkennung und -vermessung sollte anfangs per Ultraschallsensor realisiert werden, was sich jedoch als nicht machbar erwies, da diese Technik, wenn nur ein Sensor eingesetzt wird, nur den Abstand zum nächsten Objekt erkennt, nicht aber, wo genau sich dieses befindet.

2.1.1) Testgelände und Objekttransport

Die Experimente werden auf einem achteckigen (Seitenlänge ca. 80cm), mit einer Bande begrenzten Testareal durchgeführt. Die Objekte, die verschoben werden sollen, werden aufgrund der passenden Ausmaße durch Trinkjoghurt-Flaschen dargestellt, welche durch angeklebte Pappböden gegen Umfallen geschützt werden.

Um die Flaschen zu verschieben haben wir an der Vorderseite der Roboter zwei Holme so montiert, dass die Flasche auch bei einer Drehung nicht verloren wird, aber durch gerade vowärts/rückwärts fahren aufgenommen bzw. abgestellt werden kann. Die Holme sind höher als die Bande, sodass bei einer Kollision die Taster (→siehe Kapitel 2.1.2) gedrückt werden können. (→Siehe Anhang "Asuro von Oben und von der Seite mit Flasche")

2.1.2) Sensorik

Der Asuro hat zwei einzeln ansteuerbare Hinterräder und liegt vorne auf einem Tischtennisball auf.

Um nicht ohne Reaktion gegen die Begrenzung unserer Versuchsfläche oder einen anderen Asuro zu fahren, braucht der Roboter eine Kollisionserkennung. Diese wird durch sechs Kollisionstaster (AN/AUS-Tasten) ermöglicht, die an der Vorderseite des Asuros in verschiedenen Richtungen angeordnet sind, sodass er auf möglichst jede Kollision reagieren kann. Die Sensoren werden nicht ausgelöst, wenn der Asuro eine Flasche aufnimmt oder sie schiebt. Fährt der Roboter gegen die Versuchsbegrenzung während er eine Flasche vor sich herschiebt, drückt die Flasche auf einen der Kollisionstaster. Es kann vorkommen, dass zwei Asuros zusammenstoßen (→ siehe Kapitel 2.2.7.1) In diesem Fall werden die Tastsensoren nicht zwangsläufig gedrückt und die Asuros können sich verhaken. (Wie wir damit umgehen →siehe Kapitel 2.4.2).

Der Roboter muss seine eigenen Bewegungen erfassen und kontrollieren können. Hierzu verfügt er über ein Odometrie-System, das die Drehungen der Räder misst: es besteht pro Rad aus einer IR-Diode, einem Fototransistor und einem schwarz-weiß-gestreiften Aufkleber auf dem Antriebszahnrad. Die Messung erfolgt für beide Räder unabhängig, wodurch wir den Drehwinkel des Asuros bestimmen können. Die Dioden beleuchten die schwarz-weißen Zahnräder. Das bei „weiß" zurückgeworfene Licht schließt den Stromkreis, in dem sich die Fototransistoren befinden. Die Schwarz-Weiß-Übergänge werden gezählt, woraus sich die zurückgelegte Strecke des Rades ergibt.

Diese Sensoren waren bereits im Asuro-Bausatz enthalten. Aber die wichtigste Eigenschaft zu Lösung unserer Aufgabe fehlte noch: eine optische Erkennung von Objekten.

Um Haufen erkennen und vermessen zu können haben wir die Roboter mit Infrarot-Abstandssensoren von „Sharp" ausgestattet. Diese senden einen punktförmigen Lichtstrahl aus, dessen Streulicht von einem Objekt zurückgeworfen wird und durch eine Linse auf einen „Position Sensitive Detector" projiziert wird, der je nach Position des Lichtpunkts einen anderen Strom ausgibt. So kann der Winkel gemessen werden, mit dem das Licht in die Linse einfällt. Da dieser

Winkel und der Abstand von IR-LED zu Empfänger-Linse gegeben sind, kann der Roboter den Abstand zum Objekt errechnen (→ siehe Kapitel 2.2.5).

Die IR-Sensoren haben wir so hoch montiert, dass, zum einen die Bande (Begrenzung des Testfeldes) nicht durch den IR-Sensor, jedoch durch die Kollisionstaster erfasst wird und eine Unterscheidung durch die Software zwischen Flaschen und Bande überflüssig bleibt und zum anderen die Wahrscheinlichkeit, dass andere Roboter gemessen werden minimiert wird.

2.2) Programmierung der Roboter

Wir haben die Roboter in der Programmiersprache C programmiert. Auf jedem Roboter wird derselbe Quelltext ausgeführt.

2.2.1) Ablauf

Es ist nicht vorbestimmt in welcher Reihenfolge die Flaschen verschoben werden und welchen Weg die Roboter dabei benutzen. Jeder Roboter fährt zufällig umher und trifft auf verschiedene Haufen. Die Regeln, die ein Roboter befolgt, wenn er auf einen Haufen trifft, haben wir uns ausgedacht und programmiert:

Zuerst misst der Roboter die Größe des Haufens aus und prüft, ob er größer ist als der größte Haufen, den er jemals gemessen hat. Je nachdem, ob der Roboter im Moment eine Flasche transportiert entscheidet er sich für Aufnehmen einer Flasche, Abstellen oder keine Aktion.

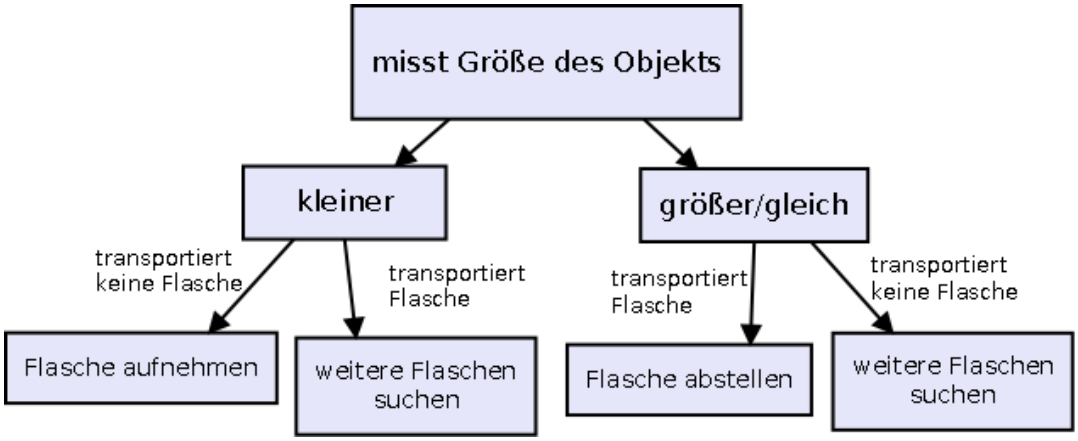


Abb. 1: Handlungsregeln jedes Roboters nach Vermesung eines gefundenen Haufens

Diese Regeln sollen dazu führen, dass Flaschen immer wieder von kleinen Haufen zu größeren Haufen transportiert werden. Die Roboter speichern nicht die Position des größten ihnen bekannten Haufens, sondern dessen Größe. Sie müssen danach wieder so lange weitere Haufen vermessen, bis sie diesen oder einen noch Größeren gefunden haben. Ein genaues Flusslaufdiagramm befindet sich im → Anhang „Flusslaufdiagramm des Asuroprogramms“.

Es ist also nicht nötig, dass die Roboter komplex sind: Sie müssen nicht mit den anderen kommunizieren können (z.B. über Bluetooth), sich nicht orten können, nicht von einer zentralen Oberaufsicht gesteuert werden und auch keine Karte von der Umgebung machen, um zu navigieren oder um sich Positionen von Objekten zu merken. Das Einzige, was jeder Roboter macht, ist auf einen Reiz mit einem bestimmten Verhalten zu reagieren (wie Termiten, vgl. „Einleitung“)

2.2.2) Wie genau wird nach Haufen gesucht?

Der Roboter fährt geradeaus und ruft ständig Messwerte des Infrarot-Abstandssensor und der Tastsensoren ab. Wenn der gemessene Abstand zum nächsten Objekt einen bestimmten Schwellenwert unterschreitet, dann wurde ein „Haufen gefunden“. (Die Variable für den Schwellenwert haben wir „objdist“ genannt, ihr Wert entspricht ungefähr 15cm). Der Roboter hält an und misst die Größe des Haufens aus. Genaueres zur Vermessungsprozedur → siehe Kapitel 2.2.3.

Der Variable, die der Roboter benutzt, um sich die Größe des größten bekannten Haufens zu merken, wird ganz am Anfang der Wert 1 zugewiesen, sodass jeder als nächstes gemessene Haufen als größer erkannt wird. Wenn der Roboter während des Versuchs auf verschiedene Haufen trifft, entscheidet er jedes Mal nach den oben beschriebenen Regeln, ob eine Flasche aufgenommen, abgestellt oder nichts von beidem ausgeführt werden soll.

Aufnehmen einer Flasche: Nach der Vermessungsprozedur ist der Roboter zum rechten Rand des Haufens gedreht. Er dreht sich nach links, bis der Abstandssensor den Haufen wieder misst. Dann fährt er etwa 15 cm nach vorne, sodass sich eine Flasche in der Schiebevorrichtung befindet. Damit dreht er sich nach rechts vom Haufen weg und fährt weiter.

Abstellen einer Flasche: Auch hier ist die Ausgangsposition der rechte Rand des Haufens. Der Roboter dreht sich nach links, bis der Abstandssensor den Haufen wieder misst. Dann fährt er solange nach vorne, bis mit dem Abstandssensor genau der Abstand zum Haufen gemessen wird, der zwischen dem Sensor und der Spitze der Schiebevorrichtung besteht. Wenn das der Fall ist, und der Haufen nicht verfehlt wurde steht die Flasche beim Haufen. Dann fährt der Roboter etwa 15 cm gerade nach hinten, sodass die Flasche nicht von der Schiebevorrichtung mitgerissen wird. Er dreht sich vom Haufen weg und fährt weiter.

Wenn das gefundene Objekt jedoch nicht kleiner sondern größer als das größte bekannte Objekt ist, wird dieses als neues größtes bekanntes Objekt definiert: seine Größe wird gespeichert. Wenn der Roboter zu diesem Zeitpunkt schon eine Flasche transportiert, stellt er sie zu dem Haufen. Wenn er gerade nichts transportiert sucht er weiter nach anderen Haufen.

2.2.3) Wie werden die Haufen ausgemessen?

Der Roboter dreht sich so lange nach links, bis der Abstandssensor die linke Kante des Objekts misst (Abstandssensor misst schlagartig einen größeren Abstand). Anschließend wird die Odometrie-Messung (Anzahl Drehungen jedes Rades) gestartet und er dreht sich nach rechts, bis die rechte Kante des Objekts erreicht ist. Da ein Haufen aber keine glatte Oberfläche hat, sondern aus mehr oder weniger nah zusammenstehenden Flaschen besteht, müssen kleinere Lücken ignoriert werden: Wenn bei der Drehung eine Lücke (oder die vermeintliche Kante des Haufens) gemessen wird, dreht sich der Roboter zunächst einen bestimmten Winkel weiter. Wenn dann nur wenig weiter rechts die Lücke ein Ende nimmt und klein genug ist, ignoriert der Roboter die Lücke. Ist der Abstand zwischen zwei Flaschen zu groß (etwa 15 cm), dann ist das Objekt per Definition kein zusammenhängender Haufen, sondern wird wie mehrere einzelne Objekte behandelt.

Wenn die rechte Kante des Objekts erreicht wurde, werden die Odometrie-Daten abgelesen und aus der Drehung des Rades der Drehwinkel bestimmt. Die Drehung erfolgt nur mit dem linken Rad, sodass der Abstandssensor, der rechts am Asuro montiert ist, nahezu auf der Drehachse (dem rechten Rad) liegt. Je größer der Drehwinkel, desto größer muss der Haufen sein. Die gemessene Größe ist aber auch vom Abstand abhängig.

2.2.3) Messgrößen

Ein Haufen lässt sich stark vereinfacht als Kreis darstellen. Da von allen Seiten Flaschen an einen Haufen gestellt werden können, ist dieses Modell brauchbar. Der **Radius** dieses Kreises ist gesucht. Der Roboter vermisst den Haufen nur von der Stelle aus, an der er den Haufen zum ersten Mal erfasst hat. Auf dieser Stelle dreht er sich, um die Größe zu bestimmen. Also registriert er nicht die komplette Breite des Haufens. Stattdessen ist die Stelle, die vom Sichtpunkt des Roboters aus der Rand des Objektes ist, nur eine Tangente an dem Kreis, der den Haufen darstellt. Um den **Abstand** zu dem Haufen zu bestimmen, wird

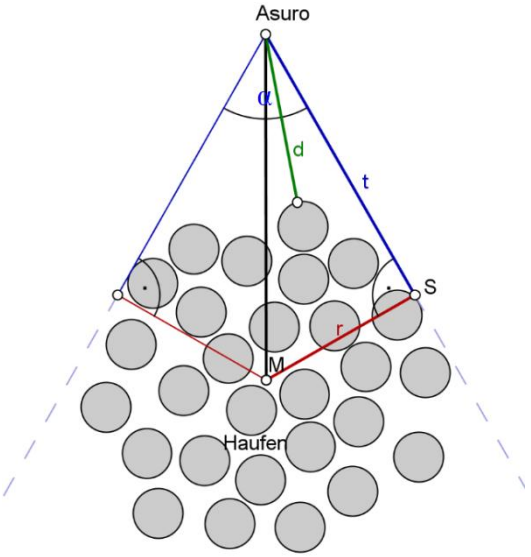


Abb. 2: Vermessung eines Flaschen-Haufens.
*M = Mittelpunkt des Umkreises; t = Tangenten am Rand des Haufens durch die Position des Roboters; d = kleinster gemessener Abstand; α = Drehwinkel zwischen den gemessenen Rändern; 2 * r = zu bestimmende Größe des Haufens*

während der gesamten Drehung vom linken zum rechten Objekt-Rand in kurzen Zeitabständen der momentane Abstand gemessen (siehe Abbildung 1, Größe d). Der kleinste Abstand wird bei einem Kreis genau in der Mitte der Drehung gemessen (siehe Abbildung 2, Größe d). Bei einem realen Haufen kann der kleinste Abstand auch an einer anderen Stelle gemessen werden (Abb. 1), doch ist dieser Abstandswert dem eines Kreises sehr ähnlich. An dieser Stelle geht es ja nicht um den Winkel.

Wenn ein Haufen nicht wie oben angenommen einem Kreis ähnlich ist, werden der Abstand und der Winkel falsch gemessen. Ein länglicher Haufen wird von der einen Seite größer gemessen, als von der anderen Seite.

2.2.4) Herleitung der Vermessungsformel

Gegeben sind der minimale Abstand d und der Winkel α zwischen den beiden Tangenten des Kreises. Wir haben eine Formel hergeleitet, die den Radius des Haufens in Abhängigkeit von d und α setzt:

Wegen des rechtwinkligen Dreiecks (siehe Abb. 1) zwischen Kreismittelpunkt M, Tangentenberührpunkt S und der Position des Asuros gilt: $\sin\frac{\alpha}{2} = \frac{r}{r+d}$; $r > 0; 0 < \propto < 180°; d > 0$

Termumformung ergibt: $\frac{1}{\sin\frac{\alpha}{2}} = \frac{r+d}{r}$ woraus folgt: $\frac{1}{\sin\frac{\alpha}{2}} - 1 = \frac{d}{r}$

Somit erhält man: $r = \frac{d}{\frac{1}{\sin\frac{\alpha}{2}} - 1} = \frac{\sin\frac{\propto}{2} * d}{1 - \sin\frac{\propto}{2}}$

Die so ermittelten Werte der Radien r verschiedener Haufen werden verglichen.

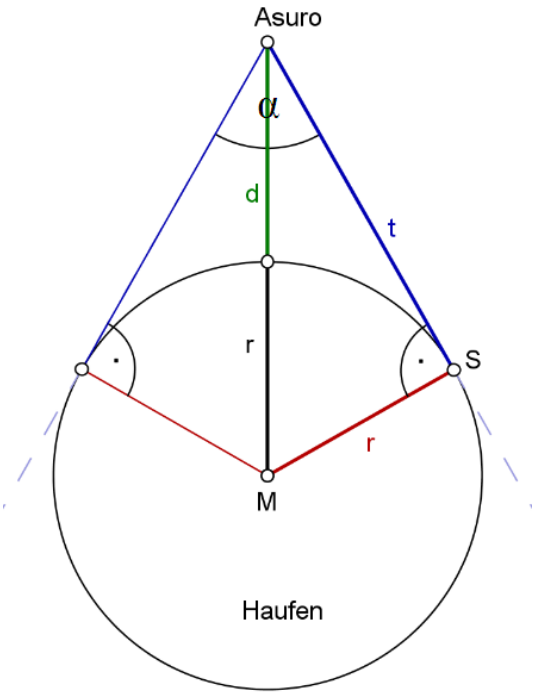


Abb. 3: Der Kreis entspricht einem vereinfachten Haufen; t = Tangenten durch die Position des Asuros; d = minimaler Abstand;

2.2.5) Schwarmfähigkeit der Programmierung

Es spielt für einen Roboter keine Rolle, ob ein Haufen von ihm selbst oder von anderen angelegt wurde, weil keine absoluten Positionen der Haufen gespeichert oder vorgegeben werden, sondern die Haufen immer wieder ausgemessen werden. Die Roboter können also theoretisch (→ siehe 3.1.2) gemeinsam an einem Haufen arbeiten oder diesen zumindest nicht zerstören.

2.2.6) Probleme und softwareseitige Lösungen

2.2.6.1) Gegenseitiges Behindern der Roboter

Problem: Ein Roboter kann einen anderen Roboter als Haufen interpretieren.

Lösung: Meistens fahren die Roboter, das macht sie von Haufen unterscheidbar. Wird bei einer Haufenvermessungs-Prozedur erkannt, dass der Haufen, der eben noch vermessen werden sollte, nicht mehr da ist, so kann angenommen werden, dass es sich um einen Roboter handelt. Dann wird die Messung ignoriert und weitergesucht.

2.2.6.2) „Vergessen“ können

Problem: Die Vermessung der Haufen kann Fehler ergeben, z.B. bei länglichen Haufen. Wenn ein Haufen zu klein gemessen wird, macht der Roboter bei diesem Haufen eine falsche Aktion (z.B. aufnehmen) und kann danach weiterarbeiten. Wird jedoch ein Haufen als zu groß gemessen und ist größer als der bisher größte bekannte Haufen, dann speichert der Roboter diese falsche Größe. Hierbei kann es leicht passieren, dass der Roboter während des restlichen Versuchs keinen Haufen dieser (viel zu großen) Größe mehr misst und uneffektiv ist.

Lösung: Wenn ein Roboter lange Zeit nur kleinere als der größte bekannte Haufen misst, kann angenommen werden, dass der oben beschriebene Fall eingetreten ist. Das bedeutet, die Größe des größten bekannten Haufens sei falsch und muss korrigiert werden. Misst ein Roboter 5 aufeinanderfolgende kleinere Haufen, so reduziert er bei jedem weiteren gemessenen kleinen Haufen die Größe des *größten bekannten Haufens* um 15%. Sobald wieder ein Haufen gemessen wird, der größer als der *größte bekannte Haufen* ist, wird dieses „Vergessen der Haufengröße“ eingestellt.

2.2.6.3) Transportierte Flasche ertasten

Problem: Trotz Schlingern kann es vorkommen, dass ein Roboter eine Flasche unabsichtlich aufnimmt.

Lösung: Von den 6 Tastsensoren an der Vorderseite des Asuros sind die zwei inneren so nach Innen gerichtet, dass sie nicht bei einer herkömmlichen Kollision gedrückt werden, sondern nur, wenn mit einer Flasche gegen die Bande gefahren wird. Werden auschließlich diese beiden Tastsensoren gedrückt und und die Variable, die speichert, ob eine Flasche geschoben wird, hat den Wert 0 (= nein), so wird diese auf 1 (= ja) gesetzt.

Die beschriebenen Fehler werden durch die Lösungen nicht ausgeschlossen, nur unwahrscheinlicher gemacht. Durch das nicht vordefinierte Schwarmverhalten ist es theoretisch möglich, dass ein Fehler später wieder ausgeglichen wird: Sollte ein Haufen von einem Roboter falsch vermessen werden, ist es bei einem Schwarm möglich, dass ein anderer Roboter diesen Haufen später korrekt vermisst.

2.3) Simulation

2.3.1) Warum wir eine Simulation entwickelt haben

Eine Simulation bietet zwei große Vorteile. Erstens können Versuche mit sehr viel größeren Schwärmen ohne weitere Kosten durchgeführt werden und zweitens können mithilfe einer Simulation Versuchsdurchläufe automatisch durchgeführt und statistisch ausgewertet werden.

2.3.2) Modellierung

Bei der Entwicklung von Simulationen im Allgemeinen muss man die Realität mithilfe von Modellen vereinfachen und auf das Wesentliche reduzieren. Konkret in unserem Fall bedeutet das: In einem zweidimensionalen Koordinatensystem werden Asuros als Vierecke und Flaschen als Kreise dargestellt (siehe Anhang „Wie Roboter und Flaschen in der Simulation dargestellt werden“). Die Roboter können sich nicht verhaken oder Flaschen umwerfen, es handelt sich also um ein Abbild „perfekter“ Asuros, die keine Anfälligkeit der Hardware (wie Verhaken, Durchdrehen, abnehmende Batteriespannung, ...) aufweisen.

Bei der Entwicklung der Simulation haben wir darauf geachtet, alles so modular wie möglich zu halten, um spätere Veränderungen an der Simulation relativ kurzfristig vornehmen zu können.

2.3.3) Struktur der Simulation

Es handelt sich bei unserer Simulation um ein Multi-Agenten-System (siehe Wikipedia.de: „es [handelt] sich um ein System aus mehreren gleichartigen oder unterschiedlich spezialisierten handelnden Einheiten, die kollektiv ein Problem lösen.“), bei dem jeder Asuro durch einen Agenten repräsentiert wird. Wir portieren den Quellcode, der auch von den Asuros ausgeführt wird, in die Simulation (einige syntaktische bzw. programmiersprachenbedingte Änderungen müssen von Hand vorgenommen werden). Die Simulationsumgebung erstellt für jeden simulierten Asuro (im Folgenden Agent genannt) einen Prozess (Thread), welcher den selben Quellcode, der sich auch auf den „realen Asuros“ befindet, ausführt, mit dem Unterschied, dass durch Befehle wie *setMotorPower* keine Motoren in Bewegung gesetzt werden, sondern lediglich eine Zustandsänderung im Agenten bewirkt wird.

Wir haben uns für die starke Trennung zwischen simuliertem (Quellcode, der auch auf den Asuros ausgeführt wird; im Folgenden Asurosoftware genannt) und simulierendem Quellcode (Die Simulationssoftware an sich) entschieden, um nicht aus Versehen verfälschende Ergebnisse zu erhalten: Ohne eine strikte Trennung wäre es z.B. möglich, dass ein Agent mehr Input erhält als den Zustand seiner Tastsensoren, seiner Odometrie und seines Abstandssensors. Dadurch könnte er effizienter arbeiten, als die realen Asuros und der Vergleich eines Roboterschwarmes mit einem einzelnen Roboter könnte verfälscht werden. Für eine Übersicht über Klassen und Packages in der Simulation → siehe Anhang „Grobe Übersicht über die Klassen/Packages der Simulation".

2.3.4) Asurosoftware ←→Asurohardware

Asurosoftware ist eine Klasse unserer Simulationssoftware, die den Quellcode der „realen Asuros" beinhaltet. Die Klasse hat Zugriff auf denselben Input, den die Asurosoftware auch in der Realität hat: Tastsensoren, Odometrie und Abstandssensor. Diesen Input erhält sie durch die Klasse Asurohardware. Sie stellt dieselbe Schnittstelle bereit, wie die reale Hardware eines Asuros (mit denselben Funktions- und Methodennamen). Die Klasse Asurohardware bildet sozusagen die Simulation der Mechanik und bietet somit der Asurosoftware die Möglichkeit, mit der simulierten Realität (andere Agenten und simulierten Flaschen) zu interagieren (→ siehe Anhang „Drei Ebenen der Simulation - Abbildung realer Versuche"). Jeder Agent besteht aus eben diesen beiden Klassen.

Die Koordination zwischen verschiedenen Agent-Instanzen erfolgt über die Klasse World. In ihr ist eine Liste mit Verweisen auf jeden Agenten abgelegt, wodurch Callbacks durchgeführt werden können. Diese Callbacks sind wichtig, um festzustellen ob zwei Agenten miteinander kollidieren.

2.3.5) Kollisionskontrolle

An diesem konkreten Fall soll die Funktionsweise der Simulation erläutert werden.

Agenten besitzen eine rechteckige, Flaschen eine kreisförmige Fläche. Auf dem Rand dieser Flächen befinden sich in regelmäßigen Abständen Punkte. Diese Punkte dienen der Kollisionskontrolle (auch Kollisionskontrollpunkte), welche verhindern soll, dass die Flächen zweier Agenten sich überlappen (→ siehe Anhang „Wie Roboter und Flaschen in der Simulation dargestellt werden").

Die Asurosoftware einer Agenten-Instanz führt den Befehl „`SetMotorPower(127,127)`" aus, was bedeutet, dass beide Motoren mit voller Geschwindigkeit vorwärts fahren sollen. Der Aufruf des Befehls setzt in der dazugehörigen Asurohardware eine Zustandsänderung aus: zwei interne Variabeln (`intMotorRight` und `intMotorLeft`), die die aktuellen Motorengeschwindigkeiten speichern, werden auf die Werte 127;127 gesetzt. Bevor die Bewegung ausgeführt wird, berechnet der Agent, ob sie überhaupt möglich wäre. Der Thread der Asurohardware fragt innerhalb seiner `run()`-Methode die beiden Motor-Variablen ab und errechnet aus ihnen eine Bewegung des Agenten als Vektor. Anschließend werden die Kollisionskontrollpunkte seiner Fläche errechnet, die die reservierte Fläche des Agenten repräsentieren würden, wenn er die Bewegung bereits ausgeführt hätte. Diese Randpunkte werden in einem Array an die `World`-Instanz übergeben. Von dort aus werden Callbacks auf jede simulierte Flasche und auf jeden Agenten (beide implementieren das Interface `ICollideable`) durchgeführt, wobei immer die Randpunkte des Agenten, der die Bewegung ausführen „will", als Parameter übergeben werden. Jeder Agent und jede simulierte Flasche prüft nun, ob mindestens einer dieser Randpunkte mit ihnen kollidiert. Sollte das der Fall sein, geben sie ein „`true`" an die World zurück, welches dann an den auslösenden Asuro weitergegeben wird (→ siehe Anhang „KommunikationsdiagrammfürFortbewegungKollisionskontrolle").

Mit anderen Worten: Hätte der Agent die errechnete Bewegung durchgeführt, hätte Kollision stattgefunden. Da zwei Asuros sich nicht an derselben Stelle befinden können, wird die Bewegung von der Asurohardware nicht ausgeführt.

Sollte aber keiner der Randpunkte mit einer Flasche bzw. einem anderen Agenten kollidieren, wird die Bewegung ausgeführt (die Variable, die die Position des Asuros speichert, wird mit dem Bewegungsvektor addiert).

2.3.6) Output der Simulation

Die Output-Klassen der Simulation sind vollkommen unabhängig vom eigentlichen Simulationsprozess. Wir unterscheiden zwei Arten von Output-Klassen: Zeitgesteuerte (`ITimedrivenOutput`) und Ereignisgesteuerte (`IEventdrivenOutput`).

2.3.6.1) Ereignisgesteuerter Output

Ereignisgesteuerte Output-Klassen können Informationen z.B. auf einer Konsole, in einem Datenstream oder auf einem Bild ausgeben. Auch ist die statistische Auswertung solcher Ereignisse möglich (z.B. eine Output-Klasse die zählt, wie oft Flaschen aufgenommen wurden).

Mithilfe von Verweisen wird eine ereignisgesteuerte Output-Klasse bei Klassen, die das Interface `IEventDrivenGenerator` implementieren (z.B. Agenten und Flaschen), registriert. Diese `IEventDrivenGenerators` können, sobald ein Ereignis eintritt, einen Callback auf die ereignisgesteuerte Output-Klasse durchführen (→siehe Anhang „Prinzip des Outputs – Registrierung / Callback"). Unabhängig vom Output, arbeitet die zugrunde liegende Simulation immer gleich (→siehe Anhang „Prinzip des outputs - SimpleFileOutput„ und „Prinzip des Outputs – Verwendung von Outputklassen untereinander").

2.3.6.2) Zeitgesteuerter Output

Das Prinzip des Callbacks wird auch hier gebraucht. Allerdings registriert sich in diesem Falle ein Objekt bei einer Output-Klasse. Dieses Objekt wird von der Output-Klasse sozusagen „beobachtet". In Zeitabständen, die von Output-Klasse zu Output-Klasse unterschiedlich sind, werden Callbacks ausgeführt um bestimmte Informationen von den registrierten Instanzen zu erhalten, um einen Output zu erzeugen (sei es ein Bild, eine Bildschirmausgabe, eine Konsolenausgabe,...)

Ein Beispiel ist die Klasse `Mapper`. Sie zeichnet mehrmals in der Sekunde alle Kollisionskontrollpunkte auf ein Formular und gibt somit Bilder aus, auf denen man schnell erkennt, was in der Simulation gerade berechnet wird (→ siehe Anhang "Bild einer Mapper-Instanz").

Auch die Effizienz wird von der Simulation mithilfe eines zeitgesteuerten Outputs gemessen. Diese Output-Klasse hat eine Liste mit Verweisen auf alle Flaschen. Alle 5 Sekunden wird die Effizienz berechnet (→siehe Kapitel 2.4.3.3) und in eine HTML-Tabelle (→siehe beiliegende DVD) geschrieben. Zusätzlich wird ein Screenshot der Flaschenumkreise gespeichert.

2.3.7) Versuchsdurchführungen mit der Simulation

Die Software startet nacheinander 50 mal eine Simulation mit exakt denselben Startparametern (Startpositionen der Agenten und Flaschen, Größe der Flaschen,...). Für jede Simulation wird ein Ordner angelegt, in dem der Verlauf der Effektivität in einer HTML-Datei und mehreren Screenshots dokumentiert werden. Zusätzlich werden die Positionen von Flaschen alle 5 Sekunden als CSV-Datei gesichert um spätere Effektivitätsmessungen (u.U. mit anderen Messverfahren) durchzuführen.

Wie bei den realen Versuchen bilden wir aus diesen einzelnen Versuchen einen Durchschnitt und vergleichen anschließend die Effizienz eines Schwarmes im Vergleich zu einem einzelnen Roboter.

2.4) Durchführung der Versuche

2.4.1)Versuchsreihen und Versuchsaufbauten

Um die Effizienz eines einzelnen Roboters im Vergleich zu der eines Roboterschwarmes zu messen, haben wir insgesamt 4 Versuchsreihen durchgeführt. Wir variierten die Anzahl der Roboter und die Startpositionen, auf denen sich die Flaschen zu Beginn der Versuche befanden. Jede Versuchsreihe wurde mit exakt derselben Versuchsanordnung vier mal durchgeführt (um bei der Auswertung Durchschnittswerte zu errechnen). Das ergibt 16 Versuche, wobei jeder je 15 Minuten durchgeführt wurde.

Versuchsreihe 1	Zufällig, reproduzierbar	1 Asuro
Versuchsreihe 2	Raster	1 Asuros
Versuchsreihe 3	Wie bei Versuchsreihe 1	3 Asuro
Versuchsreihe 4	Raster	3 Asuros

Tabelle 1: Versuchsreihen

- Versuchsreihe 1: Die erste Versuchsreihe wurde mit einem Asuro und einer zufälligen Anordnung der Flaschen durchgeführt. Dazu war es notwendig, vor der Versuchsreihe für die Flaschen und den Asuro zufälige Startpositionen zu definieren. Mithilfe eines Zufallsgenerator wählten wir diese Positionen aus und zeichneten sie auf das Testgelände um sie bei jedem Versuch der Testreihe 1 und 2 verwenden zu können.
- Versuchsreihe 2: Bei dieser Versuchreihe wählten wir eine symmetrische Anordnung der Flaschen (= „Raster“). Wie bei Versuchsreihe 1 setzten wir nur einen Asuro ein.
- Versuchsreihe 3: Um die Effektivität des einen Asuros mit einem Asuroschwarm zu vergleichen haben wir den Asuroschwarm vor genau dieselbe Aufgabe gestellt wie in der Versuchsreihe 1. Das heißt: Die Startpositionen der Flaschen sind unverändert. Der einzige Unterschied zur 1. Versuchsreihe liegt darin, dass zwei Asuros dem Versuchsaufbau hinzugefügt wurden.
- Versuchsreihe 4: Selbe Anordnung der Flaschen wie bei Versuchsreihe 2, aber mit drei Asuros.

2.4.2) Versuchsprotokoll

Um den Ablauf der Versuche später analysieren zu können haben wir die Versuche gefilmt, die Handlungen jedes Asuros und Fehler protokolliert und die Anzahl und Größe der Haufen notiert.

Für jeden Asuro haben wir während des Versuchs ein eigenes Protokoll geführt, in dem wir jedes Ereignis und die Zeit, zu dem es eingetreten ist, eingetragen haben.

Ereignisse: Handlungen der Asuros und Fehler:

- Flasche erfolgreich abgestellt: Der Asuro transportiert „absichtlich“ eine Flasche, misst einen Haufen und stellt die Flasche so zum Haufen dazu, dass sie Teil des Haufens wird. Eine Flasche ist dann „falsch abgestellt“ wenn sie nicht zu dem vermessenen Haufen (den der Asuro als vergrößerungswürdig einstuft → siehe Kapitel 2.2.1) dazugestellt wird, sodass dieser vergrößert wird (→ siehe beiliegende DVD – Videos).
- Flasche erfolgreich aufgenommen: Der Asuro transportiert „absichtlich“ keine Flasche, misst einen Haufen und nimmt eine Flasche des Haufens auf. Der Haufen kann dabei zerstört werden, wichtig ist nur, dass eine Flasche getroffen wurde. Eine Flasche wurde „falsch aufgenommen“, wenn sich nach dem Aufnahme-Manöver keine Flasche des zuvor gemessenen Haufens im Greifer befindet.
- Flasche verloren: Die Flasche gerät aus dem Greifer, ohne dass ein Abstell-Manöver durchgeführt wurde(→ siehe beiliegende DVD – Videos).
- Flasche aus Versehen aufgenommen: Eine Flasche gerät in den Greifer ohne dass ein Aufnahme-Manöver durchgeführt wurde.
- Haufen zerstört: Ein Haufen wird durch einen Asuros zerstört, ohne dass ein Aufnahme-Manöver durchgeführt wurde.
- Versehentlich dazugestellt: Ein Asuro verliert eine Flasche (siehe oben) und dabei wird ein Haufen vergrößert.
- Vermessung der Haufen:
    - Falsche Messung: Dreht sich ein Asuros während einer Messung so, dass er dabei eine Flasche permant vor dem Abstandssensor herschiebt, interpretiert er den Haufen als zu groß. Er interpretiert ihn auch als zu groß, wenn die Räder während der Messung durchdrehen. Andere Messfehler werden nicht berücksichtigt.
    - Eine „korrekte Messung“ ist dann erfolgt, wenn die Messung wie im Programmcode vorgesehen verläuft.
- Motoren drehen durch: Die Motoren drehen sich, aber der Asuro bewegt sich nicht fort.
- Ein anderer Asuro wird gemessen: Ein Asuro wird von einem anderen wie ein Haufen vermessen.
- Bleibt hängen: Wenn ein Asuro mindestens 10 Sekunden hardwarebedingt bewegungsunfähig ist und wir eingreifen mussten: kurzes Anstoßen, sodass der Asuro wieder losfahren kann.
- Mit anderem Asuro verhakt: siehe „bleibt hängen“

2.4.3) Auswertung der Messwerte

2.4.3.1) Geometrischer Schwerpunkt

Wir betrachteten nach den Versuchen die Flaschenpositionen zu Beginn und zu Ende des Versuchs und überlegten uns, wie man eine messbare Größe ermitteln kann, die den Erfolg des/der Roboter/s beschreibt. Ziel war es, die Flaschen möglichst zu einem Haufen zusammen zu schieben, allerdings mussten wir festellen, dass dies am Ende eines Versuchs (nach 15 Minuten) nie der Fall war.

Es wurden zwar einzelne Haufen (mit 2 bis 6 Flaschen) gebildet, diese waren aber auch unterschiedlich weit voneinander entfernt.

Wir erechneten den geometrischen Schwerpunkt aller Flaschen und berechneten den durchschnittlichen Abstand aller Flaschen zu diesem Punkt. Allerdings mussten wir dieses Bewertungsverfahren verwerfen, da zwei große, weit vom Schwerpunkt entfernte Haufen schlechter bewertet wurden als viele gleichmäßg verteilte Flaschen → siehe Anhang „mittlere Schwerpunktdistanz am Beispiel von zwei Szenarien“.

2.4.3.2) Effektivitätsformel

Wir haben einen Haufen so definiert, dass die Flaschen die ihn bilden untereinander einen Abstand von 15cm nicht überschreiten dürfen. Also zogen wir gedanklich Umkreise mit 7,5 cm Radius um jede Flasche. Berühren oder überschneiden sich die Umkreise zweier Flaschen bilden sie einen Haufen. Je näher sich mehrere Flaschen stehen, desto mehr Fläche überschneidet sich. Somit ist die gesamte von Umkreisen belegte Fläche kleiner, als ohne Haufen. Je kleiner die belegte Fläche ist, desto dichter stehen die Flaschen und desto näher ist man dem Ziel gekommen, einen Haufen zu bilden. (Daraus folgt: (i) $Effektivität \sim \frac{1}{belegte\ Fläche}$). Die niedrigste Effektivität erreicht man, wenn keine Überschneidungen vorhanden sind. Dann gilt: (ii) $belegte\ Fläche\ =\ Anzahl\ Flaschen\ *\ (7{,}5cm)^{\,2}\ *\ \pi.$

Will man zwei Versuche mit verschiedener Flaschenanzahl vergleichen: Je mehr Flaschen existieren, desto größer ist von Anfang an die belegte Fläche. Bei einer bestimmten belegten Fläche ist eine hohe Flaschenanzahl effektiver zu bewerten als eine geringe Flaschenanzahl. (Daraus folgt: (iii) $Effektivität \sim Anzahl\ Flaschen$)

Aus den beiden Proportionalitäten folgt: (i) & (iii) $Effektivität \sim \frac{Anzahl\ Flaschen}{belegte\ Fläche}$

Existieren keine Überschneidungen (ii), soll die Effektivität 0 betragen:

$$Effektivität\ =\ 0\ =\ \frac{Anzahl\ Flaschen}{Anzahl\ Flaschen * \ (7{,}5cm)^2\ *\ \pi} + c$$

Daraus ergibt sich $c\ =\ -\frac{1}{(7{,}5cm)^2 * \pi}$

Unsere Effektivitätsformel: $Effektivität(t)\ =\ \frac{Anzahl\ Flaschen(t)}{belegte\ Fläche(t)} - \frac{1}{(7{,}5cm)^2\pi}$

2.4.3.3) Näherungsweise Berechnung der Effektivität mithilfe von Bildern

Aus den Videos der Versuche schnitten wir den letzten Frame (Flaschenpositionen am Ende jedes Versuchs) heraus um ihn als Bild abzuspeichern. Mithilfe eines selbstentwickelten Programms markierten wir die Flaschenposition auf diesem Bild. Das Programm prüft für jedes Pixel des Bildes, ob es sich innerhalb oder außerhalb eines Flaschenumkreises befindet. Da ein Pixel einem Quadratmillimeter entspricht (Programm wurde kalibriert), ist die Anzahl aller Pixel, die sich außerhalb der Flaschenumkreise befinden gleich der freien Fläche in Quadratmillimetern.

Da umgefallene Flaschen nicht mehr von Robotern gemessen werden können, werden diese nicht mehr als Bestandteil eines Haufens gewertet. Bei einigen Versuchen fielen Flaschen um, deshalb variiert die Anzahl der Flaschen unter den Versuchen. Dies wird aber in der Effektivitätsformel berücksichtigt.

2.4.3.4) Effizienz

Die Effizienz eines Schwarmes ist seine Effektivität geteilt duch die Anzahl seiner Mitglieder, da Effizienz *ein Maß für das Verhältnis zwischen Nutzen*(Effektivität) *und Aufwand ist* (vgl. Wikipedia.de).

$$Effizienz = \frac{Effektivität}{Anzahl\ Roboter}$$

3) Ergebnisse

3.1) Versuche

3.1.1) Effektivität und Effizienz

Zwei Versuchsprotokolle und Videos der Versuchsreihe 3 gingen wegen Datenverlust verloren.

Bei zufällig angeordneten Startpositionen der Flaschen gibt es bereits beim Start Überschneidungen der Flaschenumkreise und damit ist die $Effektivität > 0$, beim Raster ist die $Effektivität = 0$. Um die verschiedenen Versuchsreihen vergleichen zu können, bildeten wir daher $\Delta Effizienz = Effizienz(t_{Ende}) - Effizienz(t_0)$.

So war z.B. die Effizienz bei zufälliger Flaschenanordnung zu Beginn des Versuchs =0,0000003927514128 und am Ende des Versuchs (Durchlauf 3, Versuchsreihe 1) = 0,00000966013. Daraus ergibt sich für $\Delta Effizienz = 0{,}00000966013 - 0{,}0000003927514128 = 0{,}0000092673785872$.

Da unser Messverfahren für die Effektivität (→siehe Kapitel 2.4.3.3) nur näherungsweise ist, erhalten wir für die Effektivität beim Start des Raster-Versuchs (es existieren keine Überschneidungen) $0{,}08 * 10^{-6}$ anstelle von 0.

Bei allen Versuchen haben wir eine positive Effizienz gemessen. Die Schwankung der Effizienz innerhalb einer Versuchsreihe war bei Versuchen mit einem einzelnen Roboter viel höher als bei Schwarmversuchen.

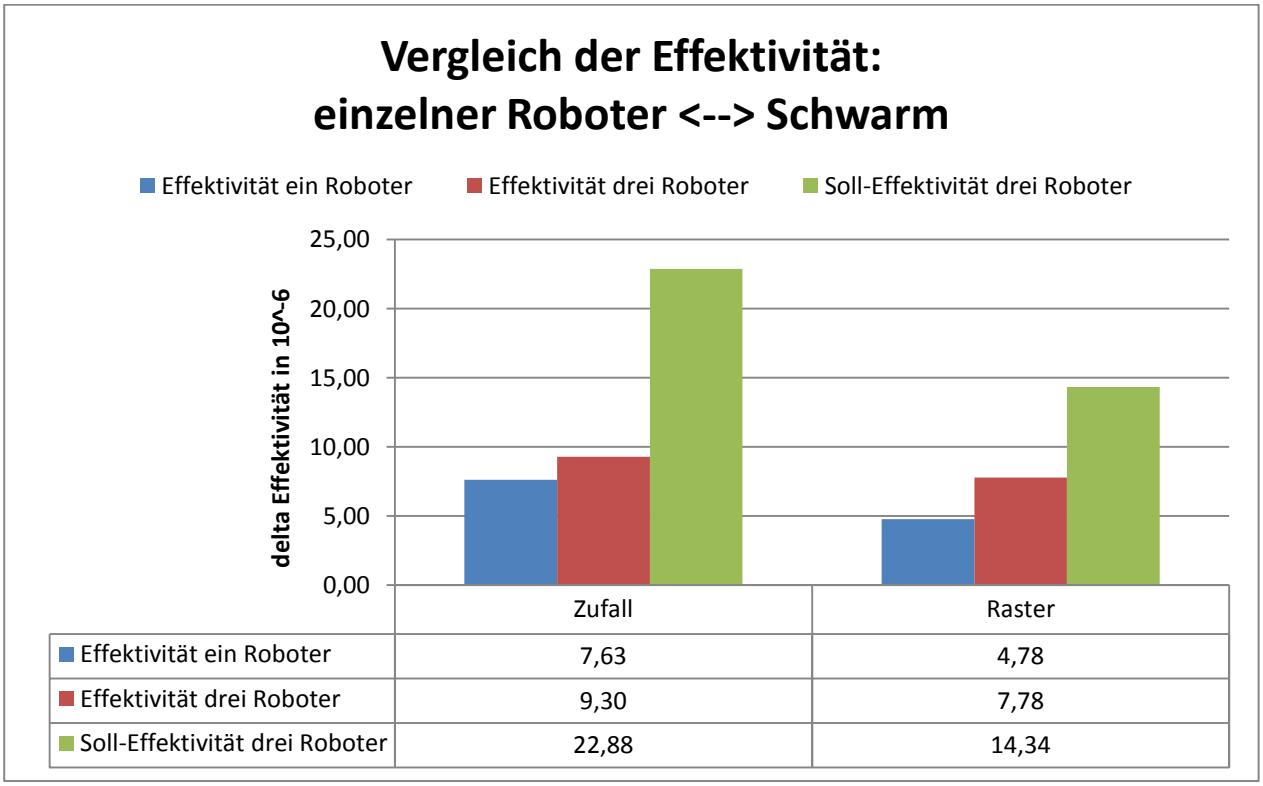


Abb. 4: Vergleich der Effektivität eines einzelnen Roboters im vergleich zu einem Roboterschwarm und dessen Soll-Effektivität

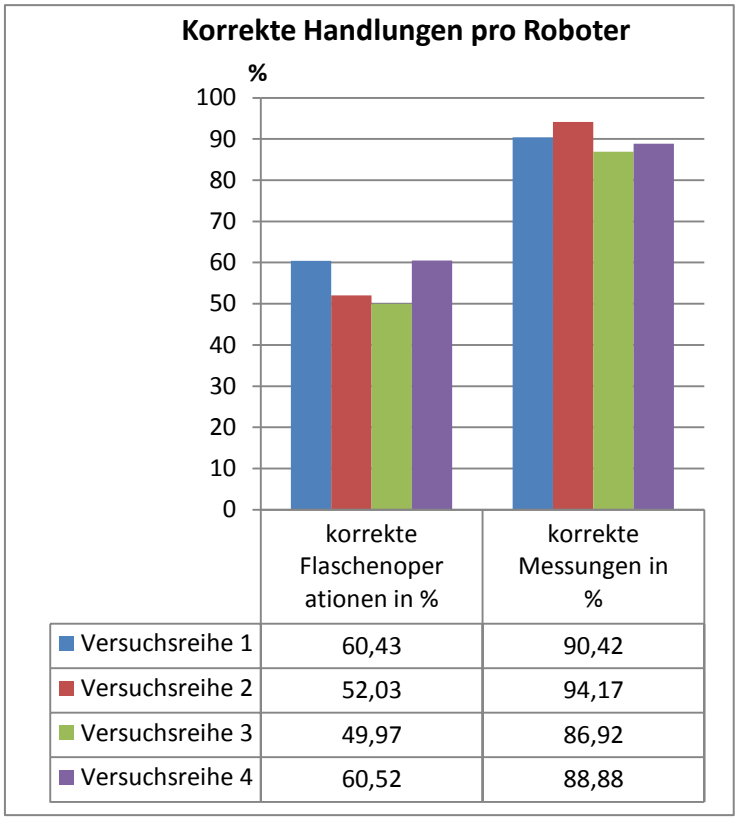
Die Effektivität eines Schwarmes bei Zufall-Versuchen war 22% höher als bei einem einzelnen Roboter mit demselben Versuchsaufbau, bei Raster-Versuchen 39%.

Soll-Effektivität des Schwarms: Damit der Schwarm dieselbe Effizienz erreicht wie ein einzelner Roboter, muss er die dreifache Effektivität eines Einzelnen erreichen (im Graphen grün eingezeichnet). Das würde eine Effektivitäts-Differenz von 200% im Vergleich zum Einzelnen ergeben.

Bei zufälliger Flaschenanordnung erreichte der Schwarm allerdings nur 41% der Soll-Effektivität, bei Raster-Versuchen 54%.

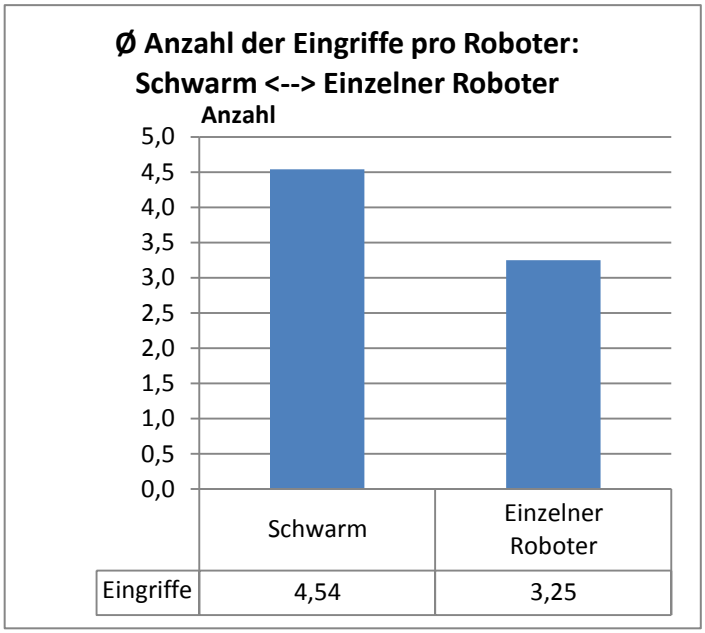
Der Schwarm ist effektiver, aber weniger effizient als ein einzelner Roboter.

3.1.2) Handlungen der Asuros



	korrekte Flaschenoperationen in %	korrekte Messungen in %
Versuchsreihe 1	60,43	90,42
Versuchsreihe 2	52,03	94,17
Versuchsreihe 3	49,97	86,92
Versuchsreihe 4	60,52	88,88

Abb. 5: Korekkte Handlungen pro Roboter (im Schnitt) bei durchgeführten Versuchen



	Schwarm	Einzelner Roboter
Eingriffe	4,54	3,25

Abb. 6: Durchschnittliche Eingriffe pro 15 Minuten. Einzelner Roboter und Schwarm im Vergleich

Wir werteten anschließend die Versuchsprotokolle aus.

Folgende Ereignisse (→siehe 2.4.2) fassten wir zusammen pro Roboter:

- Korrekte Flaschenoperationen: Aufnehmen und Abstellen von Flaschen ohne Fehler. Angegeben in Prozent von Flaschenoperationen insgesamt.
- Korrekte Messungen: keine zu groß oder zu klein interpretierten Messungen. Angegeben in Prozent von Messungen insgesamt.
- Eingriffe: Menschliches Eingreifen nötig (Roboter verhaken sich oder bleiben an der Bande hängen)

Es zeichnet sich kein signifikanter Unterschied zwischen einzelnen Robotern (Versuchsreihe 1 und 2) und Roboterschwärmen ab.

Bei Schwarmversuchen lag die Zahl der Eingriffe höher als bei Einzelversuchen, da sich bei Einzelversuchen keine Roboter verhaken können.

Die Zeit, die durch diese gegenseitige Behinderung verbraucht wurde, stand nicht mehr zur Lösung der Aufgabe zur Verfügung. Dadurch lässt sich die niedrigere Effizienz eines Schwarmes im Vergleich zu einem einzelnen Roboters teilweise erklären.

3.2) Messwerte und Interpretation der Simulation und der Zusammenhang zu den realen Versuchen

Bei der Simulation haben wir dieselben Versuche (Versuchsreihe 1-4) jeweils 50 mal durchgeführt. Die Erwartung, dass die Simulationsversuche höhere Effektivitätswerte ergeben, als die realen Versuche (da Roboter in der Simulation keine Hardwareanfälligkeit haben usw.) hat sich bestätigt: Durchschnittliche Effektivität eines Roboters bei zufällig angeordneten Flaschen in der Realität = 7,63 bei der Simulation = 45,38. Um zu überprüfen, ob die Simulationswerte trotzdem mit denen realer Versuche verglichen werden können, dachten wir uns:

$$\frac{\emptyset\ Effizienz\ dreier\ Roboter\ (real)}{\emptyset\ Effizienz\ eines\ Roboters\ (real)} \approx \frac{\emptyset\ Effizienz\ dreier\ Roboters\ (simuliert)}{\emptyset\ Effizienz\ eines\ Roboters\ (simuliert)}$$

Dies hat sich jedoch nicht bestätigt: für zufällige Flaschenanordnung $\frac{31{,}91}{45{,}38} \neq \frac{15{,}64}{31{,}65}$ und bei Rasteranordnung $\frac{2{,}59}{4{,}78} \neq \frac{15{,}13}{46{,}93}$

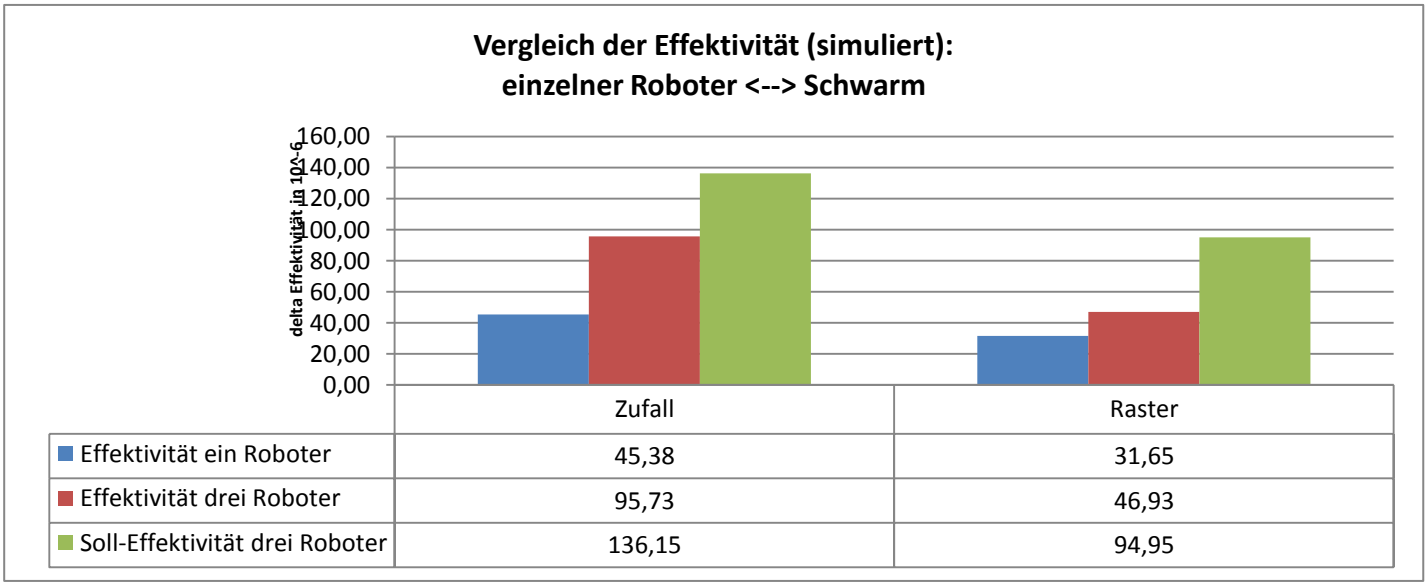


Abb. 7: Vergleich der Effektivität eines einzelnen Roboters im Vergleich zu einem Roboterschwarm und dessen Soll-Effektivität. Basierend auf simulierten Versuchen.

Trotzdem erreicht der untersuchte Roboterschwarm von drei Robotern nicht seine Soll-Effektivität: nur 70% (zufällige Flaschen) bzw. 49% (Raster).

Mithilfe der Simulation haben wir über die vier Versuchsreihen hinaus zwei weitere durchgeführt: Die Startpositionen der Flaschen und die Anzahl der Asuros sind gleich der Versuchsreihe 1 bzw. 3, aber die Versuchslänge dauert 60 Minuten. Hierbei erreichte der Schwarm nur 43% der Soll-Effektivität. (→ siehe DVD – Effizienzmessungen).

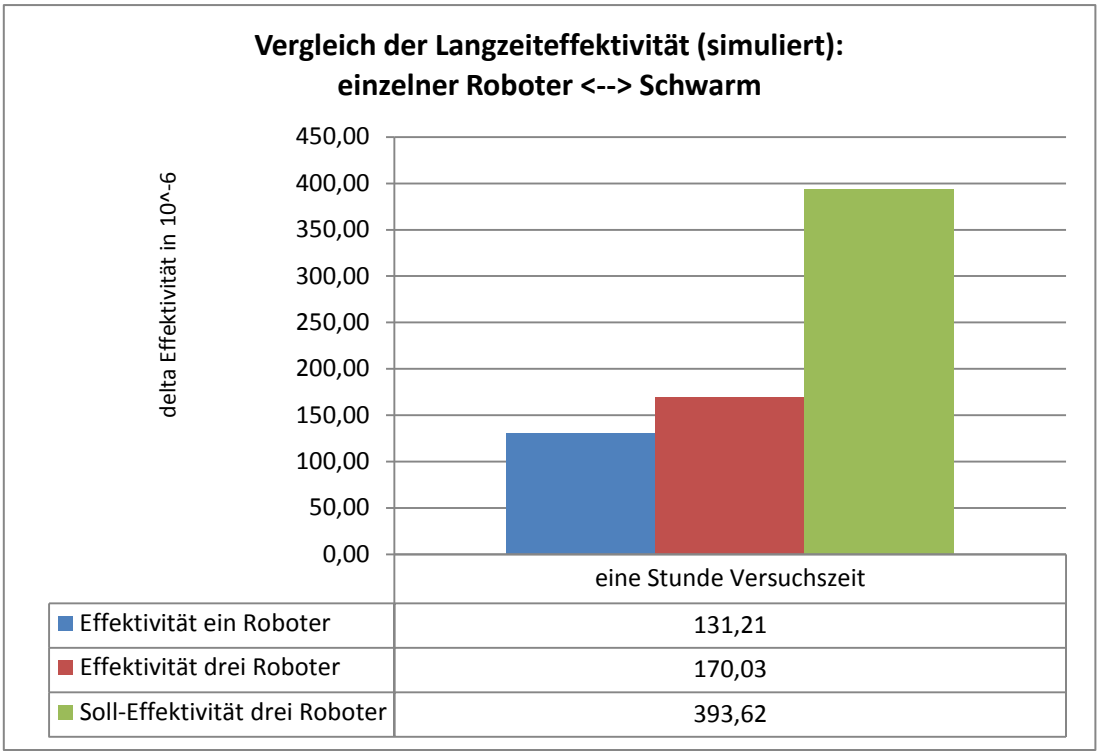


Abb. 8: Vergleich der Effektivität eines einzelnen Roboters im vergleich zu einem Roboterschwarm und dessen Soll-Effektivität. Basierend auf simulierten Versuchnen mit einer Stunde laufzeit

3.3) Fazit

Der von uns beschriebene Schwarm ist effektiver als ein einzelner Roboter. Jeder Roboter für sich genommen ist im Schwarm jedoch weniger effektiv als alleine. Er erreicht nur 41% bzw. 54%, bzw. bei simulierten Versuchen 70% bzw. 49% der Leistung eines einzelnen Roboters.

Zwar konnten wir keine Simulation entwickeln, die sich mit den realen Versuchen eins zu eins vergleichen lässt. Allerdings können wir aus den Messwerten der Simulation (wie bei den realen Versuchen) die Aussage ableiten, dass die Effektivität des Schwarms über der eines einzelnen Roboters aber unter der für einen Schwarm erwarteten Effektivität liegt.

4) Diskussion

Selbstverständlich beziehen sich unsere Ergebnisse nicht auf alle Roboterschwärme, sondern nur speziell auf unsere Versuche. Da wir pro Versuchsreihe nur 4 Versuche durchgeführt haben (insgesamt 16) sind die Ergebnisse nicht unbedingt repräsentativ. Die Differenzen zur Simulation könnten sich auch dadurch erklären.

Das Vermessungsverfahren für Haufen könnte zu ungenau sein, um die Aufgabe effizient lösen zu können. Eine Möglichkeit wäre, dass ein Roboter einen Haufen von mehreren Seiten vermisst, was jedoch mit der einfachen Hardware zu schwer zu realisieren wäre. Das planlose Umherfahren führt nicht dazu, dass innerhalb von 15 Minuten genug Flaschen zusammengestellt werden, um einen großen Haufen zu bilden. Bis der größte Haufen gefunden wird müssen sehr viele Haufen vermessen werden, was viel Zeit in Anspruch nimmt.

Eine vordefinierte, zentral gesteuerte oder direkte Kommunikation benutzende Aufgabenlösung könnte sehr viel effektiver (aber nicht zwangsläufig effizienter) sein. Aber wenn der Ablauf nicht vordefiniert oder zentral gesteuert wird, können die Roboter des dezentralen Schwarms möglicherweise flexibler auf Fehler oder veränderte Umstände reagieren.

Wir können annehmen: Weil die von uns untersuchte Aufgabenlösung auf einfacher Software und Hardware basiert, wird die Aufgabe zwar ansatzweise bewältigt, dies aber nicht sehr effektiv.

5) Quellen & Literaturverzeichnis

Interrupt [Online] / Betreiber: Andreas Schwarz // Mikrocontroller.ne. - 5. 12 2008. - http://www.mikrocontroller.net/articles/Interrupt.

2D Graphics Tutorial [Online] /Betreiber: Sun Microsystems Inc // java.sun.com. - 25. 10 2008. - http://java.sun.com/docs/books/tutorial/2d/basic2d/index.html.

2D Graphics Tutorial – Drawing primitive Geometrics [Online] /Betreiber: Sun Microsystems Inc // java.sun.com. - 25. 10 2008. - http://java.sun.com/docs/books/tutorial/2d/geometry/primitives.html 25.10.2008.

Ameisen [Online] /Verf. Wagner Susanne // Planet Wissen. - 26. 10 2008. - http://www.planet-wissen.de/pw/Artikel,,,,,,,E4D4C88C524B748DE0340003BA5E0905,,,,,,,,,,,,,,.html.

Asuro [Online] /Verf. Grewe Jan und Gruber Robin // arexx.com. - 1. 7 2008. - http://arexx.com/downloads/asuro/asuro_manual_de.pdf.

Asuro meldet nur VL... [Online] / Betreiber: Frank Brall// roboternetz.de. - 3. 12 2008. - http://www.roboternetz.de/phpBB2/zeigebeitrag.php?p=315187.

asuro.h-Dateireferenz [Online] / Betreiber: Peter Rektenwald// asurowiki.de. - 2008. 11 22. - http://www.asurowiki.de/pmwiki/pub/html/asuro_8h.html.

C-Tutorial - RN-Wissen [Online] / Betreiber: Frank Brall// roboternetz.de. - 15. 11 2008. - http://www.roboternetz.de/wissen/index.php/C-Tutorial.

Die Programmiersprache C [Online] /Verf. Schellong Helmut // schellong.de. - 2008. 10 3. - http://www.schellong.de/c.htm.

Echt cooles Java Cleverer Code, Open-Soure-Bibliotheken und Projektideen [Buch] /Verf. Eubanks Brian D.. - [s.l.] : Hanser Fachbuchverlag, 2006. - ISBN 3-446-40685-9.

Effizienz [Online] /Betreiber: Wikimedia Foundation // wikipedia.de. - 5. 8. 2008. - http://de.wikipedia.org/w/index.php?title=Effizienz&oldid=49208207

Emergenz [Online] / Betreiber: Wikimedia-Foundation// Wiktionary. - 1. 1 2009. - http://de.wiktionary.org/wiki/Emergenz.

Entwurfsmuster [Online] / Betreiber: Wikimedia-Foundation // Wikipedia.de. - 6. 13 2008. - http://de.wikipedia.org/wiki/Entwurfsmuster.

Graphics2D Javadoc [Online] / Betreiber: Sun Microsystems Inc // java.sun.com. - 25. 10 2008. - http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics2D.html.

Graphics2D Javadoc [Online] / Betreiber: Sun Microsystems Inc // java.sun.com. - 2008. 10 25. - http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics2D.html.

Heuristik [Online] / Betreiber: Wikimedia-Foundation// Wikipedia.de. - 13. 12 2008. - http://de.wikipedia.org/wiki/Heuristik.

http://www.roboternetz.de/phpBB2/viewtopic.php?p=342583#342583 [Online] / Betreiber: Frank Brall // roboternetz.de. - 16. 12 2008. - Die wichtigsten Dateien, Quellcodes, Downloads.

Java – exemplarisch [Buch] /Verf. Dr. Plüss Ägidius. - München : Oldenbourg Wissenschaftsverlag, 2004. - ISBN 3-486-20040-2.

Jetzt lerne ich Java 6! [DVD] /Verf. Maus Helge. - [s.l.] : Video2Brain.com. - ISBN 978-3-8272-0767-8.

Kollektive Intelligenz [Online] / Betreiber: Wikimedia-Foundation // Wikipedia.de. - 13. 12 2008. - http://de.wikipedia.org/wiki/Schwarmintelligenz.

Kommando (Entwurfsmuster) [Online] / Betreiber: Wikimedia-Foundation // Wikipedia.de. - 2008. 6 13. - http://de.wikipedia.org/wiki/Kommando_%28Entwurfsmuster%29.

Math.sin() - Java [Online] / Betreiber: Dominik Haubrich // Tutorials.de. - 4. 12 2008. - http://www.tutorials.de/forum/java/262942-math-sin.html.

Mehr Spaß mit Asuro, Band 1 [Buch] /Verf. Gruber Robin und Hofmann Martin. - [s.l.] : AREXX INTELLIGENCE CENTRE, 2005. - ISBN 9-080-93921-8.

Mehr Spaß mit Asuro, Band 2 [Buch] /Verf. Gruber Robin und Hofmann Martin. - [s.l.] : AREXX Intelligence Centre, 2007. - ISBN 978-9080939233.

Multiagentensystem [Online] /Betreiber: Wikimedia Foundation // wikipedia.de. - 6. 11 2009. - http://de.wikipedia.org/wiki/Multiagentensystem.

Observer (Entwurfsmuster) [Online] / Betreiber: Wikimedia-Foundation // wikipedia.de. - 13. 6 2008. - http://de.wikipedia.org/wiki/Beobachter_(Entwurfsmuster).

Programmieren mit Java – Multithreading [Online] /Verf. Middendorf Stefan, Reiner Singer und Heid Jörn // dPunkt.de. - 10. 7 2008. - http://www.dpunkt.de/java/Programmieren_mit_Java/Multithreading/1.html bis http://www.dpunkt.de/java/Programmieren_mit_Java/Multithreading/17.html.

Sensorarten - RN-Wissen [Online] /Betreiber: Frank Brall // roboternetz.de. - 16. 12 2008. - http://www.roboternetz.de/wissen/index.php/Sensorarten#Sharp_GP2D12.

Swarm Intelligence: From Natural to Artificial Systems [Buch] /Verf. Bonabeau Eric, Dorigo Marco und Theraulaz Guy. - [s.l.] : Oxford University Press, 1999. - Bd. ISBN 0195131592.

Thema: Tasterwerte an Hyperterminal übergeben [Online] / Betreiber: Frank Brall // roboternetz.de. - 5. 12 2008. - http://www.roboternetz.de/phpBB2/zeigebeitrag.php?p=32820202.

Tutorials [Online] /Betreiber: Vladimir Mironjuk // java-forum.org. - 28. 10 2007. - http://www.java-forum.org/de/topic7152_tutorials.html.

UML 2 Das Einsteigerseminar [Buch] /Verf. Erler Thomas: Vmi Buc, 2004. - ISBN 3-826-67363-8.

Zustand (Entwurfsmuster) [Online] / Betreiber: Wikimedia-Foundation // Wikipedia.de. - 13. 6 2008. - http://de.wikipedia.org/wiki/Zustand_%28Entwurfsmuster%29.

Zuständigkeitskette [Online] / Betreiber: Wikimedia-Foundation // Wikipedia.de. - 13. 6 2008. - http://de.wikipedia.org/wiki/Zust%C3%A4ndigkeitskette.