Automome Systeme
ROBOINO 2.0

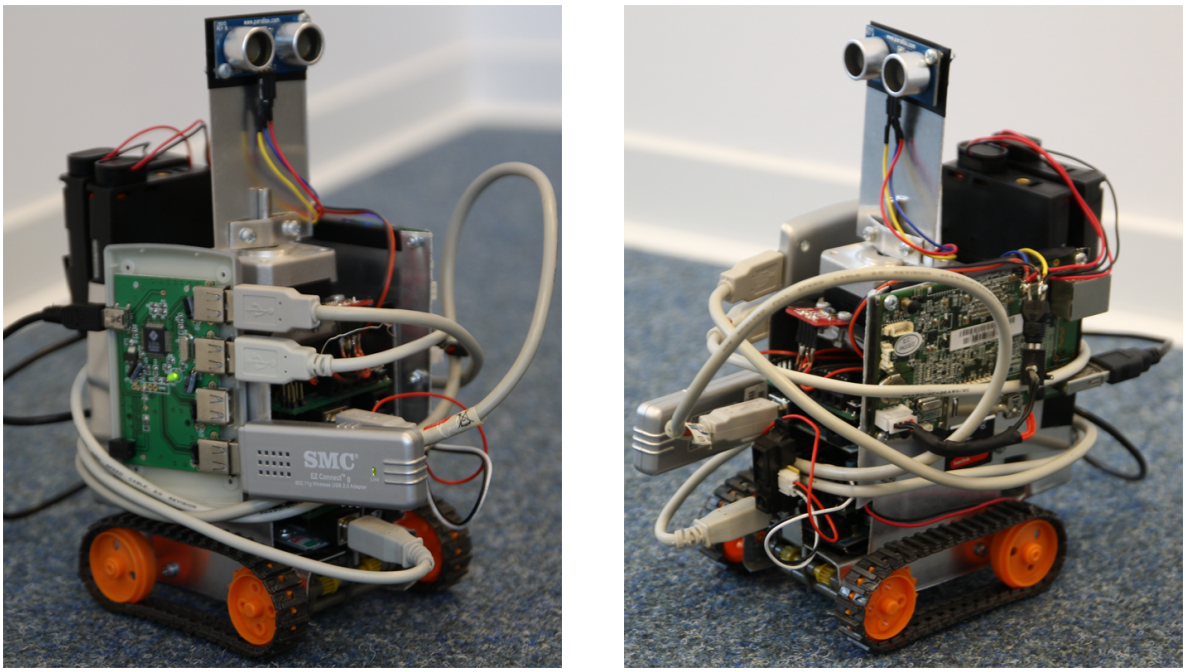
Stephan Bergemann, Ingo Kabus, Max Ludwig

29. Juli 2011

Inhaltsverzeichnis

Abstract

In diesem Artikel beschreiben wir den Entwurf und die Umsetzung eines auf Arduino und Sheevaplug basierenden Roboters zum Einmessen und somit Kartografieren eines Raumes. Diese Kartografierung wird dabei mit einem Roboter vorgenommen, welcher als einzigen Sensor zur Kartografierung einen drehbaren Ultraschall-Sensor verwendet. Daher müssen Drehwinkel und Weite der Fahrt des Roboters selbst gemessen und daraus errechnet werden.



(a) linke Seite (b) rechte Seite

Abbildung 1: Entstandener Roboter

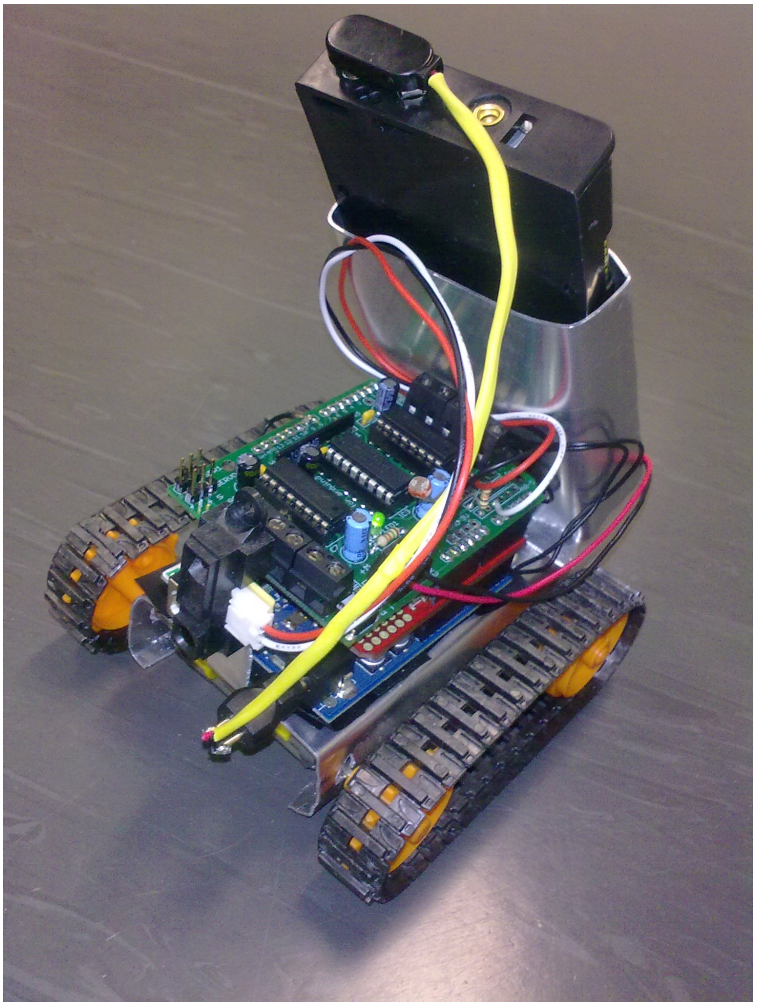


Abbildung 2: Die Ursprungsversion des Roboters

1 Einführung

In diesem Abschnitt wird erläutert, was Ausgangspunkt für die Entwicklung des Roboters war, was konkret die Erweiterungen sind und welche Hardware dazu notwendig ist.

1.1 Ausgangspunkt

Als Startpunkt für die Entwicklung unseres Roboters stand ein im Rahmen des Kurses „Aktuelle Themen Multimedia“ entstandener, per WLAN fernsteuerbarer Roboter auf Arduinobasis.
Dazu wurden ein Arduino-Board, ein Motor-shield, sowie ein WLAN-shield auf einem selbst gebauten fahrbaren Untersatz verbaut. Zur Erweiterung des Roboters zu einem autarken System wurden zusätzlich zweiBatteriepackete aus je 4 Batterien montiert.

Als Fernsteuerung diente ebenso ein auf Arudino basierendes System aus einem Arduino-Board, einem Ethernet-Shield, sowie einem Joy-Stick-Shield. Diese Fernsteuerung war per Ethernet-Kabel mit einem WLAN-Router verbunden, zu dem wiederum der Roboter verbunden war.

Da im Rahmen dieses Projektes von einem Gruppenmitglied bereits Erfahrungen mit der Arduinoplattform und von allen Gruppenmitgliedern schon Erfahrungen mit dem Sheevaplug (einem Steckdosencomputer, der wegen seinen äußerst kompakten Abmessungen bei akzeptablen Kenndaten auch für Embedded-projekte sehr interessant ist) gesammelt werden konnte, war schnell die Herausforderung jenseits der vorgeschlagenen Lego-Mindstorms-Plattform gefunden.

1.2 Idee zum Projekt

Bei einem fahrenden Roboter ist es zunächst unerheblich, irgendeine Kollisionserkennung einzuführen - er wird ferngesteuert und es liegt allein im Ermessen des Steuernden, ihn „gegen die Wand zu fahren“, oder nicht. Wenn der Roboter jedoch autonom werden soll, sollte er beim herumfahren möglichst selbst erkennen, ob er Gefahr läuft, gegen eine Wand zu fahren, oder nicht. Das erfordert also einen Entfernungssensor, der möglichst ständig Daten liefert.
Da wir eine Kombination aus Arduino und Sheevaplug schaffen wollten, lag es nah, die Daten, die der Entfernungssensor liefert, vom Sheevaplug weiter verarbeiten zu lassen.
Da wir hier auf einer Plattform sind, die wesentlich mehr Speicher zur Verfügung hat (je nach Größe der verwendeten SD-Karte beispielsweise 16 GB anstatt der 32 KB Flash-Speicher, über den der Arduino verfügt) kann auch einiges mehr mit den Daten gemacht werden. So kamen wir auf die Idee, einen Roboter zu bauen, der nicht nur in Fahrtrichtung Entfernungen messen kann, sondern auch in alle 360° um ihn herum. Dazu müsste dem Roboter ein weiterer Motor hinzugefügt werden. Mit einem Roboter, der nun in alle Richtungen Entfernungen messen kann und herumfahren kann, überlegten wir uns einen Roboter, der seine Umgebung einmisst und eine Karte von ihr erstellt.

Wenn man nun eine Karte vom Raum hat und weiß, wo der Roboter in ihr steht, wäre es wünschenswert, ihn zielgerichtet an eine Position im Raum navigieren zu lassen.
An der Stelle ist es schließlich auch nötig, eine gewisse Infrastruktur zu schaffen. Der Roboter soll weiterhin möglichst ohne Kabel verwendbar sein, weswegen wieder WLAN genutzt werden sollte. Zum WLAN sollte sich der Roboter dann allerdings nicht mehr vom Arduino verbinden, sondern möglichst vom Sheevaplug direkt. Dies ermöglicht uns die Programmierung einer Webanwendung, welche die Karte zur Verfügung stellt, den Befehl, an eine bestimmte Stelle auf ihr zu fahren, entgegennimmt und an den Roboter die entsprechenden Befehle sendet.

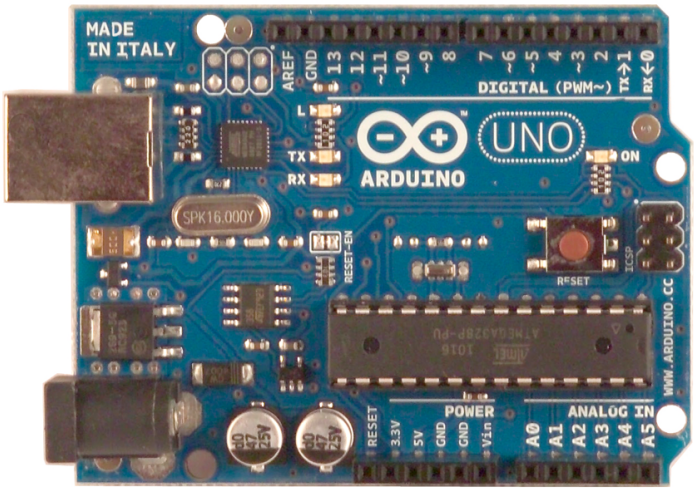


Abbildung 3: Das Arduino-Grundboard [i_ard]

2 Grundlagen

In diesem Abschnitt werden wichtige Grundlagen über die von uns verwendeten Plattformen und Technologien kurz beschrieben.

2.1 Arduino

Arduino[1] ist eine Rapid-Hardware-Prototyping-Plattform mit eigener IDE und eigener Programmiersprache. Diese Programmiersprache basiert auf Processing, wie auch die IDE, während die Hardwareplattform auf Wiring basiert. In dieser Kombination bildet die Arduino-Plattform eine sehr gute Einsteigerplattform für die Mikrocontrollerprogrammierung. Diese bietet durch ein standardisiertes Pinlayout und entsprechender Anordnung dieser Pins (in Reihen), bietet sich die Plattform für Shield-basierte Erweiterung an. Ein Shield erweitert dabei durch Aufstecken auf das Arduino-Board dieses um die entsprechende Funktionalität des jeweiligen Shields. Es gibt beispielsweise WLAN-Shields, Motor-Shields und viele mehr. Duruch die Einfachheit der IDE, Programmiersprache und des Boards, erfreut sich die Arduino-Plattform einer stetig wachsenden Nutzergemeinde und viele Hardwarehersteller für Hobbyelektronik, aber auch im professionellen Bereich erstellen Shields, wodurch die Plattform immer weiter erweitert werden kann. Dabei ist allerdings stets bei der Kombination verschiedener Shields (durch Aufstecken eines weiteren Shields) kein Pin doppelt belegt ist.

Abbildung 4: Das Sheevaplug [i_wik]

2.2 Sheevaplug

Das Sheevaplug ist ein von der Firma Marvell[2] herausgebrachter sog. Plugcomputer. Diese Computer zeichnen sich durch eine besonders kompakte Bauform aus, bei der der gesamte Computer inklusive Gehäuse nicht wesentlich größer als eine handelsübliche, manuelle Zeitschaltuhr ist. Dadurch kann auf diesem Computer nicht die Leistung eines Desktop-PCs erreicht werden, im Sheevaplug muss dafür auch auf einen Display-Anschluss verzichtet werden, für die meisten privaten Serveranwendungen ist das Sheevaplug jedoch eine sehr gute Alternative zu einem ständig laufendem ausgemusterten Desktop-PC als Server. Seine Leistungsaufnahme beträgt unter Volllast der CPU und ohne angeschlossene Hardware ca. 7 Watt, in Idle sogar nur 3 Watt.
Das Sheevaplug verfügt über einen USB-Host-Adapter, eine Ethernetschnittstelle und einen SD-Kartenslot.
Durch eine Board-Spannung von 5V, seine knappen Abmessungen und dennoch einer akzeptablen Leistung (1,2GHz CPU, 512 MB RAM), eignet sich das Sheevaplug auch hervorragend für Embedded-Projekte. Der einzige Nachteil für Embedded-Projekte ist, dass keine sog. General I/O-Pins zur Verfügung stehen und somit externe Hardware entweder über die Pins des SD-Karten-Slots (welcher im wesentlichen eine SPI-Schnittstelle darstellt), oder per USB angeschlossen werden kann.

2.3 Simbad

Simbad ist ein Simulator für mobile Roboter. Er ist in Java geschrieben und durch die GNU General Public License Open-Source-lizensiert. Simbad wurde vorrangig für Forschung und Lehre entwickelt, wodurch der Quelltext gut lesbar und verständlich ist. Ziel ist es, Studenten und anderen Personen, die keine weiteren Vorkenntnisse in der

[1]Webseite des Arudino-Projektes: `http://www.arduino.cc`
[2]Webseite des Herstellers vom Sheevaplug: `http://marvell.com`

Robotik haben, den Einstieg in diesen Bereich zu erleichtern. Simbad stellt eine Plattform zur Verfügung, die unter anderem das Simulieren mehrerer Roboter gleichzeitig unterstützt. Weiterhin bietet es eine eigene Physik-Engine. Durch Java ist Simbad auf allen großen Plattformen lauffähig. Auch reale Roboter werden in begrenztem Maße unterstützt. Ein Framework stellt einfache Klassen und Methoden bereit, mit denen man schnell eigene Algorithmen für Roboterverhalten erstellen kann. Außerdem kann man simple Umgebungen schaffen, in denen sich der Roboter bewegt. Das Ergebnis kann dann in einer grafischen Oberfläche getestet werden. Diese enthält standardmäßig einen Debug-Monitor, ein Fenster, welches die 3D-Umgebung und den darin befindlichen Roboter darstellt, sowie mehrere Bedienelemente, um die Simulation zu steuern.

3 Entwurf

In diesem Abschnitt beschreiben wir alle für den Entwurf des Roboters notwendigen Komponenten.

Da zunächst noch keine Hardware (sprich: kein Roboter) zur Verfügung stand, da dieser erst gebaut werden musste, standen wir vor dem Problem, dass wir das erstellte Konzept nicht implementieren und somit nicht testen und weiterentwickeln konnten. Die Lösung brachte die in den Grundlagen bereits erwähnte Simbad-Plattform. Mit deren Hilfe konnten die Algorithmen bereits frühzeitig implementieren und testen. Da man für Simbad auch in Python - was wir für den realen Roboter verwenden - programmieren kann, ließ sich der geschriebene Quelltext relativ einfach für den realen Roboter portieren.

3.1 Odometrie

Da der Roboter keine Odometriedaten erfassen kann, müssen wir sowohl für die zurückgelegte Entfernung, als auch für die Berechnung des Drehwinkels die Daten des Entfernungssensors verwenden.

Um den Winkel herauszubekommen, um den sich der Roboter bei einem Drehmanöver dreht, nehmen wir vor und nach dem Manöver rundum Entfernungen zur Umgebung auf. Die beiden Messungen werden gegeneinander verschoben und die beste Übereinstimmung ermittelt (Faltung).

Die Qualität der Übereinstimmung berechnen wir, indem wir die Differenzen der Messwerte für die einzelnen Winkel quadrieren und aufsummieren. Eine gute Übereinstimmung zeichnet sich nun durch eine kleine Summe der quadrierten Differenzen aus (eine möglichst auf der x-Achse liegende Funktion).

Dies hat den Vorteil, dass einzelne ausgerissene Messwerte das Ergebnis kaum beeinflussen. Der Algorithmus ist auch unempfindlich gegen einen bei unserem Roboter leider unvermeidbaren und unkontrollierbaren Abstand zwischen der Drehachse des Roboters und der Drehachse des Entfernungssensors.

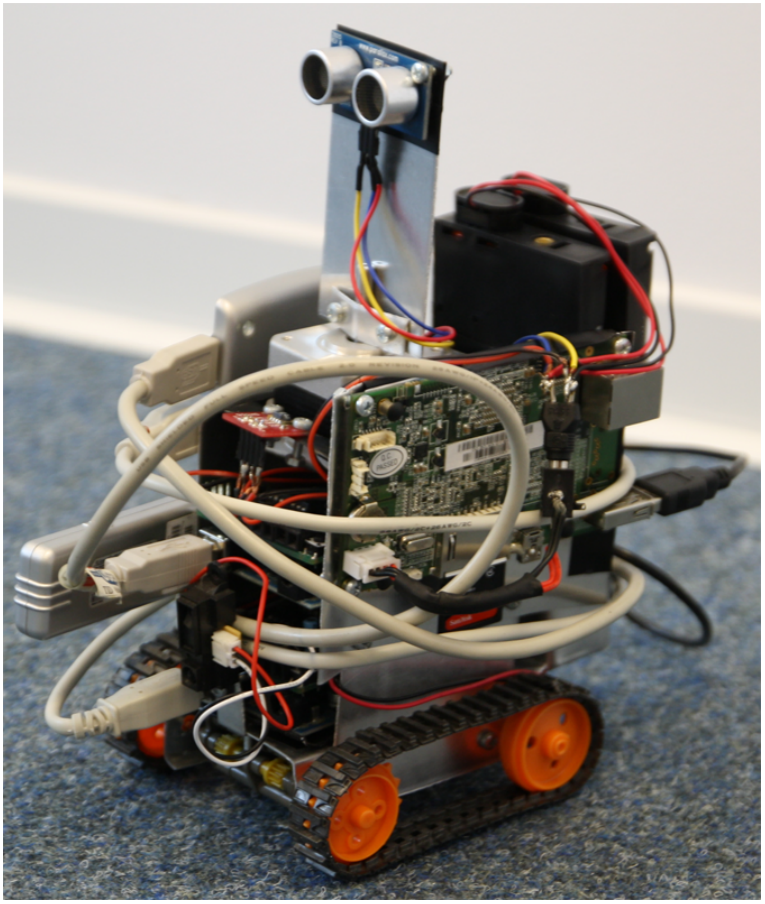


Abbildung 5: Der Roboter nach allen vorgenommenen Erweiterungen

3.2 Aufbau des Roboters

Auf den ursprünglichen Roboter (siehe Abbildung 2) muss zunächst das Sheevaplug aufgebracht werden. Damit alle (geplanten) Sensoren und Motoren auf dem Roboter ohne Pin-Doppel-Belegungen gesteuert bzw. ausgelesen werden können, muss ein zweites Arduino angebracht werden. Das WLAN-Shield entfällt, dafür muss an einen USB-Hub, welcher an das Sheevaplug angeschlossen werden muss, um die zwei Arduinos steuern zu können, ein WLAN-Stick angebacht werden, damit die Messungen des Roboters für den Nutzer sichtbar werden.

Dem Stapelkonzept des Arduinos in abgewandelter Form folgend, war die Idee, den Arduino zusätzlich auf den anderen Arduino zu montieren, was eine zusätzliche Ebene erforderte. Da der Rundum-Messkopf möglichst auf dem Drehpunkt der Ketten angebracht werden soll, muss er über allen anderen Aufbauten aufgebaut werden. Das erfordert eine weitere Ebene. Das Sheevaplug wird an eine Seite des Roboters angebracht. Auf der anderen Seite wird der USB-Hub montiert.
Als Material für die Aufbauten wurden ausschließlich gebogene, gesägte und vor allem geschraubte (nicht geklebte) Aluminiumplatten verwendet.

3.3 Aufbau des Roboters in Simbad

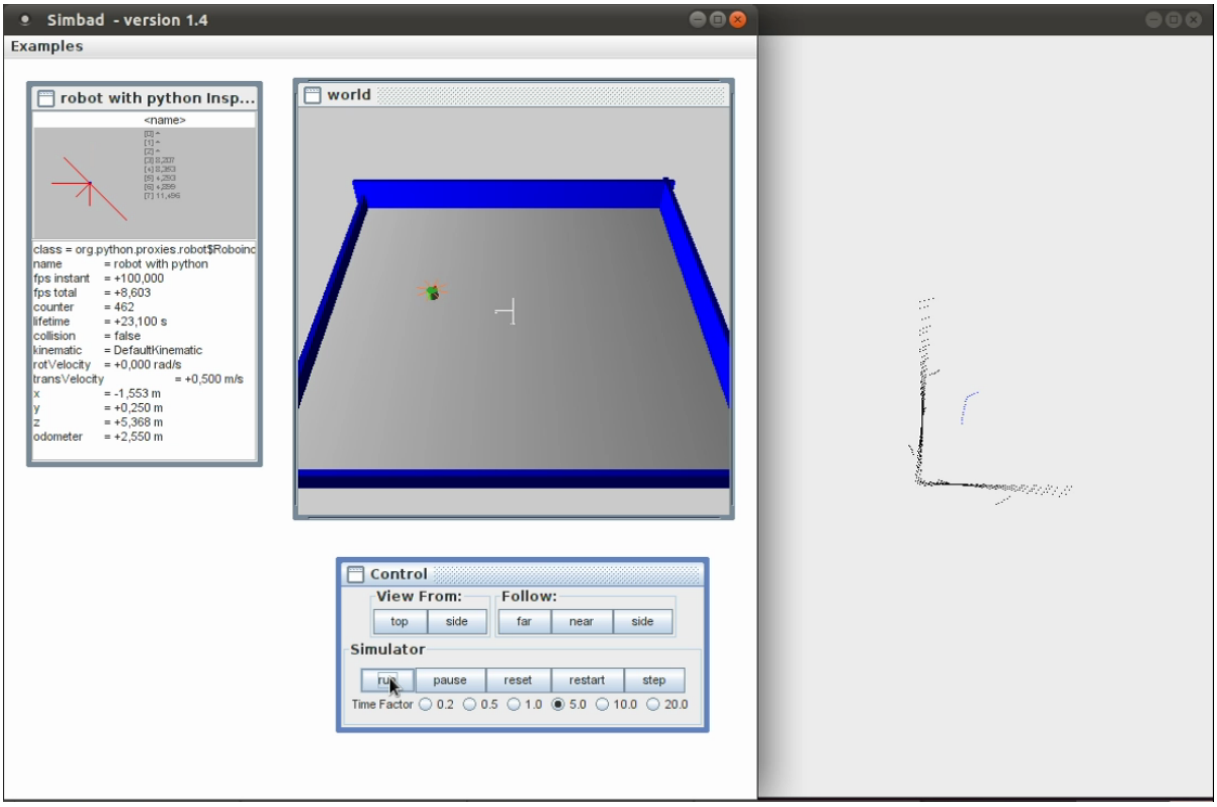


Abbildung 6: Der Roboter-Prototyp in Simbad (links) mit teils gezeichneter Karte (rechts)

Der Roboter in Simbad unterscheidet sich in geringem Maße von dem realen Roboter. Während der reale Roboter über einen drehbaren Ultraschallsensor verfügt, ßitzenäuf dem simulierten Roboter in regelmäßigen Abständen mehrere Sensoren, die durch Simbad als Ultraschallsensoren simuliert werden. Der reale Roboter fährt, stoppt und dreht den Sensor. Der simulierte Roboter hingegen fährt, stoppt und dreht sich selbst. Dieser Unterschied wird allerdings durch entsprechende Quelltextänderungen ausgeglichen. Weiterhin verfügt der simulierte Roboter über ein Odometer, der reale Roboter nicht. Auch für diesen Unterschied mussten letztendlich nur geringe Quelltextänderungen durchgeführt werden.

4 Implementierung

Für den seriellen Nachrichtenaustausch zwischen den Arduions und dem Sheevaplug musste zunächst ein Protokoll entworfen werden, welches alle Aktoren und Sensoren anspricht, ihnen Werte liefert und von ihnen Werte empfangene Werte zurückgeben kann. Aufgrund der Flexibilität der Sprache, vorhandener Bibliothek für serielle Kommunikation und schließlich persönlicher Präferenz wählten wir für die Programmierung

der Sheevaplug-Komponente unseres Systems Python als Programmiersprache. Mit Python als Skriptsprache konnten wir programmierte Funktionen komfortabel „in-time“ im Python-Interpreter testen.

4.1 Arduinoprogrammierung

Da die Nachrichtenverteilung an die Arduinos im Python-Programm vorgenommen wird, ist es auf den Arduinos nicht unbedingt notwendigt, jedem ein eigenes Programm zu schreiben - jeder Arduino kann potenziell auf alle Nachrichten reagieren. Somit musste nur ein Programm geschrieben werden, wovon durch die verschiedenen Nachrichten nur die jeweils wirklich angeschlossenen Sensoren und Aktoren angesprochen werden.

Zur Arduinoprogrammierung gehört bequemerweise auch die Programmierung des Python-Host-Modules, welches dann vom Rest des Programms genutzt werden kann und sämtliche serielle Kommunikation kapselt. Damit stellt dieses Modul eine API für den Roboter dar.

```
1  pinMode(pingPin, OUTPUT);
2  digitalWrite(pingPin, LOW);
3  delayMicroseconds(2);
4  digitalWrite(pingPin, HIGH); // Schall aussenden
5  delayMicroseconds(5); // für 5 Microsekunden
6  digitalWrite(pingPin, LOW); // Schall ausschalten
7  pinMode(pingPin, INPUT);
8  // messen, wann er zurück kommt
9  duration = pulseIn(pingPin, HIGH);
10 // umrechnen nach cm
11 cm = uint16_t (duration / 29 / 2);
```

Listing 1: Ultraschallimpuls senden und empfangen

Listing 1 zeigt das Messen der Entfernung durch den Ultraschallsensor und damit einen zentralen Bestandteil der Logik. Man sieht, dass hier tatsächlich alles „händisch“ gemacht werden muss: zunächst den Schallpin als Ausgang setzen, auf LOW schalten, anschließend auf HIGH (Ultraschall aussenden) und nach einer kurzen Pause wieder auf LOW. Anschließend wird auf die Antwort gewartet (`pulseIn(PingPin, HIGH)`) und diese Zeit wird in cm umgerechnet.
Auf der Python-Seite wird hierbei nur noch die serielle Schnittstelle mit der entsprechenden Nachricht angesprochen und auf die Antwort gewartet, welche von der Funktion an die Programmlogik übergeben wird (siehe Listing 2).

```
1  def get_distance(step):
2      send_message(create_message(DIST, step), ser1)
3      if ord(ser1.read()) is DIST:
4          dist = bytearray(2)
5          dist[0] = ser1.read()
6          dist[1] = ser1.read()
```

```
7         answer = dist[1] + (dist[0] << 8)
8         return answer
```

Listing 2: Pythonmethode, um die Distanz zu messen

Listing 3 zeigt die notwendige Funktionalität, um den Step-Motor zu drehen, welcher den Ultraschallsensor dreht. Hier ist es aufgrund der Kabelführung zum Sensor erforderlich, dass sich der Stepmotor nie weiter als 200 Steps dreht (genau die Anzahl an Schritten, bei denen der Roboter sich um 360° dreht).

```
1 method = constrain(method, 0, 199);
2 if (method > rotationpos) {
3   stepper.step(method - rotationpos, FORWARD, precision);
4   stepper.release();
5 } else if (method < rotationpos) {
6   stepper.step(rotationpos - method, BACKWARD, precision);
7   stepper.release();
8 }
```

Listing 3: Drehung des Steppers, sodass das Kabel nicht im Weg steht

4.2 Algorithmus zum Berechnen des Drehwinkels

Der Algorithmus aus Unterabschnitt 3.1 sieht in Python wie folgt aus und liefert die Drehung in Grad:

```
1  def ingalgo(a, b):
2      sqSums = [] # Liste von Abweichungsquadraten
3      for i in range(0, STEPS, STEP):
4          sqSum = 0.0 # Summe der Quadrate aller Abweichungen
5          for j in range(0, STEPS, STEP):
6              sqSum += (a[j/STEP] - b[j/STEP]) * (a[j/STEP] - b[j/STEP])
7          sqSums.append(sqSum)
8          b.append(b.pop(0))
9      #niedrigsten Index finden
10     lowIndex = 0
11     lowVal = 999999.9
12     for i in range(0, STEPS, STEP):
13         if (sqSums[i/STEP] < lowVal):
14             lowIndex = i
15             lowVal = sqSums[i/STEP]
16     return (lowIndex * 360) / STEPS
```

Listing 4: Pythonmethode, um den Winkel der Drehung zu errechnen

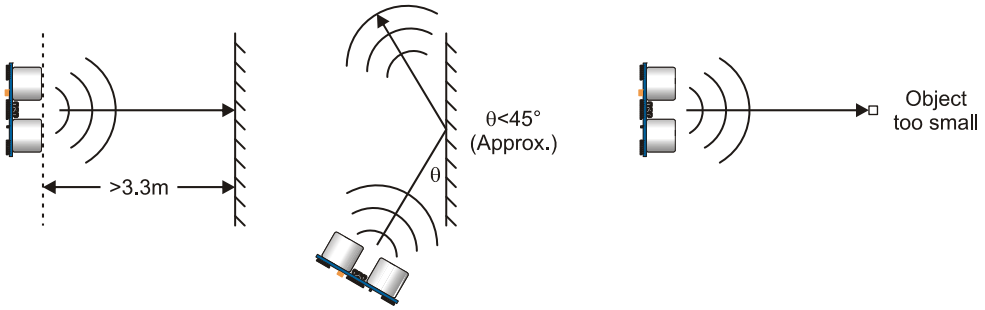


Abbildung 7: Ausleuchtung der Umgebung durch den verwendeten Ultraschallsensor [i_PAR]

5 Zusammenfassung und Ausblick

Ziel war es, einen Roboter zu bauen, der einen gegebenen Raum einmisst, indem er ihn abfährt und dabei misst.
Wünschenswert war die Erstellung einer interaktiven Karte, auf der ein Benutzer nach ihrer Erstellung wählen kann, wo der Roboter hinfahren soll.
Herausgekommen ist ein Roboter, der einen gegebenen Raum abfährt und dabei Messwerte aufnimmt. Diese Messwerte korellieren nach einigen Optimierungen in etwa mit dem Umriss des Raumes, werfen jedoch einige Unzulänglichkeiten auf, welche auf verschiedene Ursachen zurück zu führen sind:

Prinzip des Entfernungssensor Der verwendete Ultraschallsensor ist - senkrecht auf eine Wand gerichtet - zunächst sehr genau, bei Drehungen zwischen 0 und 45° (siehe Abbildung 7) jedoch kommt der ausgesendete Schall nicht mehr von der Stelle im Raum, die anvisiert wird zurück, sondern wird von dieser Stelle zu einer anderen gebrochen und verfälschen somit die gemessene Distanz (siehe Abbildung 9).

Kettenantrieb Durch den Kettenantrieb und generell unterschiedlich starke Motoren ist eine Fahrt geradeaus nicht genau geradeaus. Diese Fehler addieren sich über die gesamte Messung auf.

Zunächst befürchteten wir, dass der Kettenantrieb vor allem auch bei der Drehung unberechenbare Resultate erzeugt. Durch den verwendeten Algorithmus zur Berechnung des Drehwinkels gibt jedoch sehr zuverlässig den Winkel zurück.

Zur Umsetzung von weiteren Funktionalitäten (auslesen weiterer Sensoren, Fahren zu Punkten auf der Karte und überhaupt einer interaktiven Karte) sind wir vor allem aus Zeitgründen nicht gekommen. Teils ist die Funktionalität allerdings schon vorhanden: der Arduino-Code enthält bereits, ebenso wie die API auf Pythonseite sämtliche Funktionen, der Robter sämtliche Sensoren und eine Ajax-Oberfläche für die interaktive Karte ist auch bereits erstellt. Jedoch war das Erstellen der Karte als solche so aufwändig, dass für den Rest keine Zeit blieb.

Abbildung 8: Umgebung der Messung mit Roboter

Abbildung 9: Messung eines rechteckigen Raumes von der Mitte aus

(a) Messung 1 - quadratischer Raum (b) Messung 2 - quadratischer Raum

Abbildung 10: Beispielmessungen in einem quadratischem Testgelände

Bildquellen

[i_ard] ARDUINO.CC: *Arduino Uno*. `http://arduino.cc/en/uploads/Main/ArduinoUnoFront.jpg`. – [Online; abgerufen am 16. Mai 2011]

[i_PAR] PARALAX: *PING)))TM Ultrasonic Distance Sensor*. `http://www.pololu.com/file/0J214/PING_documentation.pdf`. – [Online; abgerufen am 28. Juli 2011]

[i_wik] WIKIPEDIA: *Sheevaplug*. `http://de.wikipedia.org/wiki/SheevaPlug`. – [Online; abgerufen am 28. Juli 2011]