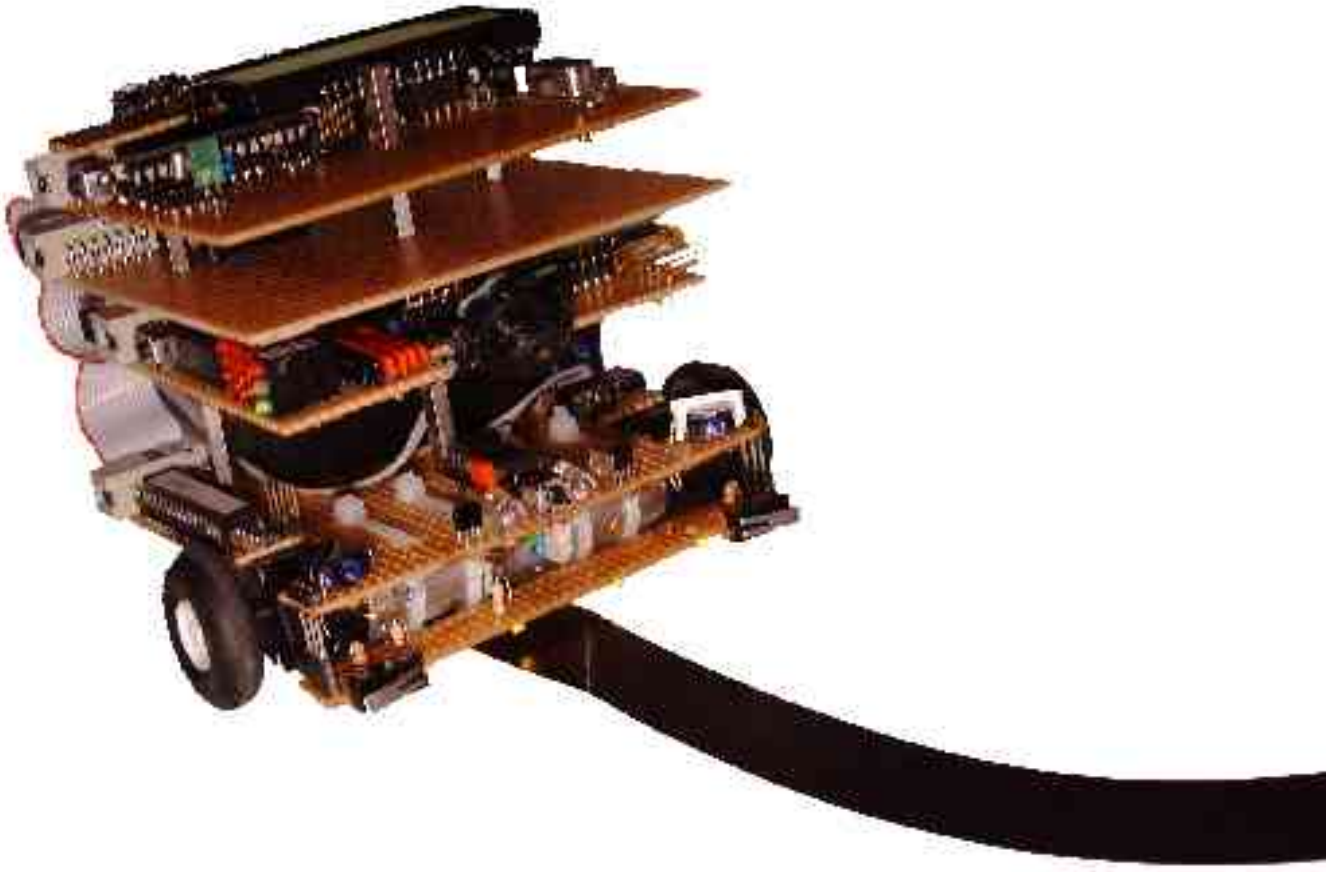
Steuerung mobiler Roboter



MICHAEL STROSCHE 13 WB

Inhalt

1. ferngelenkte Roboter

Bei ferngelenkten Robotern unterscheidet man zwischen zwei Typen, den vollständig manuell gesteuerten und den halb-autonomen Robotern. Diese beiden Steuerungsprinzipien sollen nun im Folgenden näher erläutert werden.

1.1. manuelle Steuerung

Bei manuell ferngelenkten Robotern wird ein meist auf Rädern befindliches Objekt über ein Steuerkabel oder eine Funkverbindung durch einen Operator an einem Steuerpult durch bekanntes oder unbekanntes Gelände gesteuert. Der Operator bekommt mittels bidirektionaler Datenverbindung Sensorwerte von Umwelteinflüssen, sowie auch visuelle und akustische Informationen geliefert, welche er selbstständig (oder im Team mit anderen Menschen) bewertet und entsprechende Reaktionen auf die momentan vorhandene Situation entwickelt. Diese Reaktionen werden in entsprechende Richtungs- und Geschwindigkeitsänderungen des Roboters, oder auch durch die Verwendung anderer Aktuatoren, wie Greifer oder andere Werkzeuge umgesetzt.

Die Vorteile dieser Art von mobilen Robotern liegen auf der Hand:

einfache Steuerung

kostengünstige Herstellung

kleine Bauweise => Roboter können an Stellen eingesetzt werden, wo Menschen nur schwer oder gar nicht herankommen.

Abb.1 Kanalreinigungsroboter

Abb.2 Roboter beseitigt dekontaminiertes Material

Allerdings ist diese Art der Steuerung bei großen Entfernungen, die gleichzeitig eine hohe Zeitdifferenz bewirken, nicht sehr praktisch. Beispielsweise wäre eine vollkommen erdgebundene Steuerung eines Marsrovers oder einer extraterrestrischen Sonde aufgrund großer Signallaufzeiten (ca.30min. für ein Steuersignal von der Erde zum Mars und zurück) in heiklen Situationen, welche ein Scheitern der Mission provozieren würden, nicht anwendbar. Aus diesem Grund gibt es eine weitere Art der Steuerung mobiler Roboter, welche im nächsten Kapitel vorgestellt werden soll.

1.2. halb-autonom Steuerung

Bei halbautonomen Robotern besteht kein großer Unterschied zur Steuerung durch einen Menschen, da sich dieser immer noch um die Erstellung und grobe Abarbeitung von Missionszielen kümmern muss. Der wesentliche Unterschied ist allerdings, dass der Roboter ein eigenes „Gehirn“ mit einer primitiven KI (Künstliche Intelligenz) besitzt, welches ihn dazu befähigt, eine eigene Bahnplanung (Berechnung des Weges von Punkt A zu Punkt B) auszuführen und somit auf wechselnde Gegebenheiten zu reagieren. Somit ist es ihm auch möglich zu verhindern, dass eine kostspielige Mission zum Scheitern verurteilt ist, da ein kleiner Steinbrocken den Weg des Rovers blockiert, welcher nicht von den Video-Kameras erfasst werden kann und somit für die Operatoren auf der Erde „unsichtbar“ ist.

Die KI des Roboters bezieht ihre Informationen meist über „einfache“ Sensoren, wie Stoßkontakte, 2D-Laser-Scanner oder Photozellen.

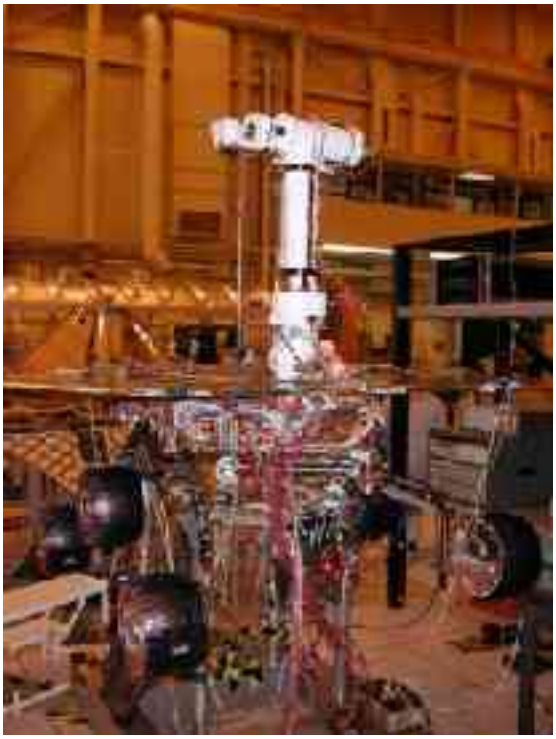


Abb.3 Mars Exploration Rover 1

Abb.4 Predator Unmanned Aerial Vehicle (UAV)

2. autonome Roboter

Autonome Roboter erfordern keine Steuerung durch den Menschen. Diese können vollkommen selbstständig in bekanntem, sowie unbekanntem Gelände navigieren und sich fortbewegen. Dies gelingt ihnen durch den Einsatz hochentwickelter Sensor-, Aktuator- und Prozessorsysteme.

Als Sensoren werden auch hier Kameras, Abstandssensoren (Ultraschall, Infrarot, Laser), Berührungssensoren, etc. verwendet.

Die Besonderheit hier steckt nun aber in der vollkommen autonomen Weiterverarbeitung der Sensorsignale in Echtzeit, der eventuellen Kartographierung der Umgebung, der selbstständigen Kalibrierung von Sensoren und Aktuatoren, sowie dem selbstständigen Erlernen neuer Verhaltensmuster bzw. Objekte, die beispielsweise von der Kamera erkannt werden sollen.

Eine solch hohe Komplexität lässt sich oft nicht durch den Einsatz eines einzelnen Prozessors erreichen, sondern erfordert das Zusammenspiel mehrerer parallel arbeitender und kommunizierender Prozessoren, die über künstliche neuronale Netze (KNN) einkommende Signale bzw. Signalmuster ähnlich wie Nervenzellen verarbeiten und auch neue Muster erlernen können.

Näheres zu neuronalen Netzen kann unter den Quellen [Q4] und [Q5] erfahren werden.

Ein Musterbeispiel für das Zusammenspiel einer parallelen Prozessoreinheit und dem Verarbeiten einer großen Anzahl von Sensorwerten ist der humanoide Roboter Asimo von Honda [Q0]. Dieser ist in der Lage einen Raum laufend zu durchqueren, Treppen zu steigen, sich auf unebenem Untergrund zu bewegen und durch Spracherkennung auch Befehlen von Menschen zu gehorchen, ihnen Fragen zu beantworten, oder ihnen zu folgen. Des weiteren ist es ihm sogar möglich, Personen durch Gesichtserkennung wieder zu erkennen und diese mit ihrem Namen anzusprechen.

Eines der größten Probleme bei mobilen Robotern und besonders bei solch komplexen Robotern ist die Stromversorgung. Da diese eigentlich unabhängig agieren sollen, wäre es ja eigentlich unlogisch, wenn diese an einem etwas längeren Kabel durch die Gegend laufen, fahren, oder gar fliegen müssten. Dies würde ihren Aktionsradius erheblich einschränken und manche Konstruktionen gar unmöglich machen. Daher ist eine eigenständige Energieversorgung extrem wichtig. Asimo beispielsweise kann mit vollständig aufgeladenem Akku nur maximal 20 Minuten agieren. Nach Ablauf dieser Zeit muss er allerdings wieder seine Ladestation aufsuchen.

Abb.5 ASIMO

Zur Steuerung bedarf es, wie bereits gesagt, nicht mehr der Steuerung durch den Menschen.

Dieser fungiert oft nur noch als Beobachter und korrigiert wenn nötig einige Funktionen, um einen sicheren Betrieb zu gewährleisten.

Dieser sichere Betrieb zeichnet sich vor allem dadurch aus, dass die drei Regeln der Robotik [Q1] eingehalten werden:

1) **Ein Roboter darf einem menschlichen Wesen keinen Schaden zufügen oder durch Untätigkeit zulassen, dass einem menschlichen Wesen Schaden zugefügt wird, es sei denn, dies würde das Nullte Gesetz der Robotik verletzen.**

-

2) **Ein Roboter muss dem ihm von einem menschlichen Wesen gegebenen Befehl gehorchen, es sei denn, dies würde das Nullte oder das Erste Gesetz der Robotik verletzen.**

-

3) **Ein Roboter muss seine Existenz schützen, es sei denn, dies würde das Nullte, das Erste oder das Zweite Gesetz der Robotik verletzen.**

-

Diese Gesetze wurden 1942 von Isaac Asimov in seiner Science-Fiction Geschichte „Runaround“ (Herumtreiber) veröffentlicht und bilden die höchste Kontrollinstanz aller Roboter. Diese Gesetze sind in so gut wie jedem mobilen Roboter unserer Tage in irgend einer Form realisiert.

Das in den oben genannten Gesetzen angesprochene „Nullte Gesetz“ wird in einer späteren Geschichte von Isaac Asimov von einem Roboter aufgestellt. Dieses lautet [Q2]:

0) Ein Roboter darf der Menschheit keinen Schaden zufügen oder durch seine Untätigkeit gestatten, dass die Menschheit zu Schaden kommt.

Auf der folgenden Seite finden Sie ein Beispiel (in DELPHI) für das Folgen einer Linie mit Hilfe von zwei optischen Sensoren, die unterhalb des Roboter in einem Abstand, der etwas breiter ist als die zu erkennende Linie, angebracht sind: Die Variablen *a* und *b* sind Byte-Variablen, die den Helligkeitswert des jeweiligen Photo-Sensors enthalten (*a* = linker Sensor, *b* = rechter Sensor). Die Funktionen FORWARD, LEFT und RIGHT sind dafür zuständig, dass die Motoren die entsprechenden Kommandos ausführen.

Mit diesem Programm ist es allerdings nicht möglich zu erkennen, wann das Ende einer Linie erreicht ist, somit fährt der Roboter weiter geradeaus, obwohl die Linie bereits aufgehört hat.

```
// Umrechnen, ob ein bestimmter Schwellenwert überschritten wurde
if(a >= TRESHOLD)
then
   a := 1
else
   a := 0;
if(b >= TRESHOLD)
then
   b := 1
else
   b := 0;

// Prüfen, welcher Sensor die Linie detektiert hat.
// Linie wurde erkannt, wenn der Wert in der Variable 1 ist.
if((a = 0) and (b = 0))
then
   {
    es wurde keine Linie gefunden
    es wird angenommen, dass sich die Linie nun zwischen den Sensoren
    befindet.
    geradeaus fahren
   }
   FORWARD
else if((a = 0) and (b = 1))
then
   {
    der rechte Sensor hat eine Linie gefunden
    leicht nach Rechts fahren, um die Linie wieder zwischen die Sensoren
    zu bekommen.
   }
   RIGHT
else if((a = 1) and (b = 0))
then
   {
    der linke Sensor hat eine Linie gefunden
    leicht nach Links fahren, um die Linie wieder zwischen die Sensoren
    zu bekommen.
   }

    nach Rechts fahren und abwarten, was passiert
    man könnte auch nach Links fahren!
   }
   RIGHT;
```

Listing 1

Solche Programme können natürlich erheblich komplexere Strukturen annehmen und einiges an Speicherplatz verbrauchen. Da diese Programme mit steigender Komplexität natürlich auch mehr Rechenzeit benötigen, ist es für einen Roboter extrem gefährlich, wenn dieser 2 Sekunden braucht, um eine Bildverarbeitung durchzuführen und kein anderer Prozess während dieser Zeit die Chance hat andere Berechnungen oder Kurskorrekturen vor zu nehmen. Daher werden oft einfache und kleine Betriebssysteme für diese geschrieben, welche dann die Programmabarbeitung kontrollieren und überwachen. Somit ist es dann auch möglich, Programme in einer Art Multitask-Betrieb zu betreiben, welcher es ermöglicht, dass mehrere Programme scheinbar simultan ablaufen.

Als Beispiel für ein solch komplexes Programm möchte ich die Bahnplanung erwähnen. Die Bahnplanung ermöglicht es einem Roboter anhand seiner gegenwärtigen Position, einer Zielposition und einer mehr oder weniger detaillierten Karte selbstständig einen Weg zum Ziel zu finden,. Dieser sollte Erstens möglichst kurz und Zweitens um Hindernisse herum führt. Der Ursprung der Karte kann hierbei unterschiedlichster Art sein.

Die Karte kann beispielsweise vom Roboter selbstständig erstellt werden. Dazu fährt der Roboter zunächst ziellos in der Gegend herum, bis er ein grobes Grundgerüst der Umgebung in der Karte aufgebaut hat. Dieses benutzt er dann als Grundlage zur Navigation und verfeinert die Karte immer weiter. So ist es ihm auch möglich die Karte ständig aktuell zu halten, so dass er nicht plötzlich vor einem unbekannten Hindernis steht. Sollte dies dennoch auftauchen, so wird es einfach in der Karte eingetragen und eine weitere Kollision wird somit vermieden. Wenn dieses Objekt nun aber wieder weggenommen wird, so kann der Roboter dies erkennen und es aus der Karte entfernen.

Eine andere Möglichkeit besteht darin, dass ein Mensch eine freie Handskizze der Umgebung anfertigt, einen Zielpunkt einträgt, am Computer einscannt und an den Roboter schickt. Dieser benutzt dieses grobe Umgebungsbild zur Navigation und verfeinert es, während er seine Arbeit verrichtet. [Q3]

Natürlich können im Zeitalter drahtloser Kommunikationsmittel auch die Roboter untereinander kommunizieren und sich somit gegenseitig Befehle geben. Dies hat den Vorteil, dass mit einem geringen Maß an zu implementierender KI ein hohes Maß an Flexibilität des entstehenden „Gesamtorganismusses“ entsteht und dieser dann in Dinge bewerkstelligen kann, die ein Roboter alleine nie geschafft hätten. [Q6]

Dies ermöglicht es auch, dass die Roboter untereinander einen „Führer“, bzw. einen Alpha-Roboter ernennen können, dem dann die ganze Gruppe folgen kann.

Die Interaktion in Gruppen wird auch bei der Durchführung von verschiedenen Sportarten mit Robotern verwendet. Beispielsweise im Roboter-Fußball, dem sogenannten RoboCup. Hier agieren alle Roboter der eigenen Mannschaft interaktiv miteinander oder werden von einem zentralen Computer grob mit Befehlen versorgt, die dann von den einzelnen Robotern verfeinert ausgeführt werden. [Q7]

3. Entwicklung des autonomen mobilen Roboters „Kassiopeia“

3.1 Mechanik

Aufgrund finanzieller Faktoren habe ich mich bei der Mechanik für ein Chassis entschieden, welches direkt aus den einzelnen Platinen besteht, welche durch Distanzbolzen verbunden werden. Für die Fortbewegung wurden auf der untersten Platine zwei 12V Getriebemotoren montiert, die es aufgrund der getrennten Ansteuerung ermöglichen, den Roboter auch auf der Stelle wenden zu lassen. Zur Stabilisierung ist auch ein drittes Rad am hinteren Ende des Roboters montiert. Dieser Dreiradantrieb ermöglicht es, dass auch bei leicht unebenem Boden stets alle Räder Bodenkontakt haben.

Die ersten Versuche mit diesem Steuerungskonzept wurden mit modifizierten Servomotoren (Servomotoren aus dem Modellbau, welche so modifiziert wurden, dass sie sich um 360° drehen können) durchgeführt. Diese Lösung erschien anfangs praktisch, da die Stromversorgung über Akkus mit einer Gesamtspannung von 4,8V erfolgen sollte, da diese leicht auf dem Roboter unterzubringen und die Kosten niedrig waren. Allerdings erwiesen sich die Servomotoren nach einiger Zeit als zu leistungsschwach und hatten einen zu hohen Stromverbrauch. Daraufhin wurden 12V Getriebemotoren auf eine neue Platine für den Unterboden installiert, welche anfangs noch mit der 4,8V Spannung der Akkus versorgt wurden.

Als sich nach einiger Zeit gezeigt hatte, dass die Motoren mit dieser Versorgungsspannung kein stabiles Drehverhalten entwickeln konnten, habe ich mich entschlossen einen DC/DC-Wandler (ein Baustein, der eine unstabilisierte Eingangsspannung, hier Gleichstrom (= DC), in eine geregelte Ausgangsspannung, ebenfalls Gleichstrom, umwandelt, ähnlich einem Transformator) einzubauen, welcher die vorhandene Versorgungsspannung für die Motoren auf 12V regelt.

Abb.6 Die modifizierten Servomotoren

3.2 Elektronik

Die Elektronik des Roboters wurde auf verschiedene Platinen aufgeteilt, welche speziell für die jeweiligen Aufgaben ausgerüstet sind. Ich habe mich für diese Lösung entschieden, da dies ermöglicht, dass neue Entwicklungen sehr schnell verwirklicht und implementiert werden können, ohne gleich das komplette Platinenlayout zu verändern. Des Weiteren wird in diesem Konzept ein Multi-Prozessor-System verwendet, welches es ermöglicht, dass den einzelnen Aufgaben, die bewältigt werden müssen, ein höchstes Maß an Rechenkapazität zur Verfügung gestellt wird, da

jede Platine mit mindestens einem Mikrocontroller ausgerüstet ist. Im derzeitigen Entwicklungsstadium besteht der Roboter aus vier Platinen.

Die oberste Platine ist für die Benutzersteuerung ausgelegt. Auf ihr befindet sich ein Mikrocontroller des Typs AT4433, der ein 2-zeiliges Display ansteuert, auf welchem verschiedene Werte angezeigt werden, welche über eine kleine Tastatur ausgewählt werden. Da diese Platine sozusagen eine Verbindung zwischen Mensch und Roboter darstellt, wird diese auch als Human-Interface-Device (kurz: HID) bezeichnet.

Seit der Entwicklung der Platine hat sich diese nicht verändert.

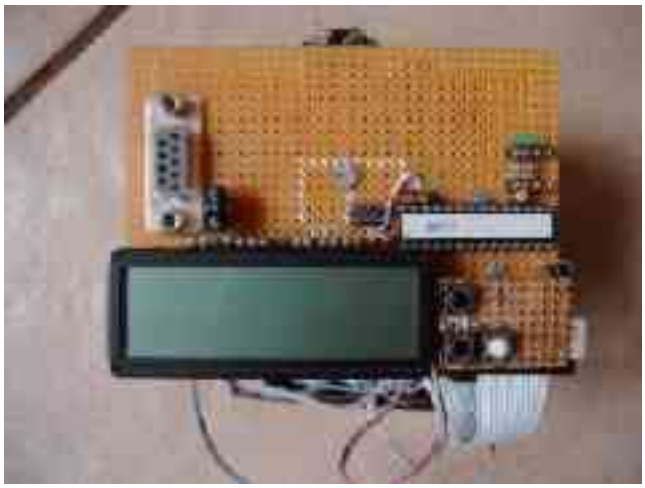
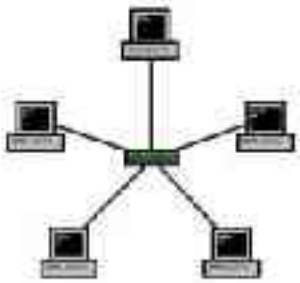


Abb.7 Die HID-Platine

Die zweite Platine bildet sozusagen das „Herz“, sowie „Rückenmark“ des Roboters. Auf ihr befindet sich der MDN, der Master-Data-Node (= HOST) vom Typ Atmega8515. Dieser ist dafür zuständig, dass Roboter interne Kommunikationssystem nach Anfragen auf Kommunikationen eines Controllers im Netzwerk (= CLIENT) zu überwachen. Im Falle einer Anfrage übernimmt der MDN schließlich die Kontrolle über den Kommunikationsprozess über eine globale Taktleitung, welche an jedem Controller zum Zweck der Synchronisation der Kommunikation angeschlossen ist, übernimmt. Des Weiteren stehen für jeden Controller, der an das Kommunikationsnetz angeschlossen wird, zwei Signalleitungen zur Verfügung. Eine Datenleitung, über welche die Daten in beide Richtungen (bidirektional) transferiert werden, sowie eine Leitung, welche vom jeweiligen Controller gesetzt wird, um einen Datentransfer anzufordern (DR= Data request).

Die Topologie des Netzwerkes ist eine Stern-Topologie, welcher sich sternförmig um den MDN aufbaut. Der Vorteil eines solchen Netzwerks ist, dass ein Ausfall eines Clients keinen Einfluss auf den Rest des Netzwerks hat. Sollte allerdings der Host ausfallen, so ist das gesamte Netzwerk betroffen und funktionsunfähig.



Der MDN stellt 12 Anschlüsse für Clients zur Verfügung und ist daher auch für größere Erweiterungen in der Zukunft gerüstet. Das verwendete Netzwerkprotokoll hat den Namen STAR-Bus von mir erhalten, da es sich, wie zuvor beschrieben, sternförmig ausbreitet.

Das „Herz“ wird hierbei durch zwei Quarz-Oszillatoren (8 MHz und 16 MHz) repräsentiert, welche den Takt für die vielen verschiedenen Controller angibt, so dass diese optimal und schnell arbeiten können. Auch diese Signale werden auf den gesamten Roboter verteilt, so dass nur diese zwei Oszillatoren nötig sind, um jeden Controller zu takten.

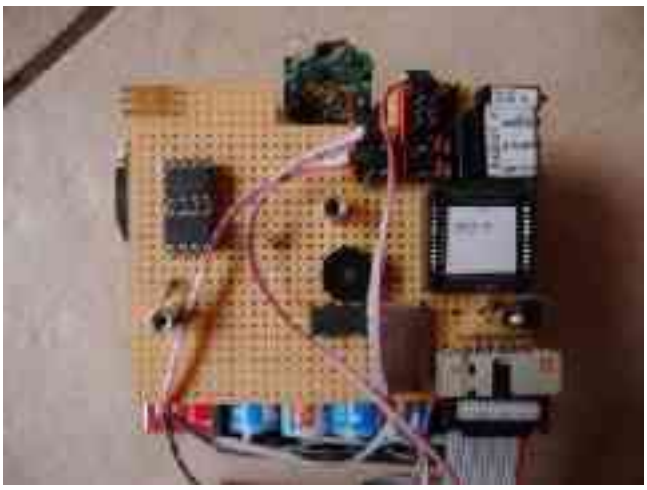


Abb8. Die MDN-Platine

Die dritte Platine enthält den sogenannten VDSP (**V**ideo **D**igitaler-**S**ignal-**P**rozessor). Dies ist ein Atmega128, welcher einer der leistungsstärksten 8-bit Mikrocontroller ist. Sein ohnehin üppiger Arbeitsspeicher von 4kB wird durch einen externen Arbeitspeicher auf 64kB erweitert und bietet somit genug Platz für das Speichern und Verarbeiten von Bildern. Für zukünftige Erweiterungen sind zwei zusätzliche Speicherbänke für Speicherbausteine verfügbar, so dass maximal 184kB zur Verfügung stehen. Für den Programmspeicher stehen 128kB zur Verfügung. Dies erlaubt es, auch relativ komplexe Bildverarbeitungsalgorithmen zu implementieren. Als Kamera wird hier eine GameBoy-Kamera verwendet, welche eine Auflösung von 128x128 Bildpunkten (= Pixel) mit einer Grauabstufung von 8-bit (= 256 Grauwerte) aufweist. Über eine serielle Schnittstelle können Bilder an einen Computer geschickt werden.

Abb9. Die VDSP-Platine

Die vierte und zugleich unterste Platine enthält den Motorcontroller (MCU), welcher den Motortreiberchip (L293d) ansteuert, die Infrarot-Abstandswarner und die Kollisionsschalter auswertet. Des Weiteren liest er über das Netzwerk von allen Controllern Motorkommandos ein und führt diese dann entsprechend aus. Auf dieser Platine sind auch die Motoren „angebunden“. Diese werden über den zuvor erwähnten DC/DC-Wandler mit der erforderlichen Versorgungsspannung von 12V versorgt. Die Versorgungsspannung des Roboters (4,8V) wird aus vier Akkus bezogen, welche auf dieser Platine angeschlossen werden. Des Weiteren befindet sich hier der Hauptschalter,

über den der Roboter an- und abgeschaltet wird.

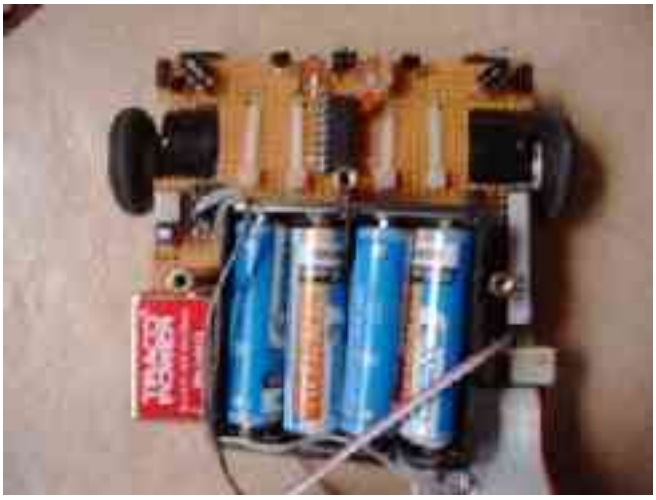


Abb.10 Die MCU-Platine

Um die Sicherheit des Roboters, sowie seiner Benutzer, zu gewährleisten, befindet sich auf jeder Platine ein Not-Aus-Schalter, welcher später näher erläutert wird.

Zum Platinenlayout ist noch zu sagen, dass alle Ebenen aus Lochrasterplatinen bestehen und die Verbindungen zwischen den einzelnen Bauteilen über Kupfer-Lack-Drähte hergestellt wurden. Diese Kupfer-Lack-Drähte haben den Vorteil, dass sie sehr dünn sind und daher auf der Unterseite der Platinen nur sehr wenig Platz einnehmen. Allerdings kann es durchaus vorkommen, dass mit zunehmender Zahl an Verbindungen auch die Übersichtlichkeit abnimmt, was eventuelle Reparaturen erschwert.

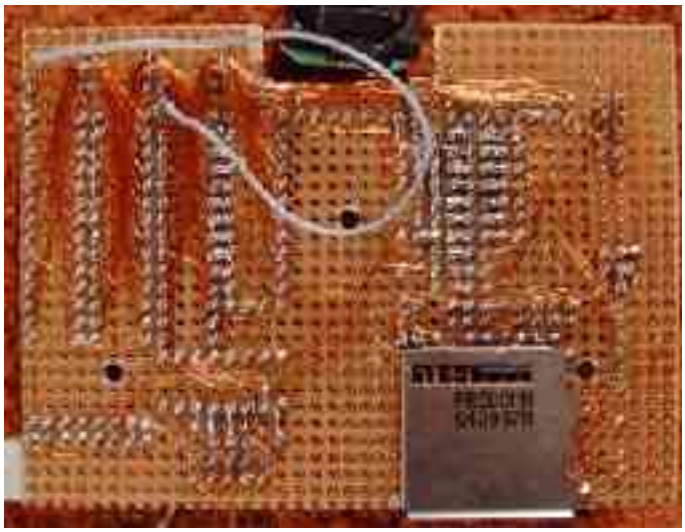


Abb.11 „Verkabelung“ der einzelnen Komponenten

Alle verwendeten Mikrocontroller stammen von der Firma ATMEL [Q8] und wurden bei Conrad Elektronik [Q9], der Firma Reichelt [Q10] und Schuricht [Q11] bestellt. Diese Bezugsquellen treffen auch auf alle anderen Bauteile zu. Diese große Anzahl an Bezugsquellen ist aufgrund der unterschiedlichen Verfügbarkeit der Bauteile erforderlich.

3.3 Software

Die Software des Roboters ist aufgrund der Verteilung auf mehrere Controller nicht im Ganzen zu Beschreiben und relativ komplex. Daher sollen nur einige ausgewählte Gebiete beschrieben werden.

3.3.1 Controller-Modi

In Zukunft soll jeder Controller, mit Ausnahme des MDN und des HID, drei Controller-Modi besitzen. Diese sind Reset, Pause und Operate. Der jeweilige Modus ist im MDN für jeden Controller gespeichert und wir von diesen entsprechend abgefragt. Diese Modi werden unter dem Begriff „spezial Mikro-Controller commands“ (kurz „sµCc“) zusammengefasst.

Der Modus „Reset“ initialisiert der jeweilige Controller die von ihm verwaltete Hardware, sowie Variablen in der Software und Software-Treiber, welche später näher beschrieben werden. Nach einem erfolgten Reset des Controllers setzt sich dieser automatisch in den „Pause“ - Modus, indem er an den MDN den entsprechenden Wert schickt.

Befindet sich ein Controller nun im „Pause“ - Modus, so führt dieser nur die wichtigsten Aktionen aus, die bisher allerdings undefiniert sind. Daher wird im „Pause“ - Modus bisher nur gewartet, bis der Controller arbeiten muss.

Der „Operate“ - Modus veranlasst den Controller schließlich dazu sein reguläres Programm auszuführen.

Derzeit ist dieses Konzept in den Controllern VDSP und MCU integriert.

Nach dem Anschalten des Roboters ist jeder Controller automatisch im „Reset“ – Modus. Der HID fordert nun auf dem Display auf, den Roboter mit einem Tastendruck auf die weiße Eingabetaste, den Roboter zu starten, da sich die Controller nun im „Pause“ - Modus befinden. Nach dem Tastendruck gibt der MDN jedem Controller das Kommando, seine Arbeit aufzunehmen.

3.3.2 Not-Aus

Die Not-Aus-Funktion ist sicherheitstechnisch eigentlich nicht erlaubt, da es sich hierbei um eine indirekte Abschaltung der Roboterfunktionen handelt und nicht etwa sofort die Stromversorgung zu den Motoren unterbrochen wird. Hierbei wird vielmehr über die Controller-Modi veranlasst, dass jeder Controller in den „Pause“ - Modus wechselt.

Da der VDSP derzeit ohne relevante Funktion ist, sind die Auswirkungen eines Not-Aus bei diesem nicht zu beobachten. Der MCU allerdings stoppt sofort die Motoren und wertet die Motorkommandos anderer Controller nicht aus, so dass der Roboter sofort stehen bleibt und somit bei Fehlfunktionen keine Gefahr darstellt.

3.3.3 Software-Treiber

Um die Programme für die einzelnen Controller möglichst übersichtlich zu halten, habe ich mich entschlossen für jede externe Hardware, welche durch einen Controller verwaltet wird, einen Software-Treiber zu schreiben, welcher speziell für die jeweiligen Hardwarebausteine, wie GameBoy-Kamera oder Motortreiberchip (L293d), zugeschnitten sind und sowohl grundlegende Funktionen, wie das Initialisieren des Treibers und der zugehörigen Hardware, als auch spezielle Funktionen enthalten.

Eine solche Aufteilung in Software-Treiber und Hauptprogramm hilft, den Programmcode übersichtlicher zu machen und ist auch bei der Fehlersuche sehr nützlich.

Die nebenstehende Abbildung zeigt deutlich die Aufteilung eines Projekts, hier der Programmcode für den VDSP, in das Hauptprogramm (M2-VDSP.asm) , die Software-Treiber (M64282FP.inc [GameBoy-Kamera], SPI-Treiber.inc [Software-Treiber für einen 3-Draht-Bus], PCF8833.inc [Handydisplay], STAR.inc [Netzwerkprotokoll], vision_basics.inc [Bildverarbeitungsroutinen]) und in Dateien, welche Konstanten enthalten, die für die Software-Treiber benötigt werden (PCF8833-CommandSummary.inc [alle Kommandos für die Ansteuerung des Handydisplays], STAR-Database.inc [Adressen für die einzelnen Werte im Netzwerk], suCc.inc [Auflistung der Controller-Modi]).

Auf den Seiten 14 - 17 ist der Software-Treiber für den STAR-Bus als Beispiel angehängt.

3.3.4 Lernen

Das Lernverhalten des Roboters beschränkt sich auf ein sehr einfaches und leicht nachzubildendes Verhalten. Es handelt sich hierbei um eine Art klassischer Konditionierung.

Die klassische Konditionierung zeichnet sich dadurch aus, dass man versuch, das Verhalten, welches ein Tier auf einen bekannten Gegenstand bzw. eine bekannte Situation zeigt, mit einem neutralen Gegenstand bzw. einer neutralen Situation, zu verknüpfen, so dass bei Letzterem schließlich das selbe Verhalten zu beobachten ist, wie bei dem Bekannten.

Die Rolle der bekannten Situation nehmen im Falle meines Roboters die Kollisionssensoren ein, welche ein „angeborenes“ Verhalten besitzen (Ausweichen von erkannten Hindernissen). Die Infrarot-Abstandsensoren lösen jedoch kein „angeborenes“ Verhalten aus. Dies muss der Roboter erst erlernen, indem er die Infrarot-Sensoren mit den Kollisionssensoren verknüpft. Dies geschieht nach folgendem Muster:

Wurde ein Hindernis durch einen der Kollisionssensoren erkannt, dann wird in einer Unterfunktion geprüft, ob gleichzeitig zu diesem Ereignis auch einer (oder mehrere) IR-Sensoren ausgelöst wurden. Ist dies der Fall, dann wird eine Variable im Arbeitsspeicher des Roboters, welche dem entsprechenden IR-Sensor zugeordnet ist, inkrementiert (um eins erhöht).

Sollte nun zu einem späteren Zeitpunkt einer der IR-Sensoren aktiviert werden, ohne, dass ein Kollisionssensor ein Hindernis meldet, so wird in einer anderen Unterroutine geprüft, ob der Wert der Variable, die zu dem entsprechenden Sensor gehört, einen Schwellenwert überschreitet. Ist dies der Fall, so wird das „angeborene“ Verhalten des Kollisionssensors aktiviert und der Roboter weicht dem Hindernis schon aus, bevor es zu einer Kollision kommt. Man könnte also sagen, dass der Roboter gelernt hat.

Anhang

Assembler-Code des Software-Treibers für den STAR-Bus. Der hier gezeigte Treiber ist für die Verwendung in den Client-Controllern gedacht.

```
; ----STAR----STAR----STAR----STAR----STAR----STAR----STAR----STAR----STAR----
; Name:             STAR-Bus-Treiber
; Autor:            Michael Strosche
; Datum:            24.08.2004
; Version:          1.0

; benötigte Variablen:      r16       (z.B. .def r16 = r16)
;                           r17       (z.B. .def r17 = r18)

; folgende Konstanten müssen im Hauptprogramm individuell gesetzt werden:

; DR-Pin:           oDR  für den PORT, an dem die Leitung angeschlossen wird (z.B. .set oDR = PORTC)
;                   dDR  für den DDR, an dem die Leitung angeschlossen wird (z.B. .set dDR = DDRC)
;                   DR   für den Pin, an dem die Leitung angeschlossen wird (z.B. .set DR = 0)
; DATA-Pin:         oDATA  für den PORT, an dem die Leitung angeschlossen wird (z.B. .set oDATA = PORTC)
;                   iDATA  für den PIN, an dem die Leitung angeschlossen wird (z.B. .set iDATA = PINC)
;                   dDATA  für den DDR, an dem die Leitung angeschlossen wird (z.B. .set dDATA = DDRC)
;                   DATA   für den Pin, an dem die Leitung angeschlossen wird (z.B. .set DATA = 1)
; CLK-Pin:          STARpClk  für den PIN, an dem die Leitung angeschlossen wird (z.B. .set STARpClk = PINC)
;                   STARdClk  für den DDR, an dem die Leitung angeschlossen wird (z.B. .set STARdClk =
DDRC)
;                   STARclk   für den Pin, an dem die Leitung angeschlossen wird (z.B. .set STARclk = 0)

; Speicheradressen für Treiberdaten im SRAM:
;           ComStatusAdr            (z.B. .set ComStatusAdr = 0x61)
;           DataAdr                 (z.B. .set DataAdr = 0x62)
;           ComAdr                  (z.B. .set ComAdr = 0x63)
;           loopDataAdr             (z.B. .set loopDataAdr = 0x64)

; des Weiteren können die zu übermittelnden Kommandos eigenen Namen zugewiesen werden.
; Es gibt insgesamt 128 Kommandos (Com0000 bis Com127)
;  Beispiel: .set MotorKommando = Com053



; ComStatus-byte
;           Bit-Nr.       | 7 | 6 | 5 | 4 |3 | 2  | 1   |      0
```

```
;           -------------+----+--+--+--+-+----+-----+---------
;           Value        |128|64|32|16|8|  4   | 2   |      1
;           -------------+----+--+--+--+-+----+-----+---------
;           Discription|     |   |   |   |  |C/D|R/W|Com-Error

; STAR-Bus-Treiber

;  Konstanten des STAR-Bus-Treiber
.set        RetryCyclesDR           =          90
.set        MaxErrorNum             =          10
.set        DataLengthData          =          8
.set        DataLengthComand        =          7
.set        DataBit                 =          7
.set        ComBit                  =          6

;  Bus-Leitungen initialisieren
initSTAR:
            ; Clk-Eingang
            cbi         STARdClk,       STARclk
            cbi         STARpClk,       STARclk

            ; DR-Ausgang
            sbi         dDR,            DR
            sbi         oDR,            DR

            ; DATA-Ein-(/Aus-)gang
            cbi          dDATA,                 DATA
            cbi         oDATA,                 DATA

            ret

; read-bit setzen
DATAr:
            sts         ComAdr,         r16

            clr         r16

            rjmp        useSTAR
```

```
; write-bit setzen
DATAw:
        sts     ComAdr,         r16
        sts     dataAdr,                r17

        clr     r16
        sbr     r16,            2

        rjmp    useSTAR

; STAR-Bus benutzen um neue Daten zu transferieren
useSTAR:
        sts     ComStatusAdr,   r16

        cbi     oDR,            DR

        cbi     dDATA,                  DATA
        ldi     r16,            RetryCyclesDR
waitCom:
        sbic    iDATA,          DATA
        rjmp    handleData
        rjmp    waitCom

noCom:
        sbi     oDR,            DR

        ret

handleData:
        sbi     oDR,            DR

        ldi     r17,            DataLengthComand

sendRW:
        sbi     dDATA,                  DATA
        cbi     oDATA,                  DATA
```

```
        lds     r16,            ComStatusAdr

        sbrc    r16,            1
        sbi     oDATA,          DATA

waitRWclkHIGH:
        sbis    STARpClk,       STARclk
        rjmp    waitRWclkHIGH

waitRWclkLOW:
        sbic    STARpClk,       STARclk
        rjmp    waitRWclkLOW

        cbi     dDATA,          DATA
        cbi     oDATA,          DATA


loopCom:
        sts     loopDataAdr,    r17

        sbi     dDATA,          DATA
        cbi     oDATA,          DATA

        lds     r17,            ComAdr

        sbrc    r17,            ComBit
        sbi     oDATA,          DATA

        lsl     r17
        sts     ComAdr,         r17

waitCOMclkHIGH:
        sbis    STARpClk,       STARclk
        rjmp    waitCOMclkHIGH

waitCOMclkLOW:
        sbic    STARpClk,       STARclk
```

```
        rjmp    waitCOMclkLOW

        cbi     dDATA,          DATA
        cbi     oDATA,          DATA

        lds     r17,            loopDataAdr
        dec     r17
        brne    loopCom


        ldi     r17,            DataLengthData

loopData:
        sts     loopDataAdr,    r17

        lds     r16,            ComStatusAdr
        sbrs    r16,            1
        rjmp    readData

writeData:
        sbi     dDATA,          DATA
        cbi     oDATA,          DATA

        lds     r17,            DataAdr

        sbrc    r17,            DataBit
        sbi     oDATA,          DATA

        lsl     r17
        sts     DataAdr,        r17

waitDATAWclkHIGH:
        sbis    STARpClk,       STARclk
        rjmp    waitDATAWclkHIGH

waitDATAWclkLOW:
        sbic    STARpClk,       STARclk
```

```
        rjmp    waitDATAWclkLOW

        cbi     dDATA,          DATA
        cbi     oDATA,          DATA

        rjmp    nextBit

readData:

waitDATARclkHIGH:
        sbis    STARpClk,       STARclk
        rjmp    waitDATARclkHIGH

        lds     r17,            DataAdr
        lsl     r17

        sbic    iDATA,          DATA
        sbr     r17,            1

        sts     DataAdr,        r17

waitDATARclkLOW:
        sbic    STARpClk,       STARclk
        rjmp    waitDATARclkLOW

nextBit:
        lds     r17,            loopDataAdr
        dec     r17
        brne    loopDATA

exitCom:
        clr     r16
        lds     r17,            DataAdr

        ret
; ---------------------------------------------------------------------------
```

Listing 2

Quellennachweis:

[Q0] http://asimo.honda.com

[Q1] http://www.roboterwelt.de/info/assimov.

[Q2] Artikel: „Die Gesetze der Robotik nach Isaac Asimov“.

Online im Internet: <http://www.computerbase.de/lexikon/Robotik>

(abgerufen am 15.Mai 2005)

[Q3] Dr.-Ing. Hartmut Weber: „Navigation einfacher mobiler Roboter mit Handskizzen“

Spektrum der Wissenschaft, Dossier 4/1998, Seite 40ff.

[Q4] „Praktische neuronale Netzwerke“ Teil 1 bis 4

Elektor 3-6/2003

[Q5] Artikel: „Neuronale Netze (Einführung)“. Stand: 05.Juli 2003.

Online im Internet: URL < http://www.elektronik-projekt.de/include.php?path=content/articles.php&contentid=47&PHPKITSID=e81b9f11ed1957cc4f3afcf901041b7d >

(abgerufen am 15. Mai 2005)

[Q6] Artikel: „Robots“. Stand: 29. November 2004.

Online im Internet: URL <http://www.cirg.reading.ac.uk/robots/index.htm>

(abgerufen am 15. Mai 2005)

[Q7] http://www.robocup.org/

[Q8] http://www.atmel.com

[Q9] http://www.conrad.de

[Q10] http://www.reichelt.de

[Q11] http://www.schuricht.de

Bildnachweis:

Abb.1 http://www.kmg.de/images/3_dienstleistungen/33_sanierung/331_reparatur/3315_sidefit/331561.jpg

Abb.2 Spektrum der Wissenschaft, Dossier 4/1998, Seite 26, Bild 1

Abb.3 http://marsrovers.jpl.nasa.gov/gallery/spacecraft/images/mer1_021003_br.jpg

Abb.4 http://upload.wikimedia.org/wikipedia/de/thumb/c/c4/800px-Predator.300pix.jpg

Abb.5 http://asimo.honda.com/ASIMO_DCTM/News/images/highres/18_ASIMO_Handshake.jpg

Die Abbildungen 6 – 11 sind Aufnahmen des Roboters mit einer Digitalkamera.