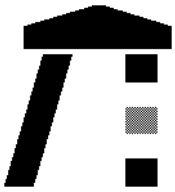


Universität Kaiserslautern
Fachbereich Informatik

Robotik-Praktikum

SS 1999

AG Robotik & Prozeßrechentechnik
Prof. Dr. E. von Puttkamer

Praktikumsbetreuung durch Michael Kasper
Teile der Ausarbeitung von Uwe Zimmer

Inhalt

1. Vorbemerkungen 3

2. Handhabungsautomat 4

2.1 Modulare Programmierung in Pascal 5

2.2 Der Roboter Mitsubishi Move-Master-EX 6

2.3 Echtzeitprogrammierung ..... 10 — *Aufgabe 1*

3. Autonome Mobile Roboter 13

3.1 Der Roboter Rug Warrior 14

3.2 Library-Funktionen ..... 16 — *Aufgabe 2.1*

3.3 (v,ω)-Schnittstelle ..... 24 — *Aufgabe 2.2*

3.4 Motorregelung ..... 27 — *Aufgabe 2.3*

3.5 Kontrollstrukturen ..... 31 — *Aufgabe 2.4*

3.6 Sensordatenverarbeitung ..... 34 — *Aufgabe 2.5*

4. Anhang A-1

4.1 Macintosh Tastaturbelegung A-1

4.2 Pascal Hilfsroutinen A-2

1. Vorbemerkungen

Das Praktikum umfaßt zwei Versuchsblöcke mit je einem Arbeitsplatz sowie weiteren Rechnern für die Programmierung. Die Arbeitsplätze stehen jeder Gruppe einen halben Tag in der Woche *reserviert* zur Verfügung. In der übrigen Zeit gilt die Regel "Wer zuerst kommt, malt zu erst".

Zeitrahmen

Zu jeder Aufgabenstellung ist eine Zeitspanne angegeben, die als Orientierung dienen soll. Diese Zeit ist nicht zwingend einzuhalten. Das Praktikum geht bis zum Ende der Vorlesungszeit und solange werden auch Termine angeboten. Achten Sie allerdings darauf, daß Sie nach Möglichkeit keine andere Gruppe behindern, die noch auf Ihren Arbeitsplatz wartet.

Besprechung

Alle zwei Wochen - bei Bedarf auch häufiger - wird eine Praktikumsbesprechung stattfinden, in welcher Fragen geklärt und Lösungsansätze diskutiert werden sollen. Die genauen Inhalte werden von mal zu mal bekanntgegeben, wobei auch regelmäßig Präsentationen von den Gruppen zu erstellen sind.

Die funktionierenden Lösungen sind jeweils vorzuführen (zuerst den Hiwis und anschließend dem betreuenden Assistenten). Die nach den Versuchen abzugebende Dokumentation kann sich i.a. auf gut kommentierte Programm-Listings beschränken. Bei Verwendung von mehreren Modulen sollten jedoch auch die Modul-Abhängigkeiten (Import-Beziehungen) schriftlich (graphisch) festgehalten werden.

Scheine

Am Ende des Praktikums (voraussichtlich in der letzten Vorlesungswoche) findet für jede Gruppe ein Kolloquium statt. Der Erhalt des Praktikumscheins hängt von der erfolgreichen Durchführung der Versuche und vom Kolloquium ab.

Viel Erfolg und Spaß bei der Arbeit!

2. Versuchsblock Handhabungsautomat

Die Praktikumsversuche sollten einen Einblick in die Praxis vermitteln (wie der Name sagt). Damit verbunden ist aber auch die Möglichkeit Fehler zu begehen - insbesondere bei diesem Versuch mit einem Handhabungsautomaten. Um die Auswirkungen dieser (unvermeidlichen) Fehler in finanzierbaren Grenzen zu halten, folgen hier einige Sicherheits-Regeln, die unbedingt zu beachten sind (Nichtbeachtung führt zum Ausschluß vom Praktikum!).

Sicherheits-Regeln:

1. Der Roboter ist während des gesamten Versuchsablaufs "im Auge " zu behalten.
2. Sollte sich irgendetwas Unvorhergesehenes ereignen, ist sofort einer der drei Notaus-Schalter zu betätigen.
4. Nach einer Störung (Notaus oder Fehler-Anzeige am Roboter) ist Ihr Programm in jedem Fall neu zu starten (dadurch wird der Roboter erneut initialisiert).
5. Während des Austestens der Programme ist höchstens die Robotergeschwindigkeit 3 erlaubt.

2.1 Modulare Programmierung in Pascal

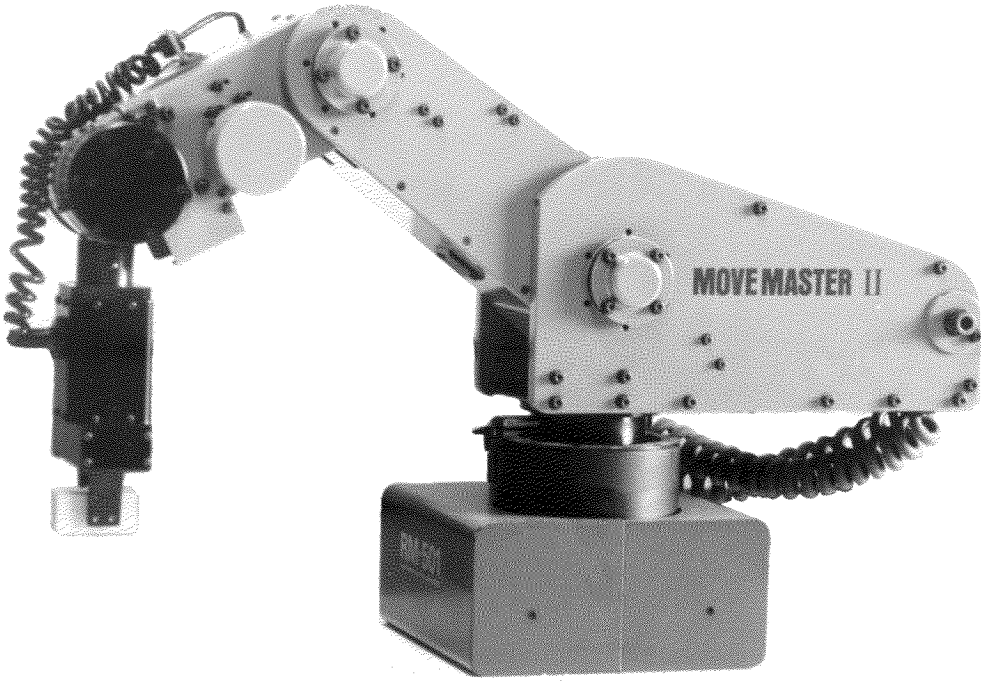
Als Implementierungssprache des ersten Versuchsblocks wurde Pascal gewählt. Die hier verwendete Sprach-Version verbindet die Schlichtheit der prozeduralen Programmiersprachen (Standard-PASCAL) mit den Vorteilen der modularen Programmierung (Unit-Konzept). Die modulare Programmierung bildet hier die Grundlage des methodischen Programmierens, dessen wichtigste Prinzipien hier erwähnt seinen:

- *Hierarchiebildung,*
  d.h. die Zerlegung eines Programmentwurfs in eine Hierarchie von Modulen mit zunehmendem Abstraktionsgrad. Eine strenge Hierarchie wird dann erreicht, wenn sämtliche Module immer nur Funktionen von Modulen niedrigerer Abstraktion importieren.
- *Geheimnisprinzip,*
  d.h. ein Modul gibt seiner Umgebung nicht mehr Information preis als nötig.
- *Prinzip der schrittweisen Verfeinerung,*
  d.h. das schrittweise Vordringen vom Abstrakten (dem Modell der Lösung) zum Konkreten (Top-Down-Entwurf).
- *Strukturierte Programmierung,*
  d.h. Verwendung weniger, sorgfältig ausgewählter algorithmischer Grundstrukturen.
- *Problemangepaßte Datentypen und Datenstrukturen,*
  d.h. Strukturieren der Daten nach den Gesichtspunkten: Übersichtlichkeit und Effizienz.
- *Standardisierung,*
  d.h. jede Unit sollte einem gewissen Standard in Form, Gliederung, Größe und Namensgebung genügen. Bei der Namensgebung ist natürlich auf Verständlichkeit zu achten (nicht: "hilf", "Wert" oder "i" verwenden).

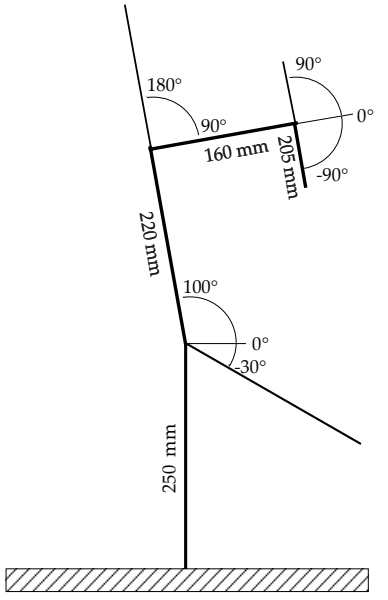
Speziell bei diesem Versuch und der damit verbundenen Echtzeitprogrammierung ist auf eine durchdachte (nach Möglichkeit strenge) Hierarchiebildung zu achten.

2.2 Der Roboter: Mitsubishi Move-Master-EX

Der Roboter: Mitsubishi Move-Master-EX ist ein Gleichstrom-Servomotor-gesteuerter Roboter mit fünf Freiheitsgraden. Die Auflösung der Encoder beträgt 0.02 Grad/Puls oder besser. Der Roboter läßt sich unmittelbar nur durch Angabe der Impulse für jedes Gelenk steuern. Um Ihnen eine mühsame Umrechnung zu ersparen und das Gesamtsystem sicherer zu gestalten steht eine Pascal-Schnittstelle ("Robbi") zur Verfügung, die Ihnen die Steuerung des Roboters durch kartesische und zylindrische Koordinaten erlaubt.



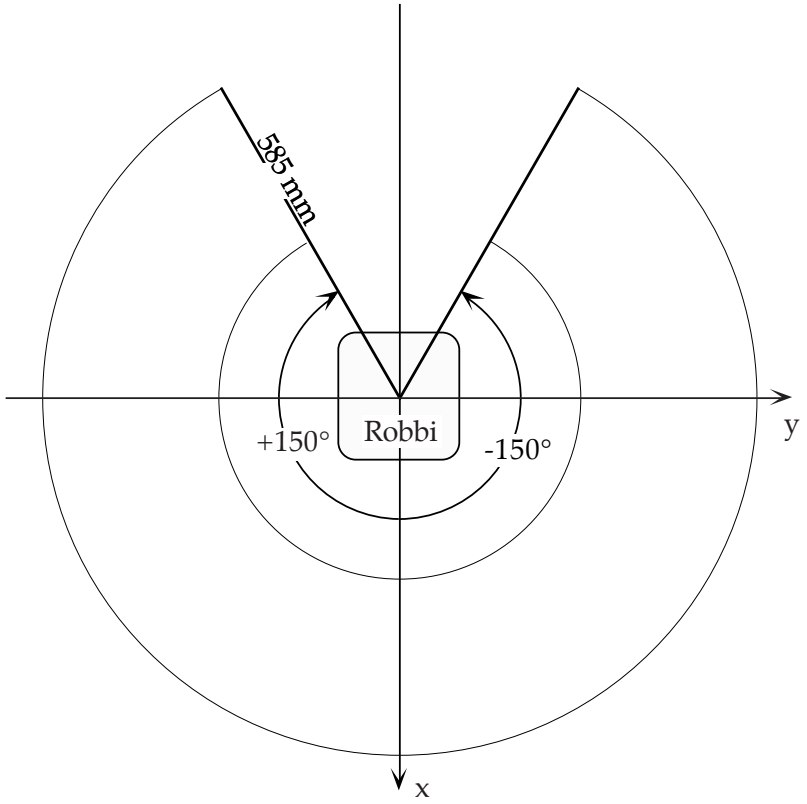
Die Orientierung der Koordinatenachsen, sowie die Nullstellung des Mittelpunktwinkels (Waist) ist der auf der nächsten Seite dargestellten Sicht von oben zu entnehmen. Die Koordinate "z" bezeichnet die Höhe des Greiferzentrums über der Tischoberfläche. Verfahren Sie das Greiferzentrum, um etwas aufzunehmen in der Höhe z = 18 mm, und um etwas niedrig über den Tisch zu bewegen in der Höhe z = 20 mm. Beachten Sie, daß das Schnittstellen Modul "Robbi" keine Anfahrpunkte niedriger als 18 mm akzeptiert.



Seiten-Ansicht

Koordinaten

Die Parameter "Inclination" und "Rotation" bezeichnen die Neigung und die Rotation des Handgelenkes in der Greifposition. Die Position des Greiferzentrums wird von diesen beiden Parametern nicht beeinflußt. Die Kombination: Inclination und Rotation = 0° bezeichnet die Stellung der Hand senkrecht zum Tisch (Hand greift nach unten) und die Ausrichtung der Greifer parallel zur y-Achse.



Drauf-Sicht

Im einzelnen exportiert die Unit "Robbi" die folgenden Typen und Prozeduren.

- ... die Typen "`mm`" und "`Degree`", die beide als REAL definiert sind.
- ... die parameterlosen Prozeduren

```
SynchronousMode
AsynchronousMode
WaitForRobbi
```

  Es gibt zwei verschiedene Betriebsarten, in denen das Modul mit dem Roboter kommunizieren kann. In der synchronen Betriebsart blockieren sämtliche Roboteraufrufe den Programmfluß, bis der Befehl vollständig abgearbeitet ist. In der asynchronen Betriebsart wird die Kontrolle unmittelbar wieder zurückgegeben, auch wenn der Roboter seine Bewegung noch nicht vollständig ausgeführt hat. Um sich dennoch an kritischen Punkten mit dem Roboter synchronisieren zu können, kann die Prozedur "`WaitForRobbi`" verwendet werden, die solange blockiert, bis der Roboter alle Aktionen beendet hat. Nach dem Start befindet sich der Roboter in der Betriebsart "Asynchron"!

- ... die parameterlose Prozedur

```
MoveToHome
```

  die den Roboter in die Ausgangsstellung ("ausgestreckter" Arm nach vorne) zurück bringt.

- ... die Funktionen

```
MoveToCartesian (  x, y, z          : mm;
                   WristInclination,
                   WristRotation    : Degree)
      : Boolean
                                    und

MoveToCylindrical(Waist             : Degree;
                   Radius, z        : mm;
                   WristInclination,
                   WristRotation    : Degree)
      : Boolean
```

  die das Greiferzentrum in eine durch 5 Parameter festgelegte Position verfahren. Ist das Anfahren der gewünschten Position nicht möglich (Position unterhalb der Tischfläche, außerhalb des physikalisch erreichbaren Bereichs, ...), so liefern die Funktionen "`False`" zurück.

  Achtung! Beachten Sie, daß der Greifer nur um ca. ±172° gedreht werden kann, so daß Sie zum Drehen von Werkstücken um 180° möglicherweise umgreifen müssen.

- ... die parameterlosen Prozeduren

```
OpenGrip
CloseGrip
```

  zum Öffnen und Schließen des Greifers an der momentanen Position.

In die Tischoberfläche des Versuchsaufbaus sind fünf Photosensoren (Reflexlichtschranken), drei Schalter und vier LEDs eingelassen. Die Photosensoren dienen zum Detektieren von Teilen an den jeweiligen Positionen. Die Schalter sind den drei Photosensoren an den Übergabepunkten D, E und F (Ausgang 1,2,3) zugeordnet und dienen zur Flußsteuerung. Die erste LED ist dem Übergabepunkt A (Eingang) zugeordnet und dient dort der "Überlastungsanzeige" der simulierten Roboterzelle. Die drei übrigen LEDs geben den Zustand der o.g. Schalter wieder.

Modul DiditalIO

Die genaue Semantik dieser Elemente wird später erläutert. Die Schnittstelle zu diesen Elementen realisiert das Modul "`DigitalIO`", welches die hier aufgeführten Konstanten, Typen, Prozeduren und Funktionen zur Verfügung stellt:

- ... die Konstanten:

  ```
  SensorA, SensorC, SensorD, SensorE, SensorF,
  ControlD, ControlE, ControlF,
  ```

  welche die Photosensoren und die Schalter bezeichnen.

- ... die Typen:

  ```
  SensorState   = (Light, Dark)
  ControlState  = (Stop, Go)
  ```

  welche die möglichen Stellungen der Schalter und Photosensoren bezeichnen. (Dark = Klotz liegt auf Photosensor)

- ... die Prozedur:

  ```
  SetInFlowControl (FlowControl : ControlState)
  ```

  welche die Flußkontrollanzeige am Übergabepunkt A schaltet. (Also die LED am "Eingang" ein- oder ausschaltet.)

- ... die Funktionen:

  ```
  GetLightSensor    (SensorNr   : LightSensorRange)
                : SensorState
  ```

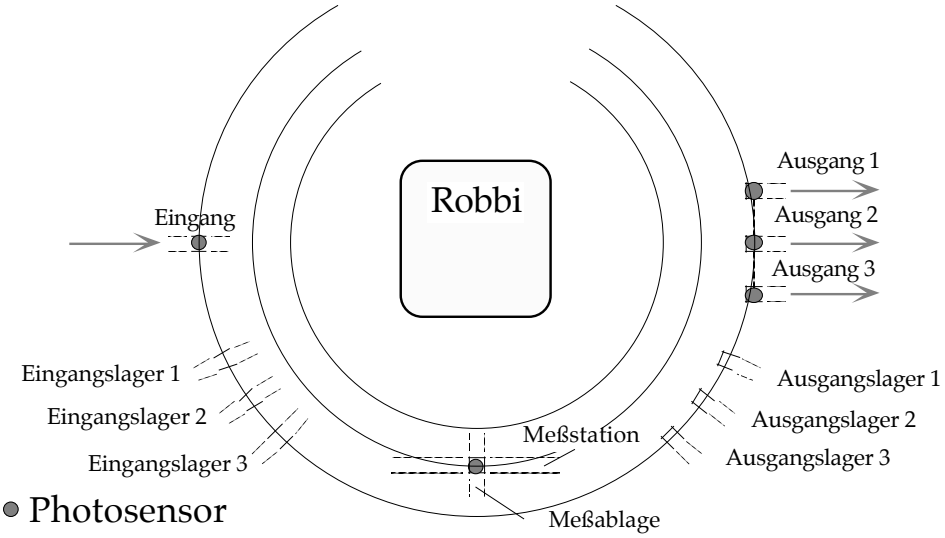
  und

  ```
  GetOutFlowControl  (ControlNr  : FlowControlRange)
                : ControlState
  ```

  welche die Schalterstellungen und die Photosensoren abfragen. Als Parameter sind hier nur die exportierten Konstanten erlaubt.

2.3 Echtzeitprogrammierung

Der Roboter sei im folgenden in eine Verarbeitungsgruppe (Pipeline) integriert. Die Aufgabe des Roboters besteht darin, Teile (Holzklötze) am Eingang entgegenzunehmen, sie nach der Länge in 3 Gruppen zu sortieren und sie an den Ausgängen einer nachfolgenden Einheit zur Verfügung zu stellen.



Roboter-Positionen

Punkt	x [mm]	y [mm]	Waist [°]	Radius [mm]	Inclination [°]	Rotation [°]
Eingang	7,8	-320			-2	90
Meßstation	310	0			-2	-90
Meßablage	310	0			-2	0
Ausgang 1	-50,8	310			0	-90
Ausgang 2	17,9	310			0	-90
Ausgang 3	87,7	310			0	-90
Eingangslager 1			64	377	0	64
Eingangslager 2			54	377	0	54
Eingangslager 3			44	377	0	44
Ausgangslager 1			-62,5	362	0	-62,5
Ausgangslager 2			-52,5	362	0	-52,5
Ausgangslager 3			-42,5	362	0	-43

Roboter-Positionen (Koordinaten)

Messen

Das Vermessen der Teile soll durch systematisches Positionieren über dem Photosensor C an der Meßstation erfolgen (z.B. binäre Suche). Die Übergabepunkte "Eingang" sowie "Ausgang 1,2,3" sind ebenfalls mit Photosensoren ausgestatten.

Die Verarbeitungspipeline, deren Sortierkomponente hier zu implementieren ist, hat keinen festgelegten Arbeitstakt. Die Übergabe zwischen den Komponenten erfolgt asynchron, d.h. es können zu jedem beliebigen Zeitpunkt Teile angeliefert bzw. abgenommen werden. Dies hat zur Folge, daß sich jede Komponente den variablen Taktraten an ihren Zu- und Ablieferungspunkten anpassen muß, so auch unser Roboter. Um einen möglichst großen Durchsatz der Verabeitungskolonne zu gewährleisten müssen zwei Maßnahmen getroffen werden:

i. **Einrichten von Puffern**

ii. **Definition von Prioritäten**

Puffer

(zu i.) Zwischen dem Eingang und der Meßstation sind mindestens drei Ablageplätze für angelieferte Teile einzurichten (z. B. die Punkte: Eingangslager 1, 2, 3). Ebenso zwischen der Meßstation und den Ausgängen (z. B. die Punkte: Ausgangslager 1, 2, 3), wobei hier die Teile gestapelt werden sollen (Höhe z.B. bis 3 Klötze), da Länge und Position an dieser Stelle bekannt sind. Kann das aktuell gegriffene Teil direkt weiterverarbeitet werden, so sind die Zwischenlager natürlich zu übergehen.

Prioritäten

(zu ii.) Die Prioritäten innerhalb der Roboterstation sind wie folgt zu vergeben (höchste Priorität = 1):

1. Priorität:
Unterbrechung des Arbeitsablaufs durch den Bediener (Tastendruck).
Dies könnte zum Beispiel nötig werden, falls eine Störung zu beheben oder eine Werkzeugwechsel vorzunehmen ist. Dazu ist der Roboter in die Home-Position zu verfahren.

2. Priorität:
Abnahme der Teile vom Eingang
Wird z. B. die Übergabe zwischen den Komponenten durch ein Fließband realisiert, so muß nach Ankunft eines neuen Teils der Eingangsbereich innerhalb einer kurzen Frist freigeräumt werden, da sonst weitere ankommende Teile gefährliche Überläufe produzieren könnten.

3. Priorität:
Weiterleitung der vermessenen Teile an die Ausgänge, falls ein entsprechendes Teil zur Verfügung steht.
Diese Maßnahme verhindert, daß nachfolgende Einheiten zum "Däumchen drehen" verurteilt werden, während sich in unserer Station die Teile im Auslieferungslager häufen.

4. und niedrigste Priorität:
Ausmessen der Teile über der Meßstation

Sollte der Roboter aktuell nichts sinnvolles arbeiten können, so ist er in die Home-Position zu verfahren.

Granularität

Der Roboter muß nach jeder einzelnen Bewegung (ein Steuerbefehl!) seine momentane Arbeit zugunsten einer höheren Priorität unterbrechen können! Beachten Sie, daß dabei der u.U. gerade bewegte Klotz zunächst abgesetzt werden muß (d.h. **Reaktionszeiten so kurz wie möglich!**). Ausnahme: An der Meßstation kann zur Ablage die Meßablage verwendet werden.

Zur Verwirklichung dieser Steuerung stehen keine Interrupts zur Verfügung, d.h. die Implementierung muß eine synchrone Steuerung realisieren. Dabei wird jede Aktion in kleine, einen Robotersteuerbefehl beinhaltende Teile zerlegt, nach denen jeweils neu entschieden wird, welche Operation als nächstes ausgeführt werden muß. Als Anregungen zur Implementierungen seien hier die Steuerung durch Auswahl einer Regel (Prolog) oder die Technik des Event-Handling (graphische Oberflächen) erwähnt.

Flußkontrolle

Trotz der oben genannten Maßnahmen kann eine langsame oder blockierte Station die Verarbeitungskolonne zum Überlaufen bringen. Daher existiert an jedem Übergabepunkt zusätzlich ein Signal (LED), welches dem Produzenten "Bitte keine weiteren Teile an diesem Übergabepunkt anliefern!" anzeigt. Dieses Signal muß gesetzt werden, bevor das letzte verarbeitbare Teil vom Konsumenten abgenommen wurde. Am Eingang muß das Signal vom Roboter gesetzt werden. An den Ausgängen müssen die Signale vom Roboter ausgewertet werden. (Verwenden Sie die Routinen: `GetOutFlowControl` sowie `SetInFlowControl`)

Aufgabe 1.a

(1 Woche)

Entwerfen Sie eine Programm- (Modul-) Struktur, die den oben genannten Anforderungen gerecht wird. (Die eigentliche Programmierung geschieht erst in Aufgabe 1.b!) Achten Sie speziell auf eine geschickte Entscheidungsfindung ("What to do next?"). Schreiben Sie Ihren Entwurf in einer geeigneten Form nieder und bereiten Sie einen kurzen Vortrag (10-15 Minuten) vor, in dem Sie ihren Ansatz den übrigen Gruppen vorstellen (am besten anhand eines Fallbeispiels). Diskutieren Sie dabei auch die Lösung "kritischer Fälle".

Aufgabe 1.b

(2-3 Wochen)

Implementieren Sie das in Aufgabe 1.a entwickelte Konzept in der Programmiersprache Pascal unter dem Entwicklungssystem Think-Pascal Version 4.x. (Beachten Sie die Hinweise und Hilfsroutinen im Anhang.)

3. Autonome Mobile Roboter

Das Ziel dieses zweiten Versuchsblocks ist die Einführung in die Prinzipien und Probleme autonomer mobiler Roboter (AMR). Dabei werden insbesondere die Themenbereiche Motorregelung, Sensordatenverarbeitung, Hindernisvermeidung, Exploration und Navigation angesprochen.

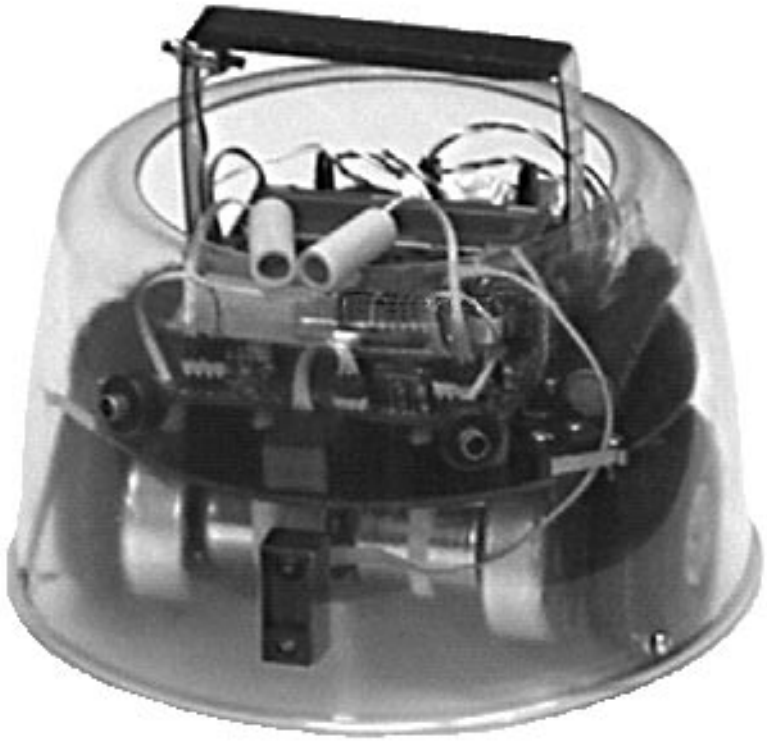
Da in dem Praktikum großer Wert auf einen realen Kontakt zur Robotik gelegt wird, erfolgt die Bearbeitung der Versuchsaufgaben nicht in einer Simulationsumgebung, sondern mit Kleinrobotern. Die dabei gewonnenen Erkenntnisse lassen sich in der Regel direkt auf die derzeit in der Forschung üblichen größeren autonomen Systeme übertragen.

Im Vergleich zu einer reinen Simulation bieten die kleinen Roboter den Vorteil, sich in einer realen Umgebung zu bewegen. Das gilt insbesondere für die generellen Probleme autonomer Systeme wie fehlerbehaftete Sensordaten, die in Simulationen meist ausgespart werden

Auf der anderen Seite halten die kleinen Maschinen das Risiko von Unfällen (für den Roboter und seine Umgebung) gering. Trotzdem gibt es auch bei diesem Versuch eine Notaus-Taste, so daß die Sicherheits-Regeln analog zu Kapitel 2 auch hier zu beachten sind.

3.1 Der Roboter Rug Warrior

Der am Massachusetts Institute of Technology (MIT) entwickelte Rug Warrior ist ein kleiner mobiler Roboter mit etwa 20 cm Durchmesser.



Das Herz des Roboters ist ein 8-Bit Mikrocontroller vom Typ 68HC11 mit 32 KByte RAM. Dieser kann über ein serielles Kabel mit einem Host-Computer verbunden werden. Die Programmierung erfolgt auf dem Host in der Programmiersprache Interactive-C (IC), einer parallelen, interaktiven Variante von C. Der Host dient dabei als Editor und Compiler für die C-Quelltexte. Die compilierten Routinen werden auf dem Mikrocontroller von einem P-Code-Interpreter abgearbeitet. Eine Beschreibung des Sprachumfangs von IC findet sich im Anhang.

Das Antriebssystem des Roboters besteht aus zwei durch Gleichstrommotoren angetriebenen Rädern sowie einem Freilaufrad. Durch diese "Differential Drive"-Anordnung ist der Roboter in der Lage, sich auf der Stelle zu drehen. Außer den beiden durch Pulsbreitenmodulation ansteuerbaren Motoren, befinden sich als Aktoren noch eine zweizeilige alphanumerische LC-Anzeige, vier Leuchtdioden sowie ein Piezo-Lautsprecher.

Aktoren

Auf dem LC-Display können während des Betriebs beliebige Texte oder Test-/Debug-Informationen ausgegeben werden. Textausgaben auf dem Host-Computer sind nicht ohne weiteres, mit einem Trick aber dennoch möglich (später hierzu mehr). Die Leuchtdioden zeigen die Drehrichtung der Elektromotoren und die Funktion der Infrarotsensoren an. Der am Heck des Roboters angebrachte Magnet dient - anstelle eines Greifers - zum Aufnehmen von Gegenständen.

Die einzigen internen Sensoren des Systems sind zwei Rad-Encoder, welche an den beiden Antriebsrädern angebracht sind. Sie erzeugen bei einer Drehung der Räder etwa alle 1.5 cm einen Impuls, der vom Mikrocontrollertroller mitgezählt wird.

Sensoren

Als externe Sensoren stehen dem Rug Warrior zwei Fotowiderstände, zwei Infrarot-LEDs mit einem Infrarotempfänger, ein Infrarot-Entfernungssensor sowie eine Bumper-Schürze mit drei Mikroschaltern zur Verfügung. Ein ebenfalls vorhandenes Mikrophon wird für unsere Zwecke nicht benötigt.

Mit Hilfe der an Analogeingängen angeschlossenen Fotowiderstände ist der Roboter in der Lage, hellere und dunklere Stellen in der Umgebung zu unterscheiden. Sie sind an der Front des Roboters angebracht.

Dort befinden sich auch die beiden IR-LEDs. Sie können unabhängig voneinander eingeschaltet werden und senden dann ein mit 40 kHz moduliertes Infrarotlicht aus. Der in ihrer Mitte angebrachte Infrarotempfänger detektiert dieses Licht (sofern es von Hindernissen reflektiert wird) und meldet das Ergebnis als binäres Signal an den Mikrocontroller. Es läßt sich also keine Abstandsabschätzung vornehmen, sondern lediglich das Vorhandensein von Hindernissen in einem ca. 30 cm großen Bereich feststellen.

PSD

Zur Bestimmung von Abständen dient ein weiterer Infrarotsensor. Er ist in der Lage, Objekte in einem Bereich von etwa 8 - 80 cm zu detektieren und eine Schätzung des Abstandes vorzunehmen. Die Auflösung liegt im Nahbereich bei etwa 1 cm und fällt mit zunehmender Entfernung auf ca. 10 cm ab.

Das dabei angewandte Verfahren ähnelt einer Laser-Triangulation, wobei als Empfänger ein sogenanntes Position Sensitive Device (PSD) eingesetzt wird. Der Sensor kann 10-15 Messungen pro Sekunde durchführen und liefert die gemessene Entfernung als synchrones 8-Bit Datenpaket an den Mikrocontroller zurück.

Wird ein Hindernis von den optischen Sensoren übersehen, so wird es spätestens bei Kontakt durch die Bumper-Schürze erkannt. Realisert wurde dieser Kontaktsensor durch drei Mikroschalter, die im Kreis um die Chassisgrundplatte angeordnet sind. Daher spricht immer mindestens einer der Schalter an, gleichgültig, an welcher Stelle der Roboter auf ein Hindernis trifft. Durch Auswerten aller Mikroschalter läßt sich sogar die Richtung des Hindernisses ermitteln.

Energie

Durch zwei Akkupacks kann der Rug Warrior tatsächlich autark, d.h. ohne externe Energiezufuhr, betrieben werden. Dies ist vor allem für den gleichzeitigen Einsatz mehrerer Maschinen wichtig. Während der Test- und Programmierphase wird der Roboter sinnvollerweise über das ohnehin notwendige serielle Kabel mit Energie versorgt.

Obwohl der Roboter insgesamt nur über eine sehr eingeschränkte Sensorik, wie auch eine geringe Rechenleistung verfügt, lassen sich bereits komplexe Aufgabenstellungen mit ihm realisieren.

3.2 Library-Funktionen

Um die Programmierung des Rug Warriors zu vereinfachen, sind grundlegende sowie hardwarenahe Funktionen in den Bibliotheken *Lib_RW11.c*, *Drive_cl.c* , *Drive_ol.c* und *Serial.c* zusammengefaßt, welche im folgenden näher beschrieben werden.

Die Library Lib_RW11.c

Diese Library wird beim Starten des Interactive-C automatisch geladen. Sie enthält Funktionen für das Zeitmanagement sowie den Zugriff auf die Hardware. Im einzelnen sind dies:

Timer

`void reset_system_time()`

Setzt den internen Millisekunden-Zähler zurück

`long mseconds()`

Gibt die Systemzeit in Millisekunden zurück. Der Zähler startet nach einem Hardwarereset oder dem Aufruf der Funktion reset_system_time() jeweils wieder bei 0,

`float seconds()`

Gibt die Systemzeit in Sekunden zurück. Die Auflösung entspricht jedoch Millisekunden.

`void sleep(float sec)`

Wartet die angegebene Zeit in Sekunden. (busy-waiting!)

`void msleep(long msec)`

Wartet die angegebene Zeit in Millisekunden. (busy-waiting!)

`void timer_start(int i, long timeout)`

Startet einen von 5 virtuellen Timern ($0 \le i \le 4$). Intern wird die Zeit `mseconds()+timeout` in einem Array vermerkt.

`long timer_time(int i)`

Gibt die Restlaufzeit des angegeben Timers in Millisekunden zurück. Ist der Wert kleiner 0, so ist der Timer abgelaufen.

Töne

`void beep()`

Erzeugt für 0.3 Sekunden einen 500 Hz Ton.

`void tone(float frequency, int length)`

Erzeugt einen Ton mit der angegebenen Frequenz und Dauer (in Millisekunden)

I/O

`int digital(int port)`

Gibt den Zustand der digitalen Eingänge `PA1` bzw `PA2` zurück. Wird für interne Zwecke benötigt.

`void digital_out(int port, int value)`

Setzt einen der acht digitalen Ausgänge des Erweiterungsboards auf High (`value=1`) oder Low (`value=0`).

Sensoren

`int analog(int port)`

Gibt den Wert des angegebenen Analogports zwischen 0 und 255 zurück. Für die Ports sind unter anderem die Konstanten `photo_right`, `photo_left` und `microphone` definiert. Bei den Fotosensoren entsprechen kleinere Werte hellerem Licht. Das Abfragen des Mikrophons ergibt bei Ruhe einen Wert von 128 und andernfalls einen Wert darüber oder darunter. Um den Geräuschpegel zu bestimmen, ist es sinnvoll mehrere Werte aufzunehmen und zu mitteln.

`int bumper()`

Gibt einen 3-Bit-Wert entsprechend dem Zustand der Bumper zurück. Die einzelnen Bits entsprechen dabei den einzelnen Tastern. Ein Wert von 3 bedeutet demnach, daß die Taster 1 und 2 gedrückt sind.

`int ir_detect()`

Gibt einen 2-Bit-Wert mit folgender Bedeutung zurück: 0 - kein Hindernis erkannt, 1 - Hindernis auf der rechten Seite erkannt, 2 - Hindernis auf der linken Seite erkannt, 3 - Hindernis auf beiden Seiten erkannt. Dieser Test benötigt mindestens 2 Millisekunden.

`int RLS()`

Fragt die am Heck des Roboters angebrachtet Reflexlichtschranke ab. Der Rückgabewert bedeuted: 0 - kein Objekt, 1 - unsicher, 2 - Objekt vorhanden. Dabei wird ein Schwellwert benutzt, welcher in der globalen Variablen `rls_threshold` gespeichert ist. Je nach Lichtverhältnissen und Roboter kann es nötig sein, diesen Schwellwert anzupassen.

Zahlen

`int abs(int arg)`

Gibt den Absolutwert des ganzzahligen Arguments zurück.

`int absf(float arg)`

Gibt den Absolutwert des reellen Arguments zurück.

`int p_byte(int val)`

Druckt ein Byte als genau 3-stellige Zahl aus. Dies ist hilfreich zur formattierten Zahlenausgabe.

Zufallswerte

`int randomint(float arg)`

Berechnet aus der Zystemzeit einen Zufallswert im Bereich [0 .. range-1].

`int randomfloat(float arg)`

Berechnet aus der Systemzeit einen Zufallswert im Bereich [0 .. (range-1)/teiler].

Scheduling

`void hog_processor()`

Stellt dem aktuellen Prozess 256 weitere Millisekunden bis zur Prozeßumschaltung zur Verfügung. Wiederholtes Aufrufen vor Ablauf dieser Zeit, verhindert die Abgabe der CPU an einen anderen Prozeß. Mit Hilfe des Befehls `Defer()` kann die Abgabe erzwungen werden.

Beachten Sie in diesem Zusammenhang auch die Befehle im "Multi-Tasking"-Kapitel des IC 3.1 - Handbuches (Seite 59 ff.).

Die Libraries Drive_OL.c und Drive_CL.c

An den Rädern des Rug Warriors sind reflektierende Scheiben mit 16 Unterbrechungen angebracht, welche von Reflexlichtschranken abgetastet werden. Die Libraries Drive_OL.c und Drive_CL.c dienen der Abfrage dieser Radencoder. Dabei wird jede Unterbrechung der Lichtschranken in Variablen aufsummiert.

Der Unterschied zwischen Drive_OL.c und Drive_CL.c betrifft das Zurücksetzen dieser Zählvariablen. Dabei ist Drive_CL.c für den Einsatz in in einer geschlossenen Regelschleife (closed loop) vorgesehen, in der die Routinen `get_right/left_clicks` regelmäßig aufgerufen werden, wogegen Drive_OL.c für den Betrieb ohne Regelung (open loop) gedacht ist.

*.lis

Im Gegensatz zu der Library Lib_RW11.c muß Drive*.c explizit geladen werden. Dies geschieht am einfachsten durch Anlegen einer Listendatei, wie das folgende Beispiel der Datei `Aufg1.lis` zeigt:

```
Drive.icb
Drive_OL.c
Square.c
Aufg_1.c
```

Nun kann in IC mit dem Befehl `load Aufg1.lis` das Laden der Libraries und Quelltexte auf "einen Rutsch" geschehen. Man beachte, daß es nur eine Version der Assemblerdatei `Drive.icb` gibt. Sie installiert Interruptroutinen, welche sowohl von Drive_OL.c als auch von Drive_CL.c benutzt werden. Generell ist es sinnvoll für jede Teilaufgabe eine Entsprechende Listendatei und jeweils neue Quelltexte anzulegen!

Im einzelnen lauten die Funktionen der Library Drive*.c wie folgt:

Motoren

`int init_motors()`

Initialisiert die Pulsbreiten-Generatoren für die Motorregelung. Diese Routine sollte zu Beginn eines Programmes aufgerufen werden.

`void motor(int index, int vel)`

Setzt die Energiezufuhr eines Motors (`index`=`LEFT` bzw. `RIGHT`) auf den angegebenen Prozentwert der Maximalenergie (-100 ≤ vel ≤ +100). Positive Werte entsprechen der Vorwärts- und negative Werte der Rückwärtsfahrt. Werte um Null halten den Motor an. Da es sich um eine *open loop control* handelt, entsprechen die eingestellten Energiewerte nicht unbedingt der gewünschten Geschwindigkeit.

`void init_velocity()`

Initialisiert die Interruptroutinen für das Zählen der Encoderclicks. Die Clicks des linken Motors werden dabei durch eine Interruptroutine gezählt, während die des rechten Motors durch einen Hardwarezähler aufsummiert werden.

Clicks

`int get_left_clicks()`

Liefert die Anzahl der Clicks seit dem letzten Aufruf und setzt die Zählvariable auf Null zurück.Wird diese Funktion in regelmäßigen Abständen aufgerufen, so entspricht ihr Wert der Geschwindigkeit pro Zeitintervall. Ein Click entspricht etwa 1,2 cm gefahrener Wegstrecke.

`int get_right_clicks()`

Analog zu `get_left_clicks()`. Zu beachten ist, daß die get_clicks-Funktionen keine Unterscheidung der Drehrichtung vornehmen. Sie addieren die Clicks bei Vorwärts- und Rückwärtsdrehung gleichermaßen!

Odometer

`float odometer(int wheel)`

Liefert den Stand des "Tageskilometerzählers". Es wird die vom linken bzw. rechten Rad seit dem letzten Odometer-Reset zurückgelegte Entfernung in Metern wiedergegeben (`wheel`=`LEFT` bzw. `RIGHT`). Diese Routine kann dazu benutzt werden Strecken abzumessen. Die Genauigkeit liegt allerdings nur bei etwa 1.2 cm.

`void reset_odometer(int wheel)`

Setzt den linken bzw. rechten "Tageskilometerzähler" auf Null.

Die Library RWTSerial.c

Diese Library dient der Übermittlung von Daten des Rug Warriors an den Hostcomputer. Interactive-C bietet selbst keine Möglichkeit, Daten wie z.B. den Inhalt eines Arrays vom Roboter zum Macintosh zu übertragen. Die einzige Möglichkeit besteht normalerweise darin, den Inhalt des Feldes nach und nach auf dem LC-Display auszugeben und manuell abzuschreiben. Um dies zu umgehen, wurden die folgenden Library-Funktionen implementiert:

Initialisieren

`void InitSerial()`

Initialisiert die serielle Schnittstelle des Rug Warriors zum Senden an den Host. Auf dem Macintosh kann nun das Programm *RW-Tool* gestartet werden. IC braucht während der Datenübertragung nicht geschlossen zu werden, jedoch sollte es während dieser Zeit nicht (via Kommandozeile) benutzt werden. Die Benutzung von IC, z.B. zum Laden von Programmen oder Starten von Prozessen setzt die serielle Schnittstelle wieder in ihren ursprünglichen Zustand zurück, so daß `Init-Serial` erneut aufgerufen werden muß.

Text

`int RWTsendTextString(char string[])`

Sendet einen String an das Textfenster des RW-Tools. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0. Im Normalfall kann der Rückgabeparameter ignoriert werden. Das Zeichen "\n" bewegt den Cursor in eine neue Zeile.

`int RWTsendTextInt(int value)`

Sendet eine Integerzahl ziffernweise an das Textfenster des RW-Tools. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0.

`int RWTsendTextFloat(float value, int digits)`

Sendet eine reelle Zahl ziffernweise an das Textfenster des RW-Tools. *digits* gibt die Anzahl der Nachkommastellen an. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0.

Graphen

```
int RWTsendCoordinateHead(int xMin, int xMax,
                          int yMin, int yMax,
                          int xDigits, int yDigits,
                          int xDistance, int yDistance,
                          char xLable[], char yLable[])
```

Initialisiert die Datenübertragung für die Koordinatensystemdarstel-

lung. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0. *xMin*, *xMax*, *yMin* und *yMax* geben den darzustellenden Bereich des Koordinatensystems ganzahlig an. *xDigits* und *yDigits* geben die Anzahl der Nachkommastellen an. Die später übertragenen Integer-Daten werden als Fixpunkt-Zahlen interpretiert. *XDistance* und *yDistance* legen den Abstand der Einheitenstriche fest. *xLable* und *yLable* dienen der Beschriftung der Achsen. Im RW-Tool Verzeichnis befindet sich ein Demo-Programm, welches die Funktionsweise verdeutlicht.

```
int RWTsendCoordinateData(int x, int y, int concatenate)
```

Überträgt ein einzelnes Datum für die Koordinatensystemdarstellung. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0. Neben den x/y-Koordinaten eines Punktes kann durch `concatenate` angegeben werden, ob und mit welcher Farbe dieser mit seinem Vorgänger verbunden wird: 0 = keine Verbindung, 1=black, 2=white, 3=red, 4=green, 5=blue, 6=cyan, 7=magenta, 8=yellow

Sector Scans

```
int RWTsendSectorHead(int sectornumber, int oriantation,
                      int angle, int displacement)
```

Initialisiert die Datenübertragung für einen Sector Scan. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0. *sectornumber* gibt die Anzahl der Sectoren an, *orientation* die Drehrichtung (1=linksherum, -1 =rechtsherum), *angle* den überstrichenen Winkel und *displacement* den Winkelversatz bei der Darstellung (mathematisch positiv). Im Rug Warrior-Verzeichnis befindet sich ein Demo-Programm, welches die Funktionsweise verdeutlicht.

```
int RWTsendSectorData(int sectorlength)
```

Überträgt ein einzelnes Datum eines Sector Scans. Liefert eine 1, falls kein Fehler aufgetreten ist, sonst 0.

Low-Level

```
void SendByte(int Byte)
```

Sendet ein Byte als ASCII-Zeichen an den Macintosh. Diese Funktion sollten Anwendungsprogramme nicht benutzen.

Aufgabe 2.1

(3 Tage)

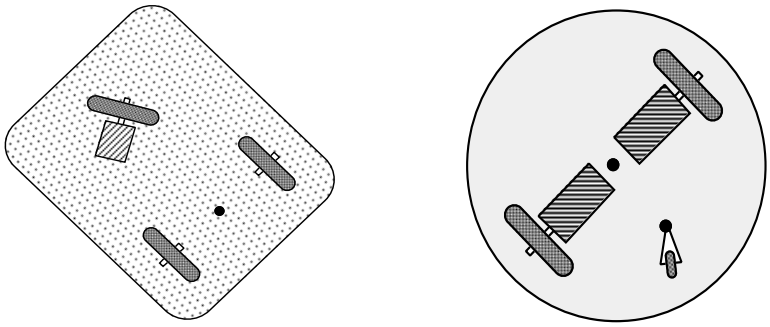
Machen Sie sich mit dem Rug Warrior und der Programmiersprache IC vertraut. Laden Sie das Demo-Programm zu dieser Aufgabe in dem Sie IC starten und `load Aufg1.lis` eingeben. (Zuvor muß - wie in der IC-Anleitung angegeben - der P-Code-Interpreter auf den Roboter geladen werden. Achten Sie auf die richtige Stellung des Schalters am Rug Warrior! Zum Download des Interpreters muß dieser zum LCD hin zeigen, und anschließend zum Laden der C–Programme umgeschaltet werden.)

Nach dem Drücken des Resettasters am Roboter startet das Programm. Erneutes Drücken stoppt es. Achtung: der Roboter braucht viel Platz! Benutzen Sie rechtzeitig die Reset- oder die Notaus-Taste, damit er keinen Schaden nimmt!

Was stellen Sie beim Laufenlassen des Programms fest? Untersuchen Sie den Quellcode und erklären Sie das Verhalten des Roboters.

3.3 *(v, ω)-Schnittstelle*

Zur Steuerung der Geschwindigkeit und Richtung eines Roboters sind je nach Anordnung und Funktion der Räder verschiedene Schnittstellen denkbar. So kann beispielsweise bei einer Anordnung mit einem angetriebenen, gelenkten Vorderrad und zwei starren, freilaufenden Hinterrädern, der Lenkwinkel und die Geschwindigkeit des Antriebsmotors als Softwareschnittstelle dienen.



Zwei Antriebskonfigurationen mit jeweils drei Rädern

Bei einer Radanordnung mit zwei starren, angetriebenen Rädern und einem freilaufendem Schlepprad, also einer Anordnung, wie sie beim Rug Warrior eingesetzt wird, wird man als Schnittstelle die Vorgabe der Geschwindigkeiten der Antriebsmotoren wählen.

Es sind also für die unterschiedlichsten Antriebskonfigurationen von mobilen Robotern jeweils angepaßte Low-Level-Schnittstellen nötig. Um die Entwicklung und Portierung von Kontrollalgorithmen zu erleichtern, wird in der Regel über diese Low-Level-Schnittstellen eine weitere Schnittstelle gelegt, welche weitgehend unabhängig von der realen Kinematik des Fahrzeugs ist.

(v,ω)

Eine solche Schnittstelle ist die (v,ω)-Schnittstelle. Ihr liegt die Idee zugrunde, zur Definition eines Bewegungszustandes, die **Linear-** und **Winkelgeschwindigkeit** des Roboters zu benutzten. Die Lineargeschwindigkeit bezieht sich dabei auf die Verschiebung (Translation) des kinematischen Zentrums des Roboters, während sich die Winkelgeschwindigkeit auf die Rotation dieses Punktes (oder eines beliebigen anderen Punktes auf dem Roboter!) bezieht. Übliche Einheiten zur Angabe von (v,ω) sind [m/s] für die Lineargeschwindigkeit und [Grad/s] für die Winkelgeschwindigkeit.

Das kinematische Zentrum liegt bei den oben angegebenen Dreiradanordnungen jeweils auf der Achse zwischen den starr montierten Rädern, da hier die kinematischen Beschränkungen am größten sind.

Im einzelnen gelten für die Radanordnung des Rug Warriors die folgenden Beziehungen:

$$v_{rechts} = v + \frac{\omega \cdot d}{2} \qquad v_{links} = v - \frac{\omega \cdot d}{2}$$

wobei d den Abstand der beiden Räder bezeichnet, und umgekehrt:

$$v = \frac{v_{links} + v_{rechts}}{2} \qquad \omega = \frac{v_{rechts} - v_{links}}{d}$$

In dem Beispielprogramm aus Aufgabe 2.1 wird die Umrechnung der (v,ω)-Werte auf die Geschwindigkeiten der beiden Antriebsmotoren durch die Funktion `move(float v, float w)` realisiert. Allerdings ist dies eine sehr einfache Implementierung, in der nicht die tatsächlichen Geschwindigkeiten der Motoren berücksichtigt werden (keine Regelung!), sondern vielmehr davon ausgegangen wird, daß die Motorgeschwindigkeit proportional zur Energiezufuhr durch das PWM-Signal ist. Wie wir in Aufgabe 2.1 gesehen haben, ist diese Annahme prinzipiell falsch.

Aufgrund der Exemplarschwankungen der einzelnen Motoren fährt der Roboter nämlich nicht geradeaus, obwohl beide Motoren die gleiche Energie (den gleichen PWM-Prozentwert) zugeführt bekommen. Außerdem können natürlich auch die beiden Raddurchmesser unterschiedlich sein, was wir im folgenden aber vernachlässigen wollen.

Ein erster Ansatz zur Verbesserung des Verhaltens besteht darin, die beiden Motoren aneinander anzupassen. Dabei kann man entweder den langsameren an den schnelleren, oder den schnelleren an den langsameren Motor anpassen.

Aufgabe 2.2a

(1 Woche)

Schreiben Sie ein Programm, welches die Motoren eine feste Zeit (z.B. jeweils 3 Sekunden) mit PWM-Werten zwischen -100% und +100% im Leerlauf laufen läßt. Ermitteln Sie, nachdem Sie den Motoren eine kurze Anlaufzeit gegönnt haben, die gezählten Impulse der beiden Rad-Encoder für die einzelnen PWM-Werte und übertragen Sie diese mit Hilfe der Graphen-Funktionen aus `RWTSerial.c` auf den Macintosh. Starten Sie hierzu auf dem Mac das Programm "RWTool", welches u.a. diese Graphen darstellen kann. (Interactive-C braucht dazu nicht geschlossen zu werden.) Ein Beispiel zum Gebrauch der seriellen Funktionen finden Sie in der Datei `Koordinaten_Demo.c`.

Die gemessenen Impulse entsprechen den Motorkennlinien der beiden Motoren im Leerlauf. Führen Sie diese Prozedur je einmal für Akku- und

Netzbetrieb des Roboters durch. (Achtung: Nur volle Akkus verwenden! Gegebenenfalls die Akkus vorher laden.) Inwieweit unterscheiden sich die Kurven?

Aufgabe 2.2b

Passen Sie einen der beiden Motoren an den anderen an, indem Sie eine Tabelle der Form [PWM-Motor-1, PWM-Motor-2] in die Funktion `move` integrieren. Diese Tabelle kann durch ein eindimensionales Array implementiert werden, welches die Abbildung des PWM-Wertes des ersten Motors auf den entsprechenden Wert des zweiten Motors realisiert.

Hinweis: Um Platz im Speicher des Roboters zu sparen, sollte diese Tabelle vom Typ Integer sein. Außerdem ist es nicht erforderlich alle Prozentwerte zwischen -100% und +100% aufzunehmen.

(3 Tage)

Ergänzen Sie Ihr Programm aus Aufgabe 2.2a so, daß es neben der graphischen Kennlinie die benötigten Tabellenwerte in C-Syntax (z.B.: `int Anpassung[] = { 10, 15, 20, 25, ... }`) an das RWTool sendet. Diese Ausgabe können Sie direkt in Ihren C-Code übernehmen. Ein Beispiel zum Gebrauch der seriellen Text-Funktionen finden Sie in der Datei `Text_Demo.c`.

Überprüfen Sie die Anpassung durch Fahrenlassen des Roboters mit verschiedenen Geschwindigkeiten - Er sollte möglichst geradeaus fahren. Korrigieren Sie die Tabelle gegebenenfalls. (Es genügt, wenn der Roboter "einigermaßen" geradeaus fährt. Sehr hohe und sehr niedrige Geschwindigkeiten brauchen dabei nicht berücksichtigt zu werden.)

Lassen Sie nun das Programm aus Aufgabe 2.1 erneut laufen.

Hinweis: Eine bessere Alternative besteht darin, statt *einer* Tabelle *jeweils eine* Tabelle der Form [Geschwindigkeit, PWM-Wert] für jeden Motor zu implementieren. Mit solchen Tabellen läßt sich dann bereits eine einfache, ungeregelte (v,ω)-Schnittstelle realisieren. Beachten Sie jedoch den erhöhten Speicherbedarf im Vergleich zur obengenannten Lösung.

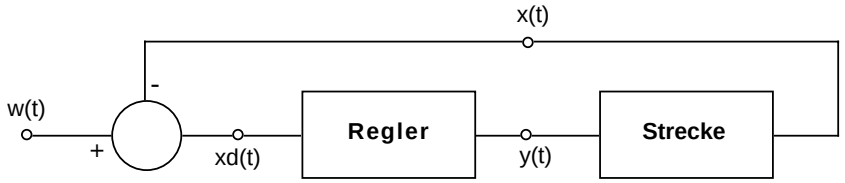
3.4 Motorregelung

Die im letzten Abschnitt eingeführte Benutzung von Motorkennlinien führt bereits zu einer erheblichen Verbesserung des Fahrverhaltens des Roboters. Dennoch stellt die verbesserte `move`-Funktion noch keine echte (v,ω)-Schnittstelle dar, da die übergebenen Geschwindigkeiten vom Roboter nicht unbedingt eingehalten werden.

So wird der Roboter bei gleichen Geschwindigkeitswerten z.B. eine Rampe viel langsamer hinauf- als herabfahren. Besonders problematisch ist dieses Verhalten im Hinblick auf einen gewünschten Kurvenradius, mit dem der RugWarrior fahren soll. Je nach Belastung der Motoren ist der Radius größer oder kleiner als die Vorgabe.

Regelung

Um dieses Problem zu lösen, muß der Roboter seine tatsächliche Geschwindigkeit permanent messen und diese bei der Ansteuerung der Motoren berücksichtigen. Es ist also eine *Regelung* der Motorgeschwindigkeiten nötig. Den prinzipiellen Aufbau einer Regelschleife zeigt die folgende Abbildung:



Die Sollgröße w(t) wird mit der tatsächlichen Regelgröße x(t) durch Berechnung der Differenz verglichen. Dieser Wert $x_d(t)$ wird durch den Regler in eine Stellgröße y(t) umgesetzt.

In dieser Terminologie entspricht w(t) der Sollgeschwindigkeit (z.B. in Klicks pro Zeitintervall), $x_d(t)$ der Differenz aus vorgegebener und gemessener Geschwindigkeit (ebenfalls in Klicks pro Zeitintervall) und die Stellgröße y(t) dem PWM-Wert, welcher dem Motor zugeführt wird. Hinter dem Begriff *Strecke* verbirgt sich der Motor zusammen mit einem Rad-Encoder, so daß der Ausgang x(t) der Strecke der gemessenen Geschwindigkeit entspricht. (Achtung: Dies ist i.d.R. nur eine Abschätzung der tatsächlichen Geschwindigkeit, da es hier sowohl zu Diskretisierungs-, als auch zu Quantisierungsfehlern kommt!)

PI-Regler

Ein in der klassischen Regelungstechnik häufig eingesetzter Motorregler ist der sogenannte *Proportional-Integral-Regler (PI-Regler)*. Er besteht aus einem proportionalen Verstärkungsglied (P-Glied) und einem integrierenden I-Glied. Im Gegensatz zu dem einfacheren P-Regler ist der PI-Regler in der Lage, eine bestehende Regelabweichung tatsächlich auszuregeln. Hierzu wird die Regelabweichung x_d über die Zeit mit einem Vorfaktor T_N

(der Nachlaufzeit) integriert und zusammen mit dem Proportionalanteil, nach Multiplizieren mit dem Verstärkungsfaktor V, als Stellgröße für die Strecke verwendet. Mathematisch läßt sich dies wie folgt ausdrücken:

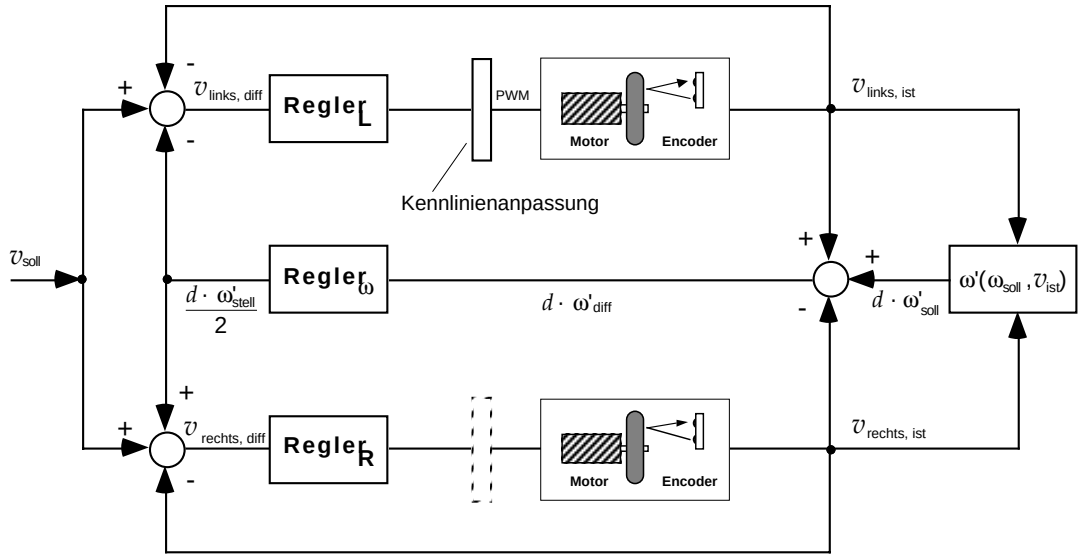
Formeln

$$y(t) = V\left(x_d(t) + \frac{1}{T_N}\int_0^t x_d(\tau)\,d\tau\right)$$

Durch Diskretisieren und Subtrahieren erhält man eine rekursive Formel für den PI-Regler:

$$y_n = y_{n-1} + V\left(x_{dn}-x_{dn-1} + \frac{1}{T_N}\left(\frac{x_{dn}+x_{dn-1}}{2}\right)\Delta t\right)$$

Mit diesem Wissen läßt sich bereits eine unabhängige Regelung der einzelnen Motoren realisieren. Um jedoch neben den beiden Motorgeschwindigkeiten auch das Kurven- bzw. Geradeausverhalten des Rug Warriors zu kontrollieren, ist eine weitere Regelschleife notwendig, welche explizit die Winkelgeschwindigkeit ausregelt.



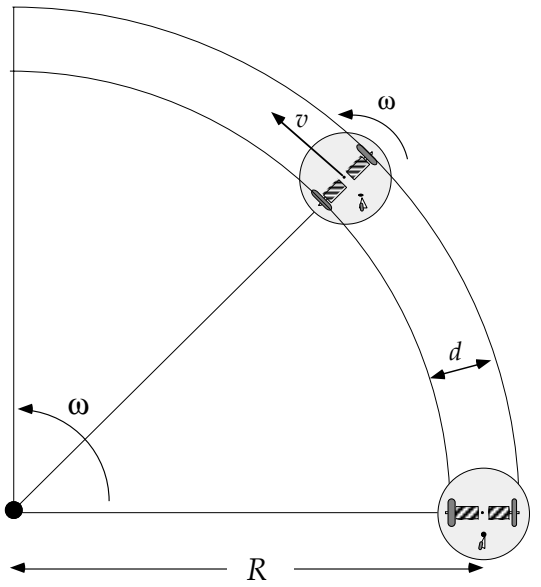
Regelschleifen für die (v,ω)-Schnittstelle

Für die im mittleren Zweig liegende Regelung der Winkelgeschwindigkeit wird nicht direkt die Winkelgeschwindigkeit ω als Sollgröße verwendet, sondern ein ω', welches jeweils in Abgängigkeit von der aktuellen Geschwindigkeit berechnet wird. Dies hat den Vorteil, daß der gesamte Regler nicht den v - und ω- Anteil getrennt ausregelt, sondern daß vielmehr der Kurvenradius das entscheidende Regelkriterium ist.

Anschaulich bedeuted dies, daß die Regelung versucht, einem durch (v,ω) vorgegebenem Kurvenradius mit höherer Priorität zu folgen, als der vorgegebenen Geschwindigkeit. Dadurch kann der Roboter beim Anfahren und bei unterschiedlicher Belastung der einzelnen Motoren genauer der vorgegebenen Kreisbahn folgen. Bei einer getrennten Regelung würde der Roboter häufig einen falschen Kurvenradius fahren, z.B. wenn die Regelung der Winkelgeschwindigkeit schneller der Vorgabe entspricht, als die Regelung der Lineargeschwindigkeit. Eine zu enge Kurve wäre die Folge.

Kurvenradius

Wie muß nun ω' bestimmt werden, damit bei jeder Geschwindigkeit der gewünschte Kurvenradius R gefahren wird? Betrachten wir hierzu nebenstehende Abbildung.



Fahrt des Roboters auf einem Kreisbogen

Für die Lineargeschwindigkeit des sich auf dem Kreisbogen *R* bewegenden kinematischen Zentrums des Roboters gilt nach der Kreisgleichung:

$$v = R \cdot \omega$$

so daß sich mit den Formeln aus Abschnitt 3.3 ergibt:

$$R_{soll} = \frac{v_{soll}}{\omega_{soll}} = \frac{v_{rechts} + v_{links}}{v_{rechts} - v_{links}} \cdot \frac{d}{2}$$

Um nun diesen Bruch und damit den Radius *R* unabhängig von der Momentangeschwindigkeit v_{ist} konstant zu halten, muß die Winkelgeschwindigkeit ω_{ist} angepaßt werden. Schließlich soll gelten:

$$R_{ist} \stackrel{!}{=} R_{soll}$$

$$\Leftrightarrow \quad \frac{v_{ist}}{\omega_{ist}} = \frac{v_{soll}}{\omega_{soll}}$$

$$\Rightarrow \quad \omega' := \omega_{ist} = \frac{v_{ist}}{v_{soll}} \cdot \omega_{soll}$$

$\omega'(\omega_{soll}, v_{ist})$

Fährt der Roboter also nur mit einem Teil seiner vorgegebenen Lineargeschwindigkeit, so darf er sich auch nur mit dem gleichen Teil der vorgegebenen Winkelgeschwindigkeit drehen, damit der durch (v,ω) gegebene Kurvenradius eingehalten wird.

Aufgabe 2.3

Implementieren Sie eine geregelte (v,ω)-Schnittstelle mit Berücksichtigung des Kurvenradiuses. Wählen Sie jeweils geeignete Regelparameter V und T_N aus. Wie verhält sich der Roboter bei schlechter Wahl der Parameter? Testen Sie Ihre Regelung indem Sie den Roboter über die bereitgestellte Rampe fahren lassen.

Realisieren Sie außerdem eine (v,K)-Schnittstelle, welche aus einer gegebenen Geschwindigkeit und einer Krümmung (dem Kehrwert des Kurvenradiuses) die Parameter für die (v,ω)-Schnittstelle berechnet. Für welche Bewegungen ist eine solche Schnittstelle nicht geeignet? Berücksichtigen sie entsprechende Sonderfälle!

(1 Woche)

Hinweis: Eine exakte Regelung hängt wesentlich von einem genauen Zeitintervall Δt ab! Am einfachsten ist es, ein festes Δt zu verwenden, d.h. die Regelungsroutine prüft zunächst ob die Zeit Δt verstrichen ist und gibt andernfalls den Prozessor vorzeitig wieder ab. Aufgrund der geringen Encoder-Auflösung werden jedoch nur wenige Klicks pro Zeiteinheit erfaßt, so daß ein Regelintervall unter 150 - 250 ms kaum sinnvoll ist. Damit sich der dennoch recht hohe Diskretisierungsfehler nicht zu sehr bemerkbar macht, sollte der Regler entsprächend "träge" eingestellt werden.

Eine Alternative zu einem festen Δt liegt in der Adaption des Regelzykluses an die Robotergeschwindigkeit. Dieser Ansatz soll im Praktikum allerdings nicht weiter verfolgt werden.

Achtung: Bei der Regelung muß die Library Drive_CL.c anstelle von Drive_OL.c in die entsprechende *.lis-Datei eingetragen werden!

3.5 Kontrollstrukturen

Damit ein autonomer mobiler Roboter auch in einer a priori unbekannten Umgebung, vorgegebene Aufgaben zielgerichtet bearbeiten kann, ist eine Programmstruktur notwendig, welche die Eingaben der Sensoren in geeignete Ausgaben für die Aktuatoren umsetzt. Diese Programmstruktur, sowie ihre Verbindung zur Hardware, wird als *Kontrollstruktur* eines AMR bezeichnet. Man versteht darunter das generelle Organisationsmuster der Datenverarbeitung eines autonomen mobilen Systems. In der englischsprachigen Literatur findet man hierfür häufig den Begriff *Robot Control Architecture*.

Anforderungen

An die Kontrollstruktur eines autonomen Systems werden verschiedene Anforderungen gestellt, so etwa eine hohe Robustheit gegen interne oder externe Fehler, eine kurze Reaktionszeit, eine einfache Erweiterbarkeit sowie der Zwang, mit relativ begrenzten Ressourcen wie Rechenzeit und Speicherplatz auszukommen.

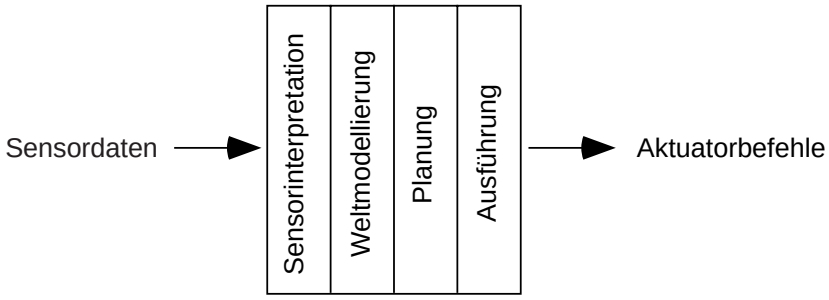
In der Vergangenheit wurde eine Vielzahl verschiedener Kontrollstruktur-Paradigmen vorgestellt, wovon zwei prinzipielle Ansätze hier betrachtet werden. Darüberhinaus existieren jedoch viele Mischformen, welche sich in der Praxis durchaus bewährt haben. Als Beispiel sei die *orthogonale Kontrollstruktur* genannt, welche im Rahmen des MOBOT-Projektes in der AG von Puttkamer seit 1985 entwickelt wurde.

Funktionale Kontrollstruktur

Abstraktion

Diese auch als *horizontale* oder *hierarchische Dekomposition* bezeichnete Architektur, stellt den klassischen Ansatz dar. Ihr liegt die Idee zugrunde, daß die Dynamik der Umgebung mit dem steigendem Abstraktionsniveau von Funktionsmodulen abnimmt. Spezifische Aufgaben werden in Teilaufgaben zerlegt, welche an die untergeordneten Hierarchieebenen delegiert werden. Diese versuchen mit einem größeren Detailwissen die Teilaufgaben zu erledigen.

Ein typisches Beispiel für den Einsatz der funktionalen Kontrollstruktur ist das *sense-model-plan-act Paradigma*. Kern dieses Paradigmas ist ein Weltmodell, in dem die gestellten Aufgaben zunächst auf einem hohen Abstraktionsniveau geplant werden, um dann zur konkreten Ausführung an untergeordnete Module delegiert zu werden. Diese Module arbeiten mit einem sequentiellen Datenfluß von den Sensoren zu den Aktuatoren. Der Detaillierungsgrad der Sensordaten nimmt bis zur Planungsinstanz kontinuierlich ab, während die dort generierten Befehle in den nachgeordneten Funktionseinheiten expandiert werden.



Detenfluß nach dem Sense-Model-Plan-Act Paradigma

Jede einzelne Funktionseinheit muß vollständig implementiert sein, damit das Gesamtsystem arbeiten kann. Der Ausfall eines Teilsystems führt zwangsläufig auch zum Ausfall des Gesamtsystems. Kritikpunkte sind ferner der hohe Speicherbedarf für das Weltmodell sowie die prinzipiell langen Reaktionszeiten durch den sequentiellen Datenfluß.

Probleme

Verhaltensorientierte Kontrollstruktur

Diese auch als *vertikale Dekomposition* bezeichnete Architektur, geht auf Arbeiten von Rodney A. Brooks am M.I.T aus dem Jahre 1985 zurück. Das Gesamtverhalten des Roboters wird bei diesem Ansatz von einzelnen Verhaltensschicht bestimmt, welche untereinander konkurrieren. Jede Verhaltensschicht verfolgt ein bestimmtes Ziel und hat prinzipiell Zugriff auf alle Sensordaten. Eine Hierarchie diesbezüglich besteht nicht.

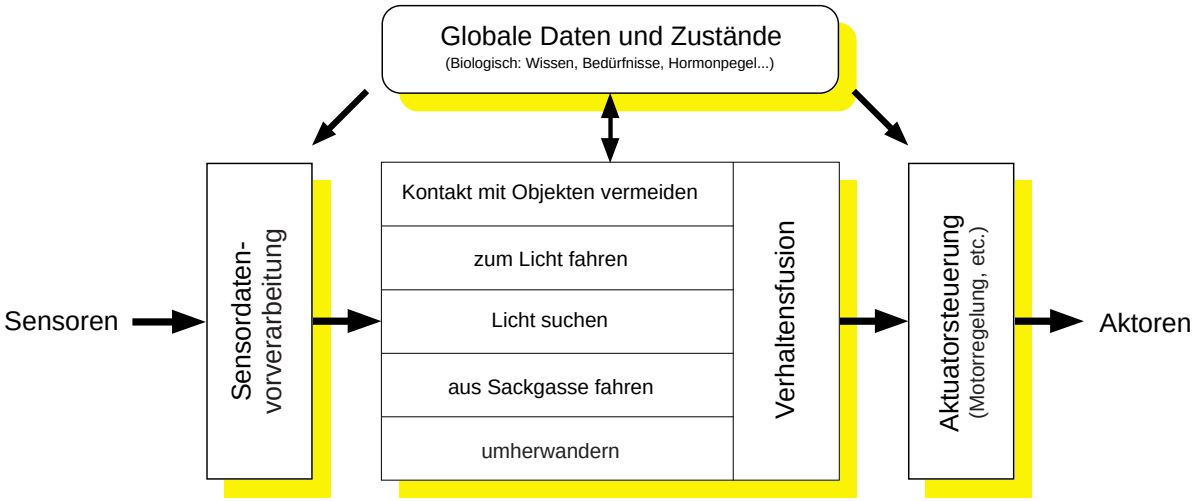
Welche Verhalten sich in einer gegebenen Situation auf die Aktuatoren auswirken, bestimmt eine als *Verhaltensfusion (behavior fusion)* bezeichnete Instanz. Brooks hat hier die *subsumption architecture* vorgeschlagen, in der höhere Schichten die Ausgaben darunterliegender Schicht unterdrücken können. Das Verhalten einer höheren Schicht enthält als Teilmenge das Verhalten der darunterliegenden Schichten. Umgekehrt sind tiefere Verhaltensschichten von höheren völlig unabhängig. Bei Ausfall einer Schicht können alle darunterliegenden weiterarbeiten.

Subsumption

Bei Brooks werden die einzelnen Verhalten durch endliche Automaten realisiert. Globale Daten sind nicht zulässig, allerdings kann eine höhere Schicht auf Ergebnisse untergeordneter Schichten zugreifen.

Dieser strikte Ansatz wird bei verschiedenen verallgemeinerten Ansätzen aufgeweicht. So kann es durchaus sinnvoll sein, globale Daten und Zustände zuzulassen. Auf diese Weise ist es z.B. möglich, auch bei einem verhaltensorientierten Ansatz eine Karte oder sogar ein Weltmodell einzubringen.

Allgemein



verallgemeinerte verhaltensorientierte Kontrollstruktur

Vorteile

Da auch bei diesem Ansatz jede Verhaltensebene für sich autonom arbeiten kann, ergibt sich eine hohe Fehlertoleranz. Die einzelnen Module können zudem weitgehend unabhängig voneinander implementiert und getestet werden. Verzichtet man auf ein internes Weltmodell, so kommt eine verhaltensorientierte Kontrollstruktur mit sehr geringen Ressourcen aus. Durch den direkten Datenfluß von den Sensoren zu jeder beliebigen Verhaltensschicht, sind kurze Reaktionszeiten gewährleistet.

Aufgabe 2.4

Der Rug Warrior soll mit Hilfe seiner Fotosensoren eine am Spielfeldrand befestigte Lichtquelle finden. Schreiben Sie hierzu ein Programm auf der Basis einer verhaltensorientierten Kontrollstruktur, welches sowohl Hindernissen ausweichen, als auch aus "Sackgassen" herausfinden kann.

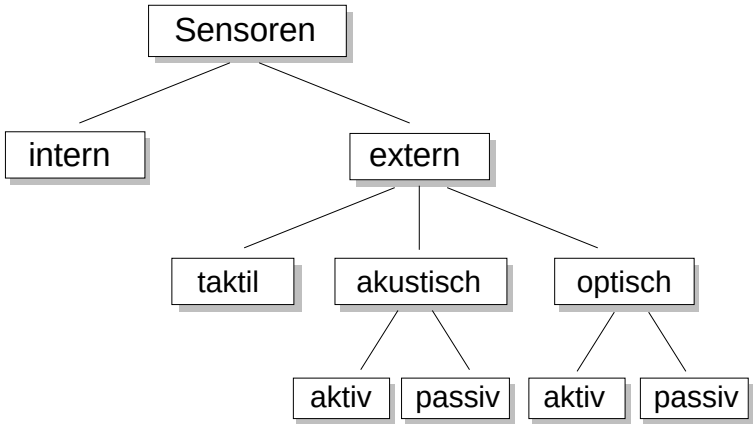
Die Hindernisdetektion soll dabei mit den beiden Infrarot-LEDs und dem Infrarot-Empfänger realisiert werden. Die Bumper-Schürze ist nur für Notfälle gedacht. Als Inspiration beim Entwurf der Verhaltensschichten kann das Beispiel oben auf dieser Seite dienen.

(1 Woche)

3.6 Sensordatenverarbeitung

Sensordaten bilden die Grundlage für die autonome Entscheidungsfindung eines AMR. Sie stammen entweder von *internen Sensoren* zur Auswertung interner Systemdaten, oder von *externen Sensoren*, welche die Umgebung des Roboters wahrnehmen.

Sensoren



Klassifikation der Sensoren eines autonomen mobilen Roboters

Zu den Aufgaben der internen Sensorik gehört die Überwachung von Systemzuständen wie des Ladezustands von Akkumulatoren, des Öffnungszustandes von Klappen, Türen und Bremsen, sowie die generelle Überwachung von Grenzwerten (Temperatur kritischer Bauteile, Öldruck, usw). Ferner fallen unter diesen Begriff Sensoren, welche die Lage und Position des Roboters im Raum *ohne* Bezug auf externe Landmarken bestimmen, also z.B. Rad-Encoder oder Kreiselsysteme.

Mittels externer Sensorik werden Daten aufgenommen, die Rückschlüsse auf die Beschaffenheit der Umwelt zulassen; insbesondere Informationen über Distanzen, statische oder bewegte Hindernisse sowie Bilder der Umgebung. Beispiele sind Bumper, Infrarot- oder Ultraschall-Entfernungssensoren oder auch Videokameras.

Fehler

In der Praxis erweisen sich die externen Sensoren als besonders kritisch, da hier die zu ermittelnde Meßgröße (beispielsweise ein Entfernungswert) häufig nur indirekt über eine Hilfsgröße (z.B. die Schallaufzeit) bestimmt werden kann. Zudem ist kein Meßprinzip vor Störungen und Meßfehlern sicher. Ideale Messungen, bei denen der ermittelte Meßwert exakt dem tatsächlichen Wert der Meßgröße entspricht, existieren in der Realität nicht. Stattdessen treten zufällige, systematische und dynamische Fehler auf, die sich in Rauschen, in Diskretisierungsstufen oder in Temperaturabhängigkeiten, etc. äußern.

Aufgaben

Die Aufgabe der Sensordatenverarbeitung ist es nun, durch geeignete Verfahren, die von den Sensoren gelieferten Daten zielgerichtet aufzubereiten und so weit wie möglich von Fehlern zu bereinigen. Dabei kann es sinnvoll sein, statt eines einzelnen Sensors, mehrere unterschiedliche Sensoren zu verwenden, die auf jeweils anderen Meßprinzipien beruhen. Das Zusammenfassen der Sensordaten *(Sensorfusion)* liefert dann erheblich bessere Ergebnisse als es die einzelnen Sensoren könnten.

Insgesamt lassen sich die folgenden Teilaufgaben der Sensordatenverarbeitung unterscheiden:

- Verdichtung von Sensordaten
- Zusammenfügen der Daten verschiedener Sensoren
- Erkennen von Objekten
- Aufbau von Karten
- Darstellung und Repräsentation von Wissen über die Umwelt

Die Leistungsfähigkeit eines Sensorsystems hängt also nicht nur von der Qualität der (verschiedenen) Sensoren ab, sondern auch entscheidend von der nachfolgenden Aufbereitung der Sensordaten. Im folgenden wollen wir die Teilaufgabe *Erkennen von Objekten* näher betrachten.

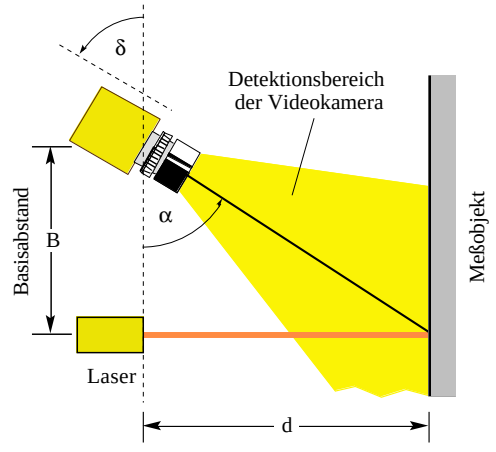
Die Sensorik des Rug Warriors

Rug Warrior

Der Rug Warrior besitzt eine Reihe externen Sensoren, welche in den vorangegangenen Aufgaben bereits eingesetzt wurden. Hierzu zählen die beiden Fotowiderstände, der IR-Näherungssensor bestehend aus zwei Infrarot-LEDs und einem Infrarotempfänger sowie die Bumper-Schürze. Außerdem gibt es eine Reflexlichtschranke, welche über dem Magneten am Heck des Roboters angebracht ist.

Zur Bestimmung von Abständen dient ein Infrarot-Entfernungssensor, auf den wir uns in diesem Abschnitt konzentrieren wollen. Er ist in der Lage, Objekte in einem Bereich von ca. 8-80 cm zu detektieren und eine Schätzung des Abstandes vorzunehmen.

Das dabei angewandte Verfahren ähnelt einer aktiven Triangulation mit einer Zeilenkamera. Dabei wird ein fokussierter Lichtstrahl (üblicherweise ein Laserstrahl) auf das zu messende Objekt gerichtet, wobei die Kamera in einem festen Basisabstand und Winkel zu dieser Lichtquelle angebracht ist. Abhängig von der Entfernung des Meßobjektes wird der gestreute Lichtpunkt auf einer bestimmten Stelle der CCD-Zeile abgebildet.

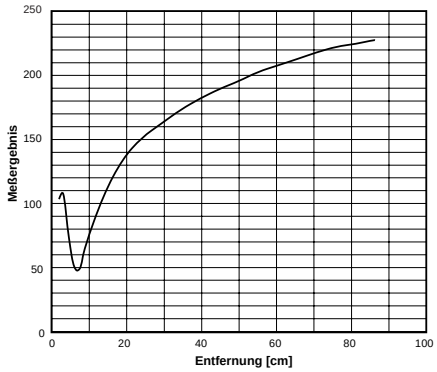


Triangulationsprinzip

Aufgrund des nichtlinearen Zusammenhangs zwischen dem Abstand zum Objekt und der Abbildung auf der CCD-Zeile, ist die Auflösung im Nahbereich deutlich besser, als in der Ferne, was ein durchaus wünschenswerter Effekt ist.

PSD

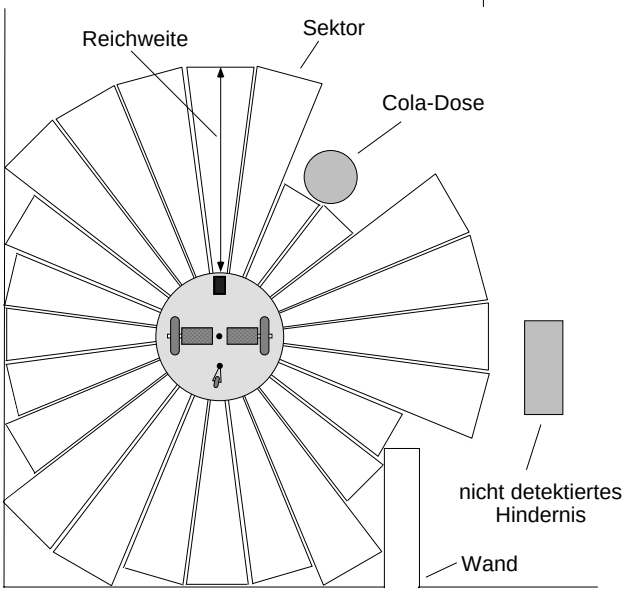
Bei dem am Rug Warrior eingesetzten Sensor GP2D02 der Firma Sharp, wird als Lichtquelle eine fokussierte Infrarot-LED eingesetzt. Statt einer CCD-Zeilenkamera wird ein wesentlich einfacherer, sogenannter *Position Sensitive Detector (PSD)* benutzt. Er ist vergleichbar mit einem länglichen Fotowiderstand, dessen Widerstandswert aber nicht von der Lichtintensität abhängt, sondern vorallem von der Stelle, an welcher der Widerstand beleuchtet wird.



Die Genauigkeit des verwendeten PSDs reicht nicht an die einer CCD-Zeile heran; dafür ist der Sensor sehr klein und preiswert. Die Entfernungsauflösung ist im Nahbereich besser als 1 cm und fällt mit zunehmender Entfernung auf ca. 10 cm ab. Auch hängt das Meßergebnis von der Reflektivität des betrachteten Objektes ab.

Der Sensor überträgt die aus 16 Einzelmessungen gemittelten Entfernungswerte als serielles Telegramm mit 8-Bit Wortlänge an den Mikrocontroller des Rug Warrior. Für einen Meßzyklus werden etwa 75 ms benötigt, so daß sich eine maximale Datenrate von 13 Messungen pro Sekunde ergibt.

Der Öffnungswinkel des PSD-Sensors beträgt in der Vertikalen etwa ±6 und in der Horizontalen ±1 Grad. Er ist mittig an der Frontseite des Rug Warriors angebracht und überstreicht so bei einer Punktdrehung des Roboters einen Kreisabschnitt, dessen Mittelpunkt im kinematischen Zentrum liegt. Mißt man nun bei einer gleichmäßigen Drehbewegung in regelmäßigen Zeitintervallen, so erhält man einen sogenannten *Sector Scan* (auch *Sector Map*) der Umgebung.



Sector Scan mit detektierten Hindernissen

Mit dem RugWarrior läßt sich auf diese Weise ein Sector Scan in ca. 8 Sekunden bei einer Winkelauflösung von 4.5° und besser aufnehmen, was einer Anzahl von 80 Sektoren auf 360 Grad entspricht.

Sector Scans bilden bei mobilen Robotersystemen häufig die Grundlage für die Hindernis- und

Situationserkennung. Typische Sensoren zur Aufnahme solcher Scans sind ringförmig angeordnete Ultraschallsensoren oder ein rotierendes Laserradar, welches nach einem Laufzeit-, Phasenversatz- oder Triangulationsverfahren arbeitet.

So fehlerfrei wie in der obigen Abbildung sind reale Sector Scans allerdings nicht. Es treten häufig Fehlmessungen aufgrund von Spiegelreflexionen, zu starker Absorption, oder der technisch bedingten Diskretisierung auf. Zudem lassen sich Objekte nicht in jeder Situation unterscheiden, weil sich ihre Repräsentation durch die Sensordaten gleicht. Im Beispiel ist daher keine Unterscheidung zwischen der Cola-Dose und dem Wandvorsprung möglich.

Dennoch lassen sich mit Sector Scans unter Einsatz einfacher mathematischer Berechnungen gute Ergebnisse erzielen, insbesondere, wenn viele Scans zu einer globalen Karte zusammengefügt werden. Dabei werden verstärkt auch neuronale Netzte zur Situations- und Objekterkennung eingesetzt.

Aufgabe 2.5a

Messen Sie die Entfernungskennlinie des im Rug Warrior eingebauten PSD-Sensors je einmal mit einem weißen und einem schwarzen Blatt Papier als Ziel aus (Sie brauchen nur jeden zweiten Wert des Sensors zu berücksichtigen). Wie stark unterscheiden sich sich die Kennlinien?

(3 Tage)

Nehmen Sie die erste Kennlinie in Ihr Programm auf und benutzen Sie sie, um *Sector Scans* ähnlich dem oben abgebildeten aufzunehmen. Nehmen Sie Scans für die Situationen

a) kein Hindernis im Bereich des Sensors
b) eine Wand im Bereich des Sensors
c) eine Eine Cola-Dose im Bereich des Sensors

sowie für Kombinationen daraus auf. Verwenden Sie hierzu das Programm "RWTool" auf dem Macintosh. Die Datei Scan.c enthält ein Beispiel für die Datenübertragung vom Rug Warrior zu dem Visualisierungsprogramm.

Aufgabe 2.5b

Entwickeln Sie ein Verfahren, um aus den gewonnenen Sensordaten Objekte der Kategorie "Cola-Dose" zu erkennen. Dabei können Sie folgende Ideen verwenden:

Eine Cola-Dose ist gekennzeichnet durch einen "Sprung" der Entfernungswerte auf den Roboter zu sowie einen nachfolgenden Sprung vom Roboter weg. Diese Sprünge müssen eine bestimmte Mindestgröße besitzen

und einen, von der gemessenen Entfernung abhängigen, zeitlichen Abstand aufweisen. Aus diesem Abstand läßt sich auf den Durchmesser des Objektes schließen. Erfolgt nach dem ersten Sprung der Rücksprung nicht in dem berechneten Zeitfenster, so handelt es sich bei dem Objekt vermutlich nicht um eine Cola-Dose sondern z.B. um eine Wand oder einen anderen Roboter. Wesentlich ist auch eine Beschränkung der maximalen Reichweite der Messungen, da die Genauigkeit bei zu großen Entfernungen für die Objekterkennung nicht ausreicht.

(3 Tage)

Aufgabe 2.5c

Benutzen Sie Ihr Verfahren, um mit dem Rug Warrior Cola-Dosen einzusammeln. Die Cola-Dosen sollen in dem Spielfeld erkannt und zum Ziel (markiert durch eine Energiesparleuchte) gebracht werden. Dabei gilt es - wie in der vorangegangene Aufgabe - Hindernissen auszuweichen und auch einen Weg aus „Sackgassen“ zu finden. Die Cola-Dosen können mit dem am Heck angebrachten Magneten hinter dem Roboter hergezogen werden. Am Ziel können die Dosen durch Drehen des Roboters "abgestreift" werden.

Das Vorhandensein einer Dose am Magneten läßt sich durch die Reflexlichtschranke abfragen, welche an den Analogeingang 7 des Controllerboards angeschlossen ist. Zur Abfrage dient die Prozedur `RLS`. Gegebenenfalls können Sie die globale Variable `rls_threshold` in Ihrem Programm (nicht in der Library!) anpassen, um die Schwelle zur Erkennung von Dosen zu justieren.

(1 Woche)

Wettbewerb!

Der besondere Reiz dieser letzten Aufgabe des Robotik-Pratikums besteht darin, daß unter den einzelnen Praktikumsgruppen ein Wettbewerb durchgeführt wird. Diejenige Gruppe, welche in einer vorgegebenen Zeit die meisten Cola-Dosen zum Ziel bringt, gewinnt. Dabei wird weder die Anordnung der Hindernisse, noch die Positionen der Dosen vor dem Wettbewerb bekannt gegeben.

Viel Erfolg und Spaß bei dieser Aufgabe!

4. Anhang

4.1 Macintosh Tastaturbelegung

Tastatur

Die Belegung der Mac-Taststur unterscheidet sich zum Teil erheblich von der eines Standard-PCs. Insbesondere die Pascal-typischen Sonderzeichen wie eckige und geschweifte Klammern sowie das Hochkomma sind schwer zu finden. Daher die folgende Tabelle:

Tastenbelegung	Sonderzeichen
Alt-5	[
Alt-6	]
Alt-7	|
Alt-8	{
Alt-9	}
Alt-´	'
Alt-Shift-1	@
Alt-Shift-7	\
Ctrl-Alt-N	~

Think-Pascal

Die Think-Pascal Entwicklungsumgebung hat ebenfalls ihre Eigenheiten wie den syntaxgesteuerter Editor oder die Projektverwaltung und den Debugger. Informationen hierüber entnehmen Sie bitte dem Think-Pascal-Manual Teil 3 (Kapitel 6 - 14).

Fragen hierzu beantworten auch gerne die Hiwis, die Ihnen ebenso eine kurze Einführung in das System geben können.

```
UNIT Hilfsroutinen;
INTERFACE

FUNCTION OeffneFenster (Datenbereich: Ptr; oben, unten, links, rechts: integer):
WindowPtr;
  (* Ein neues Fenster wird erzeugt. Datenbereich muß auf eine Variable vom  *)
  (* vordefinierten Typ WindowRecord zeigen (Übergabe: @WindowRec). Das Er-   *)
  (* gebnis von OeffneFenster ist ein Zeiger vom Typ WindowPtr. Er muß an     *)
  (* WaehlePort übergeben werden, bevor das Fenster angesprochen werden kann.*)

PROCEDURE SchliesseFenster (FensterZeiger: WindowPtr);
  (* Das Fenster, auf das FensterZeiger zeigt, wird geschlossen. Bei Pro-   *)
  (* grammende werden alle Fenster, die vom Programm geöffnet wurden,       *)
  (* automatisch geschlossen.                                               *)

PROCEDURE WaehlePort (FensterZeiger: WindowPtr);
  (* Das angegebene Fenster wird zum aktuellen Grafik-Port erklärt. *)
  (* Alle Grafikausgaben beziehen sich nun auf dieses Fenster.      *)

PROCEDURE StrichGroesse (Breite, Hoehe: integer);
  (* Eine neue Strichgröße wird eingestellt, Voreinstellung: 1, 1  *)

PROCEDURE SetzeCursor (horizontal, vertikal: integer);
  (* Der Cursor wird an eine neue Stelle gesetzt.            *)
  (* Der Ursprung (0,0) befindet sich links oben im Fenster. *)

PROCEDURE SetzeCursorRelativ (deltaHorizontal, deltaVertikal: integer);
  (* Der Cursor wird relativ zur alten Position verschoben *)

PROCEDURE WriteDraw (p1, p2, …);
  (* Analog zum Standard-Pascal Write; Ausgabe auf den aktuellen Graphik-Port *)

PROCEDURE LinieNach (horizontal, vertikal: integer);
  (* Wie SetzeCursor, aber zwischen der alten und neuen Position wird eine *)
  (* Linie gezogen.                                                        *)

PROCEDURE Linie (horizontal, vertikal: integer);
  (* Wie SetzeCursorRelativ, aber zwischen der alten und neuen Position wird *)
  (* eine Linie gezogen. *)

PROCEDURE Helligkeit (Toenung: Pattern);
  (* Die Helligkeit der zukünftig zu zeichnenden Linien und Zeichen wird  *)
  (* eingestellt. Pattern umfaßt white, ltGray, gray, dkGray und black.   *)
  (* Voreinstellung ist black.                                            *)

PROCEDURE TextGroesse (Groesse: integer);
  (* Ausgedrückt in Punkten, Voreinstellung 12 Punkte. *)

PROCEDURE Loesche (oben, unten, links, rechts: integer);
  (* Löscht einen rechteckigen Ausschnitt im aktuellen Grafik-Port. *)

FUNCTION PruefeTasten: boolean;
  (* Prüft, ob eine Taste gedrückt oder festgehalten wurde. *)

FUNCTION LeseZeichen: char;
  (* Liest einen Tastendruck ein, nur gültig, wenn der vorherige Aufruf von *)
  (* PruefeTaste TRUE lieferte.                                             *)

FUNCTION TickCount: longint;
  (* Liefert die Systemzeit des Macintosh in 60stel Sekunden zurück *)
```