

Studienarbeit-Nr. 1617

Ansteuerung einer sechsbeinigen Gehmaschine

Rainer Leonhardt

1997

Universität Stuttgart
Fakultät Informatik

Vorwort

Und sie bewegt sich doch - die Rede ist von der sechsbeinigen Gehmaschine aus Stuttgart. Die erste war die, von Waldemar Spädt gebaute „Rug Walker“. Die zweite ist „the Beast“ aus den U.S.A.

Wie haben diese „Walker“, „Hexapods“, sechsbeinigen „Gehmaschinen“ oder „Schreitroboter“ ihre Namen erhalten? Ganz einfach: In der Universität Stuttgart existieren schon seit längerer Zeit kleine Roboter, welche in einem Fachpraktikum verwendet werden. Diese heißen Rug Warrior. Da die Gehmaschine geht, war der neue Name schon durch eine Wortspielerei entstanden. Wie kam „the Beast“ zu seinem Namen? Ganz einfach: Der Hersteller war schneller; er hatte diesem Fabrikat schon den Spitznamen „the Beast“ gegeben (siehe [Lynxmotion 97]).

Und da sie auch niemanden bekriegen, höchstens ihre Akkus, die kann man ja zum Glück wieder aufladen kann, sind die Namen ja auch relativ adäquat. Ansonsten gehen sie, wie es sich für anständige Gehmaschinen gehört, sie sehen - besser als Spinnen sogar. Nur hören können sie noch nicht. Doch niemand weiß, was die Zukunft bringen wird.

Wer hat dies ermöglicht? Natürlich in erster Linie PD. Dr. habil. Thomas Bräunl, unter dessen Leitung dieses Projekt aufwuchs. Die zweite Säule war Günter Mamier, ein wissenschaftlicher Mitarbeiter, ein Doktorand der Universität-Stuttgart, der in Thomas Bräunls Abwesenheit alles am Laufen hielt und sich in beispielhafter Weise für dieses Projekt einsetzte. Im Rahmen von Studienarbeiten waren noch folgende Studenten beteiligt: Gerrit Heitsch, Jörg Henne, Michael Kasper, Thomas Lampart, Barb Linn, Normann Ness, Frank Sautter, Klaus Schmitt, Waldemar Spädt.

Ich möchte hiermit Thomas Bräunl danken, daß er mir diese Studienarbeit an einem so begehrten Utensil ermöglicht hat, trotz der Schwierigkeiten mit denen er zu dieser Zeit selbst in Australien zu kämpfen hatte. Ich möchte Günter Mamier danken für seine physische und vor allem mentale Unterstützung, die hier die zarte Flamme des Eyebotprojekts wohl oftmals rettete. Des weiteren möchte ich Thomas Lampart danken für die Erklärungen über die Hard- und Software. Zu guter letzt möchte ich Gerrit Heitsch, Hans Georg Filipp und Frank Weitmann für Unterhaltung im Robotiklabor danken, welche die Arbeit oftmals amüsant machte.

1 Gehmaschinen

Gehmaschinen sind ein interessantes Gebiet der Informatik, welches vermutlich in Zukunft immer mehr an Bedeutung gewinnen wird. Bis heute scheiterten derartige Projekte an der nicht ausreichend entwickelten Technologie. Doch durch die Entwicklung von elektrischen Antrieben wie Schrittmotoren, Servos etc. und durch die Entwicklung von elektronischen Bausteinen wie Tastsensoren, Beschleunigungssensoren, CCD-Kameras und integrierten Schaltkreisen wird die Realisierung derartiger Projekte zunehmend ermöglicht und da ein Bedarf existiert, wohl auch durchgeführt.

Eine Gehmaschine ermöglicht es, durch ihre Geländegängigkeit ein Terrain zu beschreiten vor dem ein Fahrzeug auf Rädern hoffnungslos kapitulieren muß. Mit einer ausgereiften Gehmaschine sind selbst Sprünge über Hindernisse denkbar, was man sich bei einem Räderfahrzeug kaum vorstellen kann. Diese Befähigungen machen eine Gehmaschine für Einsätze interessant, in denen nicht auf ebenem Boden gefahren wird. Da Gehmaschinen selbst Kettenfahrzeugen in Sachen Geländegängigkeit überlegen sein werden, werden auf lange Sicht militärische Einsätze, Planetenexplorationen, Bombenentschärfungen etc. von Schreitrobotern durchgeführt.

Im Anschluß werden nun einige aktuelle Gehmaschinen vorgestellt, wobei ihre Schwächen und Stärken erläutert werden.

1.1 Katharina

Katharina ist eine sechsbeinige Gehmaschine, die am Fraunhofer Institut für Fabrikbetrieb und -automatisierung entwickelt wurde. Sie besitzt 18 Antriebe, d.h. pro Bein drei Freiheitsgrade, welche von Mikrocontrollern gesteuert werden. Besonders interessant bei Katharina ist die Sensorik, welche sich aus „Beinkraftsensoren“ und einem Stereokamerakopf (zwei CCD-Kameras) zusammensetzt.



Abbildung 1.1: Katharina

Diese Gehmaschine ist mit Hilfe ihrer 3-Komponenten-Kraftsensoren und Krafrückführung in der Lage die mechanischen Eigenschaften des Untergrundes festzustellen, was ihre größte Besonderheit ist. Damit ist sie in der Lage Standsicherheit auf kompliziertem Relief zu gewährleisten, Linear- und Winkelverschiebungen des Körpers bei Ausführung von technologischen Operationen (wie Bohren oder Montieren) zu gewährleisten.

Die Stereokamera ermöglicht 3D-Vermessung von Objekten, dreidimensionale Visualisierung von Arbeitsräumen und Objektverfolgung.

Alles in allem ist dieser wohl relativ teure Gehroboter eine relativ ausgereifte Maschine, und sie wird dieses Forschungsgebiet mit ihren 21 kg gewichtig nach vorne bringen. (Siehe Abbildung 1.1 und [Schmucker 97])

1.2 Dante II

Im Rahmen des Dante II Vulkan-Explorationsprojekts der NASA wurde Mount Spurr, ein Berg in Alaska erforscht. Der gleichnamige Roboter, der diese Expedition durchführte ist mit acht Beinen ausgestattet, welche in zwei Gruppen aufgeteilt sind. Die Bewegung wird durchgeführt, indem jeweils vier korrespondierende Beine auf dem Boden stehen und die vier anderen vorwärtsbewegt werden.

Diese Gehmaschine hat von allen hier angeführten am ehesten einen Sinn, denn sie ist gut für ihre Aufgabe ausgestattet, sie hat zusätzlich zu der herkömmlichen Roboterausstattung Sensoren für H₂S, SO₂ und CO₂. Sie hat ein Stereokamerapaar, und vier Bein-Beobachtungskameras. Die Rechenleistung wird von einem einzigen Board ausgestattet mit einem Spark oder M68030 Prozessor erbracht, welcher über Satellit mit 1024 Kbit/sec an der Außenwelt hängt.

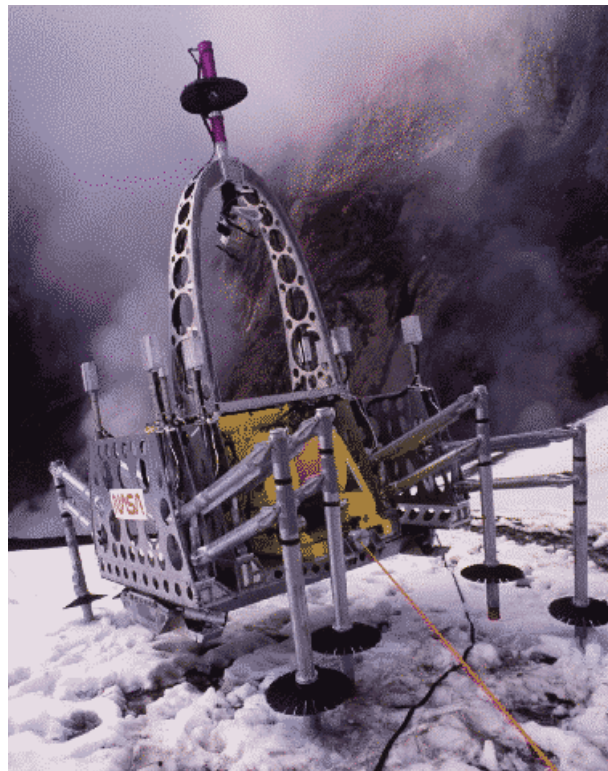


Abbildung 1.2: Dante II

Die Kosten und das Gewicht sind mit 1700000 \$ und mit 850 kg dementsprechend hoch. Dieser Roboter dient in diesem Fall nur als Anschauungsobjekt, jedoch nicht als konkurrierendes Produkt. (Siehe Abbildung 1.2 und [Erik et al. 97])

1.3 Rodney3

Rodney3 ist eine sechsbeinige Gehmaschine, die mit dem in dieser Studienarbeit verwendeten Objekt vergleichbar ist. Sie wurde von Frank Scott aus London entwickelt und hatte, wie der Name es schon sagt, zwei Vorgängermodelle, Rodney1 und Rodney2.

Die Bewegungen werden von 12 Servos veranlaßt, allerdings sind hier durch den Preis bedingt Servos verwendet worden, welche ein Drehmoment von nur 0.3 Nm haben, was wohl für viele Projekte als zu gering gelten würde.

Angesteuert wird Rodney3 wie auch die Rugwarriors, welche in der Uni-Stuttgart in einem Fachpraktikum verwendet werden, mit Hilfe eines MC68HC11-Microcontrollers, der einen externen Speicher von 32 kB RAM hat. Rodney3 verfügt, wenn man bedenkt, daß er nur 1100g wiegt und 34 cm lang ist, über eine beachtliche Anzahl an Sensoren: IR-Detektoren, Tastsensoren an den Beinenden, Pyro-Detektoren, Bumper, etc. wieder ähnlich zu den Rugwarriors. (Siehe auch Abbildung 1.3 und [Scott 97]).

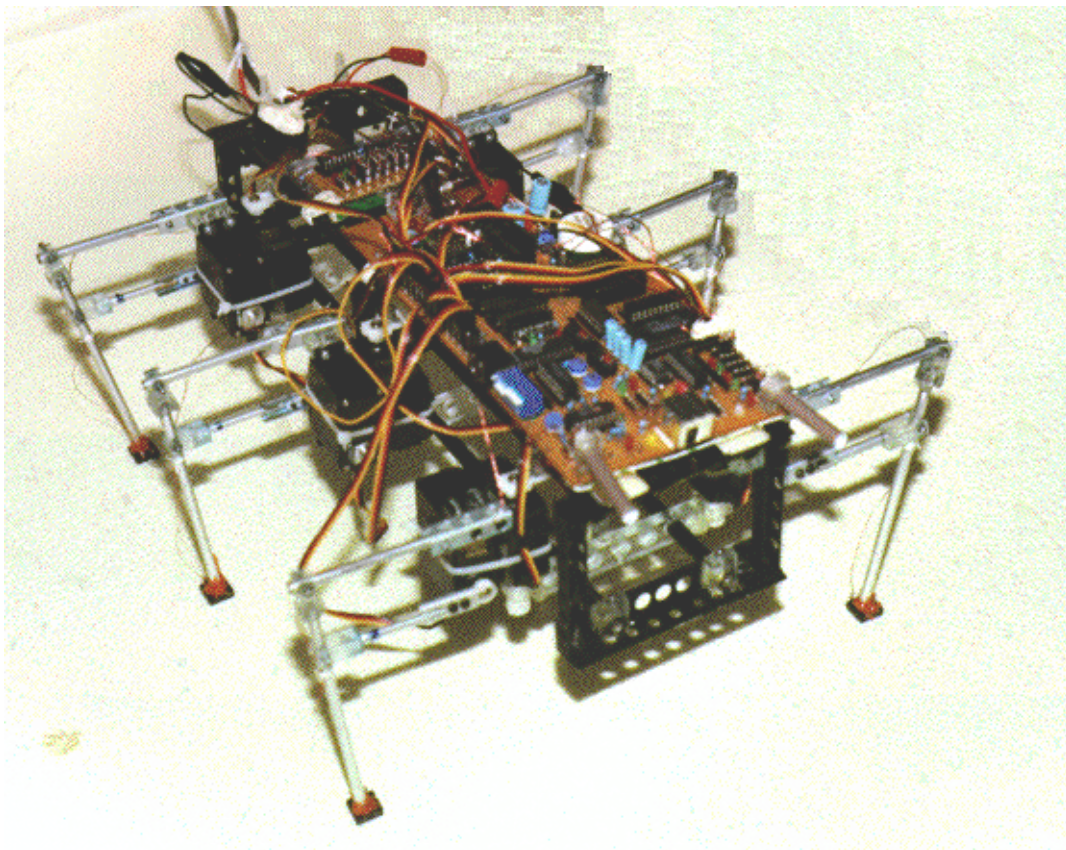


Abbildung 1.3: Rodney3

1.4 Thing

Thing ist eine vierbeinige Gehmaschine, die am MIT im Laboratory for Preceptual Robotics entwickelt wurde. Gesteuert wird sie durch fünf Prozessorboards, welche in einer sternförmigen Konfiguration angebracht sind, die Sensordaten verarbeiten und die Aktoren ansteuern. Alternativ kann sie auch über eine serielle Verbindung von einer Sparc aus gesteuert werden. Programmiert wird Thing mit HC11-Assemblercode und mit C. Das besondere an Thing ist, daß es auch als Hand verwendet werden kann, d.h. man dreht es auf den Rücken und läßt von ihm einen Globus balancieren.

Als Aktoren werden bei Thing so wie bei den meisten anderen Gehmaschinen auch PWM Servos verwendet, als Sensoren dienen Infrarotdetektoren (siehe auch Abbildung 1.4).

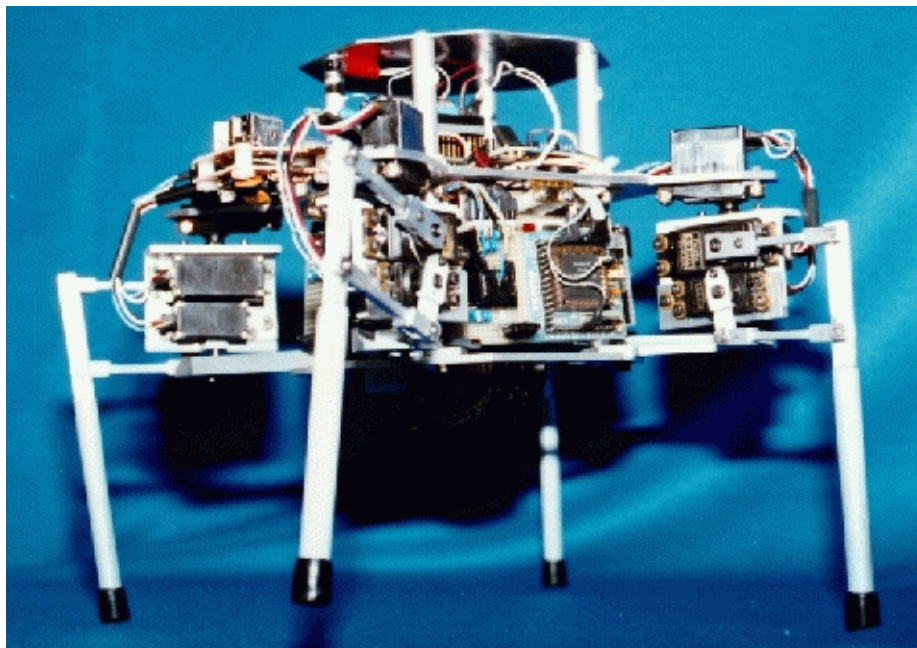


Abbildung 1.4: Thing

1.5 Rug Walker

Die Gehmaschine Rug Walker (siehe Abbildung 1.5), die im Rahmen einer Diplomarbeit von Waldemar Spädt gebaut wurde, hat sechs Beine, die seitlich wie bei einem Käfer angebracht sind. Dabei besitzt ein Bein zwei Servos, die dieses Bein in zwei Dimensionen bewegen können, nämlich: Ein etwas stärkeres Servo zum heben bzw. senken und ein etwas schwächeres aber dafür auch leichteres zum vorwärts- bzw. rückwärtsbewegen des Beines. An den Enden der Beine sind Tastsensoren angebracht, anhand derer abgefragt werden soll, ob ein Bein Bodenkontakt hat oder nicht. Des weiteren wird die Gehmaschine jeweils vorne links und vorne rechts mit einem Entfernungssensor ausgestattet, welcher der Gehmaschine ein gewisses Orientierungsvermögen zur Verfügung stellt. Diese Sensoren können entweder auf der Basis von Ultraschall funktionieren oder es werden Infrarot-Entfernungssensoren angebracht.

Die Gehmaschine wird von einer kleinen Computerplatine gesteuert, die von Frank Sauter gebaut wurde und für die das Betriebssystem von Gerrit Heitsch geschrieben wurde (siehe [Heitsch 96]). Im wesentlichen enthält diese Platine einen MC68332 Microcontroller, der ihr

Herzstück bildet, 1 MB RAM, ein EEPROM in das die Betriebssoftware über einen Parallelport geladen wird und einige Ports. Die Steuerung, welche diese Platine nun zur Aufgabe hat, bewältigt sie über diese Ports.

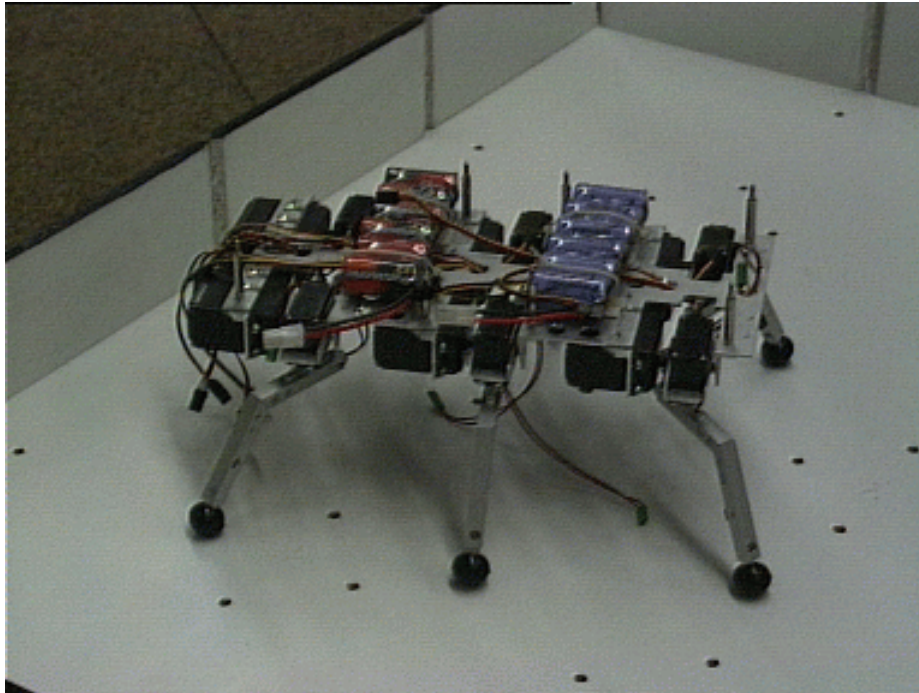


Abbildung 1.5: Rug Walker

Einige Technische Daten zu Rug Walker sind:

Gewicht ohne Akkus und Steuerelektronik: ~2kg

Länge: 323 mm

Stromversorgung: 2 mal 6 Akkupacks a 1.2V

- ein Pack für die Steuerplatine, eines für die Mechanik

maximale theoretische Höhe eines Hindernisses: 127 mm

- dies entspricht dann auch gleichzeitig der Beinlänge

Breite (ohne Beine): 154 mm

Mechanik: 12 Servos (davon 6 starke vertikal, 6 leichte horizontal)

Sensorik: - 6 Tastsensoren an den Beinenden (Mikroschalter)

- 2 Entfernungssensoren vorne links und rechts

1.6 The Beast

The Beast ist der Spitzname für eine sechsbeinige Gehmaschine der Firma Lynxmotion (siehe auch [Lynxmotion 97]), welche ursprünglich den Modellnamen Hexapod II hatte. Diese Gehmaschine kann man bei der amerikanischen Firma Lynxmotion bestellen, weswegen sie auch von der Stuttgarter Elektronik angesteuert werden soll. Geliefert wird sie als Rohling, nicht zusammengebaut und kostet in diesem Zustand 375 \$. In zusammengebautem Zustand ist sie etwas teurer nämlich 475 \$. Die Maße sind im wesentlichen gleich wie beim Rug Walker nur besitzt „the Beast“ die Qualität, daß die Beine über einen speziellen Mechanismus vom Boden gehoben werden, was eine gewisse Grundreibung, die beim Rug Walker bei hoher Gangart

zwangsläufig auftritt, verhindert. Durch diesen Mechanismus ermöglicht die Gehmaschine es auch kleinere Servos für die vertikale Bewegung zu verwenden, was den gesamten Aufbau wesentlich leichter macht. Ebenfalls durch diesen Mechanismus wird auch Strom gespart, wodurch der Sechsheiner länger aktiv sein kann (siehe auch Abbildung 1.6).

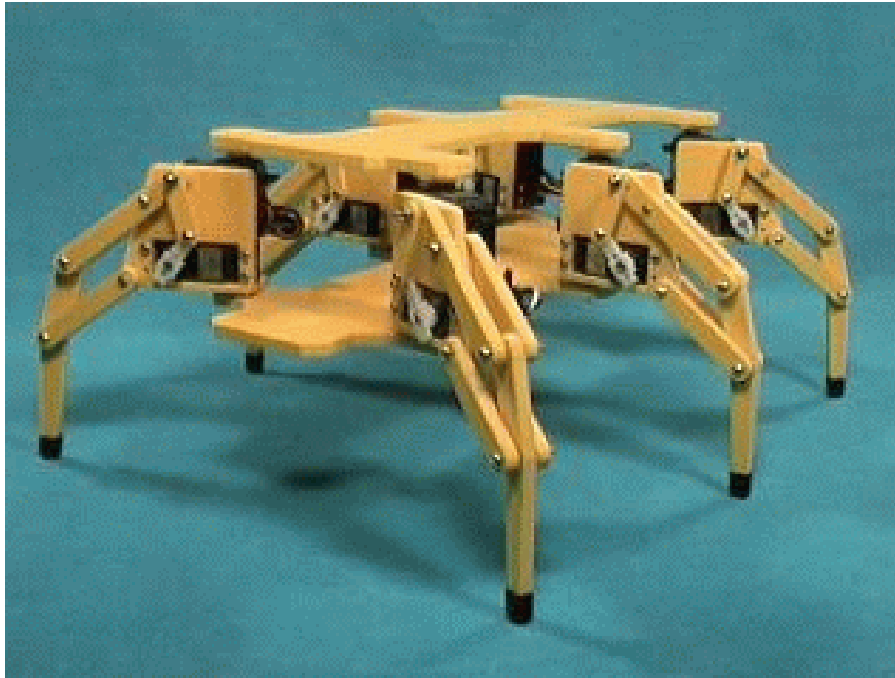


Abbildung 1.6: The Beast

Einige Technische Daten zum Beast sind:

Gewicht mit voller Beladung: ~2,4 kg

Länge: 331 mm

Stromversorgung: 2 mal 6 Akkupacks a 1.2V

- ein Pack für die Steuerplatine, eines für die Mechanik
maximale theoretische Höhe eines Hindernisses: 81 mm

- dies entspricht dann auch gleichzeitig der Beinlänge

Breite (mit Beinen): 273mm

Mechanik: 12 Servos (12 leichte)

Sensorik: 2 Infrarot-Entfernungssensoren vorne links und rechts

- PSDs von Sharp mit einem Meßbereich von 8 - 80 cm

2 Schichtenmodell zur Fortbewegung

Das Gehmaschinen-vier-Schichtenmodell, siehe Abbildung 2.1 besteht aus 4 Schichten, die es ermöglichen sollen einen Roboter über einen modularen Programmaufbau übersichtlicher zu steuern. Diese Schichten sind nicht nur für dieses spezielle Projekt geeignet, sondern lassen sich durchaus auch in zukünftigen Projekten einsetzen. Das Schichtenmodell soll den modularen Programmaufbau realisieren und die Implementierung eines solchen Projektes erleichtern.

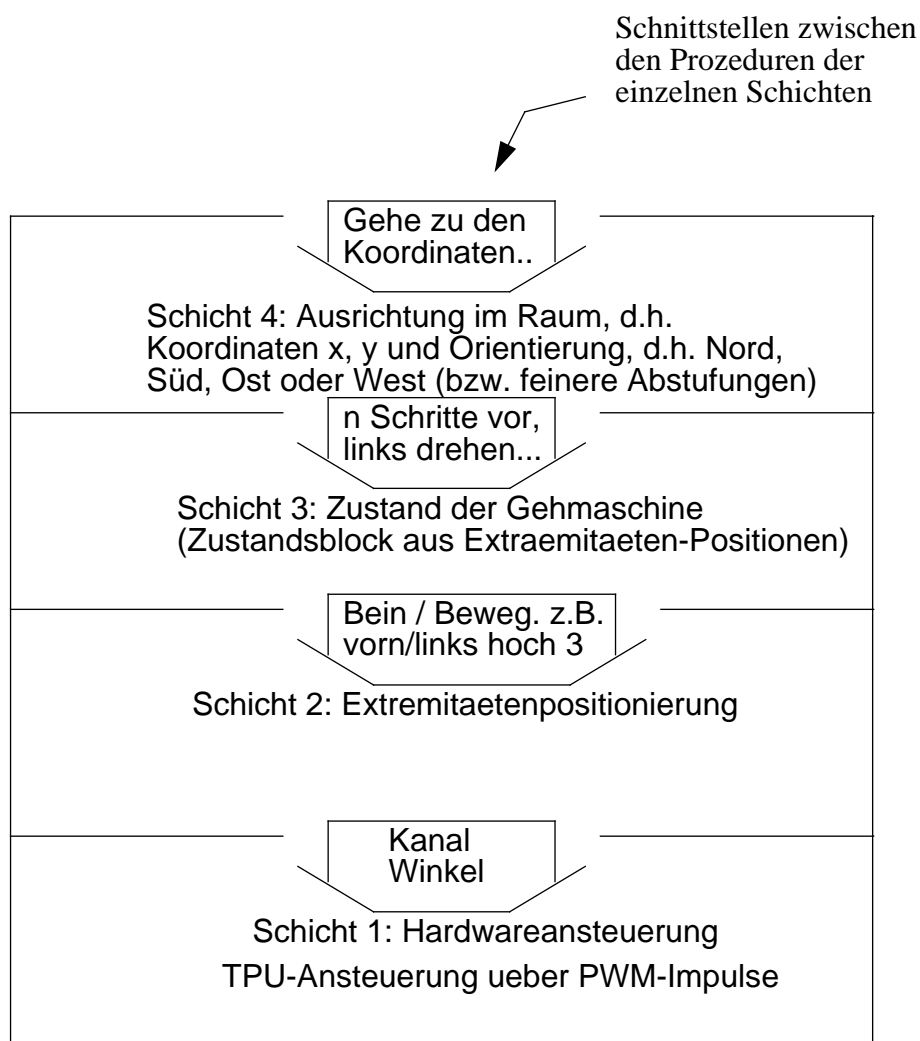


Abbildung 2.1: Vierschichtenmodell

2.1 Schicht eins: Hardwaresteuerung

Die Schicht eins kümmert sich um die Hardwareansteuerung und wird daher typisch in Assembler programmiert. Sie stellt für die nächst höhere Schicht Funktionen bereit, die es ermöglichen ein Gelenk des Roboters in eine bestimmte Position zu bringen. Als Hardware wurden in diesem Fall Servos benutzt, die über Impulse von der TPU angesteuert werden (siehe Kapitel 3: Implementierung), es sind jedoch auch durchaus andere Realisierungen möglich wie z.B. Servo-Motoren, Schrittmotoren oder gar Antriebe, die nur zwei diskrete Zustände haben. Der nächst höheren Schicht werden in diesem Fall Funktionen bereitgestellt, die einen Parameter zum selektieren des Servos bzw. des TPU-Kanals übernehmen, sowie einen zum Bestimmen des Winkels, der in Altgraden angegebene wird. Diese Parameter sind dann bei den Servo-Motoren die Nummer des Motors sowie die prozentual angegebene Geschwindigkeit, welche ebenfalls als Parameter einer dafür gemachten Funktion übergeben werden.

Als Hardwaresteuerung gilt auch das Auslesen der Sensoren (Tastsensoren bzw. Ultraschallansteuerung). Wie diese Komponenten programmiert wurden, kann man im einzelnen im Kapitel Implementierung nachlesen.

Diese Schicht kann man auch dazu verwenden um bei größeren Robotern die einzelnen Motoren anzusteuern, auf welche Art dies auch immer geschehen mag. Sie soll dazu dienen die Hardware anzusteuern, um der nächst höheren Schicht eine etwas abstraktere Funktion anzubieten, die als Parameter eine Aktornummer sowie einen Winkel oder eine andere Rationalskala besitzt.

Die hierzu passenden Funktion sind in Anhang A dokumentiert.

2.2 Schicht zwei: Extremitätenpositionierung

Die Schicht zwei behandelt die Positionierung der Extremitäten, in diesem Fall sind dies Beine, die vertikal und horizontal jeweils fünf verschiedenen Positionen (bzw. Winkel) einnehmen können, dies könnte bei einem Fahrzeug die Geschwindigkeit eines Rades sein. Diese Schicht baut auf der ersten auf, d.h. sie benutzt Funktionen der ersten Schicht um selber Dienste für die nächst höhere Schicht zur Verfügung zu stellen. Die Funktionen, welche angeboten werden, bekommen Parameter, welche die Extremität genauer beschreiben, sowie einen der die Ausprägung der Bewegung bestimmt. Welches Gelenk eines Beines bewegt wird, bestimmt die aufgerufene Funktion.

In diesem konkreten Fall existiert eine Funktion zum Beinheben (lift) wobei VUP (Very Up) der höchst Zustand ist und VDN (Very Down) der niederste. Des weiteren existiert eine Funktion zum rückwärts bewegen (move) eines Beines, wobei VBK (Very Back) der Zustand am weitesten hinten, VFN (Very Front) der Zustand ist, bei dem das Bein am weitesten vorne ist. In Abbildung 2.2 sieht man, wie aus einer abstrakten Ansteuerungsfunktion für das Gelenk, welches am rechten, vorderen Bein für die Bewegung nach hinten zuständig ist, eine konkreter Funktion für eine Winkelorientierte Bewegung des Servos wird, d.h. es wird in Schicht zwei eingegeben: „Bewege rechtes, vorderes Bein nach hinten“. Schicht zwei ruft dadurch eine Funktion aus Schicht eins auf, der sie das Kommando übergibt das Servo Nr. 3 zum Winkel 68 Grad zu bewegen.

Die Erklärung zu den entsprechenden Funktionen steht in Anhang B.

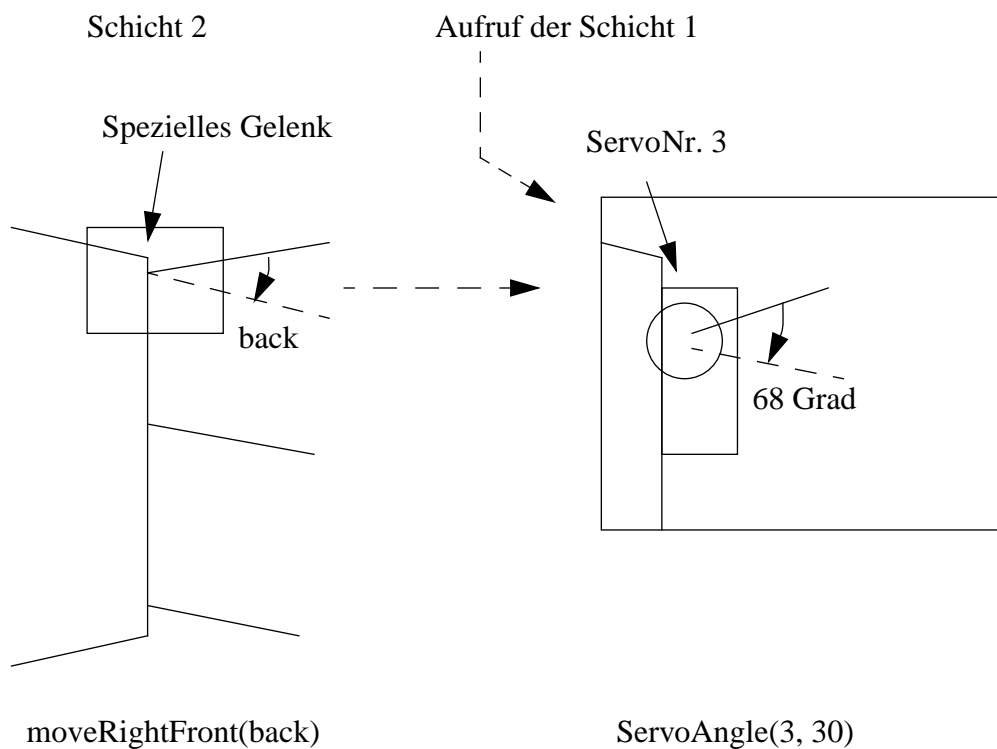


Abbildung 2.2: Gelenkansteuerung durch Schicht zwei

2.3 Schicht drei: Zustand des Agenten

Die Schicht drei liefert an Schicht vier, basierend auf den Funktionen der Schicht zwei, verschiedene *Koreographien* bzw. Bewegungsmuster, welche die Gehmaschine ausführt, um sich selbst im Raum vorwärts, rückwärts oder um die eigene Achse drehend zu bewegen. Solche Koreographien sind beispielsweise „mache n Schritte vorwärts“. Diese Koreographien werden über sogenannte *Zustandsblöcke* (s.u.) programmiert.

Ein Zustandsblock ist ein Vektor, welcher für jedes Gelenk eine bestimmte Position enthält. Reiht man nun alle in einer Koreographie enthaltenen Zustandsblöcke aneinander, so entsteht eine Koreographie. Diese Vorgehensweise kennt man aus der Automatentheorie, weshalb man die Übergänge innerhalb einer Koreographie bis zum Endzustand oder *stabilen Zustand* als *Lambda-Übergänge* bezeichnen kann. Ist eine Koreographie zu Ende geführt, so kann sie wiederholt werden, bzw. eine neue gestartet werden.

In diesem Fall ist der Zustandsblock ein Feld, aus sechs Paaren (pro Bein ein Paar: Vertikales Gelenk, horizontales Gelenk), welches für jedes Bein der Gehmaschine eine bestimmte Position enthält.

In Abbildung 2.3 ist die Koreographie für „oneStep“ d.h. einen Schritt dargestellt. Jede Skizze der Gehmaschine repräsentiert einen Zustandsblock. In der Abbildung sind die Beine, welche den Boden berühren, mit einem Kreis am Ende dargestellt. Der letzte Zustand ist auch gleichzeitig der stabile Zustand, weil die Gehmaschine in diesem verbleibt. Es ist sinnvoll den stabilen Zustand bei jeder Koreographie gleich zu wählen, damit sich eine Art Baukasten aus Koreographien für die vierte Schicht ergibt, mit welchem sich dann relativ flüssig größere

Bewegungsfolgen zusammenbauen lassen, ohne daß man jeweils noch Übergänge von einem stabilen Zustand in einen anderen einfügen muß. Dies macht den gesamten Aufbau solider, weil in der vierten Schicht auf den stabilen Zustand nicht mehr Rücksicht genommen werden muß und somit weniger Fehler entstehen können. (Wäre dies nicht so, dann könnte die Gehmaschine bei Nichtberücksichtigung des stabilen Zustandes Schaden nehmen!)

Die Erklärungen zu den entsprechenden Funktionen befinden sich in Anhang C.

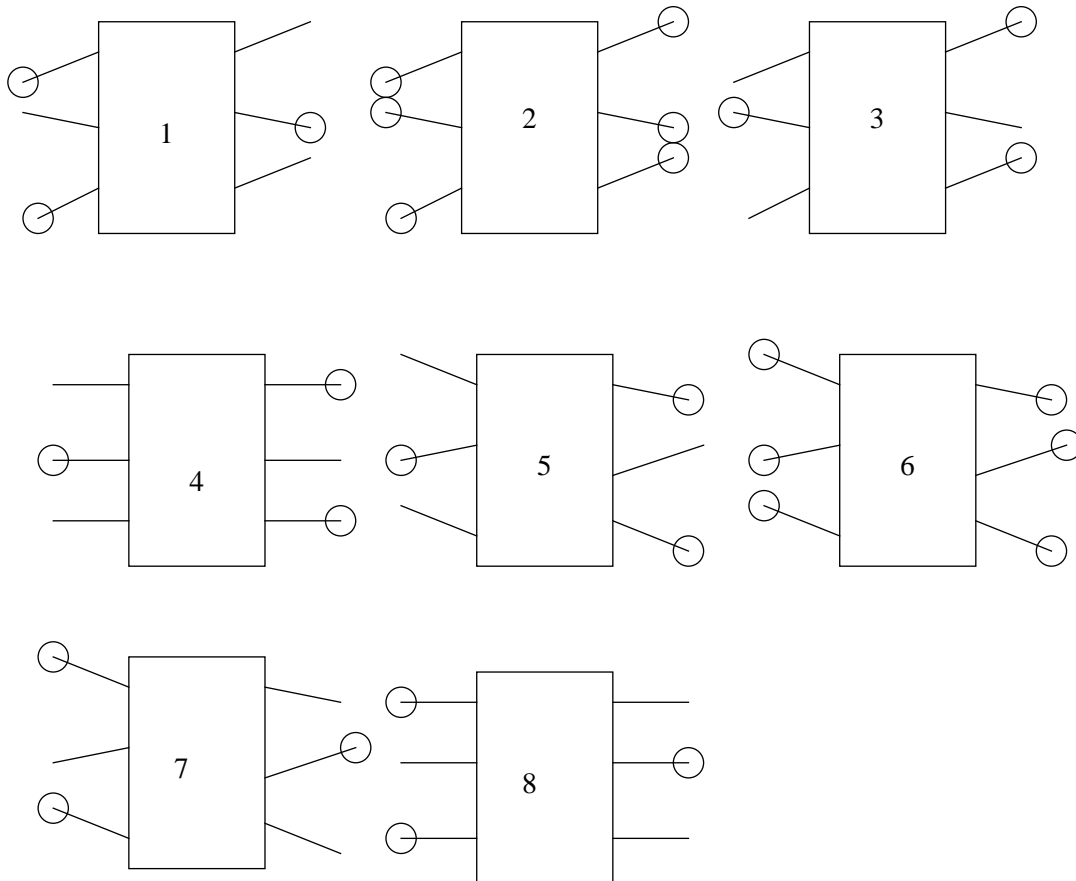


Abbildung 2.3: Koreographie „oneStep“

2.4 Schicht vier: Ausrichtung im Raum

Die vierte und oberste Schicht baut auf der dritten Schicht auf und ermöglicht somit eine gezielte Fortbewegung im Raum. Die vierte Schicht ermöglicht es dem Roboter Zielkoordinaten einzugeben, wonach er sich selbst den Weg zu diesem Platz sucht und eventuell im Weg stehenden Hindernissen ausweicht.

Das in dieser Situation verwendete Verfahren arbeitet wie folgt: Zuerst bekommt der entsprechende Programmteil die Quell- und die Zielkoordinaten eingegeben. Die Hauptschleife wird solange wiederholt, bis der Zielpunkt erreicht ist, bzw. solange der Abstand zum Zielpunkt nicht geringer als eins ist.

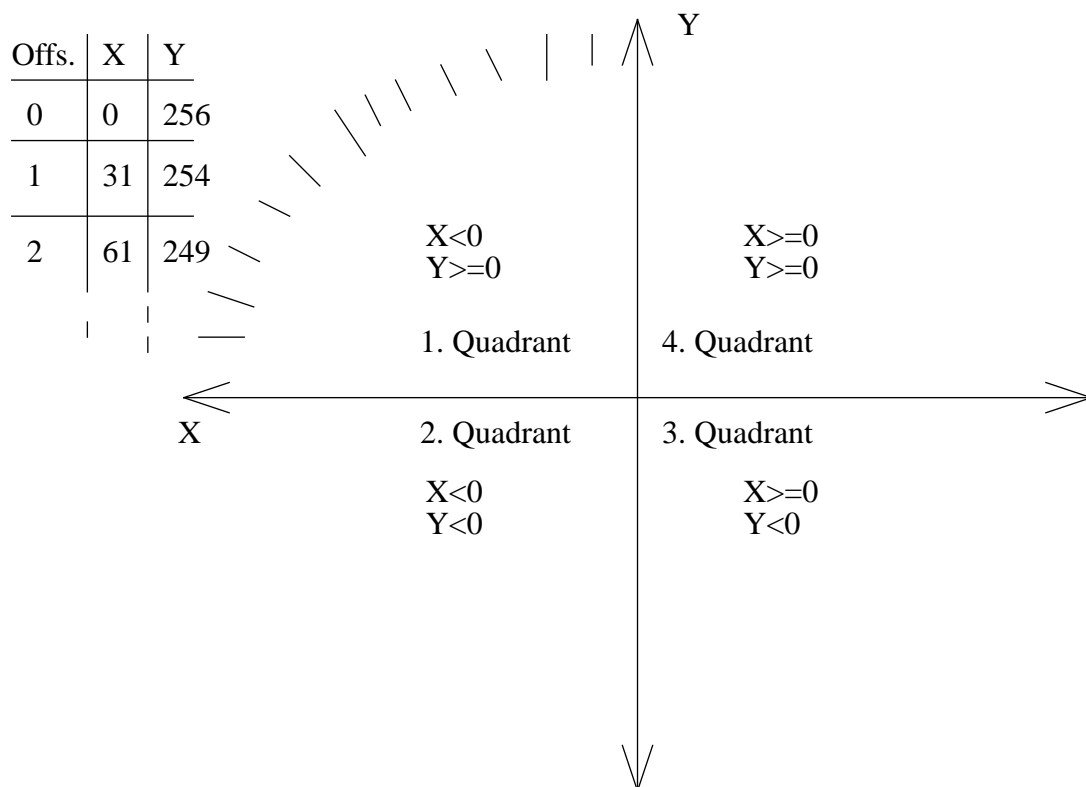


Abbildung 2.4: Ermittlung der zu Ziel-Himmelsrichtung

- Darin wird als erstes die Richtung berechnet, in welche die Gehmaschine auf dem schnellsten Weg zu ihrem Ziel kommt. Mit den Schrittweiten, die mit den drei unteren Schichten realisiert wurden, lassen sich 52 verschiedene Orientierungen einstellen, in die die Gehmaschine sich bewegen kann. Es wird also als erstes die Richtung ausgesucht, mit deren Hilfe das Ziel auf dem direktesten Weg erreicht werden kann, was wie folgt geschieht: Da der verwendete Rechner keine Realzahlen verwendet, muß man dies mit Integerzahlen realisieren, was schon das erste Problem darstellt. Es wird deswegen als erstes der Quadrant ermittelt (siehe Abbildung 2.4), in dessen Richtung die Gehmaschine gehen soll, was aufgrund der Differenz zwischen Ziel und Quelle geschieht. Die Bedingungen, die erfüllt sein müssen, damit ein Quadrant ausgesucht wird, stehen in der Mitte in Abbildung 2.4. Hat man den Quadranten, so tastet man sich in der entsprechenden Tabelle (links oben in Abbildung 2.4) an den Zieloffset heran, welcher dann auch die Richtung bestimmt, in welche die Gehmaschine dann gehen muß.

- Wenn die Gehmaschine losgeht, so überprüft sie nach jedem Schritt, ob vor ihr ein Hindernis steht und bricht die Gehbewegung unter Umständen ab. Nachdem die Gehbewegung beendet wurde, wird als Ergebnisparameter die Anzahl der Schritte zurückgegeben, welche mit Hilfe des Winkels und der alten Position die Berechnung der neuen Position zuläßt.

- Wurde die Gehbewegung durch ein Hindernis unterbrochen, d.h. es sind weniger Schritte gegangen worden als vorgegeben waren, so könnte die Gehmaschine zuerst theoretisch versuchen das Hindernis zu übersteigen.

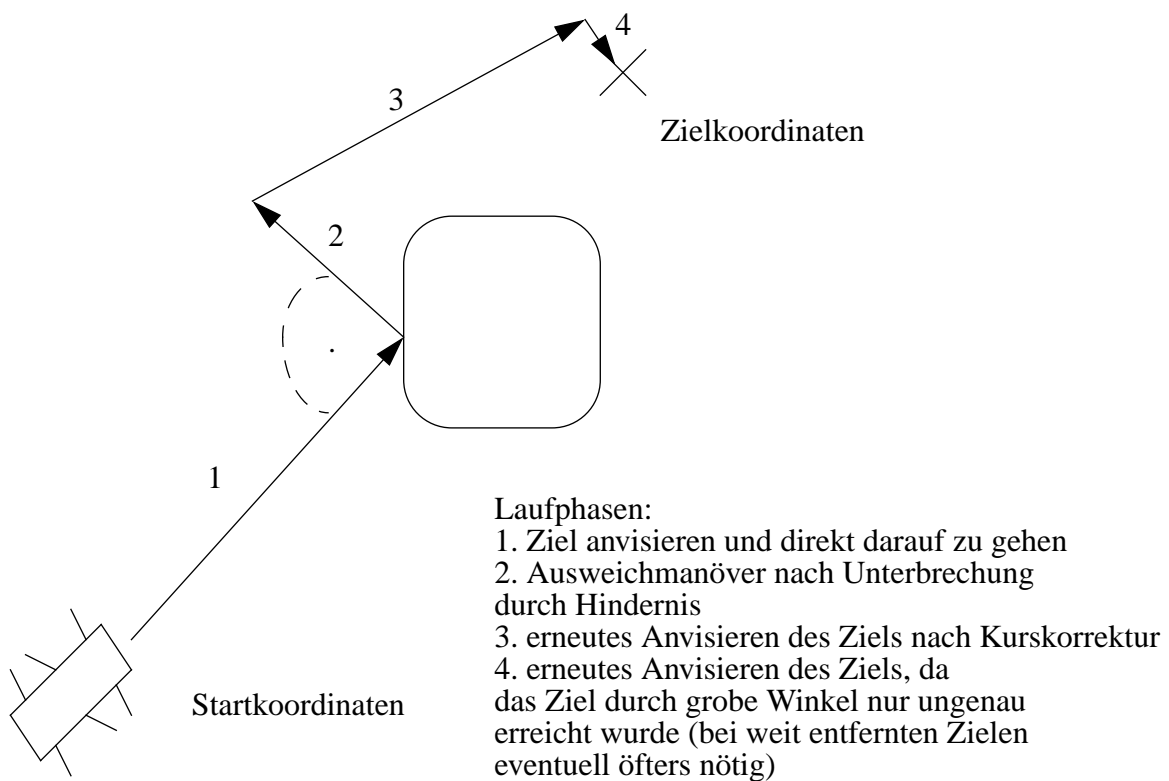


Abbildung 2.5: Bahn der Gehmaschine zum Ziel

- Wurde die Gehbewegung durch ein Hindernis unterbrochen, welches zu hoch zum übersteigen ist, bzw. durch eines, auf das kein Übersteigeversuch folgte, so wird ein Ausweichmanöver gestartet (siehe Abbildung 2.5: Bahn der Gehmaschine zum Ziel). Da die Gehmaschine sowohl links als auch Rechts einen Sensor hat, der Hindernisse wahrnehmen kann, wird versucht das Hindernis auf der Seite zu umgehen, auf der das Objekt eine größere Entfernung zur Gehmaschine hat, bzw. auf der kein Objekt wahrgenommen wird. Bei der Umgehung des Hindernisses wird versucht, dementsprechend ob links oder rechts ein größerer Freiraum wahrgenommen wird, zehn Schritte in die entsprechende Richtung, orthogonal zur direkten Zielrichtung zu machen, so daß eventuell kein Hindernis mehr ihm Weg steht. Danach wird die Schleife noch einmal angesprungen, um das Ziel erneut anzuvisieren und auf es zu gehen. Wurde das Hindernis umgangen, bzw. existierte keines und die vorgegebene Lauffolge wurde komplett beendet, so kann es trotzdem sein, daß die Zielkoordinaten immer noch nicht erreicht wurden, weswegen die Schleife noch einmal angesprungen wurde.

3 Implementierung

3.1 Ansteuerung eines Servos

Ein Servo ist ein kleiner Motor, dessen eigentlicher Verwendungszweck die Lenkbewegung von Rädern bzw. Rudern in ferngesteuerten Modellbaueinheiten ist. Die Steuersignale für das Servo werden dabei vom Empfänger im Fahrzeug über die Steuerleitung an das Servo gegeben. Diese Steuersignale sind pulsbreitenmodulierte Rechtecksignale.

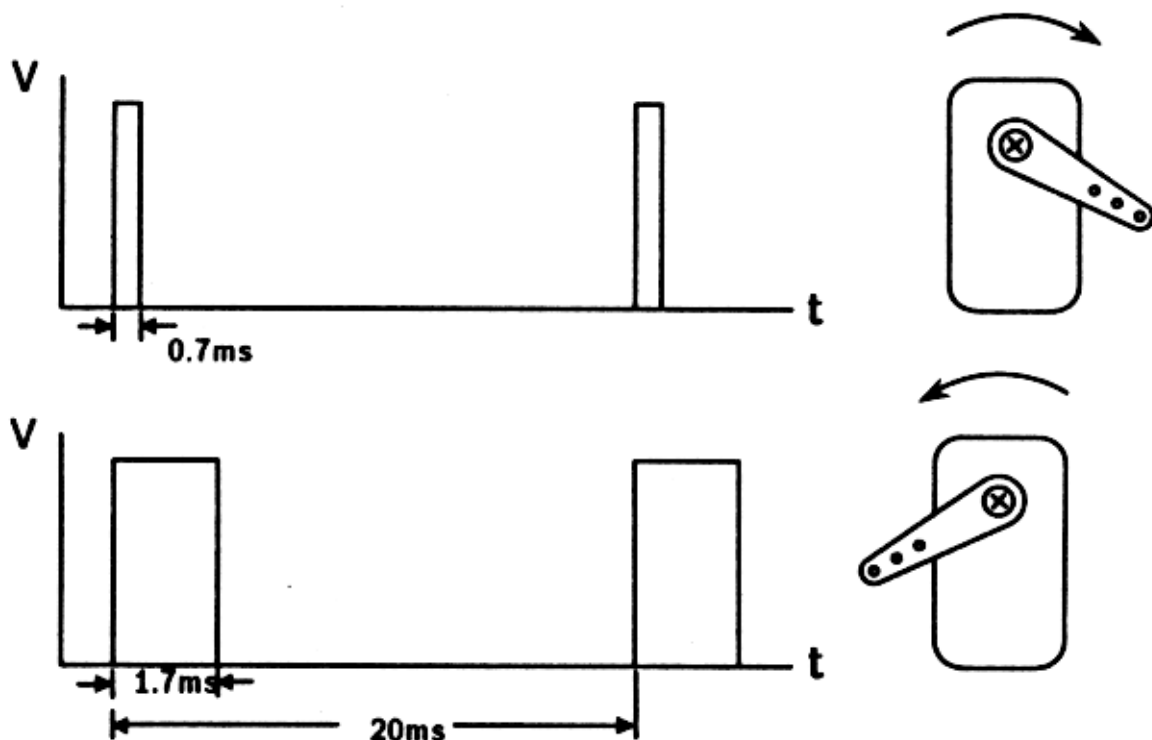


Abbildung 3.1: Ansteuerung eines Servos

Des Weiteren besitzt das Servo noch die Eingangsleitungen, Masse und Spannungspotential H_i , was alles von der zur Verfügung gestellten Platine ausgegeben werden kann, weswegen es sich hervorragend für dieses Projekt eignet.

Die Steuerung über die Steuerleitung funktioniert nach [Jones et al. 93] wie folgt:

Auf die Steuerleitung wird ein positives Rechtecksignal ausgegeben, wobei die Dauer während der es H_i bleibt den Winkel bestimmt, den das Servo einnimmt. Diese Dauer schwankt zwischen 0,7 ms und 1,7 ms, wobei der Wert für den mittleren Winkel bei 1,3 ms liegt. Danach wird das Signal auf L_o gesetzt, was zusammen mit der Passage, in der das Signal H_i ist, 20 ms dauert (Periodendauer). Ist ein solcher Steuerzyklus abgelaufen, kann ein neuer begonnen werden. Siehe auch Abbildung 3.1

3.2 Anschluß der Servos an die Eyebot-Platine

Die Platine führt über den „Connector 3 (Servo Port)“ die von den Servos benötigten Signale der TPU (Time Processor Unit) eines Controllerbausteins des MC68332 nach außen. Des Weiteren stellt der „Connector 3“ zusätzlich zu jeder Signalleitung je einen Massenausgang und je einen Hi-Ausgang für insgesamt 16 Servos zur Verfügung.

Mit zwei weiteren nichtbelegten Polen besitzt „Connector 3“ also fünfzig Leitungen, von denen für die Servoansteuerung jedoch nur zwölf mal drei benötigt werden.

Die Belegung von „Connectors 3“ ist aus Abbildung 3.2 abzulesen.

Pin	Description	Pin	Description
1	TPU Channel 15	2	Servo Supply (+6V)
3	Digital Ground	4	TPU Channel 14
5	Servo Supply (+6V)	6	Digital Ground
7	TPU Channel 13	8	Servo Supply (+6V)
9	Digital Ground	10	TPU Channel 12
11	Servo Supply (+6V)	12	Digital Ground
13	TPU Channel 11	14	Servo Supply (+6V)
15	Digital Ground	16	TPU Channel 10
17	Servo Supply (+6V)	18	Digital Ground
19	TPU Channel 9	20	Servo Supply (+6V)
21	Digital Ground	22	TPU Channel 8
23	Servo Supply (+6V)	24	Digital Ground
25	TPU Channel 7	26	Servo Supply (+6V)
27	Digital Ground	28	TPU Channel 6
29	Servo Supply (+6V)	30	Digital Ground
31	TPU Channel 5	32	Servo Supply (+6V)
33	Digital Ground	34	TPU Channel 4
35	Servo Supply (+6V)	36	Digital Ground
37	TPU Channel 3	38	Servo Supply (+6V)
39	Digital Ground	40	TPU Channel 2
41	Servo Supply (+6V)	42	Digital Ground
43	TPU Channel 1	44	Servo Supply (+6V)
45	Digital Ground	46	TPU Channel 0
47	Servo Supply (+6V)	48	Digital Ground
49	NC	50	NC

Abbildung 3.2: Belegung des Connector 3 (Servo Port) auf der Eyebot-Platine

3.3 Anschluß der Servos an die TPU

Die Gehmaschine besitzt, wie oben schon erwähnt, sechs Beine, die über TPU-Kanäle und Servomotoren bewegt werden. Pro Bein gibt es zwei Servomotoren, welche jeweils ein Gelenk bzw. eine Dimension pro Bein bewegen. Jeweils ein Servo bewegt ein Bein vertikal, und ein weiteres Servo dient zur horizontalen Bewegung. Die Verteilung der Servos an die TPU-Kanäle wurde wie folgt vorgenommen, siehe auch Abbildung 3.2, was allerdings nicht so sein muß, da die Beinbelegung in der HDT eingestellt werden kann/muß:

- TPU-Kanal0 - Bein hinten links vor-zurück
- TPU-Kanal1 - Bein hinten links heben
- TPU-Kanal2 - Bein hinten rechts vor-zurück
- TPU-Kanal3 - Bein hinten rechts heben
- TPU-Kanal4 - Bein mitte links vor-zurück
- TPU-Kanal5 - Bein mitte links heben
- TPU-Kanal6 - Bein mitte rechts vor-zurück
- TPU-Kanal7 - Bein mitte rechts heben
- TPU-Kanal8 - Bein vorne links vor-zurück
- TPU-Kanal9 - Bein vorne links heben
- TPU-Kanal10 - Bein vorne rechts vor-zurück
- TPU-Kanal11 - Bein vorne rechts heben

3.4 Ansteuerung der TPU (Time Processor Unit)

Die TPU, der Baustein mit dem die Servos angesteuert werden, ist eine eigene microcodeprogrammierte CPU, die im MC68332 enthalten ist. Sie besitzt einen eigenen Schreib-Lese-Speicher, der auch gleichzeitig die Schnittstelle zum Prozessor (einem MC68000) bildet, der in dem MC68332 das Herzstück ist. Diese TPU kann verschiedene Aufgaben übernehmen, mit deren Bewältigung der Prozessor, alleine (ohne Coprozessoren) schon ganz ausgelastet wäre. Die TPU kann pulsbreitenmodulierte Rechtecksignale erzeugen, die für diesen Zweck ideal geeignet sind, des weiteren kann man mit der TPU Schrittmotoren steuern, man kann Zeitmessungen vornehmen und sogar selber kleine Mikro-Programme schreiben, die von der TPU abgearbeitet werden, siehe auch [Knülle-Wenzel 94]. Mit der TPU kann man auch Signale auswerten, wie z.B. Infrarotsignale oder Ultraschallsignale.

Die Programmierung der TPU, so daß eine PWM (Pulsbreitenmodulation - Puls Width Modulation) abgegeben wird, geschieht über Register, die sich in einem Zweiwegespeicher befinden. Parameter, welche die Länge des Impulses bzw. die Wellendauer bestimmen, werden im sogenannten PRAM (Parameter RAM) abgelegt. Es muß einer der beiden in der TPU enthaltenen Zähler ausgewählt werden, der nach einem Systemstart nur einmal initialisiert werden kann. In diesem Projekt wurde Zähler zwei gewählt, da der erste höher getaktet werden kann und somit für entsprechende Aufgaben zur Verfügung stehen muß. Hierbei ist zu beachten, daß die Leitung T2CLK auf Hi gelötet sein muß, siehe [Harman 91][Motorola 93], damit der zweite Zähler abhängig vom CPU-Takt als solcher verwendbar ist (dies entspricht einem Zusammenlöten der Pinns). Der Zähler wurde mit $1/32$ des CPU-Taktes initialisiert, also mit 2^{19} Hz ($\sim 1/2$ MHz). Bei höheren Frequenzen bekommt man das Problem, daß der Zähler um 20 ms lang hochzuzählen s.o. einen Wert größer als 8000_{16} erreicht. Die Differenz zwischen dem Zählerstand, nach 20 ms langem zählen und dem Zählerstand bei der fallenden Flanke, darf allerdings nicht höher als 8000_{16} werden.

Ein weiteres zeitraubendes Problem war die Initialisierung der Priorität, die pro Kanal zwei Bit belegt: Die Priorität muß auf einen Wert, höher als null, gesetzt werden, sonst wird dem TPU-Kanal keine Rechenzeit zukommen gelassen. Bei der Rechenzeitverteilung gibt es abhängig von der Priorität ein ganz bestimmtes Scheduling innerhalb der TPU, wobei mit steigendem Wert in den zwei Prioritätsbits auch die „Genauigkeit steigt“ mit der eine Flanke stattfindet. Es handelt sich hierbei um einen schwach fairen Timingalgorithmus, siehe [Fuchs 95]. Die Priorität ist außerdem der Wert, der die Initialisierung der gesamten TPU-Funktion bewirkt, und somit immer als letztes zu schreiben. Will man eine TPU-Funktion deaktivieren, muß man lediglich die Priorität auf null setzen.

Die TPU wird wie oben erwähnt, auch noch zur Aufnahme von Sensordaten verwendet. Hierzu braucht man die IC/ITC-Funktion (Input Capture/ Input Transition Counter), deren Aufgabe es ist, nach einer vorgegebenen Anzahl von positiven, negativen oder allgemeinen Spannungsspiegelanken ein Signal abzugeben. Dies kann sich in Form eines TPU-Interrupts äußern, indem z.B. nach 1000 Flanken ein Programm aufgerufen wird, in dem ein Timerstatus gerettet wird, was für eine Ultraschall-Entfernungsbestimmung wichtig ist. Diese Funktion kann aber auch ohne Interrupt schlicht dazu verwendet werden, einen Wert im PRAM, dem Parameter-RAM der TPU hochzuzählen und somit die Aktivitäten eines anderen Kanals zu verändern.

3.5 Ansteuerung der Tastsensoren

Die Gehmaschine besitzt an jedem Bein einen Tastsensor, der durch einen Mikroschalter realisiert ist, welcher betätigt wird, sobald die Gehmaschine das entsprechende Bein auf den Boden setzt. Die Tastsensoren sind sogenannte Sensoren, die im Gegensatz zu Aktoren eine beobachtende Verbindung mit der Außenwelt herstellen sollen und somit ein Ersatz für „Menschliche Sinnesorgane darstellen“.

Die Abfrage der Tastsensoren geschieht beim Rug Walker ohne intelligente Zwischeninstanz, d.h. es ist kein weiterer Prozessor zwischen den 68332 und die Sensoren geschaltet. Die Ansteuerung erfolgt somit direkt über den Adreßbus. Hierbei wurden nur die Adreßleitungen A23, A22, A21 und A2, A1, A0 verwendet, i.a.W. die dazwischenliegenden Adreßleitungen A3 - A20 können beliebig belegt werden. Die Belegung der Adreßleitungen muß nun so gewählt werden, daß A23 und A0 Hi sind, die restlichen der aufgezählten müssen Lo sein.

Danach wird das untere Byte des Datenbusses mit den Signalen der Tastsensoren belegt, wobei zu beachten ist, daß diese Signale allesamt Lo-active sind.

Die unteren sechs Bits sind mit den Tastsensoren verbunden, die obersten beiden Bits sind noch unbelegt und könnten eventuell mit weiteren Tastsensoren z.B. an der Unterseite der Gehmaschine bestückt werden, um eine häufig eintretende Situation zu erkennen - die Gehmaschine liegt auf dem Bauch.

Theoretisch müßte der Assemblerausdruck „move.b 0x800001,%d0“ die entsprechenden Bits auslesen, wonach man sie verarbeiten könnte, dies funktioniert aber noch nicht, es gibt dabei lediglich einen Systemabsturz.

Bei der Gehmaschine Beast sollen diese Tastsensoren an die digitalen Eingänge 2 - 7 angeschlossen werden, hierfür existieren nun auch entsprechende Abfrageroutinen, siehe Anhang A.

3.6 Ultraschallerzeugung

Da die Gehmaschine Sensoren benötigt, die brauchbare Werte liefern, auf Grund derer sie sich sinnvoll im Raum bewegen kann, wurde entschieden, daß Rug Walker mit Ultraschalldetektoren ausgestattet werden soll. Ultraschall wird dazu verwendet um die Entfernung zu im Weg stehenden Hindernissen zu messen. Dies geschieht schlicht indem man die Laufzeit des Signals mißt und aufgrund der Laufzeit mit Hilfe der Schallgeschwindigkeit die Strecke berechnet.

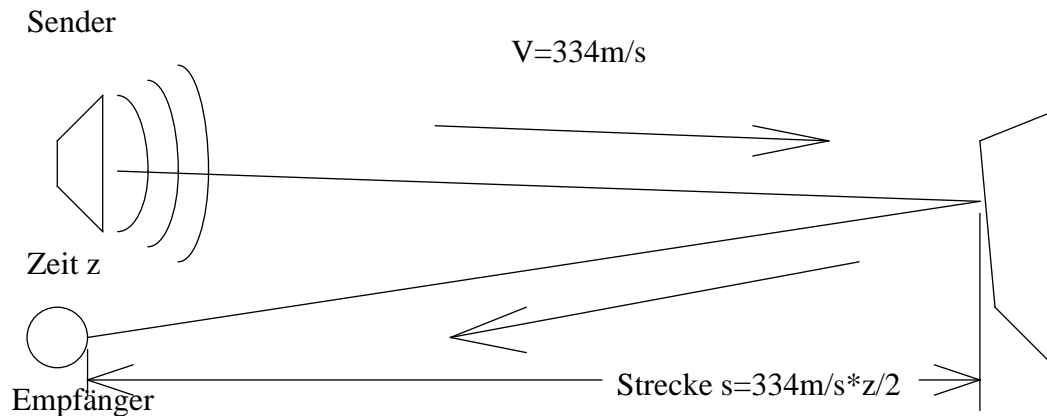


Abbildung 3.3: Entfernungsmessung mit Ultraschall

Die Signallaufzeit z vom Sender zum Empfänger muß durch zwei geteilt werden, da das Signal die Strecke, welche bestimmt werden soll, zweimal abläuft, nämlich einmal hin und einmal zurück. D.h. als Zeitkomponente wird $z/2$ genommen. Da der Schall sich mit Schallgeschwindigkeit fortbewegt, welche 334 m/s beträgt, kann man laut der Formel $s=V\cdot t$ mit $t=z/2$ die Entfernung zum nächsten schallreflektierenden Objekt messen, welche $s=334\text{m/s}\cdot z/2$ beträgt (siehe Abbildung 3.3).

Die Gemaschine ist u.U. mit zwei Ultraschalldetektoren ausgestattet, welche dann über TPU-Kanal 12 und 13 angesteuert werden. Natürlich soll dies nicht statisch sein, sondern von der HDT erkannt werden. Der an TPU-Kanal 13 angeschlossene „Ultraschall-Lautsprecher“ soll Ultraschall-Signale erzeugen und das an TPU-Kanal 12 angeschlossene „Ultraschall-Mikrofon“ soll sie auffangen.

Da nur zwei TPU-Kanäle existieren, mit denen nur ein „Ultraschalldetektor“ betrieben werden kann, da aber zwei „Ultraschalldetektoren“ existieren, hat Waldemar Spädt eine Schaltung gebaut, welche ähnlich zu der ist, die bei den Tastsensoren verwendet wurde: Es werden wieder die Adreßleitungen abgegriffen; wenn eine bestimmte Adresse (800000_{16}) angelegt wird, wobei nur die Adreßleitungen 0, 1, 2 und 21, 22, 23 von Bedeutung sind kann ein Bytewert eingegeben werden, welcher darüber entscheidet ob der linke oder der rechte Ultraschallsensor aktiviert sein soll (hier hat sich wohl das gleiche Problem wie bei den Tastsensoren eingeschlichen).

Ist ein Detektor ausgesucht, so kann der Kanal 12 mit der IC/ITC-Funktion initialisiert werden. Dabei wird der Zähler zwei ausgesucht, denn mit 2^{19} Hz ist er schnell genug, so daß der Schall pro Zählersignal $1/2\text{ mm}$ zurücklegt, allerdings kann eine größere Entfernung als bei Zähler eins gemessen werden. Der TPU-Kanal wird auf Continuous Mode eingestellt, damit immer das letzte Signal das Festhalten des Zählerstandes veranlaßt. Bei dieser Initialisierung kann auch noch eingestellt werden, wieviele Signale ankommen sollen, damit der Kanal einmal reagiert, hierbei wird ein Signal eingestellt, und als letzter Punkt wird eingestellt, daß sowohl positive als auch

negative Signalwechsel eine Reaktion auslösen, denn nach dem letzten Signal, das der Sender absendet, gibt er keine Signale mehr ab (auch nicht solche um in den Urzustand zurückzukehren).

Als nächstes wird der Kanal 13 mit der Funktion PWM auf Senden eingestellt. Ausgewählt wird ebenfalls Zähler zwei (passend zum Empfänger). Danach müssen passende PWM_Per- bzw. PWM_Hi-Werte (in diesem Fall die Hälfte von PWM_Per) eingestellt werden, damit Ultraschall erzeugt wird. Mit Ultraschall ist Schall gemeint, der eine höhere als die vom Menschen hörbaren Frequenzen hat, was wichtig ist, damit man nicht durch die ständigen Entfernungüberprüfungen von der Gehmaschine belästigt wird. Der Schall soll hier eine Frequenz von 30 kHz haben, was einen PWM_Per Wert zur Folge hat, der wie folgt berechnet wird: $\text{PWM_Per} = T_{\text{Ultraschall}} / T_{\text{Zähler}} = f_{\text{Zähler}} / f_{\text{Ultraschall}} = 2^{19} \text{Hz} / 30 \text{kHz} = 17.46$, hier wird dann 18 genommen, da dieser Wert auch adäquat ist (T ist hierbei die Periodendauer also der Kehrwert der Frequenz). Nachdem ausreichend Zeit vergangen ist, damit mindestens ein Signal abgesendet wurde, kann Kanal 13 wieder abgestellt werden und der gemerkte Zählerstand ausgelesen werden. Danach muß erneut gewartet werden bis das letzte Schallsignal zurückgekommen ist (also 1/100 Sekunde), wonach auch Kanal 12 abgeschaltet und ausgelesen wird. Nun kann die Differenz d zwischen dem letzten Empfängerzählerstand und dem letzten Senderzählerstand berechnet werden (bei einem Negativen Wert muß 10000_{16} zu d dazuaddiert werden, da es im 16 Bit langen Zähler zwischen Sendesignal und Empfangssignal einen Überlauf gab), wonach nach obiger Formel die Entfernung zum Hindernis berechnet werden kann. Dabei berechnet sich z durch die Formel $z = d * T_{\text{Zähler}} = d / f_{\text{Zähler}} = d / 2^{19}$.

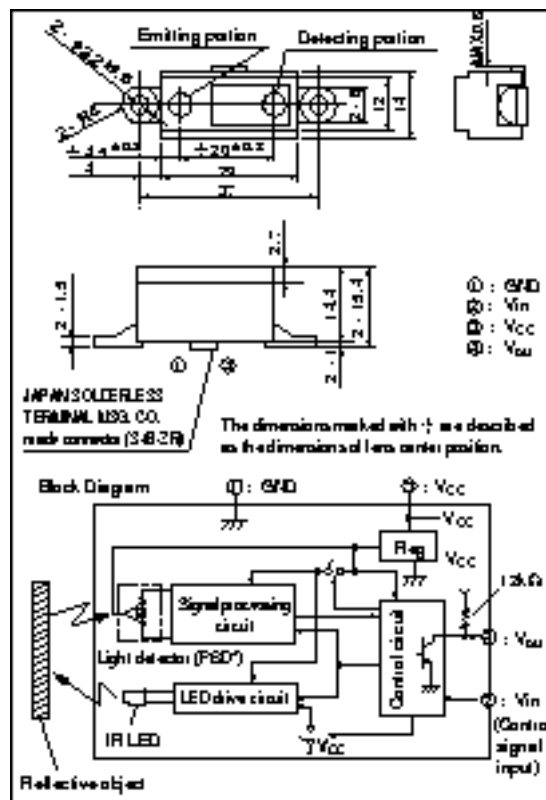


Abbildung 3.4: PSD GP2D02 2Z von Sharp

3.7 Verwendung von PSDs

PSD ist eine Abkürzung für Position Sensitiv Device. Ein PSD ist ein Baustein, welcher mit Hilfe von Infrarotlicht die Entfernung mißt. Bei der in der Studienarbeit verwendeten Gehmaschine Beast wurde die Entfernung statt mit Ultraschallsensoren mit PSDs von der Firma Sharp, mit der Kennnummer GP2D02 2Z gemessen. Diese PSDs besitzen vier Anschlußpole: Masse, V_{in} , Hi, V_{out} (siehe auch Abbildung 3.4). Diese vier Pole finden bei der Gehmaschine sowohl für das linke PSD als auch für das rechte ihren Anschluß am Connector 6, siehe Abbildung 3.5. Dabei ist der Eingang V_{in} des linken PSDs an den digitalen Ausgang Port 6 über einen Spannungsteiler s.u., der Ausgang V_{out} des linken PSD an den Digitalen Eingang Port 0 des Connectors 6 angeschlossen. Der Eingang V_{in} des rechten PSDs ist an den digitalen Ausgang Port 7 und der Ausgang V_{out} des rechten PSDs ist an den digitalen Eingang Port 1 des Connectors 6 angeschlossen. Die Masseanschlüsse beider PSDs sind an Digital Ground Pol 10 und die Hi anschlüsse der PSDs sind an +5V Pol 11 des Connectors 6 angeschlossen.

Pin	Description	Pin	Description
1	+5V	2	Digital Output Port 0
3	Digital Output Port 1	4	Digital Output Port 2
5	Digital Output Port 3	6	Digital Output Port 4
7	Digital Output Port 5	8	Digital Output Port 6
9	Digital Output Port 7	10	Digital Ground
11	+5V	12	Digital Input Port 0
13	Digital Input Port 1	14	Digital Input Port 2
15	Digital Input Port 3	16	Digital Input Port 4
17	Digital Input Port 5	18	Digital Input Port 6
19	Digital Input Port 7	20	Digital Ground
21	Analog Input 6	22	Analog Ground
23	Analog Input 7	24	Analog Ground
25	Analog Input 8	26	Analog Ground
27	Analog Input 9	28	Analog Ground
29	Analog Input 10	30	Analog Ground
31	Analog Input 11	32	Analog Ground
33	Analog Input 12	34	Analog Ground
35	Analog Input 13	36	Analog Ground
37	+5V	38	Analog Input 14
39	Analog Input 15	40	Analog Ground

Abbildung 3.5: Belegung des Connector 6 auf der Eyebotplatine

Bevor das Signal eines Digitalen Ausgangs der Eyebotplatine an das PSD herangeführt wird muß die Spannung auf die Hälfte geteilt werden, was sich durch eine einfache Spannungsteilerschaltung mit Zwei Widerständen realisieren läßt (siehe [Ferner 96]).

Die Ansteuerung der PSDs erfolgt über einen Treiber. Da die PSDs in der HDT eingetragen sind, ist es nur erforderlich das PSD mit der entsprechenden Semantik zu initialisieren, wie z.B.: PSDInit(PSD_FRONTLEFT). Als Rückgabewert erhält man einen Handler, mit dem man am entsprechenden PSD die Entfernung abfragen kann. Hierbei sind Entfernungen zwischen 8 und 80 cm möglich.

Mit PSDStart(Handler, FALSE) veranlaßt man das einmalige Auslesen einer Entfernung aus einem PSDs durch die Eyebotplatine (Wird der zweite Parameter TRUE, so wird eine Endlosschleife zum Entfernungsmessen begonnen), mit PSDCheck() kann überprüft werden, ob inzwischen schon ein brauchbarer Wert vorliegt (falls einer vorliegt ist der Rückgabewert TRUE), und über PSDGet(Handler) erhält man den Integerwert, der die Entfernung angibt.

3.8 Demoprogramm: Walker

Das Demoprogramm muß über den seriellen Port mit einem Programm, wie z.B. Kermit, geladen werden. Nachdem es gestartet wurde, kann man sich durch Menüs wählen, bis man zu den Menüs vordringt, in welchen die entsprechenden Funktionen der Gehmaschine demonstriert bzw. getestet werden können.

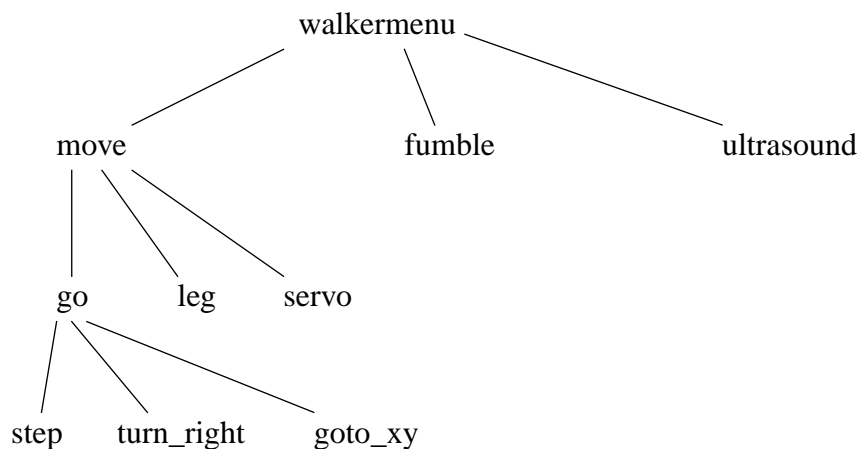


Abbildung 3.6: Demoprogramm

-move: Dieser Menüpunkt führt selbst wieder zu einem Menü, in welchem die Menüpunkte go, leg, servo gewählt werden können.

-Der Menüpunkt go führt in ein Untermenü, in welchem die eigentlichen Laufmöglichkeiten der Gehmaschine vorgeführt werden können. Die erste Taste läßt die Gehmaschine acht käferartige Schritte machen. Die zweite Taste läßt die Gehmaschine acht Rechtsdrehschritte ausführen und bei der dritten Taste erhält man ein weiteres Menü, in dem man die Gehmaschine auf folgende Art und Weise zu einer spezifizierten Stelle gehen läßt: Man kann mit den ersten beiden Tasten die Werte in einem gegebenen Menü verändern, mit der dritten Taste schaltet man zum nächsten Menüpunkt weiter.(siehe auch Schichtenmodell - Schicht 4).

-Nach Anwählen des Menüpunktes leg kann man mit den ersten beiden Tasten die Werte in einem gegebenen Menü verändern, mit der dritten Taste schaltet (genau wie bei dem Menü

für die vierte Schicht) man zum nächsten Menüpunkt weiter. Die Menüpunkte sind: Seite der Gehmaschine bestimmen, Beinpaar bestimmen, im dritten und vierten Menüpunkt kann man die vertikale und die horizontale Position des spezifizierten Beines ändern.

-Der Menüpunkt Servo führt in ein ähnliches Menü wie das Legmenü, mit dem Unterschied, daß hier die Nummer des Servos und der Winkel des entsprechenden Servos eingestellt werden kann.

- fumblemenu: In diesem Menü werden die Tastfähigkeiten der Gehmaschine getestet. Mit der ersten Taste der Eyebotplatine erhöht man die Nummer des aktiven Tastsensors, mit der zweiten Taste erniedrigt man diesen Wert, die dritte Taste dient nur dazu den Wert zu erneuern, d.h. der Tastsensor wird abgefragt, was bei der ersten und der zweiten Taste auch automatisch gemacht wird.

- ultrasound: In diesem Menü werden die Entfernungsmesser abgefragt, wobei die ersten beiden Tasten wie im fumblemenü zum wechseln der Entfernungssensors und zum Entfernungsmessen dienen, die dritte dient nur dazu eine erneute Entfernungsmessung durchzuführen (dies kann je nach verfügbarer Hardware mit Ultraschall oder Infrarotwellen geschehen).

Die vierte Taste dient immer dazu zum übergeordneten Menü zurückzukehren. (siehe auch Abbildung 3.6)

4 Die Hardware Description Table

Die Hardware Description Table (kurz HDT) ist eine Firmware des Eyebot-Projekts, die den Zweck hat die Hardware zu beschreiben, wie der Name es schon sagt, und somit zu verhindern, daß unkontrolliert auf eine Hardware zugegriffen wird, die nicht existiert. Das Format der HDT wurde vorwiegend von Klaus Schmitt einem Studenten aus Kaiserslautern und Thomas Lampart entwickelt, siehe auch [Lampart 97]. Die Hardwaredeskriptiontable wird extra, d.h. nicht zusammen mit dem Rest des BIOS geflasht. Die Adresse, an die es geladen wird, war beim letzten Stand während dieser Studienarbeit $1c000_{16}$, der Flashvorgang ist genau gleich wie der des BIOS, bis auf diese Adresse und die zu überschreibenden Blöcke. Beim Flashen der HDT wird statt $11111110, 00000001$ eingegeben, also der inverse Wert zum Rest des BIOS.

4.1 Typdefinitionen

Um die Hardwarebestückung festzulegen, muß erst ein Format, d.h. die Datentypen festgelegt werden, mit dem die Hardware beschrieben werden kann, was in der Datei `mc/hdtdata/hdtwalk.c` erscheint. Hierbei wird auf die Hardware Bezug genommen, d.h. die HDT sollte mit der tatsächliche vorhandenen Hardware übereinstimmen. Bei Servos wird z.B. die Version festgehalten, so daß ältere Treiber, als die nötige Version für die Hardware nicht zugelassen wird. Hier wird außerdem der TPU-Kanal, der Timer etc. festgelegt. In dieser Datei wird zusätzlich der Typ der HDT-Einträge festgelegt. Eine Routine soll dann in dieser Tabelle (in der HDT) vor dem freigeben eines Handlers beim Öffnen eines Gerätes nachschauen, ob es überhaupt existiert, d.h. ob es in der HDT eingetragen ist. Die entsprechenden genauen Erklärungen finden sich bei der Beschreibung zur Schicht eins in Anhang A.

4.2 Hardwarebeschreibung

Die Wertdefinitionen in der Datei `mc/hdtdata/hdtwalk.c` (und alle weiteren Dateien in diesem Verzeichnis) stellen die eigentliche HDT dar, welche unabhängig vom BIOS ins EEPROM geflasht wird, wie das geht, kann man in der Datei `mc/HOWTODO` nachlesen (siehe auch Einleitung dieses Kapitels). Hier wird die Hardware, die an die entsprechende Eyebot-Platine angeschlossen ist (möglichst in Übereinstimmung mit der tatsächlich angebauten Hardware) beschrieben. Dies geschieht über die Hardwaredescriptiontable, die für jedes „Device“, welches an die Eyebotplatine angeschlossen ist, einen Eintrag enthält, in welchem die Devicebezeichnungen, die Semantik (genauere Beschreibung z.B. SER_LFUD für Servo-left-front-updown) und ein Zeiger auf die Struktur des Gerätes steht. Die entsprechenden genauen Erklärungen findet man, wie die der Typdefinitionen, in der Beschreibung zur Schicht eins in Anhang A.

4.3 Schnittstellenroutinen

Um auf die in der HDT enthaltene Hardware zugreifen zu können, existieren einige Schnittstel-

lenroutinen, welche die Treiberrountinen der vorhandenen Hardware ansprechen. Diese liegen in der Datei mc/robios/hdt.c, die Erklärungen dazu befinden sich wie die der restlichen HDT-Anteile in Anhang A.

Es gibt jeweils Routinen zum öffnen von Devices. Diese Routinen schauen erst einmal in der HDT nach, ob dort Einträge für die angeforderten Devices mit der entsprechenden Device-Semantik existieren (durch Aufruf der Funktion HDT_FindEntry), als positives Ergebnis erhält man einen Zeiger auf die dazugehörige Struktur. Ist ein solcher Zeiger zurückgegeben worden, so wird dieser im Normalfall der Hardwarenahen Initialisierungsroutine des Devices übergeben, welche einen Handler liefert. Dieser Handler ist später als Parameter notwendig, wenn man dieses Gerät ansprechen will (z.B. wenn man den Winkel eines Servos ändern will), bzw. wenn man die Hardware wieder freigeben will. (siehe auch Abbildung 4.1)

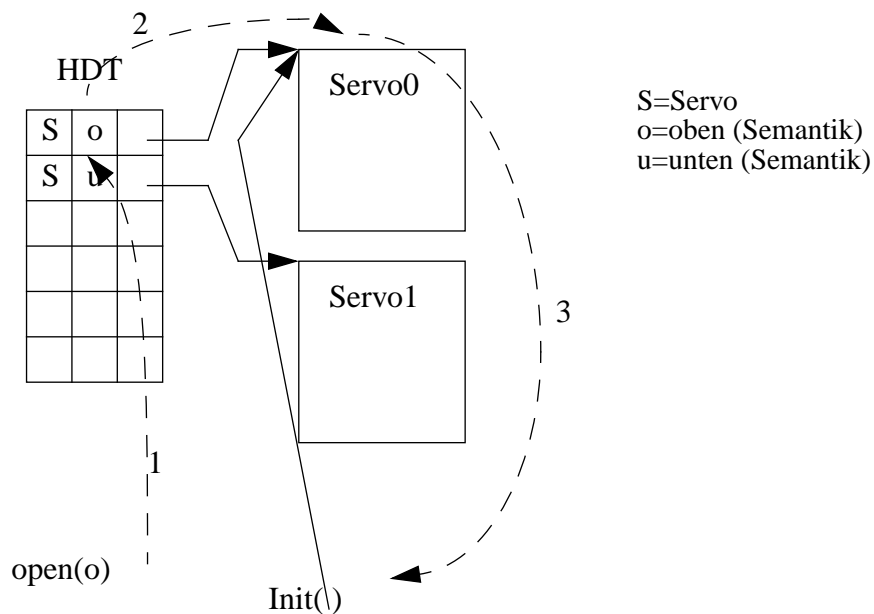


Abbildung 4.1: Öffnen eines Devices mit Hilfe der HDT

5 Ausblick

Auf Beast wurde mittlerweile eine CCD-Kamera aufgebaut. Passend dazu existieren von Thomas Lampart bereits Routinen, die aus einem Bild dieser Kamera Kanten extrahieren. Diese Kanten kann man vektorisieren (siehe [Dudler et al. 95]) und damit eventuelle Schilder, auf denen einfache Zeichen stehen, interpretieren. Solche Schilder könnte man dazu verwenden die Gehmaschine, die ganz autonom unterwegs wäre, durch ein Gelände zu lotsen. Derartige Gehmaschinen eignen sich besser um sich im Gelände vorwärtszubewegen, als Fahrzeuge mit Rädern.

Anhang A:

Schnittstellendefinitionen - Prozedurerklärungen, der Assemblerfiles:

Die Routinen der untersten Ebene, d.h. der Ebene in der direkt Servos bzw. die TPU angesteuert wird, sind in 68000 Assembler geschrieben und wurden mit dem GNU-Crossassembler auf einer Sun-Workstation bzw. auf einem 486 PC übersetzt. Ergänzt wurden diese durch C-Funktionen, welche durch einen GNU-Crosscompiler ebenfalls auf einer SUN-Workstation in 68000 Maschinencode übersetzt werden.

SERVO_PWM_Init

Parameterübergabe:

Ein Zeiger auf ein Feld der HDT auf dem Stack, in dem folgende Parameter aufgelistet, jeweils als Langwort, stehen:

- Abwärtskompatible Versionsnummer
- TPU-Kanalnummer des Kanals, der initialisiert werden soll
- Zähler auf den der Kanal zugreift: Der erste ist mit 4 MHz getaktet, der 2. mit 1/2 MHz
- PWM_PERIODidoden-Zeit

War die Initialisierung erfolgreich, d.h. der TPU-Kanal existiert, er war nicht belegt und auch sonst ist nichts schief gegangen, dann wird in d0, als Ergebnis eine 0 zurückgegeben, ansonsten erhält man eine 1.

Funktion:

Diese Routine initialisiert einen TPU Kanal, der frei ist mit einer Pulsbreitenmodulation, die z.B. für den Betrieb eines Servos benötigt wird.

Module:

mc/robios/sm_driv.s

MOTOR_PWM_Init

Parameterübergabe:

Ein Zeiger auf ein Feld der HDT auf dem Stack, in dem folgende Parameter aufgelistet, jeweils als Langwort, stehen:

- Abwärtskompatible Versionsnummer
- TPU-Kanalnummer des Kanals, der initialisiert werden soll
- Zähler auf den der Kanal zugreift: Der erste ist mit 4 MHz getaktet, der 2. mit 1/2 MHz
- PWM_PERIODidoden-Zeit

War die Initialisierung erfolgreich, d.h. der TPU-Kanal existiert, er war nicht belegt und auch sonst ist nichts schief gegangen, dann wird in d0, als Ergebnis eine 0 zurückgegeben, ansonsten erhält man eine 1.

Funktion:

Diese Routine Initialisiert einen TPU Kanal, der frei ist mit einer Pulsbreitenmodulation, die z.B. für den Betrieb eines Servo-Motors benötigt wird.

Module:

mc/robios/sm_driv.s

TPU_PWM_Release

Parameterübergabe:

Ein Handler des belegten TPU Kanals, der mit SERVO_/MOTOR_PWM_Init initialisiert wurde auf dem Stack. Falls mehrere Kanäle gleichzeitig freigegeben werden sollen, können die

Handler dieser mit „oder“ verknüpft werden und so in einem einzigen Aufruf der Routine übergeben werden.

War der Kanal mit PWM initialisiert und aktiv, so wird nach erfolgreicher Kanalfreigabe eine 0 in d0 als Ergebnisparameter zurückgegeben, an sonst ist das Ergebnis ein Handler, welcher aus einer „oder“-Verknüpfung der Handler der Kanäle hervorgeht, die freigegeben werden können, was für einen Fehler steht. Überschneiden sich keine PWM-belegten Kanäle mit freigegebenen Kanälen, so wird der Wert \$1000 zurückgegeben.

Funktion:

Diese Routine deaktiviert einen TPU Kanal, der mit einer Pulsbreitenmodulation aktiviert ist, so daß er für weitere Verwendung zur Verfügung steht.

Module:

mc/robios/sm_driv.s

SERVO_Angle

Parameterübergabe:

In d1 wird ein TPU-Kanal übergeben, der vorher mit SERVO_PWM_Init ordnungsgemäß aktiviert worden sein muß. In d2 steht der Servowinkel, der einen Wert zwischen einschließlich 0 und 210 Grad besitzen muß.

War der Kanal mit SERVO_PWM_Init initialisiert und hat alles geklappt so wird als Ergebnisparameter in d0 eine 0 übergeben, ansonsten erhält man eine 1, und die Funktion wurde nicht ausgeführt.

Funktion:

Diese Routine stellt an einem TPU-Kanal den gewünschten PWM_Hi-Wert ein, der nötig ist, um ein Servo auf den gewünschten Winkel zu trimmen.

Module:

mc/robios/sm_driv.s

MOTOR_Speed

Parameterübergabe:

In d1 wird ein TPU-Kanal übergeben, der vorher mit MOTOR_PWM_Init ordnungsgemäß aktiviert worden sein muß. In d2 steht die Servomotor-Geschwindigkeit, die einen Wert zwischen einschließlich -100 % und 100 % besitzen muß.

War der Kanal mit MOTOR_PWM_Init initialisiert und hat alles geklappt, so wird als Ergebnisparameter in d0 eine 0 übergeben, ansonsten erhält man eine 1, und die Funktion wurde nicht ausgeführt.

Funktion:

Diese Routine stellt an einem TPU-Kanal den gewünschten PWM_Hi-Wert ein, der nötig ist, um einen Motor auf die gewünschte Geschwindigkeit zu bringen.

Module:

mc/robios/sm_driv.s

FUMBLE_Getsignal

Parameterübergabe:

Ein Zeiger auf ein Feld der HDT auf dem Stack, in dem folgende Parameter aufgelistet, jeweils als Langwort, stehen:

-Abwärtskompatible Versionsnummer

-Adresse des digitalen Eingabe

-Bit der Digitalen Eingabe, welches abgefragt werden soll

War die Versionsnummer neu genug, und am digitalen Eingang liegt eine eins an (d.h. ein Tastsensor wurde gedrückt - ein Bein ist auf dem Boden), so ist der Ergebnisparameter 1. War die

Versionsnummer neu genug, und am digitalen Eingang liegt eine null an (d.h. der Tastsensor wurde nicht gedrückt - das Bein ist vom Boden erhoben), so ist der Ergebnisparameter 0. Bei veralteter Versionsnummer der Routine erhält man eine -1 zurück.

Funktion:

Diese Routine stellt an einem digitalen Eingang fest, ob das abgefragte Bein den Boden berührt, oder ob es erhoben ist.

Module:

mc/robios/fumble.s

servo_type**Funktion:**

Dies Typdefinition erzeugt eine Struktur, die eine Servo beschreiben soll. Sieh auch Kapitel über HDT. Dabei werden die folgenden Parameter festgehalten:

- driver_version: Der Treiber, der ein Servo ansteuert, überprüft vor der Servoverwendung, ob der in driver_version festgelegte Treiber eine ältere als die verwendete Treiberversion oder gleichalte Treiberversion hat, falls dies der Fall ist funktioniert die Ansteuerung, ansonsten wird ein Fehler zurückgegeben, und die Servoansteuerung wird nicht ausgeführt.
- tpu_channel: Dieser Wert legt fest, durch welchen TPU-Kanal das entsprechende Servo angesteuert wird.
- tpu_timer: Dieser Wert legt fest, welcher der beiden in der TPU vorhandenen Timer verwendet wird. Der erste Zähler ist mit 2^{22} Hz \sim 4 MHz, der zweite mit 2^{19} Hz \sim 1/2 MHz getaktet. Dies wird im Module „mc/robios/init.s“ in dem Code nach der Marke „TPU_Init“ festgelegt, siehe auch TPU.
- pwm_period: Dieser Wert legt die Wellenlänge des PWM-Signals, mit dem das Servo angesteuert wird fest, siehe auch TPU.

Module:

mc/robios/include/hdt.h

motor_type**Funktion:**

Diese Typdefinition erzeugt eine Struktur, die einen Servomotor beschreiben soll. Siehe auch Kapitel über HDT. Dabei werden die folgenden Parameter festgehalten:

- driver_version: Der Treiber, der einen Servomotor ansteuert überprüft vor der Motorverwendung, ob der in driver_version festgelegte Treiber eine ältere als die verwendete Treiberversion oder gleichalte Treiberversion hat, falls dies der Fall ist funktioniert die Ansteuerung, ansonsten wird ein Fehler zurückgegeben, und die Motoransteuerung wird nicht ausgeführt.
- tpu_channel: Dieser Wert legt fest, durch welchen TPU-Kanal der entsprechende Servomotor angesteuert wird.
- tpu_timer: Dieser Wert legt fest, welcher der beiden in der TPU vorhandenen Timer verwendet wird. Der erste Zähler ist mit 2^{22} Hz \sim 4 MHz, der zweite mit 2^{19} Hz \sim 1/2 MHz getaktet. Dies wird im Module „mc/robios/init.s“ in dem Code nach der Marke „TPU_Init“ festgelegt, siehe auch TPU.
- pwm_period: Dieser Wert legt die Wellenlänge des PWM-Signals, mit dem der Servomotor angesteuert wird fest, siehe auch TPU.
- out_pin_address: Dieser Wert legt eine Adresse fest, an der bei positiven Geschwindigkeiten des Motors ein Bit gesetzt (siehe out_pin_fbit - gleich folgend) und ein anderes zurückgesetzt werden muß (siehe out_pin_bbit - gleich folgend). Desgleichen gilt, daß die Bits bei negativen Geschwindigkeiten genau anders gesetzt werden müssen, als bei positiven.
- out_pin_fbit: Dieser Wert legt das Bit fest, welches durch Byteweisen Zugriff auf die Adresse, die durch den Wert „out_pin_address“ festgelegt wurde, gesetzt werden muß, wenn der Motor

sich vorwärts drehen soll. Ansonsten muß es zurückgesetzt werden.

- `out_pin_bbit`: Dieser Wert legt das Bit fest, welches durch Byteweisen Zugriff auf die Adresse, die durch den Wert „`out_pin_address`“ festgelegt wurde, gesetzt werden muß, wenn der Motor sich rückwärts drehen soll. Ansonsten muß es zurückgesetzt werden.

Module:

`mc/robios/include/hdt.h`

fumble_type

Funktion:

Diese Typdefinition erzeugt eine Struktur, die einen Tastsensor beschreiben soll. Siehe auch Kapitel über HDT. Dabei werden die folgenden Parameter festgehalten:

- `driver_version`: Der Treiber, der einen Tastsensor ansteuert überprüft vor der Sensorabfrage, ob der in `driver_version` festgelegte Treiber eine ältere als die verwendete Treiberversion oder gleichalte Treiberversion hat, falls dies der Fall ist funktioniert die Ansteuerung, ansonsten wird ein Fehler zurückgegeben, und die Sensoransteuerung wird nicht ausgeführt.

- `in_pin_address`: Dieser Wert gibt die Hardwareadresse an, an welcher der zugehörige digitale Eingang abgefragt werden kann.

- `in_pin_bit`: Dieser Wert gibt das Bit an, welches an der Adresse, auf die `in_pin_address` zeigt, abgefragt werden muß.

Module:

`mc/robios/include/hdt.h`

servo_type servo0 = {0, 0, TIMER2, 10486}; (etc.)

Funktion:

Dies ist der eigentliche HDT-Teil (siehe auch HDT-Hardwarebeschreibung), welcher festlegt was für Werte am `servo0` anliegen (entsprechend existiert auch ein `servo1`, etc.), die allgemeine Beschreibung der Werte kann ebenfalls im Anhang A unter `servo_type` nachgelesen werden:

- Dieses Servo läuft mit allen Treiberversionen größer als 0, also mit allen Treiberversionen, die es schon gibt oder noch geben wird.

- Dieses Servo belegt den TPU Kanal 0 (weiter unten wird noch beschrieben, wie man einem solchen Kanal noch eine Semantik, wie z.B. Bein links, vorne, vertikale Bewegungsrichtung zuordnet).

- Dieses Servo benützt den `TIMER2`, welcher mit ungefähr einem halben Megahertz getaktet ist.

- Und dieses Servo wird durch eine Pulsbreitenmodulation (s.o.) angesteuert, welche einer Wellenlänge von 10486 `CLOCK`-Signalen entspricht, d.h. $(1/2^{19}) * 10486$ Sekunden = 20 ms (siehe auch Servoansteuerung)

Module:

`mc/hdtdata/hdtwalk.c`

motor_type motor0 = {0, 0, TIMER2, 10486, (void*)(OutBase+2), 6, 7}; (etc.)

Funktion:

Dies ist der eigentliche HDT-Teil (siehe auch HDT-Hardwarebeschreibung), welcher festlegt was für Werte am `motor0` anliegen (entsprechend existiert auch ein `motor1`), die allgemeine Beschreibung der Werte kann, ebenfalls im Anhang A unter `motor_type` nachgelesen werden:

- Dieser Servomotor läuft mit allen Treiberversionen größer als 0, also mit allen Treiberversionen, die es schon gibt oder noch geben wird.

- Dieser Servomotor belegt den TPU Kanal 0 (weiter unten wird noch beschrieben, wie man einem solchen Kanal noch eine Semantik zuordnet).

- Dieser Servomotor benützt den `TIMER2`, welcher mit ungefähr einem halben Megahertz ge-

taktet ist.

- Und dieser Servomotor wird durch eine Pulsbreitenmodulation (s.o.) angesteuert, welche einer Wellenlänge von 10486 CLOCK-Signalen entspricht, d.h. $(1/2^{19}) * 10486$ Sekunden = 20 ms (siehe Servoansteuerung)

Module:

mc/hdtdata/hdtvehi.c

fumble_type fumble2 = {0, (BYTE*)InBase, 4};

Funktion:

Dies ist der eigentliche HDT-Teil (siehe auch HDT-Hardwarebeschreibung), welcher festlegt was für Werte für den 3. Tastsensor wichtig sind. (entsprechend existieren auch ein fumble0 bis fumble5), die allgemeine Beschreibung der Werte kann, ebenfalls im Anhang A unter fumble_type nachgelesen werden:

- Dieser Tastsensor läuft mit allen Treiberversionen größer als 0, also mit allen Treiberversionen, die es schon gibt oder noch geben wird.
- Dieser Tastsensor ist an den digitalen Eingang der Eyebotplatine angeschlossen.
- Dieser Tastsensor ist an den 4. digitalen Eingang angeschlossen.

Module:

mc/hdtdata/hdtvehi.c

#define KorrFrLeUD 162

#define KorrMiLeUD 152

#define KorrReLeUD 155

#define KorrFrRiUD 50

#define KorrMiRiUD 11

#define KorrReRiUD 8

#define KorrFrLeFB 28

#define KorrMiLeFB 38

#define KorrReLeFB 46

#define KorrFrRiFB 148

#define KorrMiRiFB 144

#define KorrReRiFB 159

Funktion:

Dies Konstantendefinitionen legen einen Wert für jedes Servo der Gehmaschine fest, mit dessen Hilfe es „kalibriert“ werden kann. Zu Beginn der Walkerdemos, bevor ein Menüpunkt ausgewählt wurde, wird die Funktion Stillstand als eine Art Initialisierung der Gehmaschine aufgerufen, so daß die Beine alle in einen „Urzustand“ gebracht werden. Sind die Beine dann nicht alle seitlich wegstehend, in der gleichen Höhe angeordnet, so können diese Werte entsprechend verändert werden, um die Beine richtig einzustellen.

KorrFrLeUD: Korrekturfaktor für das linke, vordere Bein - vertikale Richtung

KorrMiLeUD: Korrekturfaktor für das linke, mittlere Bein - vertikale Richtung

KorrReLeUD: Korrekturfaktor für das linke, hintere Bein - vertikale Richtung

KorrFrRiUD: Korrekturfaktor für das rechte vordere Bein - vertikale Richtung

KorrMiRiUD: Korrekturfaktor für das rechte, mittlere Bein - vertikale Richtung

KorrReRiUD: Korrekturfaktor für das rechte, hintere Bein - vertikale Richtung

KorrFrLeFB: Korrekturfaktor für das linke, vordere Bein - horizontale Richtung

KorrMiLeFB: Korrekturfaktor für das linke, mittlere Bein - horizontale Richtung

KorrReLeFB: Korrekturfaktor für das linke, hintere Bein - horizontale Richtung

KorrFrRiFB: Korrekturfaktor für das rechte vordere Bein - horizontale Richtung

KorrMiRiFB: Korrekturfaktor für das rechte, mittlere Bein - horizontale Richtung

KorrReRiFB: Korrekturfaktor für das rechte, hintere Bein - horizontale Richtung

Module:

mc/demos/walker/walker.hex

ServoHandle **SERVO_Init** (DeviceSemantics semantics)

Parameterübergabe:

In semantics wird die angeforderte DeviceSemantics übergeben, also z.B. SER_LFUD für das Servo links, vorne, für die vertikale Bewegung. Die einzelnen Semantiken sind im Modul mc/robios/include/hdt.h definiert.

Rückgabeparameter ist ein Handler für das initialisierte Device. Dieser Handler ist 0, falls das Device nicht existiert bzw. schon von einem anderen Programm belegt wurde.

Funktion:

Diese Routine versucht in der Hardwaredescriptiontable einen Eintrag mit der TypeID Servo und der übergebenen DeviceSemantik zu ermitteln (mit der Funktion HDT_FindEntry). Ist so ein Eintrag gefunden, so wird das Gerät falls noch nicht belegt, durch die Routine SERVO_PWM_Init initialisiert und gleichzeitig belegt.

Module:

mc/robios/hdt.c

int **SERVO_Release** (ServoHandle *handle)

Parameterübergabe:

Dieser Funktion wird ein Zeiger auf einen Handler eines Servos übergeben (siehe auch SERVO_Init), welches nicht mehr gebraucht wird.

Als Ergebnisparameter wird 0 zurückgegeben, falls alles geklappt hat. Falls etwas schief ging, d.h. der Handler war ungültig, so wird -1 zurückgegeben.

Funktion:

Diese Funktion gibt ein, mit SERVO_Init belegtes Device, für erneuten Gebrauch wieder frei (die hardwarenahe Routine, mit deren Hilfe dies geschieht ist TPU_PWM_Release).

Module:

mc/robios/hdt.c

int **SERVO_Set** (ServoHandle handle, int angle)

Parameterübergabe:

Dieser Funktion wird ein Handler eines Servos übergeben (siehe auch SERVO_Init).

Der zweite Parameter ist ein Winkel, den das Servo einnehmen kann (möglich sind Werte zwischen 0 und 210 Grad).

Als Ergebnisparameter wird 0 zurückgegeben, falls alles geklappt hat. Falls etwas schief ging, wird -1 zurückgegeben.

Funktion:

Diese Funktion stellt an einem durch den Handler spezifizierten Servo den angegebenen Winkel ein.

Module:

mc/robios/hdt.c

MotorHandle **MOTOR_Init** (DeviceSemantics semantics)

Parameterübergabe:

In semantics wird die angeforderte DeviceSemantics übergeben, also z.B. MOTOR_RIGHT für den Motor auf der rechten Seite eines Fahrzeugs. Die einzelnen Semantiken sind im Modul mc/robios/include/hdt.h definiert.

Rückgabeparameter ist ein Handler für das initialisierte Device. Dieser Handler ist 0, falls das

Device nicht existiert bzw. schon von einem anderen Programm belegt wurde.

Funktion:

Diese Routine versucht in der Hardwaredescriptiontable einen Eintrag mit der TypeID Motor und der übergebenen DeviceSemantik zu ermitteln (mit der Funktion HDT_FindEntry). Ist so ein Eintrag gefunden, so wird das Gerät, falls noch nicht belegt, durch die Routine MOTOR_PWM_Init initialisiert und gleichzeitig belegt.

Module:

mc/robios/hdt.c

int **MOTOR_Release** (MotorHandle *handle)

Parameterübergabe:

Dieser Funktion wird ein Zeiger auf einen Handler eines Motors übergeben (siehe auch MOTOR_Init), welcher nicht mehr gebraucht wird.

Als Ergebnisparameter wird 0 zurückgegeben, falls alles geklappt hat. Falls etwas schief ging, d.h. der Handler war ungültig, so wird -1 zurückgegeben.

Funktion:

Diese Funktion gibt ein, mit MOTOR_Init belegtes, Device für erneuten Gebrauch wieder frei (die hardwarenahe Routine, mit deren Hilfe dies geschieht ist TPU_PWM_Release).

Module:

mc/robios/hdt.c

int **MOTOR_Drive** (MotorHandle handle, int speed)

Parameterübergabe:

Dieser Funktion wird ein Handler eines Motors übergeben (siehe auch MOTOR_Init).

Der zweite Parameter ist eine Geschwindigkeitsangabe, die von -100 bis 100 Werte annehmen kann.

Als Ergebnisparameter wird 0 zurückgegeben, falls alles geklappt hat. Falls etwas schief ging wird -1 zurückgegeben.

Funktion:

Diese Funktion stellt an einem durch den Handler spezifizierten Motor die angegebenen Geschwindigkeit ein.

Module:

mc/robios/hdt.c

int **OS_GetSDSignal** (DeviceSemantics semantics)

Parameterübergabe:

In semantics wird die angeforderte DeviceSemantics übergeben, also z.B. FUM_LF für den Tastsensor links, vorne. Die einzelnen Semantiken sind im Modul mc/robios/include/hdt.h definiert.

Rückgabeparameter ist 1, falls das Bein den Boden berührt, 0, falls es aufgehoben wurde und -1 falls etwas schief ging, d.h. die vorhanden Routine hat eine zu alte Versionsnummer oder der Tastsensor existiert nicht.

Funktion:

Diese Routine versucht in der Hardwaredescriptiontable einen Eintrag mit der TypeID Fumble und der übergebenen DeviceSemantik zu ermitteln (mit der Funktion HDT_FindEntry). Ist so ein Eintrag gefunden, so wird das Gerät, durch die Routine FUMBLE_Getsignal abgefragt.

Module:

mc/robios/hdt.c

HDT_entry_type;

Funktion:

Diese Typdefinition erzeugt eine Struktur, welche einen Eintrag in der Hardwarebeschreibungstabelle bildet. Die einzelnen Werte, welche diese Struktur enthält sind:

- typeid: Typid ist ein Wert, welcher das Device festlegt um das es sich handelt, also Servo oder Motor etc.

- semantics: Die Semantik beschreibt ein Gerät genauer, also die Position eines Servos und die Richtung in die es sich bewegen kann oder die Aufgabe eines Motors. Dieser Wert ist wichtig zum identifizieren des Eintrages durch die Routine HDF_FindEntry.

- *data_area: Data_area ist die Struktur, welche das Gerät genauer beschreibt. *data_area ist ein Zeiger auf diese Struktur, welche zum betreiben des Gerätes wichtig ist und somit auch als Ergebnisparameter von HDT_FindEntry zurückgegeben wird.

Module:

mc/robios/include/hdt.h

HDT_entry_type **HDT** []

Funktion:

Diese Variablendeklaration ist die eigentliche *Hardwaredescriptiontable*, welche aus einem Feld aus lauter „HDT_entries“ besteht, d.h. aus lauter Einträgen vom Typ HDT_entry_typ. Hier wird die Hardware festgelegt, welche an die Eyebot-Platine angeschlossen ist und welche vom System ansprechbar ist.

Module:

mc/robios/hdtdata/hdtwalk.c

void ***HDT_FindEntry**(TypeID typeid, DeviceSemantics semantics)

Parameterübergabe:

typeid legt das zu suchende Device fest.

semantics spezifiziert dies genauer, z.B. MOTOR_RIGHT für den rechten Motor, falls ein solcher an die Eyebotplatine angeschlossen ist.

Ergebnisparameter dieser Funktion ist ein Zeiger auf die Struktur, welche das Device genauer beschreibt, falls es vorhanden bzw. in der HDT eingetragen ist. Findet die Funktion keinen Eintrag mit der gesuchten Semantik in der Hardwaredescriptiontable, so wird als Ergebnisparameter NULL zurückgeliefert.

Funktion:

Diese Funktion sucht nach einem durch die Parameter spezifizierten Eintrag in der Hardwaredescriptiontable, d.h. sie sucht nach einer durch die Parameter spezifizierten Hardware, die an die Eyebotplatine angeschlossen ist und gibt, falls das entsprechende Device existiert, einen Zeiger auf die Struktur zurück, welche das Device genauer beschreibt.

Module:

mc/robios/hdt.c

int **ultrasonicLeft**(void)

Parameterübergabe:

ultrasonicLeft hat keine Eingabeparameter, als Rückgabeparameter besitzt es nur einen Integerwert, der die Distanz zum nächsten Hindernis auf der linken Seite im vorderen Gesichtsfeld der Gehmaschine in mm angibt.

Funktion:

Diese Routine liefert auf der Basis von Assembler Routinen einen Wert, der als Distanz eines Ultraschall- oder Infrarotsensors, der vorne links an der Gehmaschine montiert ist, zu einem Objekt interpretiert werden kann.

Module:

mc/demos/walker/walker.c

int **ultrasonicRight**(void)

Parameterübergabe:

ultrasonicRight hat keine Eingabeparameter, als Rückgabeparameter besitzt es nur einen Integerwert, der die Distanz zum nächsten Hindernis auf der linken Seite im vorderen Gesichtsfeld der Gehmaschine in mm angibt.

Funktion:

Diese Routine liefert auf der Basis von Assemblerroutinen einen Wert, der als Distanz eines Ultraschall- oder Infrarotsensors, der vorne rechts an der Gehmaschine montiert ist, zu einem Objekt interpretiert werden kann.

Module:

mc/demos/walker/walker.c

Anhang B:

Schnittstellendefinitionen - Prozedurerklärungen, der Beinsteuerungsprozeduren

Die Routinen der zweiten Ebene, d.h. der Ebene in der indirekt Servos angesteuert werden bzw. die Beine in eine festgelegte Position gebracht werden sind in C geschrieben und wurden mit dem GNU-Crosscompiler auf einer Sun-Workstation bzw. auf einem 486 PC in 68000 Maschinencode übersetzt.

int **liftleg**(Side side, Position position, int highth)

Parameterübergabe:

Side hat zwei mögliche Ausprägungen im Wertebereich: LEFT=0 entspricht der Linken Seite, RIGHT=1 entspricht der rechten Seite der Gehmaschine.

Position hat drei Ausprägungen im Wertebereich: FRONT=10 bestimmt das vordere Beinpaar, MIDDLE =11 das mittlere und REAR=12 bestimmt das hintere Beinpaar.

Highth kann Werte zwischen 1 und 5 jeweils einschließlich übergeben werden, womit bestimmt wird, wie hoch ein Bein gehoben werden soll (bzw. welche vertikale Winkelstellung es annehmen soll). In diesem Module sind für diese fünf Werte Konstanten definiert, um die Programmierung zu erleichtern:

- VUP steht für Very UP und entspricht dem Wert 5
- UP steht für UP und entspricht dem Wert 4
- CN steht für CeNter und entspricht dem Wert 3
- DN steht für Down und entspricht dem Wert 2
- VDN steht für Very DowN und entspricht dem Wert 1

Funktion:

Die Funktion liftleg hebt ein Bein einer sechsbeinigen Gehmaschine, die mit Servomotoren in der oben angegebenen Anordnung betrieben wird. Spezifiziert wird das Bein durch Angabe der Seite und einer der drei Positionen (Beinpaare). Die letzte Angabe, die erforderlich ist, ist die neue vertikale Winkelstellung des Beines.

Module:

mc/demos/walker/walker.c

int **movleg**(Side side, Position position, int distance)

Parameterübergabe:

Side hat zwei mögliche Ausprägungen im Wertebereich: LEFT=0 entspricht der Linken Seite, RIGHT=1 entspricht der rechten Seite der Gehmaschine.

Position hat drei Ausprägungen im Wertebereich: FRONT=10 bestimmt das vordere Beinpaar, MIDDLE =11 das mittlere und REAR=12 bestimmt das hintere Beinpaar.

distance kann Werte zwischen 1 und 5 jeweils einschließlich übergeben werden, womit bestimmt wird wie weit zurück ein Bein bewegt werden soll (bzw. welche horizontale Winkelstellung es annehmen soll). In diesem Module sind für diese fünf Werte Konstanten definiert, um die Programmierung zu erleichtern:

- VBK steht für Very BacK und entspricht dem Wert 5
- BK steht für BacK und entspricht dem Wert 4
- CN steht für CeNter und entspricht dem Wert 3
- FN steht für FroNt und entspricht dem Wert 2
- VFN steht für Very FroNt und entspricht dem Wert 1

Funktion:

Die Funktion `movleg` positioniert ein Bein einer sechsbeinigen Gehmaschine, die mit Servomotoren in der oben angegebenen Anordnung betrieben wird. Spezifiziert wird das Bein durch Angabe der Seite und einer der drei Positionen (Beinpaare). Die letzte Angabe, die erforderlich ist, ist die neue horizontale Winkelstellung des Beines.

(Anschaulich: Diese Funktion bewegt ein Bein vorwärts oder rückwärts)

Module:

`mc/demos/walker/walker.c`

`int CheckleftUltra(void)`

Parameterübergabe:

`CheckleftUltra` hat keine Eingabeparameter, als Rückgabeparameter besitzt es nur einen Integerwert, der `TRUE` oder `FALSE` sein kann.

Funktion:

Die Funktion `CheckleftUltra` hat die Aufgabe mit dem linken Ultraschall- oder Infrarotsensor zu überprüfen, ob die Distanz zum nächsten Hindernis einen weiteren Schritt zulässt. Dies macht sie basierend auf der Funktion `ultrasonicLeft` aus der ersten Schicht (s.o.). Reicht die Entfernung aus so wird der Wert `TRUE` zurückgegeben, ansonsten erhält man den Wert `FALSE`.

Module:

`mc/demos/walker/walker.c`

`int CheckrightUltra(void)`

Parameterübergabe:

`CheckrightUltra` hat keine Eingabeparameter, als Rückgabeparameter besitzt es nur einen Integerwert, der `TRUE` oder `FALSE` sein kann.

Funktion:

Die Funktion `CheckrightUltra` hat die Aufgabe mit dem rechten Ultraschall- oder Infrarotsensor zu überprüfen, ob die Distanz zum nächsten Hindernis einen weiteren Schritt zulässt. Dies macht sie basierend auf der Funktion `ultrasonicRight` aus der ersten Schicht (s.o.). Reicht die Entfernung aus so wird der Wert `TRUE` zurückgegeben, ansonsten erhält man den Wert `FALSE`.

Module:

`mc/demos/walker/walker.c`

`int CheckUltra(void)`

Parameterübergabe:

`CheckUltra` hat keine Eingabeparameter, als Rückgabeparameter besitzt es nur einen Integerwert, der `TRUE` oder `FALSE` sein kann.

Funktion:

Die Funktion `CheckUltra` hat die Aufgabe an beiden Ultraschallsensoren die Distanz zum nächsten Hindernis abzufragen. Überschreiten beide Sensorenabfragen (links und rechts) eine gewisse Distanz, d.h. liegt in unmittelbarer Nähe kein Hindernis, so wird der Wert `TRUE` zurückgegeben. Diese Funktionsabfrage ist nur dann sinnvoll, wenn eine Vorwärtsbewegung ausgeführt wird, da hinten keine Sensoren angebracht sind, bzw. bei einer Drehbewegung kein Abfragegrund existiert. Sie ist „eine &-Verknüpfung“ der Funktionen `CheckleftUltra` und `CheckrightUltra`.

Module:

`mc/demos/walker/walker.c`

Anhang C:

Schnittstellendefinitionen - Prozedurerklärungen und Datenerklärungen, für die Positions bzw. Zustandsänderung

Die Routinen der dritten Schicht, d.h. der Schicht in der aufgebaut auf den Routinen der zweiten Schicht Gehbewegungen des gesamten Roboters ausgeführt werden, sind in C geschrieben und wurden mit dem GNU-Crosscompiler auf einer Sun-Workstation bzw. auf einem 486 PC in 68000 Maschinencode übersetzt.

Anmerkung: In dieser Schicht wird zwischen den normalen Kriechbewegungen immer derselbe stabile Zustand (s.o.) angenommen, was sinnvoll ist, damit ein fließender Übergang zwischen normalem Vorwärtskriechen und Krümmungsbewegungen erfolgen kann.

Dieser stabile Zustand sieht aus wie folgt: In horizontaler Bewegungsrichtung befinden sich die Beine alle in mittelstellung, in vertikaler Bewegungsrichtung sind das linke vordere, das linke hintere und das rechte mittlere Bein einfach (bzw. stark) gesenkt, die übrigen drei Beine sind vertikal in ihrer Mittelstellung (bzw. einfach gesenkt), i.a.W. vom Boden gehoben.

int **oneStep**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine einen Käferschritt vorwärts macht.

Module:

mc/demos/walker/walker.c

int **turnRight**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine eine Drehbewegung nach rechts ausführt, mit der selben Beinfolge wie `oneStep`, mit dem Unterschied, daß die rechte Seite nach hinten läuft.

Module:

mc/demos/walker/walker.c

int **turnLeft**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine eine Drehbewegung nach links ausführt, mit der selben Beinfolge wie `oneStep`, mit dem Unterschied daß die linke Seite nach hinten läuft.

Module:

mc/demos/walker/walker.c

int **backStep**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine einen Käferschritt rückwärts macht.

Module:

mc/demos/walker/walker.c

int **staysolid**[1][6][2]:

Funktion:

Dieses Feld besteht aus einem Zustandsblock und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine ihre Beine zum einfachen Stehen alle gleich, seitlich wegstehend anordnet. Diese Funktion ist zum kalibrieren der Gehmaschine wichtig (siehe Anhang A: Korr.).

Module:

mc/demos/walker/walker.c

int **oneHiStep**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine einen Käferschritt in erhobener Position vorwärts macht.

Module:

mc/demos/walker/walker.c

int **turnHiRight**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine eine Drehbewegung nach rechts in erhobener Position ausführt, mit der selben Beifolge wie `oneHiStep`, mit dem Unterschied, daß die rechte Seite nach hinten läuft.

Module:

mc/demos/walker/walker.c

int **turnHiLeft**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine eine Drehbewegung nach links in erhobener Position ausführt, mit der selben Beifolge wie `oneHiStep`, mit dem Unterschied, daß die linke Seite nach hinten läuft.

Module:

mc/demos/walker/walker.c

int **backHiStep**[8][6][2]:

Funktion:

Dieses Feld besteht aus acht Zustandsblöcken und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine einen Käferschritt in erhobener Position rückwärts macht.

Module:

mc/demos/walker/walker.c

int **stayHisolid**[1][6][2]:

Funktion:

Dieses Feld besteht aus einem Zustandsblock und bewirkt bei entsprechender Übergabe an die Funktion `make_n_steps`, daß die Gehmaschine ihre Beine zum erhobenen Stehen alle gleich, seitlich wegstehend anordnet. Diese Funktion ist zum kalibrieren der Gehmaschine wichtig (siehe Anhang A: Korr.) bzw. sie wird im Demoprogramm zu Beginn aufgerufen, damit die Gehmaschine sich vor dem Bewegen erst aufrichtet, um ein unvorbereitetes Losgehen zu ver-

meiden.

Module:

mc/demos/walker/walker.c

int **make_n_steps**(int movdata[][6][2], int k, int n)

Parameterübergabe:

Movdata ist ein Feld, das typisch aus acht Zustandsblöcken besteht (die Länge ist aber nicht obligatorisch, sondern sie kann anders gewählt werden, wenn dies sinnvoll ist).

k ist die Anzahl der Zustandsblöcke, aus denen movdata besteht.

n ist die Anzahl der Schritte, die gemacht werden sollen d.h. wie oft movdata durchlaufen werden soll.

Ergebnisparameter ist ein Integerwert in dem die Anzahl der Schritte steht, die gemacht worden sind.

Funktion:

Die Funktion `make_n_steps` führt n Gehbewegungen aus, die von den Daten, die in movdata übergeben werden bestimmt werden. D.h. wird `make_n_steps` aufgerufen, so bewegt die Gehmaschine sich in den ersten Zustand, der vom ersten Zustandsblock aus movdata bestimmt wird, wartet bis die Beine die Zielposition alle erreicht haben und geht dann in den zweiten Zustand, etc. Wird das Ende von movdata erreicht, so wird von n abhängig noch einmal von vorne begonnen. N bestimmt also wie oft movdata durchlaufen wird.

Module:

mc/demos/walker/walker.c

int **make_n_stepsC**(int movdata[][6][2], int k, int n)

Parameterübergabe:

Movdata ist ein Feld, das typisch aus acht Zustandsblöcken besteht (die Länge ist aber nicht obligatorisch, sondern sie kann anders gewählt werden, wenn dies sinnvoll ist).

k ist die Anzahl der Zustandsblöcke, aus denen movdata besteht.

n ist die Anzahl der Schritte, die gemacht werden sollen, d.h. wie oft movdata durchlaufen werden soll.

Ergebnisparameter ist ein Integerwert, in dem die Anzahl der Schritte steht, die gemacht worden sind.

Funktion:

Die Funktion `make_n_stepsC` führt n Gehbewegungen aus, die von den Daten, die in movdata übergeben werden bestimmt werden. D.h. wird `make_n_steps` aufgerufen, so bewegt die Gehmaschine sich in den ersten Zustand, der vom ersten Zustandsblock aus movdata bestimmt wird, wartet bis die Beine die Zielposition alle erreicht haben und geht dann in den zweiten Zustand, etc. Wird das Ende von movdata erreicht, so wird von n abhängig noch einmal von vorne begonnen. N bestimmt also wie oft movdata durchlaufen wird. Des weiteren kann die Anzahl der Schritte, die gemacht werden auch noch durch ein Hindernis, das vorne im Weg steht, unterbrochen werden. Da an der Gehmaschine nur vorne Ultraschallsensoren bzw. Infrarotsensoren angebracht sind, ist ein Aufruf von `make_n_stepsC` nur bei einer Vorwärtsbewegung sinnvoll.

Module:

mc/demos/walker/walker.c

int **beetle_steps**(int n)

Parameterübergabe:

n ist die Anzahl der Schritte die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Schritte nach vorne gemacht, wobei von `oneStep` (s.o.) bestimmt wird

wie dieser Schritt aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_right**(int n)

Parameterübergabe:

n ist die Anzahl der Bewegungen, die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Drehbewegungen nach rechts gemacht, wobei von `turnRight` (s.o.) bestimmt wird wie diese Bewegung aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_left**(int n)

Parameterübergabe:

n ist die Anzahl der Bewegungen, die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Drehbewegungen nach links gemacht, wobei von `turnLeft` (s.o.) bestimmt wird wie diese Bewegung aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_back**(int n)

Parameterübergabe:

n ist die Anzahl der Schritte, die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Schritte nach hinten gemacht, wobei von `backStep` (s.o.) bestimmt wird wie dieser Schritt aussieht.

Module:

mc/demos/walker/walker.c

int **stillstand**(void)

Funktion:

Die Gehmaschine richtet sich in die Startposition auf, wobei von `staysolid` (s.o.) bestimmt wird wie diese Situation aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_hi_steps**(int n)

Parameterübergabe:

n ist die Anzahl der Schritte, die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Schritte nach vorne in erhobener Position gemacht, wobei von `oneHiStep` (s.o.) bestimmt wird wie dieser Schritt aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_hi_right**(int n)

Parameterübergabe:

n ist die Anzahl der Bewegungen, die gemacht werden sollen, siehe auch `make_n_steps`.

Funktion:

Es werden n käferartige Drehbewegungen nach rechts in erhobener Position gemacht, wobei von turnHiRight (s.o.) bestimmt wird wie diese Bewegung aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_Hi_left**(int n)

Parameterübergabe:

n ist die Anzahl der Bewegungen, die gemacht werden sollen, siehe auch make_n_steps.

Funktion:

Es werden n käferartige Drehbewegungen nach links in erhobener Position gemacht, wobei von turnHiLeft (s.o.) bestimmt wird wie diese Bewegung aussieht.

Module:

mc/demos/walker/walker.c

int **beetle_Hi_back**(int n)

Parameterübergabe:

n ist die Anzahl der Schritte, die gemacht werden sollen, siehe auch make_n_steps.

Funktion:

Es werden n käferartige Schritte in erhobener Position nach hinten gemacht, wobei von back-HiStep (s.o.) bestimmt wird wie dieser Schritt aussieht.

Module:

mc/demos/walker/walker.c

int **stillHiland**(void)

Funktion:

Die Gehmaschine richtet sich in die erhobene Startposition auf, wobei von stayHisolid (s.o.) bestimmt wird wie diese Situation aussieht.

Module:

mc/demos/walker/walker.c

Anhang D:

Schnittstellendefinitionen - Prozedurerklärungen, der Ausrichtung im Raum

Die Routinen der vierten Ebene d.h. der Ebene in der basierend auf den Gehfunktionen der dritten Schicht, Koordinaten angegangen werden, sind in C geschrieben und wurden mit dem GNU-Crosscompiler auf einer Sun-Workstation bzw. auf einem 486 PC in 68000 Maschinencode übersetzt.

int **selquad**(int fromX,int fromY,int toX,int toY)

Parameterübergabe:

from X ist ein Integerwert, der Quell-X-Koordinate der Gehmaschine

from Y ist ein Integerwert, der Quell-Y-Koordinate der Gehmaschine

toX ist ein Integerwert, der Ziel-X-Koordinate der Gehmaschine

toY ist ein Integerwert, der Ziel-Y-Koordinate der Gehmaschine

Als Ergebniswert wird die Zahl des Quadranten zurückgeliefert (siehe Abbildung 2.4: Ausrichtung im Raum).

Funktion:

Die Funktion selquad liefert als Ergebniswert für die eingegebenen Quell- und Zielkoordinaten den Quadranten zurück, in den der durch die Koordinaten definierte Vektor zeigt.

Module:

mc/demos/walker/walker.c

int **seldir**(int fromX, int fromY, int toX, int toY)

Parameterübergabe:

from X ist ein Integerwert, der Quell-X-Koordinate der Gehmaschine

from Y ist ein Integerwert, der Quell-Y-Koordinate der Gehmaschine

toX ist ein Integerwert, der Ziel-X-Koordinate der Gehmaschine

toY ist ein Integerwert, der Ziel-Y-Koordinate der Gehmaschine

Als Ergebniswert wird ein Offset zurückgeliefert, der die Richtung angibt, in welche die Gehmaschine gehen muß. Dabei bedeutet 0 vorwärts, 1 eine Drehbewegung nach links, etc. Der Kreis ist in insgesamt 52 Richtungen eingeteilt, also von 0 bis 51 (siehe Abbildung 2.4: Ausrichtung im Raum).

Funktion:

Die Funktion seldir liefert als Ergebniswert für die eingegebenen Quell- und Zielkoordinaten die Richtung zurück, in welche die Gehmaschine gehen muß, damit sie das Ziel auf direktestem Weg ansteuert. Natürlich muß diese Richtung noch mit der Richtung verrechnet werden, in welche die Gehmaschine im Moment gedreht ist.

Module:

mc/demos/walker/walker.c

int **gotoxy**(int fromX,int fromY,int toX,int toY,int orient)

Parameterübergabe:

from X ist ein Integerwert, der die augenblickliche X-Koordinate der Gehmaschine auf einem gedachten Plan enthält.

from Y ist ein Integerwert, der die augenblickliche Y-Koordinate der Gehmaschine auf einem gedachten Plan enthält

toX ist ein Integerwert, der die Ziel-X-Koordinate der Gehmaschine auf einem gedachten Plan

enthält, hier gilt die Einschränkung, daß to X nicht kleiner als null und nicht größer als die definierte Konstante WIDTH werden darf, sonst wird der Wert FALSE zurückgegeben, was soviel bedeutet, wie diese Position gibt es nicht.

toY ist ein Integerwert, der die Ziel-Y-Koordinate der Gehmaschine auf einem gedachten Plan enthält, hier gilt die Einschränkung, daß to Y nicht kleiner als null und nicht größer als die definierte Konstante LENGTH werden darf, sonst wird der Wert FALSE zurückgegeben, was soviel bedeutet, wie die angestrebte Position ist nicht erreichbar.

orient ist ein Integerwert, der die augenblickliche Orientierung in der Ebene repräsentiert. Diese Variable kann mit den Konstanten Werten 0 bis 51 belegt werden, wobei eine Winkelerhöhung im Gegenuhrzeigersinn stattfindet.

Funktion:

Die Funktion gotoxy bewirkt, daß die Gehmaschine von den from-Koordinaten zu den to-Koordinaten geht, wobei die Differenz n zwischen fromR und toR nicht bedeutet, daß n Gehmaschinenlängen zurückgelegt werden müssen, sondern daß n Schritte der Größe gemacht werden müssen wie die, welche bei Aufruf von make_n_steps gemacht werden.

gotoxy versucht mit einem heuristischen Verfahren die angestrebte Position zu erreichen, wenn dies nicht gelingt wird der Wert FALSE zurückgeliefert.

Module:

mc/demos/walker/walker.c

Literaturverzeichnis (Lexikographisch geordnet)

[Beer et al. 97] Randall D. Beer, Roger D. Quinn, Hillel J. Chiel, Roy E. Ritzmann: Biologically Inspired Approaches to Robotics; Communications of the ACM, Vol. 40. No. 3, S. 31-38, ACM, 1515 Broadway, New York, USA, 1997.

[Bäunl 97a] Thomas Bräunl: <http://www.cogito.de/eyebot/Core.html>

[Bäunl 97b] Thomas Bräunl: <http://www.joker-robotics.com/legs/legs.E.html>

[Dudler et al. 95] Axel Dudler, Sebastian Heldt, Rainer Leonhardt, Frank Weitmann: Schrifterkennung von Freihandeingaben, Softwarepraktikum, Lehrstuhl Prof. Dr. D. Roller, IFI, Universität-Stuttgart, Breitwiesenstraße 20-22, 70565 Stuttgart, Deutschland.

[Erik et al. 97] Erik Nygren, Terry Fong: <http://img.arc.nasa.gov/Dante/specs.html>

[Fuchs 95] Joseph Fuchs: Zeitschneider - Die Time Processor Unit des 68332; ELRAD - Magazin für Elektronik und technische Rechneranwendungen, Heft 10, S. 58-62, Deutschland, 1995.

[Harman 91] Thomas L. Harman: The Time Processor Unit; S. 493-567, in: Thomas L. Harman: The Motorola MC68332 Microcontroller, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1991.

[Heitsch 96] Gerrit Heitsch: Echtzeitbildverarbeitung auf MC 68332; Studienarbeit, 1582, IPVR, Universität-Stuttgart, Breitwiesenstraße 20-22, 70565 Stuttgart, Deutschland.

[Jones et al. 93] Joseph L. Johnes, Anita M. Flynn: Unmodified Servo Motors; S. 189-191, in Mobile Robots - Inspiration to Implementation, A K Peters, Ltd., Wellesley, Mass., USA, 1993

[Ferner 96] Lars Ferner, Michael Kasper: http://ag-vp-www.informatik.uni-kl.de/Lehre/Praktikum/Erweiterungen_HW.html#PSD

[Knülle-Wenzel 94] Alfred Knülle-Wenzel: Raubkatze - Einplatinenrechner KAT-Ce 68332, Teil 3; TPU; ELRAD - Magazin für Elektronik und technische Rechneranwendungen, Heft 5, S. 63-66, Deutschland, 1994.

[Lampart 97] Thomas Lampart: Erstellung von Systemroutinen zur Robotersteuerung auf Mikrocontroller 68332; Studienarbeit, 1614, IPVR, Universität-Stuttgart, Breitwiesenstraße 20-22, 70565 Stuttgart, Deutschland.

[Lynxmotion 97] Lynxmotion: <http://www.lynxmotion.com/index.htm>.

[MIT 97] MIT: <http://piglet.cs.umass.edu:4321/thing/thing.html>, MIT, Massachusetts, U.S.A..

[Motorola 93] Motorola: Prescaler for TCR2; Abschn. 2.1.2, in: Motorola: TPU - Time Processor Unit - References Manual, REV2, Motorola Literature Center, P.O.Box 20912, Phoenix, Arizona, USA, 1993.

[Sharp 97] <http://www.sharp.co.jp/ecg/sys/gp2d02/gp2d02-fea.html>

[Schmucker 97] Dr. Ullrich Schmucker, Dr. Anatoli Schneider, Thomas Ihme: http://www.iff.fhg.de/iff/aut/projects/schreit/mag_e.htm.

[Schott 1997] Frank Scott: <http://www.frasco.demon.co.uk/>

Bildquellennachweis

Der Bildquellennachweis ist auf Bilder bezogen, die als Originale bzw. veränderte Darstellung in dieser Studienarbeit übernommen wurden. Alle nicht ausgeführten Bilder sind vom Autor dieser Studienarbeit erstellt worden.

[Schmucker 97] Seite 2 Abbildung 1.1: Katharina

[Erik et al. 97] Seite 3 Abbildung 1.2: Dante II

[Schott 1997] Seite 4 Abbildung 1.3: Rodney 3

[MIT 97] Seite 5 Abbildung 1.4: Thing

[Bäunl 97b] Seite 6 Abbildung 1.5: Rug Walker

[Lynxmotion 97] Seite 7 Abbildung 1.6: The Beast

[Jones et al. 93] Seite 14 Abbildung 3.1: Ansteuerung eines Servos

[Bräunl 97a] Seite 15 Abbildung 3.2: Belegung des Connector 3 auf der Eyebot-Platine.

[Sharp 97] Seite 19 Abbildung 4.3: PSD GP2D02 2Z von Sharp

[Bräunl 97a] Seite 20 Abbildung 4.4: Belegung des Connector 6 auf der Eyebot-Platine.

Index

Adreßbus 17
Akkupacks 6
Aktoren 5,17
the Beast 6
Connector3 15
Connector6 20
Crosscompiler 26
Dante II
Demoprogramm 21
Device Semantik 24
Extremitätenpositionierung 9
Eyebotplatine 15
flashen 23
Frauenhofer Institut 2
NASA 3
Pulsbreitenmodulation
PWM 26
Gehmaschine 2
Handler 24
Hardwaredescriptiontable 21,23
Hardwaresteuerung 9
HDT siehe Hardwaredescriptiontable
Hexapod 1,6
IC/ITC 17
Interrupt 17
Katharina
Koreographie 10
Lamda-Übergänge 10
Lynxmotion 6
MIT 5
Mount Spurr 3
Orientierung 12
Planetenexplorationen 2
Priorität 17
PSD 7,20
Pyro-Detektoren 4
Rodney3 4
Rug Walker 5
Rug Warriors 1,4
Scheduling 17
Servo 6,7,9,14,15,16,22,27,28
Sharp 7,19
Spannungsteiler 20
Spark 3,5
stabiler Zustand
Steuerplatine 6,7
Tastsensoren 17
Thing 5
Timer 16,28
Timerstatus 17
TPU 15,16,17
Zähler 16
Zustandsblock

„Erklärung:

Ich versichere, daß ich diese Arbeit selbständig verfaßt habe
und nur die angegebenen Quellen und Hilfsmittel verwendet habe“.

Stuttgart, den 28. Juli 1997,