

## **1. Kurzfassung**

## **2. Einleitung**

## **3. Internetrecherche**

## **4. Gehen – Wie funktioniert das?**

## **5. Aufbau und Steuerung des Roboters**

### 5.1 Aufbau des Roboters

### 5.2 Entwicklung der Steuerung

#### 5.2.1 Motor, Mikroprozessor und Programmiersprache

#### 5.2.2 Die ersten Probleme

#### 5.2.3 Die Ansteuerung des Servomotors

## **6. Ausblick**

---

---

## **1. Kurzfassung**

Die Planung und die Programmierung eines zweibeinigen Geh-Roboters ist das Ziel meiner Arbeit. Daher musste ich zuerst anhand von selbsterstellten Videoaufnahmen herausfinden wie das Gehen funktioniert. Anhand dieser Forschungsergebnisse konnte ich ein elektromechanisches Modell für einen Geh-Roboter entwerfen.

Die Servomotoren meines Modells will ich mittels eines Mikroprozessors steuern. Für diesen Prozessor schrieb ich Programme in Basic und Maschinensprache. Hierbei traten sehr viele Probleme auf, die ich bis zum jetzigen Zeitpunkt noch nicht alle gelöst habe. Ich bin aber optimistisch, dass ich bis zum Wettbewerb ein funktionsfähiges Modell erstellen kann.

## 2. Einleitung

Wer hätte gedacht, dass das Gehen ein so hochkomplexer Vorgang ist? Unser Gehirn macht es scheinbar automatisch. Dabei sind sehr viele einzelne Bewegungen nötig um sich ohne umzukippen auf zwei Beinen fortzubewegen. Wir machen es unbewusst und denken gar nicht an die einzelnen Schritte. Der Mensch ist schon ein Genie in Sachen Gehen!

Ob es die Technik auch ist ???

Und schaffe ich es einen Roboter zu konstruieren und programmieren, der fähig ist auf zwei Beinen zu gehen?

## 3. Internetrecherche

Für mein Projekt war es sehr wichtig zu wissen, welche Roboter es schon gibt und was sie können. Es könnte ja sein, dass schon ein Roboter existiert, der bereits mein Ziel erreicht hat. Bei der Suche im Internet nach Gehmaschinen fanden ich tatsächlich den Gehroboter Asimo von Honda. Er kann auf zwei Beinen gehen und er kann sogar Treppen steigen kann. Dieser Roboter ist jedoch sehr teuer. Er kostet über 100.000 €

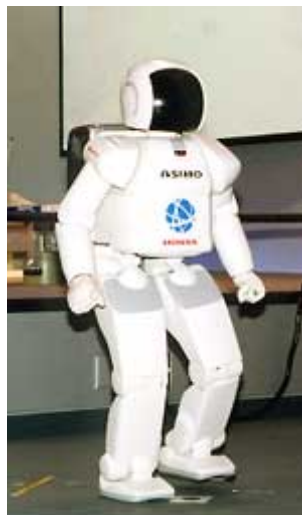


Abb. 3.1 : Der Gehroboter Asimo von Honda.

Roboter, die ich auf den Homepages von Universitäten fanden, waren meist sehr sehr komplizierte Prototypen. Somit stand mein Schüex-Thema fest:

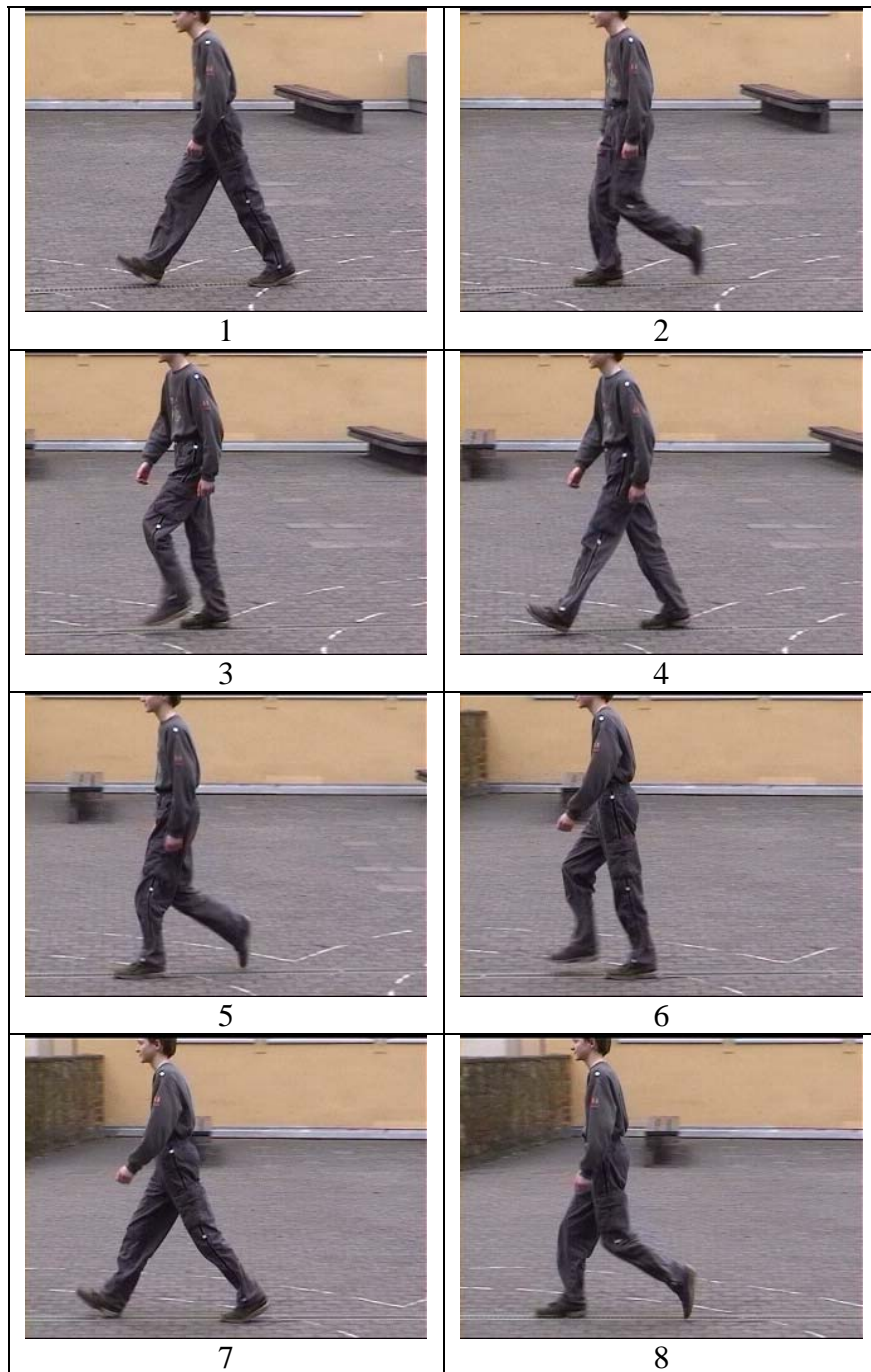
Ich baue und programmiere einen Gehroboter der

**1. billig** ist und **2. einfach gebaut** ist

## 4. Gehen – Wie funktioniert das?

Bevor man einen zweibeinigen Gehroboter plant und ein Programm hierfür entwickelt muss man sich zuerst einmal klar machen, wie das Gehen überhaupt funktioniert. Dazu bin ich oft

ganz bewusst gegangen und habe versucht über meinen Körper zu erfahren, welche Bewegungsabläufe stattfinden. Zusätzlich habe ich Videoaufnahmen gemacht, damit ich mir Bewegungsabläufe Bild für Bild ansehen konnte.



**Abb. 4.1:** Bewegungsabläufe beim Gehen



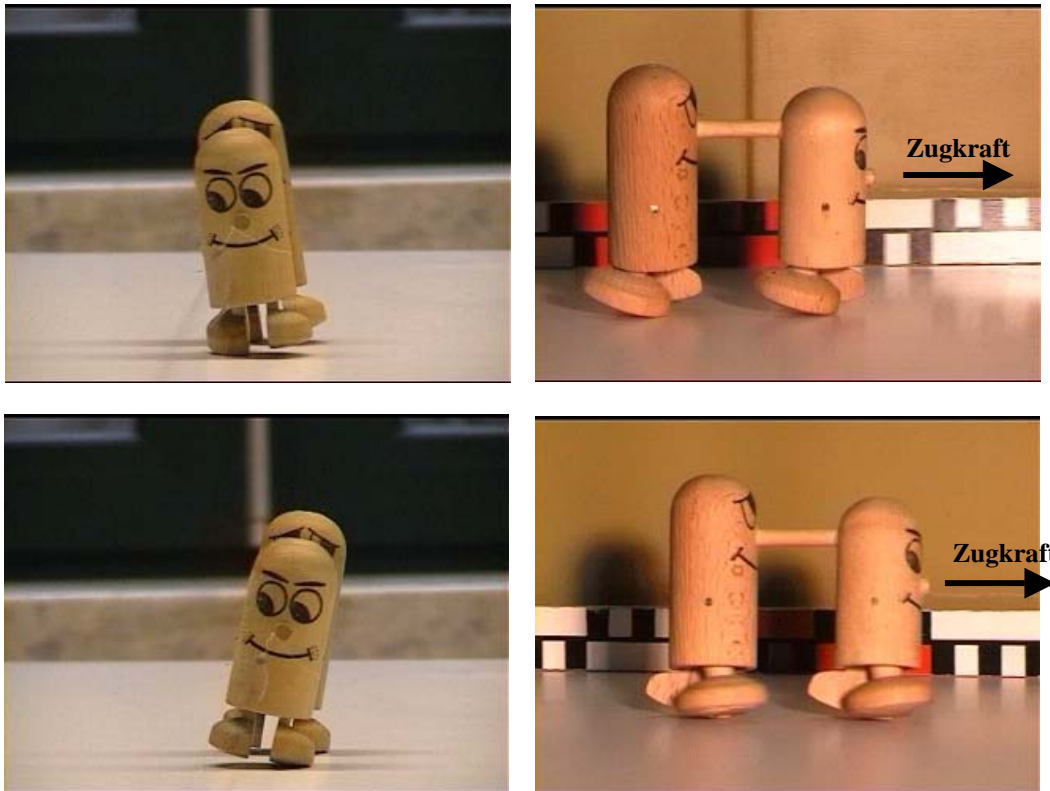
**Abb. 4.2:** Bewegungsabläufe beim Treppensteigen

Ich kam zu dem Schluss, dass das Gehen nicht bis in die letzte Kleinigkeit von unserem Gehirn gesteuert wird, sondern zum Teil auch automatisch abläuft. Unser Körper scheint so gebaut zu sein, dass, wenn er einmal in Bewegung ist, er teilweise die richtigen Bewegungen von selber ausführt. Meiner Meinung nach braucht man für das zweibeinige Gehen nicht unbedingt so ein großes Gehirn wie es der Mensch hat. Viele Vögel, wie z. B. der Strauß, können sehr gut gehen, obwohl sie viel dümmer als der Mensch sind.



**Abb. 4.3:** Ein Strauß kann nicht nur gut gehen, sondern auch bis zu 80 km/h schnell rennen.

Der Idee, dass das Gehen gar nicht so unheimlich kompliziert sein kann, kam mir, als ich ein Wackelmännchen filmte.



**Abb. 4.4:** Das Wackelmännchen ist eine vierbeinige Gehmaschine

Das „hirnlose“ Wackelmännchen ist eine Gehmaschine, die wie ein Pendel nach links und nach rechts schwingt. Dabei wird immer ein Fußpaar frei, welches in diesem Moment schnell nach vorne schwingt. So einen Wackelgang, allerdings auf zwei Beinen, haben auch die Pinguine. Sie pendeln hin und her und stürzen dabei gleichzeitig nach vorne. Das nach vorne schwingende Bein fängt dann den Sturz ab.

Die „dummen“ Vögel und das „hirnlose“ Wackelmännchen machten mir Mut. Ein Steuerungsprogramm für eine zweibeinige Gehmaschine dürfte eigentlich nicht unlösbar kompliziert sein.

## 5. Aufbau und Steuerung des Roboters

### 5.1 Aufbau des Roboters

Mein Roboter sollte natürlich nicht gehen wie ein Pinguin, sondern wie ein Mensch. Deshalb sieht mein erster Entwurf wie in Abbildung 5.1 dargestellt aus. Der Roboter hat zwei Füße, zwei Beine, einen Oberkörper und in einer späteren Ausbauphase zwei Arme. Insgesamt hat er sieben Gelenke und drei Motoren. Diese Motoren bewegen die Gelenke  $a$ ,  $b_1$  und  $b_2$ . Die Kniegelenke  $c_1$  und  $c_2$  werden jeweils von den Motoren in  $b_1$  und  $b_2$  mitgesteuert. Ich benutze also einen Motor pro Bein, der gleichzeitig das Hüftgelenk und das Kniegelenk bewegt. Verbunden werden die Gelenke durch Achsen. Dabei ist der Winkel  $\beta$  stets gleich dem Winkel  $\alpha$ . Also ist der Unterschenkel immer vertikal ausgerichtet.

Die Gelenke d1 und d2 werden nicht durch einen Motor gesteuert. Sie sind gefedert und drehen sich, der Schwerkraft zufolge etwas nach vorne, wenn sie in der Luft sind. Das garantiert das richtige Aufsetzen des Fußes. Die zwei Kontakte k1 und k2 unter den Füßen dienen dazu, um so dem Mikrocontroller mitzuteilen, welcher Fuß gerade aufkommt, um schnell genug den anderen anzuheben, damit der Roboter nicht umfällt. Dadurch ist auch das Wackeln gewährleistet, das sicheres Gehen ermöglicht. Außerdem befindet sich ein stark gedämpftes Pendel zwischen Gelenk a und b. Es ist an ein Potentiometer angeschlossen und misst die Neigung des Roboters. Diese Information gibt es an den Mikrocontroller weiter, der dann daraus die Neigung des Oberkörpers bestimmt. Ich entschied mich ein Pendel als Neigungsbestimmung zu benutzen, weil es einfach und billig ist.

Das Pendel gibt über das Potentiometer die Neigung des Roboters an. Ist der Roboter zu stark geneigt, gibt C-Control ein Signal an den Servo a, dass er sich in die entgegengesetzte Richtung neigen soll. Wenn der Roboter z.B. eine Neigung von 10 Grad nach vorne hat, neigt sich der Oberkörper solange nach hinten, bis der Roboter wieder eine stabile Lage einnimmt. Wenn der Roboter nach vorne geht, versucht C-Control das Pendel möglichst um ca. 5 Grad nach vorne geneigt zu halten. Das garantiert, dass der Roboter sozusagen immer nach vorne fällt. Da aber die Beine Schritte machen, fällt er nicht um sondern geht vorwärts. Das nennt man dann Gehen. Die Schritte der Beine fangen also das Fallen ab. Je schneller der Roboter gehen soll, desto stärker wird der Roboter nach vorne geneigt.

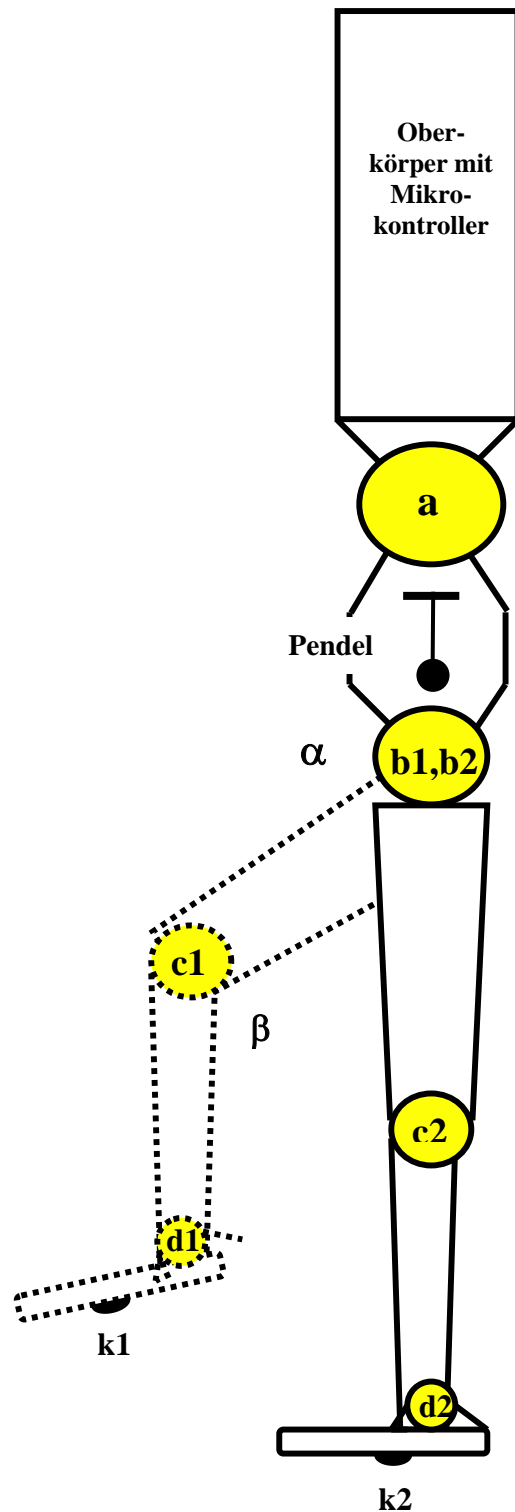


Abb. 5.1 Plan des Roboters

## 5.2 Entwicklung der Steuerung

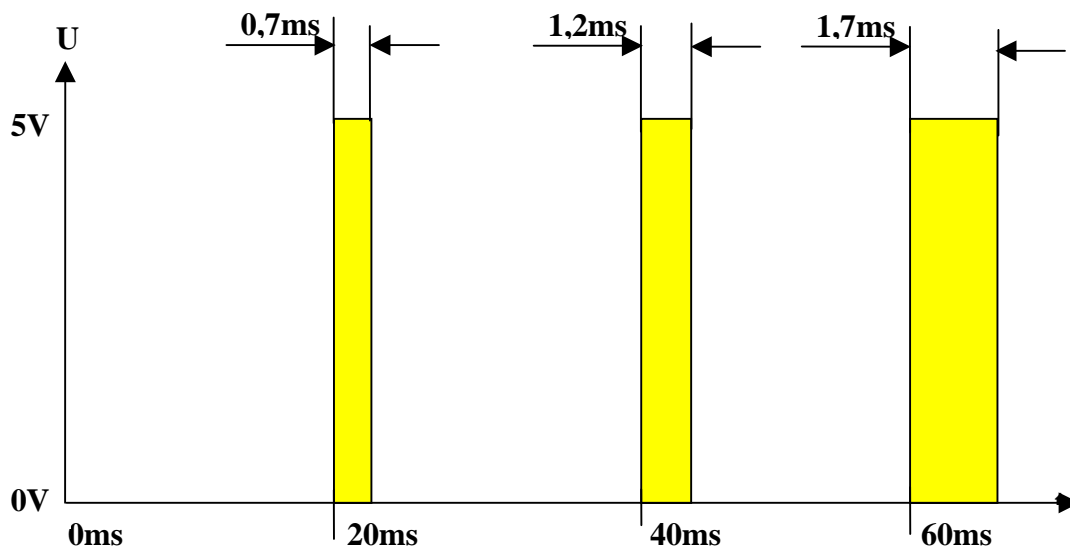
### 5.2.1 Motor, Mikroprozessor und Programmiersprache

Zur Steuerung des Roboters werde ich einen Mikrocontroller verwenden, den man vom PC aus programmieren kann. Da ich mit C-Control vertraut bin, benutzte ich C-Control. In C-Control arbeitet der Mikrocontroller vom Typ Motorola MC68HC05B6. In meiner Schule ist dieser Mikrocontroller vorhanden und ich konnte ihn sofort benutzen. Dann machte ich mir Gedanken darüber, welchen Motortyp ich für den Roboter gebrauchen konnte. Dieser muss vier Kriterien erfüllen :

1. Er muss sehr stark sein.
2. Er muss schnell reagieren
3. Er muss präzise arbeiten
4. Er muss leicht sein

Ich fand schnell einen Motortyp, der für meine Bedürfnisse optimal geeignet war. Und zwar sind Servomotoren sehr geeignet. Das sind Motoren, die ein Getriebe besitzen und zusätzlich noch eine Steuerelektronik. Diese hat drei Anschlüsse, die nebeneinander angebracht sind. Also musste ich zuerst herausfinden wie man Servomotoren ansteuert. Nach einiger Zeit des Suchens im Internet wurde ich fündig.

Und zwar wird der Servo mit Impulsen angesteuert. Die mittlere Leitung ist Träger des Impuls- oder Datensignal. Die beiden äußeren Leitungen bilden die Spannungsversorgung von 5 Volt. Dabei bestimmt das Datensignal in welchen Winkel der Servo drehen soll. Und das sieht so aus :



**Abb. 5.2: Beispiele für Impulslängen**

Alle 20 Millisekunden schickt C-Control ein Signal von 5 Volt. Dieses Signal darf zwischen 0,7ms und 1,7ms lang sein.

Bei 0,7ms dreht der Servo mit Vollausschlag nach links ( $0^\circ$ ), bei 1,2ms zur Mitte ( $90^\circ$ ) und bei 1,7ms mit Vollausschlag nach rechts ( $180^\circ$ ). Andere Werte liegen dazwischen. Mit jedem Signal dreht der Servo in die jeweilige Richtung, bis der vorgegebene Winkel erreicht ist.

Um z.B. den Servo auf  $18^\circ$  zu drehen, gibt man einen Impuls von 0,8ms Länge.

So, nun wollte ich prüfen, ob C-Control überhaupt in der Lage ist, so kurze Impulse zu geben. Zuerst wollte ich es in Basic programmieren. Diesen Gedanken verwarf ich jedoch schnell, da mir einfiel, dass der einzige Wartebefehl in Basic „pause“ ist, der nur Pausen erlaubt, die nicht kürzer als 20ms sind. Das ist für ein Signal von 0,7ms bis 1,7ms natürlich viel zu lange. Somit stand fest : Basic war zu langsam. Es gibt aber noch die Programmiersprache „Plus“ für C-Control“. Das ist eine Sprache mit grafischer Oberfläche. Doch ich merkte, dass in Plus dieselben Befehle verwendet werden wie in Basic, nur grafisch dargestellt.

Was ich brauchte war also eine sehr schnelle Programmiersprache. Ich entschied mich für die Maschinensprache. Maschinensprache ist die Sprache die der Computer versteht. Wenn man z.B. ein Programm in Basic oder Plus schreibt, wird es zuerst in die Maschinensprache übersetzt. Ein Maschinenspracheprogramm ist deswegen so schnell, weil man es eben nicht übersetzen braucht. Ein solches Programm besteht aber aus lauter Einsen und Nullen, die zwar der Computer versteht, aber ein normaler Mensch nicht. Doch es gibt sogenannte Assembler, die alle Befehle beinhalten die der Computer versteht, nur dass diese Befehle normale Wörter sind.

Also machten ich mich im Internet auf die Suche nach einem geeignetem Assembler und fand schnell den „As05“. Das ist einer, der extra für C-Control programmiert worden ist.

Dann prüfte ich erst mal, wie schnell Assemblerbefehle überhaupt ausgeführt werden. Dazu schrieb ich das folgende Programm, in der Hoffnung, dass es schnell genug ist :

```
org $101  
start:  
bset #0,$5 // Datenleitung setzen  
bclr #0,$5 // Datenleitung löschen  
jmp start // Das Ganze von vorn
```

Dieses Programm führt ständig die Befehle aus :

- 5V an die Datenleitung geben
- 0V an die Datenleitung geben

Nun leitete ich dieses Signal zuerst nicht an einen Servo, sondern an ein Oszilloskop. Ein Oszilloskop macht Signale sichtbar.

## 5.2.2 Die ersten Probleme

Als ich das Assembler-Programm kompilieren (d.h. übersetzen in „C-Controlnisch“) wollte, generierte der Assembler ein falsches Dateiformat. C-Control braucht das „s19“ Format. Der Assembler lieferte mir aber keins und zeigte auch keine Fehlermeldung an. Ich wusste nicht, dass dies nur der Anfang einer Reihe von Problemen war. Ich suchte im Internet nach einer Lösung. Nach langer Zeit fanden ich folgenden Hinweis:  
„Der Parameter -s sorgt dafür, dass der Assembler ein S19 File anlegt“

Sofort probierte ich das aus und siehe da : Es funktionierte! Neu motiviert startete ich das oben genannte Programm und stellte fest, dass C-Control ziemlich schnell ist. Viel zu schnell.

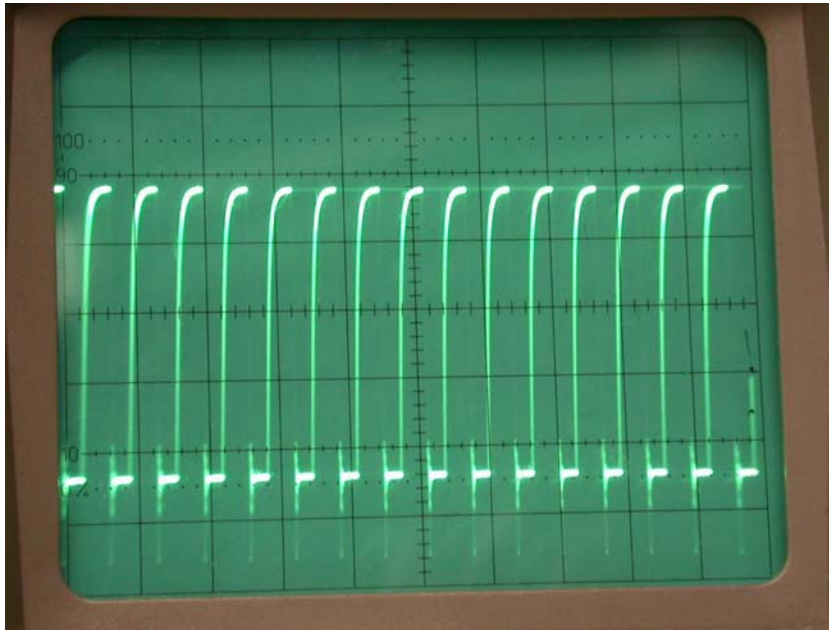


Abb. 5.3 :

**Das Oszilloskop zeigt die Ausgangssignale unseres Programms „Test“ an.**

**(Kästchenbreite = 10 $\mu$ s)**

Das Oszilloskop zeigte, dass jeder Befehl in 2 $\mu$ s ausgeführt wird. D.h. der Impuls ist nur 2 $\mu$ s lang. Er muss aber min. 700 $\mu$ s lang sein. Also mussten Warteschleifen her, um das Programm zu bremsen. Doch woher nehmen ? Die Dokumentation die dem Assembler beilag war recht mager. Dieser Tatsache zufolge war unser Wissen über diesen Assembler nahe null. Also machte ich mich mal wieder im Internet auf die Suche nach einer Assembler Befehlsreferenz. Leider erfolglos. Dann fragte ich Hendrik Krupp, einen ehemaligen Schüler unserer Schule, nach einer Lösung. Er erklärte mir die Grundstruktur von verschachtelten Warteschleifen. Leider war sein Beispielpogramm für AVR Mikrocontroller und nicht für meinen Chip. Es war trotzdem eine große Hilfe, weil mir jetzt das Grundprinzip einer verschachtelten Warteschleife bekannt war. So war es mir möglich, Warteschleifen in das Programm einzubauen, um die Impulse zu verlängern. Das Programm „Test2“ funktioniert nach dem unten dargestellten Prinzip:

**Start**

**Datensignal setzen**

**S1 = 255**

**S2 = 255**

**Schleife 1**

**S2 = 255**

**Schleife 2**

**S2 - 1**

**Solange S1 > 0 springe zu Schleife 2**

**S1 - 1**

**Solange S1 > 0 springe zu Schleife 1**

**Datensignal löschen**

**S3 = 255**

**S4 = 255**

**Schleife 3**

**S4 = 255**

**Schleife 2**

**S4 - 1**

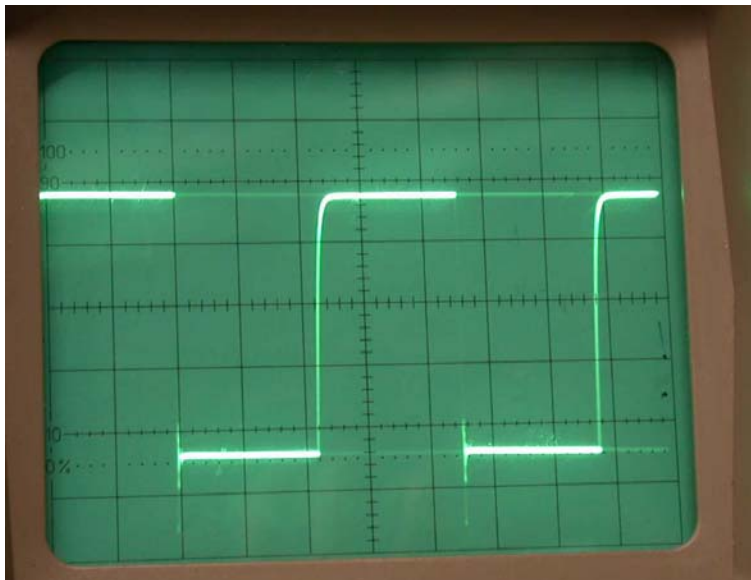
**Solange S4 > 0 springe zu Schleife 4**

**S3 - 1**

**Solange S3 > 0 springe zu Schleife 3**

**Springe zu Start**

Zunächst wird die Datenleitung auf 5V gesetzt. Dann beginnt eine einfach geschachtelte Warteschleife. Dabei ruft Schleife 1 bei jedem Durchgang, Schleife 2 auf. Somit wird erreicht, dass Schleife 2 255 Mal aufgerufen wird und dabei jedes Mal bis 255 zählt. Danach wird die Datenleitung auf 0V gesetzt und es folgt wieder eine einfach geschachtelte Schleife. Nun beginnt das Programm wieder von vorne. Beim Testen stellte sich heraus, dass die Impulse immer noch viel zu kurz waren. Jetzt waren sie 20µs lang.



**Abb. 5.4 :**

**Das Oszilloskop zeigt die Ausgangssignale unseres Programms „Test2“ an.**

**(Kästchenbreite = 10µs)**

**Das Signal ist jetzt 20µs lang.**

Selbst bei vierfacher Schachtelung waren es nur 40µs. Ich brauchte aber mindestens 700µs. Langsam sank mein Mut, doch ich gab nicht auf. Aus Verzweiflung versuchte ich es noch mal mit Basic. Basic ist zwar eigentlich zu langsam doch unter einem besonderen Umstand nicht.

*Start*

*Datenleitung setzen*  
*Datenleitung löschen*

*20 Millisekunden warten*

*zu Start springen*

Wichtig ist, dass zwischen „Datenleitung setzen“ und „Datenleitung löschen“ kein Wartebefehl, noch irgend ein anderer Befehl ist.

Beim Testen kam dann glücklicherweise das richtige Signal heraus. D.h. nachdem man die Datenleitung gesetzt hat, dauert es noch ca. 1ms bis C-Control den Befehl „Datensignal löschen“ ausführt. Natürlich ist das keine Dauerlösung, denn setzt man einen Befehl zwischen „Ein“ und „Aus“, so erhält man einen Impuls von min. 4ms. Dieser Impuls ist also deutlich länger als das benötigte Signal von 0,7 – 1,7ms Länge.

Ich konnte also nun wenigstens ein Signal von 1ms erzeugen.

### 5.2.3 Die Ansteuerung des Servomotors

Dieses Signal gab ich auf den Servo. Nun startete ich das Programm und siehe da,

**der Servo drehte sich nicht...**

Verzweiflung! Soeben wurde ich Zeuge des Unterschiedes zwischen Theorie und Praxis. Das Signal stimmte, die Kontakte stimmten. Ich war nahe daran auf einen anderen Motortyp umzusteigen. Doch vorher machte ich noch einen Versuch, um ganz sicher zu gehen, dass mein Signal stimmt, und dass mein Servomotor funktioniert. Normalerweise werden Servomotoren im Modellbau verwendet, z.B. zum Steuern der Ruder bei einem Modellflugzeug. Deshalb schließt man sie direkt an den Empfänger einer Fernsteuerung an. Das habe ich auch getan und zum ersten Mal hat sich der Servo gedreht. Nun gab ich das Signal vom Empfänger auf das Oszilloskop und stellte fest, dass es mit dem Signal von C-Control übereinstimmt.

Wieder starke Verzweiflung! Lange Zeit machte ich alles wieder und wieder, aber nie drehte sich der Servo. Ich hatte alles richtig gemacht und trotzdem konnte das Signal den Servo nicht drehen.

Jetzt baute ich eine Verstärkerschaltung zwischen C-Control und dem Servo. Es könnte ja sein, dass das Signal von C-Control einfach zu schwach für den Servo ist.

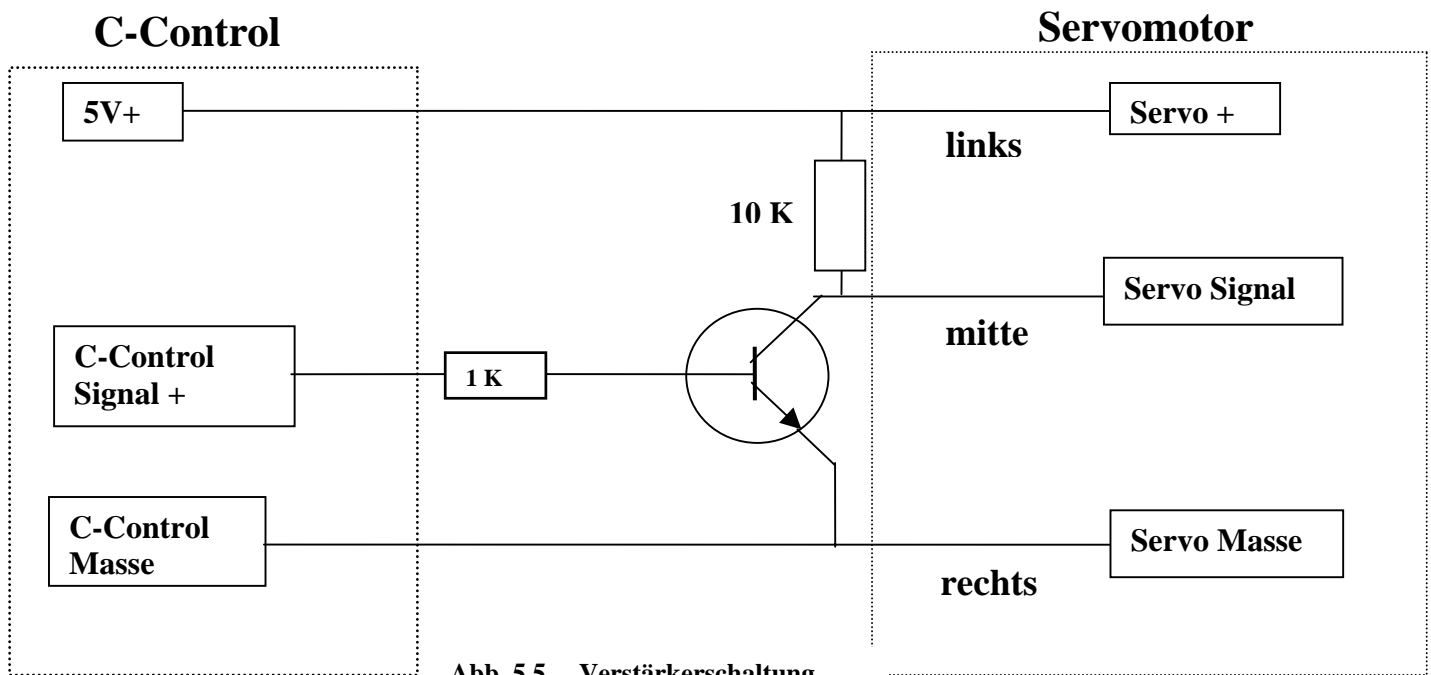


Abb. 5.5 Verstärkerschaltung

Wenn „C-Control Signal +“ nicht gesetzt ist, schaltet der Transistor nicht durch. D.h. zwischen den Anschlüssen „rechts“ und „mitte“ ist eine Spannung von 5V und zwischen „links“ und „mitte“ kaum Spannungsunterschied. Also fließt der Strom zu „Servo Signal“.

Wenn „C-Control Signal +“ gesetzt ist Schaltet der Transistor durch. D.h. die Spannung zwischen „links und „rechts“ fällt ab. Also fließt kein Strom zu „Servo Signal“.

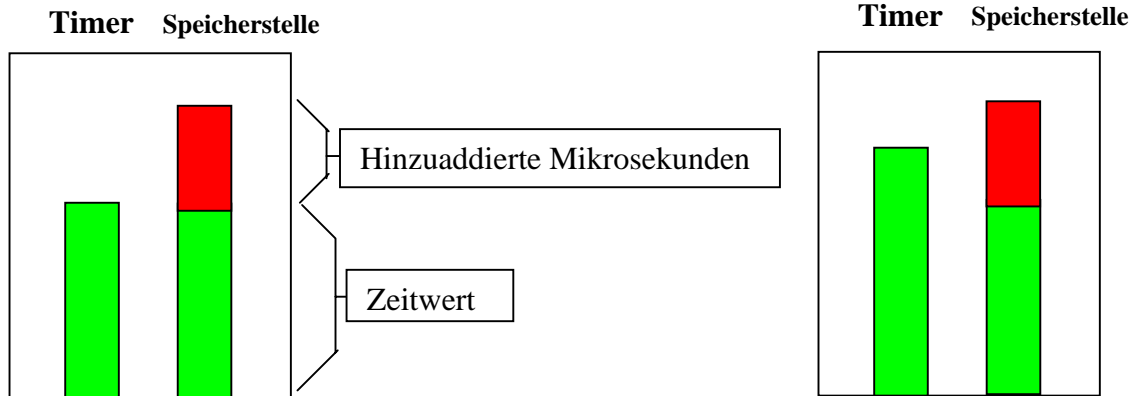
Diese Verstärkerschaltung verstärkt und invertiert also. Doch auch mit dieser Verstärkerschaltung drehte sich der Servo nicht.

Langsam wuchs der Verdacht, dass ich Opfer einer Fehlinformation war. Sollten die Informationen, die ich über die elektronische Beschaltung von Servos besaß falsch sein? Im Internet erfuhr ich dann, dass jeder Hersteller die Anschlüsse der Motoren mit unterschiedlichen Farben kennzeichnet. Nachdem ich die für unseren Servo richtige Farbkombination gefunden hatte, baute ich meine Verstärkerschaltung um.

Dann startete ich gespannt das Programm, vernehme ein Geräusch aus Richtung Servo, und sehe wie

**DER SERVO SICH DREHT !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**

Monatelange Arbeit war nicht umsonst gewesen. Aber ich hatte den Servo bis jetzt „nur“ in Basic gesteuert. Und aus o.g. Grund kann man nur in einen Winkel drehen. Um in beliebige Winkel zu drehen benötige ich Assembler. Doch der ist trotz Warteschleifen noch zu schnell. Jetzt fragte ich den Informatikstudenten Daniel Schlich nach einer Lösung. Er ist sehr erfahren in der Assemblerprogrammierung. Daniel schlug mir vor, einfach den internen Timer von C-Control zu nutzen. Der interne Timer ist quasi eine Uhr die startet, wenn man C-Control einschaltet. Jede Mikrosekunde wird der Timer und eine Speicherstelle um eins erhöht. Jetzt addieren wir die Mikrosekunden die ich warten möchte zu dieser Speicherstelle. C-Control „merkt“ nun, dass der Timer und die Speicherstelle verschieden sind. Deswegen wird jetzt jede Mikrosekunde nur noch der Timer um eins erhöht und nicht mehr die Speicherstelle.



**Abb. 5.4** Zuerst werden die Mikrosekunden addiert

**Abb. 5.5** Jetzt erhöht sich nur noch der Timer

Nun erhöht sich der Wert des Timers (Abb. 5.5), bis er mit dem Wert der Speicherstelle übereinstimmt. Wenn das der Fall ist, fährt das Programm fort, ansonsten wird weiter geprüft, ob die zwei Werte gleich sind.

Leider drehte sich der Servo mit diesem neuem Programm auch nicht.

## 6. Ausblick

Tatsache ist, dass ich mit der bisherigen Ansteuerung z.Z. nicht weiterkomme. Deswegen habe ich mir überlegt, ob es noch Alternativen gibt. Ich bin auf drei Lösungen gekommen :

1. Solange das Assemblerprogramm verbessern, bis der Servo sich dreht.
2. Den Mikrokontroller wechseln.
3. Einen Servotester benutzen. Und zwar sind das kleine Schaltungen an die man einen Servo anschließen kann. Außerdem ist ein Potentiometer angebracht mit dem man durch drehen den Winkel des Servos ändern kann. Da ich aber den Servotester mit C-Control ansteuern würde, muss ich ein digitales Potentiometer nehmen.