K U R Z F A S S U N G

(Rückseite kann auch beschrieben werden)

Bundesland: B r e m e n

Name/Anschrift/Telefon

Einzel bzw. Gruppensprecher

Heiko Purnhagen
Elsasser Str. 64
2800 B r e m e n
0421 / 3 49 96 55

2. Gruppenmitglied

3. Gruppenmitglied

Alter: 15 Jahre (geb. 2.4.69)

Schule/~~Betrieb~~/Anschrift

Gymnasium Horn, Ronzelenstr., 2800 Bremen

Fachgebiet und Thema: - T e c h n i k -

1-Bit-Spracherkennung

In meiner Arbeit beschreibe ich die Entwicklung und Funktionsweise eines Spracherkennungs-Systems, das mit einem Minimum an Hardware auf einem weit verbreiteten Homecomputer arbeiten kann. Die Idee zu diesem Gerät kam mir auf dem Jugend-Forscht-Bundeswettbewerb 1984. Die dort vorgestellten Spracherkennungs-Systeme interessierten mich sehr. Da jedoch beide einen großen Aufwand an Hardware benötigten, suchte ich nach Möglichkeiten, eine Spracherkennung mit wesentlich weniger Hardware durchzuführen.
Dabei bin ich auf das Prinzip des 1-Bit-Analog-Digital-Wandlers gestoßen. Bei meinen ersten Versuchen mit diesem Prinzip betrachtete ich nur die Sprachein- und -ausgabe. Als diese zufriedenstellende Ergebnisse zeigten, begann ich, ein 1-Bit-Spracherkennungs-System zu entwickeln. Die hierfür benötigte Hardware besteht neben dem Computer aus einem Mikrofon, 2 Operationsverstärkern und einem Eingabe-Interface für den Computer. Sie läßt sich einfach und kostengünstig selber aufbauen. Die Software zur Spracherkennung habe ich auf einem 8085-Mikrocomputersystem entwickelt und später auf den Homecomputer Sinclair ZX Spectrum übertragen. Die in Assembler geschriebenen Spracherkennungs-Programme lassen sich auf ihm einfach in BASIC-Programme einbinden.

Mit dem fertigen System erreiche ich bei 32 Wörtern eine Erkennungssicherheit von mehr als 90 %. Mit dieser Erkennungssicherheit ist mein Spracherkennungs-System durchaus für praktische Anwendungen geeignet. Da man bei solchen Anwendungen normalerweise eine Kommandosprache benutzt, habe ich die Syntax dieser Sprache mit in die Erkennung einbezogen und konnte auf diese Weise die Qualität und Benutzerfreundlichkeit noch wesentlich steigern. Als Beispiel für eine praktische Anwendung kann ich mit Hilfe der Spracherkennung Geräte ein- und ausschalten bzw. in verschiedene Stufen schalten.
So könnte diese Sprachsteuerung unter gewissen Umständen für Behinderte eine kostengünstige Hilfe darstellen.

1 - B I T - S P R A C H E R K E N N U N G

Ein Beitrag zum Wettbewerb " JUGEND FORSCHT " 1985

Gliederung

von Heiko Purnhagen, Bremen, Elsasser Str. 64

(geb. 2.4.1969) Schüler des Gymn. Horn, Bremen

1. Einleitung

In meiner Arbeit beschreibe ich die Entwicklung und Funktionsweise eines Spracherkennungs-Systems, das mit einem Minimum an Hardware auf einem weit verbreiteten Homecomputer arbeiten kann. Die Idee zu diesem Spracherkenungs-System kam mir auf dem Jugend-Forscht-Bundeswettbewerb 1984. Die dort von Dirk Graudenz sowie von Andreas und Dierk Schleicher vorgestellten Spracherkennungs-Systeme interessierten mich sehr. Da jedoch beide einen großen Aufwand an Hardware benötigten, suchte ich nach Möglichkeiten, eine Spracherkennung mit wesentlich weniger Hardware durchzuführen.
Dabei bin ich auf das Prinzip des 1-Bit-Analog-Digital-Wandlers gestoßen. Bei meinen ersten Versuchen mit diesem Prinzip betrachtete ich nur die Sprachein- und -ausgabe. Als diese zufriedenstellende Ergebnisse zeigten, begann ich, ein 1-Bit-Spracherkennungs-System zu entwickeln.

2. Mögliche Spracherkennungsverfahren

Es gibt verschiedene Arten von Spracherkennungs-Systemen, die sich darin unterscheiden, daß sie entweder einzeln gesprochene Wörter oder Wörter innerhalb von flüssig gesprochenen Sätzen erkennen können. Diese Erkennung kann entweder sprecherabhängig oder sprecherunabhängig geschehen.
Damit eine Spracherkennung durchgeführt werden kann, muß jedes Spracherkennungssystem zuerst trainiert werden, um danach in der Erkennungphase einen Vergleich mit den erlernten Referenzmustern und so die Erkennung durchführen zu können.
Dabei ist immer eine Aufbereitung des Sprachsignals vom Mikrofon notwendig. Ziel der Aufbereitung ist meistens eine spektrale Analyse des Sprachsignals, da man die menschliche Sprache anhand ihres zeitabhängigen Spektrums gut analysieren kann.
Diese spektrale Analyse, bei der eine Auflösung von 8 bis 16 Frequenzkanälen im Bereich von etwa 100 Hz bis 5000 Hz üblich ist, kann zum Beispiel über eine analoge Filterbank erfolgen. Eine andere Möglichkeit besteht darin, daß das Signal zuerst mit einem Analog-Digital-Wandler digitalisiert wird, um es dann im Computer mit einem speziellen Algorithmus spektral zu analysieren.
Bei der von mir benutzten sprecherabhängigen Einzelworterkennung wird ein Spektrum in Abhängigkeit von der Zeit als Erkennungskriterium benutzt. Ein daraus erzeugtes Muster wird für jedes Wort des Wortschatzes gespeichert. Beim Erkennen wird dann dieses Muster mit den gespeicherten Mustern verglichen. Hierbei ist die zeitliche Anpassung der zu vergleichenden Muster relativ problematisch. (siehe 3.3.4.)

3. 1-Bit-Spracherkennung

Um mit möglichst geringem Aufwand eine funktionsfähige Spracherkennung aufbauen zu können, benutze ich einen 1-Bit-Analog-Digital-Wandler (Schmitt-Trigger) und werte dessen Signale mit einem normalen Mikrocomputersystem oder einem preiswerten Home-Computer aus.

3.1. Prinzipielle Wirkungsweise

Da ich meine Spracherkennung mit möglichst wenig Hardwareaufwand konstruieren wollte, war das Prinzip der analogen Filterbank von Anfang an ausgeschlossen. So blieb nur noch die Benutzung eines Analog-Digital-Wandlers übrig. Da aber die Verwendung eines nor-

malen 8-Bit-Wandlers bei der spektralen Analyse auf den von mir benutzten Mikrocomputern zu langen Rechenzeiten geführt hätte, entschloß ich mich, die Auflösung des Wandlers auf den Minimalwert von 1 Bit zu verringern. Dieses läßt sich mit einem Schmitt-Trigger sehr einfach realisieren. Man erhält dann praktisch nur die Information über das Vorzeichen bzw. die Nulldurchgänge des Sprachsignales, aber keine Information über die Amplitude. Der Schmitt-Trigger muß eine kleine Hysterese aufweisen, damit er bei normalen Hintergrundgeräuschen nur selten seinen Zustand ändert. (siehe 3.2.2.)
Dieses 1-Bit-Signal wird abgespeichert, sobald vom Computer der Anfang eines Wortes erkannt wurde. Nachdem das Ende des Wortes erkannt worden ist, werden die gespeicherten Informationen spektral analysiert und dann als normiertes Spektrum in Abhängigkeit von der Zeit gespeichert. Dazu wird das Wort in 16 gleichlange Zeitbereiche aufgeteilt. Für jeden dieser Zeitbereiche wird die relative Häufigkeitsverteilung der Periodendauern des 1-Bit-Signales gespeichert. Die Periodendauern werden in 7 Bereiche eingeordnet, die den Frequenzen von etwa 100Hz bis >5KHz entsprechen. (siehe 3.3.2.) Pro Wortmuster werden 128 Byte abgespeichert.
Während einer Lernphase wird so für jedes Wort das charakteristische Wortmuster oder -spektrum ermittelt. Um ein möglichst typisches Wortmuster für das jeweilige Wort zu bekommen, bilde ich den Mittelwert aus 4 Mustern. (siehe 3.3.3.) Diese Wortmuster werden in einer Tabelle gespeichert. Bei meinem Programm kann diese Tabelle die Muster von 32 Wörtern aufnehmen. Sie hat daher eine Größe von 4 K Bytes.
Während der Erkennungphase vergleicht mein Programm das Wortmuster des gerade gesprochenen Wortes mit allen 32 Wortmustern der Referenztabelle. Dabei wird immer die Summe der Quadrate der Differenzen korrespondierender Werte für jedes Wort in der Referenztabelle gebildet. Das Wort mit der kleinsten Fehlerquadratsumme gilt dann als erkannt. Eine Fehlergrenze habe ich nicht eingebaut, da ich damit die Erkennungssicherheit nicht erhöhen kann.

3.2. Hardware

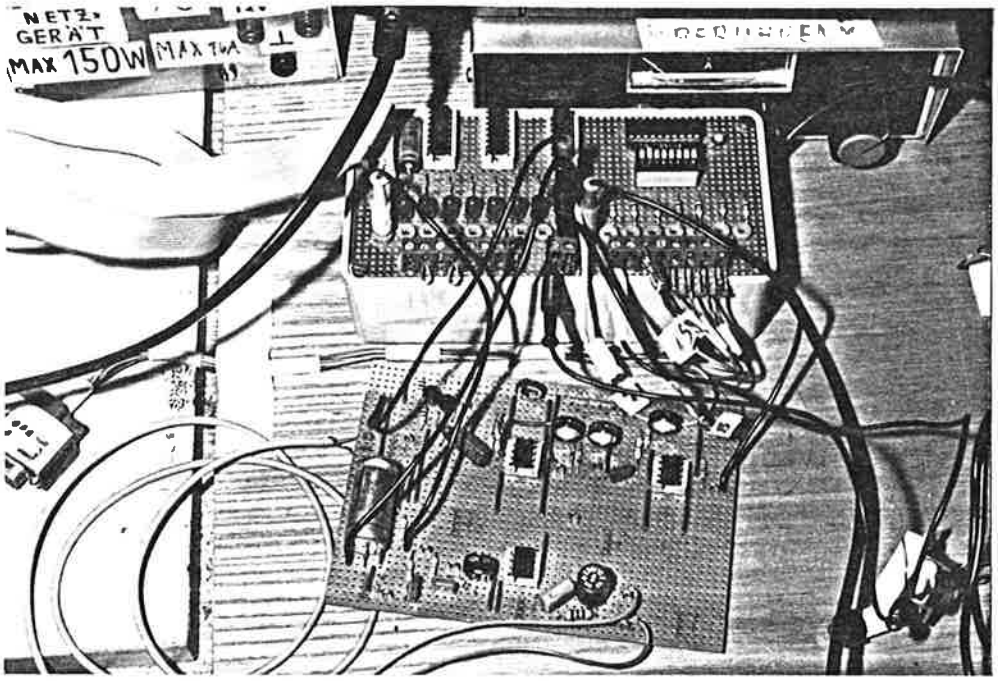
An Hardware wird nur ein Mikrofon, ein Verstärker und der Schmitt-Trigger benötigt. Für den Anschluß an einen Computer benötige ich zusätzlich ein 1-Bit-Eingabeinterface.

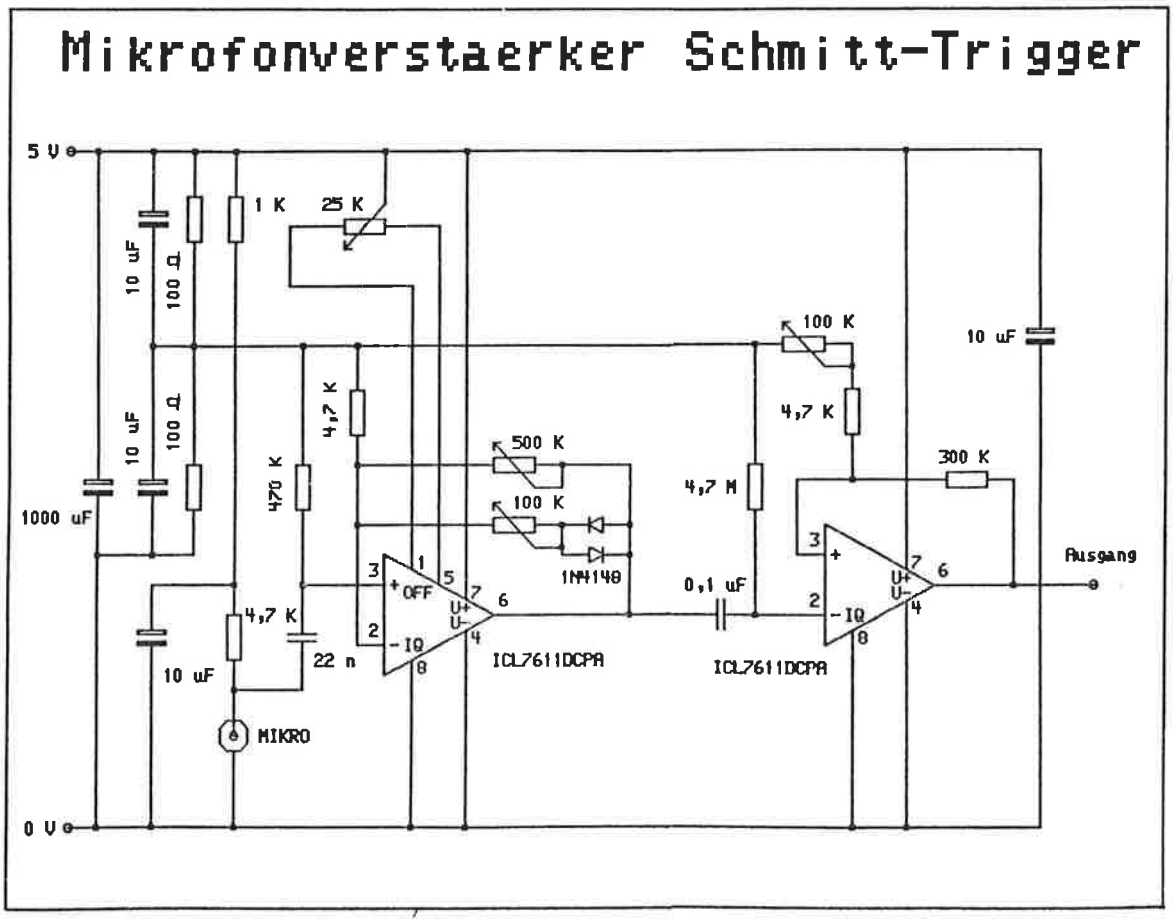
3.2.1. Mikrofon und Befestigung

Als Mikrofon benutze ich ein kleines Electret-Mikrofon, das ich mit einem festen Draht an einem leichten Kopfhörer befestigt habe. (siehe Foto S. 3) Diese Befestigung ist nötig, da sich sonst das Mikrofon nicht in einer festen Position relativ zum Mund befinden würde. Dieses hätte zur Folge, daß das Sprachsignal starke Schwankungen haben könnte, da das Mikrofon nur etwa 5 cm vom Mund entfernt ist und sich so im Bereich des Nahschalles befindet. Es ist aber auch nicht möglich, den Abstand zwischen Mund und Mikrofon zu vergrößern, da sich dann das Verhältnis von Nutz- zu Störsignal erheblich verschlechtern würde, ich aber aufgrund des Prinzips mit dem 1-Bit-Wandler auf ein gutes Nutz / Stör-Verhältnis angewiesen bin.
Ein anderes Problem bilden die Blasgeräusche, die entstehen, wenn das Mikrofon im Luftstrom des Mundes angeordnet ist. Dieses Problem läßt sich jedoch einfach dadurch lösen, daß man das Mikrofon neben den Mund positioniert, wie ich es mit der Befestigung am Kopfhörer getan habe.

3.2.2. Verstärker und Schmitt-Trigger

Das Signal vom Mikrofon wird verstärkt und dann in den Schmitt-Trigger gespeist. Für den Verstärker und den Schmitt-Trigger verwende ich den CMOS-Operationsverstärker ICL 7611 DCPA, da er einen sehr geringen Eingangsstrom hat und auch bei geringen Versorgungsspannungen (5 V vom Computer) einfach zu benutzen ist.
Der Verstärker ist über einen einfachen RC-Hochpaß mit einer Eckfrequenz von 15 Hz an das Mikrofon angekoppelt. Er hat eine einstellbare Verstärkung von etwa 50 und wirkt aufgrund von zwei Dioden über der Rückkopplungsleitung als Dynamikkompressor. Die Dioden verhindern auch ein übersteuern des Verstärkers und begrenzen das Ausgangsignal auf etwa 2 Vss.
Der Schmitt-Trigger ist über einen RC-Hochpaß an den Verstärker angekoppelt. Dieser Hochpaß hat eine Eckfrequenz von etwa 0,3 Hz. Durch diese Hochpässe werden eventuell noch vorhandene Blasgeräusche weggefiltert. Der Schmitt-Trigger hat eine veränderbare Hysterese von etwa 0,5 V. Sie ist so eingestellt, daß bei normalen Umgebungsgeräuschen die Schaltschwelle gerade noch nicht erreicht wird.
Ich habe auch untersucht, ob ein Differenzierer eine Erhöhung der Erkennungssicherheit bewirken würde. Es hat sich aber keine eindeutige Veränderung der Erkennungssicherheit ergeben. (Schaltplan und Foto der Schaltung siehe S. 4)


NETZ-
GERÄT
MAX 150W


Mikrofonverstaerker Schmitt-Trigger
5 V
0 V
1 K
25 K
10 uF
100 Ω
10 uF
100 Ω
1000 uF
470 K
4,7 K
500 K
100 K
1N4148
4,7 K
22 n
10 uF
MIKRO
ICL7611DCPA
4,7 M
0,1 uF
100 K
4,7 K
300 K
10 uF
ICL7611DCPA
Ausgang

3.3. Software

Die Software, die ich für die Spracherkennung benötige, habe ich in 8085-Assembler auf einem Mikrocomputersystem geschrieben. Es besitzt einen 8085A und etwa 29 K Byte freien Speicher. Meine Programme sind etwas länger als 2 K Byte und benötigen zusätzlich etwa 10 K Byte für die Variablen und Tabellen.

3.3.1. Speicherprinzip / Wortanfang- und -endeerkennung

Bevor ein gesprochenes Wort analysiert und erkannt werden kann, muß es zunächst einmal gespeichert werden. Hierzu benötige ich eine Methode, den 1-Bit-Datenstrom, der vom Schmitt-Trigger geliefert wird, mit einer möglichst hohen Auflösung platzsparend zu speichern. Das einfachste System besteht wohl darin, den Zustand des 1-Bit-Signales mit hoher Geschwindigkeit in den Speicher zu übertragen. Da die Abtastrate mindestens 20 K Bit / s betragen sollte, werden bei einer maximalen Wortlänge von 2 Sekunden etwa 5 K Byte benötigt. Außerdem muß die Abtastfrequenz sehr regelmäßig sein, und das Programm darf nicht nach 8 Bits etwas mehr Zeit benötigen, um ein Byte zu speichern. Zusätzlich hätte man bei der nachfolgenden spektralen Analyse die Abstände von einer Zustandsänderung bis zur nächsten anhand der Bits im Speicher auszählen müssen, was auch schwierig und zeitaufwendig geworden wäre.
Daher wird bei meiner Lösung direkt beim Einlesen eines Wortes die Periodendauer gemessen, indem in einer Schleife ein Zähler solange erhöht wird, bis sich der Zustand des 1-Bit-Signales geändert hat. Bei meinem Programm beträgt die Einheit für die Messung der Periodendauer etwa 18 us. Dies entspricht einer Abtastrate von ungefähr 55 K Bit / s. Die Längen der High- und Low-Perioden (Halbwellen) werden dann als 16-Bit-Zahlen hintereinander in den Speicher geschrieben. Dieses dauert pro Wert etwa 25 us. So benötige ich für ein Wort maximal etwa 4 K Byte Speicher. Relativ zur Abtastrate gesehen, ist das nur etwa 1/4 des Speicherplatzes, der vom oben beschriebenen System benötigt würde.
Ein anderes Problem bestand in der Erkennung des Wortanfangs und des Wortendes. Normalerweise wird hierzu eine Amplitudenschwelle benutzt. Da der Schmitt-Trigger bis auf seine Ansprechschwelle oder Hysterese keine Information über die Amplitude des Sprachsignals liefert, muß ich für die Erkennung der Wortgrenzen andere Kriterien benutzen. Durch längere Versuche habe ich festgestellt, daß 16 aufeinanderfolgende Perioden mit einer Periodendauer unter 10 ms ein gutes Kriterium für den Anfang eines Wortes sind. Bei anderen Werten würde schon ein Umgebungsgeräusch als Wort gelten, oder es ein leise gesprochenes Wort würde zu spät bemerkt werden. Wenn innerhalb des Wortes dann eine Periode länger als 150 ms dauert, wird dieses als vorläufiges Wortendekriterium benutzt. Wenn man die Grenz-Periodendauer für das Wortende verkleinerte, würde eine kleine Pause in einem Wort, wie z.B. bei "Sprach_erkennung", schon als Wortende erkannt werden. Würde man dagegen diesen Wert zu groß wählen, dann könnte ein Wort durch Blasgeräusche noch erheblich verlängert werden.
Nach dem Erkennen des Wortendes wird die Tabelle der Periodendauern dann vom Ende her durchsucht, bis wieder 16 aufeinanderfolgende Perioden mit einer Periodendauer unter 10 mS gefunden wurden. Durch diese endgültige Festlegung des Wortendes werden eventuelle gespeicherte Blasgeräusche am Ende eines Wortes herausgenommen.

Mein Programm ist also nun in der Lage, Anfang und Ende eines Wortes mit ausreichender Sicherheit zu erkennen. Eine gute Wiederholgenauigkeit dieser Erkennung der Wortgrenzen ist für mich wichtig, da das Wort zwischen diesen Grenzen einfach in 16 gleichlange Zeitabschnitte eingeteilt wird. Wäre die Wiederholgenauigkeit gering, würden bei der Erkennung eines Wortes Zeitabschnitte verglichen, die nicht zu denselben Teilen des jeweiligen Wortes gehören, und eine Erkennung wäre nicht möglich.

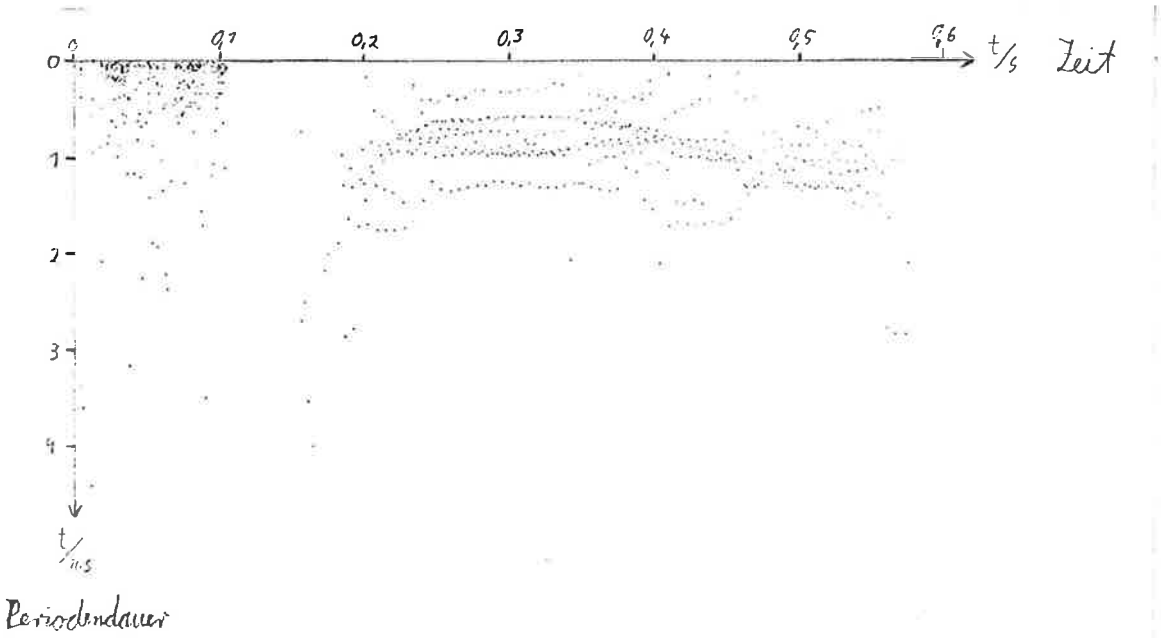
3.3.2. Spektrale Analyse und Normierung

Nach dem Einlesen wird das Wort spektral analysiert. Dafür wird es zuerst in 16 gleichlange Zeitabschnitte eingeteilt. Hierzu wird zuerst die Gesamtlänge des Wortes als Summe aller Periodenlängen berechnet. Diese Summe wird dann durch 16 dividiert und als Abschnittslänge gespeichert.
Daraufhin wird jede gespeicherte Periodendauer in einen der von mir benutzten 7 Periodendauerbereiche (Frequenzkanäle) eingeordnet. Es wird gezählt, wie oft die Periodendauer halbiert werden kann, ohne daß das Ergebnis ohne Betrachtung der Nachkommastellen Null wird. Dieser Wert entspricht dem Logarithmus der Periodendauer auf die Basis 2. So ergibt sich, daß dem höchsten Bereich Periodendauern kleiner als 0.1 ms bzw. Frequenzen über 5 KHz zugeordnet werden. Der 7. und tiefste Bereich entspricht Periodendauern kleiner als 4.6 ms bzw. Frequenzen über 108 Hz. (restliche Bereichsgrenzen siehe S. 7)
Wenn man eine Periodendauer in einen dieser 7 Bereiche einordnen kann, wird zu dem 16-Bit-Zähler für diesen Bereich die jeweilige Periodendauer addiert. Da diese Zähler vor der spektralen Analyse gelöscht wurden, kann ich aus ihnen später direkt ablesen, welchen zeitlichen Anteil Periodendauern in dem jeweiligen Bereich und Zeitabschnitt hatten bzw. wie stark dieser Frequenzbereich vertreten war. Ist dann das Ende eines Zeitabschnittes erreicht, werden die Zählerstände in einer Tabelle abgespeichert und es wird derselbe Vorgang im nächsten Abschnitt durchgeführt, bis alle 16 Zeitabschnitte bearbeitet worden sind. Ein 8. Bereich für Periodendauern wird nicht benutzt und bleibt daher Null.
Nach dieser Analyse werden die in der Tabelle gespeicherten Zählerwerte durch die Länge eines Zeitabschnittes dividiert. Dann wird diese Tabelle so normiert, daß sich alle 128 Werte in dem Bereich von 0 bis 255 befinden und ich zur Speicherung jedes Wertes nur ein Byte benötige. Ich erhalte so ein Spektrum des Wortes als Funktion der Zeit. Das durch mein Programm ermittelte Spektrum ist nicht gleich der spektralen Verteilung der Energie, da die Energieinformation durch den Schmitt-Trigger verloren geht. Es eignet sich aber dennoch ähnlich gut für die Spracherkennung wie ein Energiespektrum.
Durch die Einteilung in 16 gleichlange Zeitbereiche führe ich eine lineare Zeitnormierung durch. Diese ist im Zusammenhang mit dem Erkennungsverfahren wichtig. (siehe 3.3.4.)

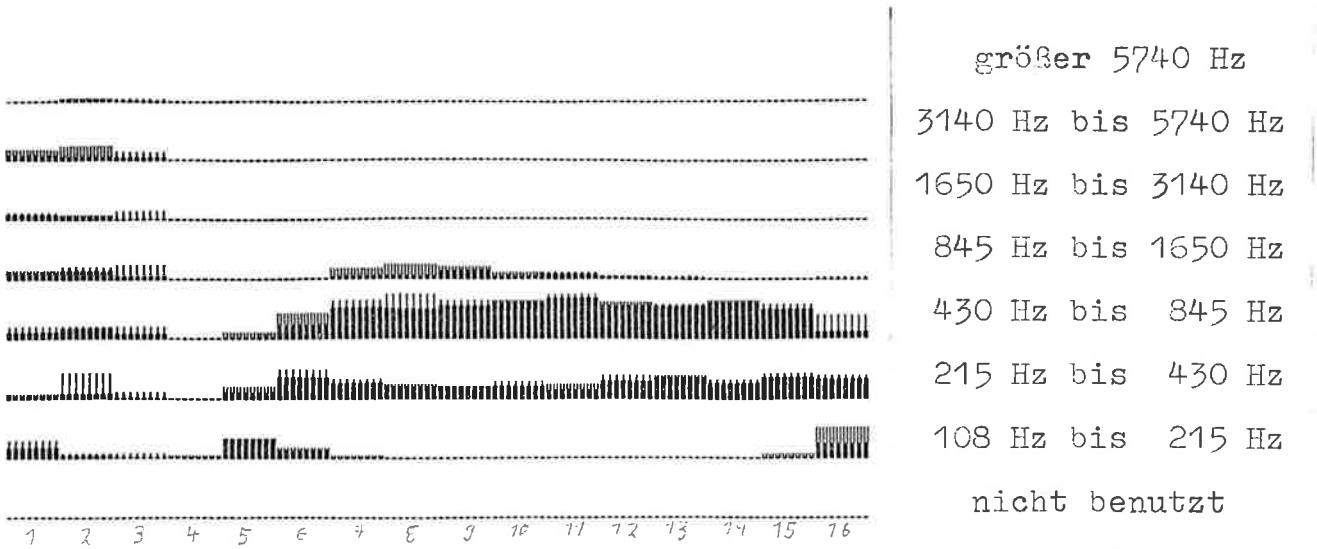
Spektrale Analyse eines Wortes

am Beispiel des Wortes "Zwei"

Gespeicherte Periodendauern als Funktion der Zeit



Analysiertes Spektrum und Referenzmuster des gleichen Wortes



Nr. des Zeitabschnittes

Frequenzbereiche

Referenzmuster: weite Striche
Gesprochenes Wort: enge Striche

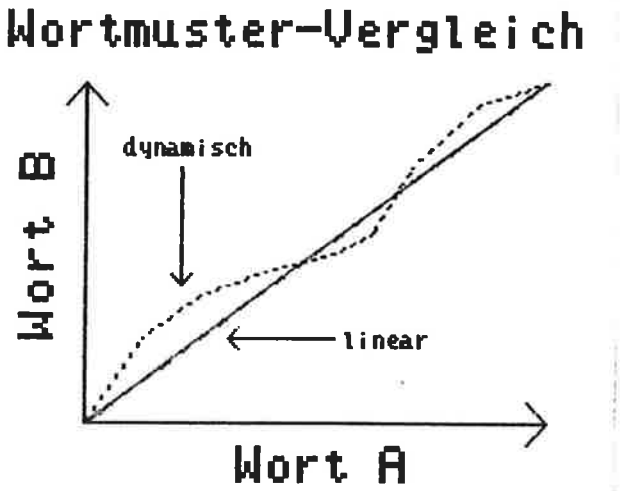
3.3.3. Vergleichsmuster

Um eine Spracherkennung durchführen zu können, muß ich zunächst alle Wörter, die erkannt werden sollen, in einer Tabelle für Vergleichsmuster bzw. Referenzmuster speichern. Bei meinem Programm habe ich in dieser Tabelle Platz für 32 Wortmuster. Sie benötigt daher 4 K Byte Speicher.
Die einfachste Möglichkeit, ein Wort zu lernen, besteht darin, es einmal einzulesen, spektral zu analysieren und dann als Referenzmuster in der Tabelle zu speichern. Da jedoch das Spektrum eines Worts bei mehrmaligem Sprechen immer geringe Unterschiede aufweist, könnte so zufällig ein untypisches Muster in der Tabelle gespeichert werden. Um diesem vorzubeugen, wird ein Wort 4 mal eingelesen und dann für jeden Bereich in jedem Zeitabschnitt das arithmetische Mittel der 4 Messungen gebildet. Das gemittelte Muster wird dann in der Tabelle gespeichert. So erhöhe ich die Erkennungssicherheit um einige Prozent gegenüber dem einmaligen Einlesen. Wenn stattdessen der Mittelwert aus 16 Messungen gespeichert wird, verbessert dieses die Erkennungssicherheit praktisch nicht mehr. Das Bilden eines solchen Durchschnittsmusters ist jedoch nur bei einer linearen Zeitnormierung möglich. (siehe 3.3.4.)
Ich habe auch versucht, die Erkennungssicherheit durch ein Nachführen der Wortmuster in der Referenztabelle zu erhöhen. Dieses habe ich bei meinen Versuchen dadurch erreicht, daß das Referenzmuster des erkannten Wortes durch ein Muster ersetzt wurde, das sich zu 3/4 aus dem alten und zu 1/4 aus dem gerade gesprochenen zusammensetzte. Zusätzlich habe ich eine Maximalgrenze für den Fehler beim Erkennen benutzt, die etwa das 1,5-fache des durchschnittlichen Fehlers betrug und genauso wie das Referenzmuster nachgeführt wurde. Trotz dieser Grenze stellte sich beim Erproben heraus, daß teilweise schon ein nicht richtig erkanntes Wort genügte, um das Referenzmuster zu verfälschen. War das einmal passiert, "driftete" dieses Muster schnell weiter ab. Es wurde also keine Erhöhung der Erkennungssicherheit sondern gerade das Gegenteil erreicht.

3.3.4. Erkennungsverfahren

Sobald in der Tabelle der Referenzmuster die zu erkennenden Wörter gespeichert sind, kann ein Wort erkannt werden. Dazu wird es zunächst eingelesen und spektral analysiert. Daraufhin muß das Wortmuster mit jedem Muster in der Referenztabelle verglichen werden. Bei der linearen Zeitnormierung genügt es, wenn man dabei jeweils alle korrespondierenden Werte der Wortmuster miteinander vergleicht, indem man die Summe der Beträge der Differenzen berechnet. Diese Fehlersummen werden dann in einer anderen Tabelle für jedes Referenzmuster gespeichert. Das Wort mit der geringsten Fehlersumme gilt als erkannt.
Da mich die Erkennungsicherheit bei diesem System des Mustervergleiches jedoch nicht zufriedenstellte, habe ich nach Verbesserungsmöglichkeiten gesucht. Eine Verbesserung besteht darin, daß ich statt der Absolutwerte der Differenzen ihre Quadrate aufsummiere. Um dieses schnell durchführen zu können, verwende ich eine Tabelle mit Quadraten. Hiermit konnte ich die Erkennungssicherheit um einige Prozent verbessern, da bei diesem System kleine Abweichungen nicht stören, große Differenzen die Fehlerquadratsumme jedoch erheblich erhöhen. Versuchsweise habe ich auch eine anderen Tabelle benutzt, in der für kleine Werte die Quadrate, für große Werte dagegen ein fester Maximalwert gespeichert war. Die Erkennungssicherheit dieses Systems war jedoch etwa gleich der des Systems ohne Quadrierung.

Das bei kommerziellen Spracherkennungssystemen oft verwendete Prinzip des dynamischen Mustervergleichs habe ich nicht erprobt, da es zum einen sehr schwierig zu programmieren ist und auf jeden Fall zu erheblichen Verlängerungen der Erkennungszeit meines Spracherkennungssystemes führen würde. Bei diesem Prinzip werden nicht einfach starr bestimmte Zeitabschnitte verglichen, sondern es werden die zu vergleichenden Wortmuster abschnittsweise so gestreckt oder gestaucht, daß jeweils zusammengehörende Wortteile oder Laute miteinander verglichen werden. (siehe Skizze unten) Ich glaube auch, daß der dynamische Mustervergleich die Erkennungssicherheit nicht mehr wesentlich erhöhen könnte, da ich wohl an die Grenzen der von meiner einfachen Hardware verfügbaren Informationsmenge gelangt bin.

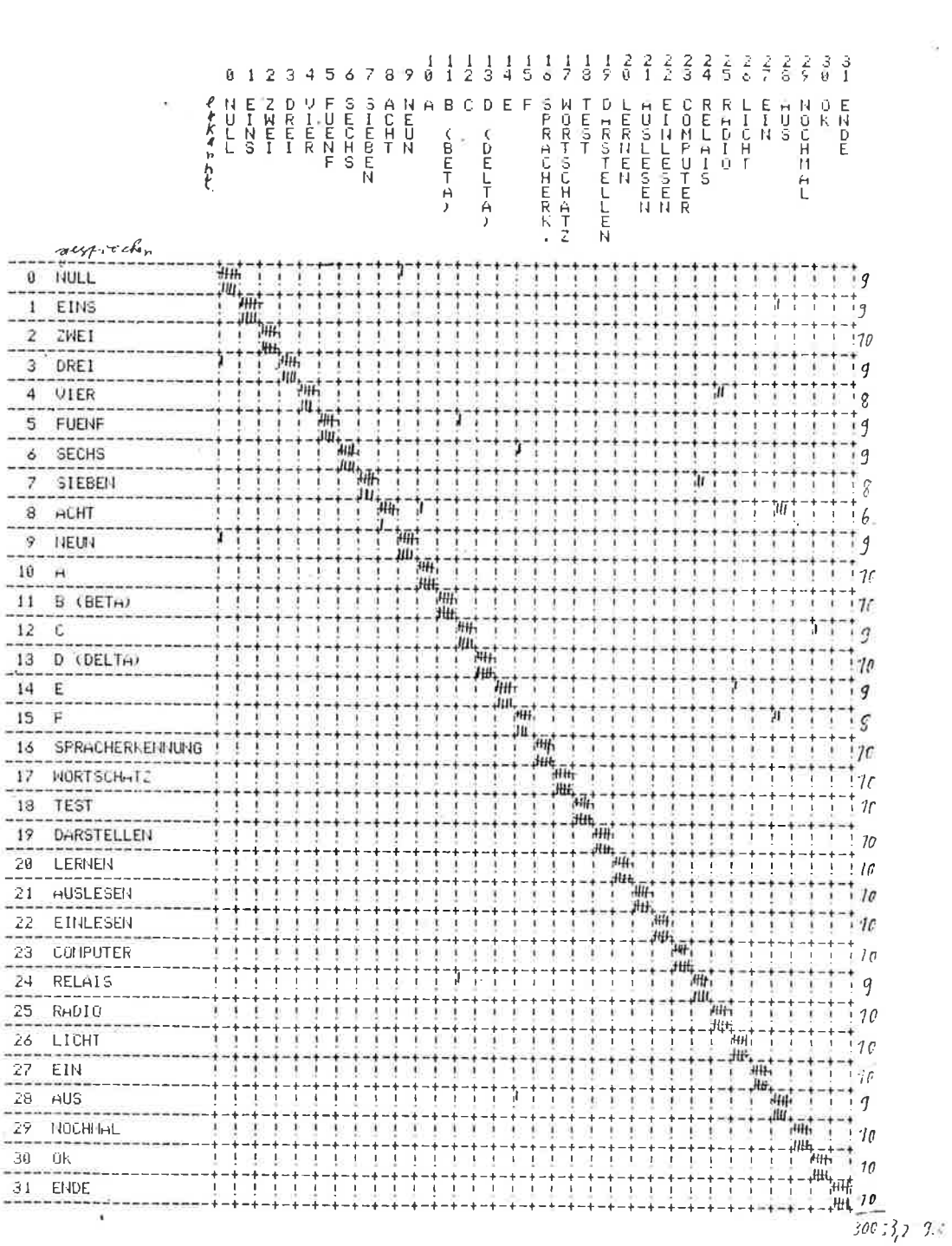


3.4. Abschließende Betrachtung der Spracherkennung

Zum Vergleich der unterschiedlichen Erkennungssyteme muß ich die Erkennungssicherheit messen können. Ich benutze dafür einen festen Wortschatz, der aus 32 Wörtern besteht. Diese Wörter habe ich so ausgewählt, daß man diesen Wortschatz auch in einer Anwendung benutzen könnte. So sind z.B. die Ziffern 0 bis 9 und häufige Kommandowörter wie "OK" und "Nochmal" enthalten. Bei der Messung spreche ich dann 10 mal hintereinander den gesamten Wortschatz klar und deutlich. Dabei notiere ich in einer Tabelle, welches Wort das Spracherkennungssystem erkannt hat. Die Erkennungssicherheit ist dann das Verhältnis der richtig erkannten Wörter zu den insgesamt 320 gesprochenen Wörtern. Ein solches Diagramm, aus dem man die Häufungen der falsch erkannten Worte ablesen kann, befindet sich auf S. 10.
Die beste Erkennungssicherheit, die ich bisher gemessen habe, lag bei 93,8%. Solche Angaben sind natürlich mit einem gewissen statistischen Fehler behaftet. Mehrmalige Messungen mit dem gleichen Erkennungsprinzip haben gezeigt, daß dieser Fehler etwa +/- 1 % beträgt. In der gleichen Weise unterliegen auch die gelernten Referenzmuster einer gewissen Streuung. Ich habe aber bei normaler Aussprache und 32 möglichen Worten auf jeden Fall eine Erkennungssicherheit von mehr als 90 %. Diese läßt sich natürlich durch die Wahl eines anderen Wortschatzes mit Wörtern, die sich untereinander stärker unterscheiden, erhöhen. Mit meinem Wortschatz kann ich jedoch auch das Verhalten bei ähnlich klingenden Wörtern untersuchen.

Noch einfacher könnte ich die Erkennungssicherheit durch eine Verkleinerung des Wortschatzes erhöhen. So wird auch klar, welche Aussagekraft z.B. eine Angabe "98 %" bei kommerziellen Spracherkennungssystemen hat. Dort wird nämlich nur selten der benutzte Wortschatz genauer beschrieben.

Für die Erkennung eines Wortes werden im Durchschnitt etwa 0,3 Sekunden benötigt. Der größte Teil hiervon wird für die spektralen Analyse gebraucht. Insgesamt erreiche ich mit einer minimalen Hardware und relativ einfacher Software so eine Erkennungssicherheit von mehr als 90 % bei einer Erkennungszeit von etwa 0,3 Sekunde. Diese Ergebnis halte ich auf jeden Fall für akzeptabel.



93,8 %

4. 1-Bit-Sprachsynthese

Ich habe auch untersucht, ob man mit einer 1-Bit-Sprachsynthese gut verständliche Sprache erzeugen kann. Dazu habe ich aus den gespeicherten Daten wieder das ursprüngliche 1-Bit-Sprachsignal erzeugt und ausgegeben. Dieses wird über einen Bandpaß einem Leistungsverstärker (LM 386) zugeführt, an dessen Ausgang ein Lautsprecher angeschlossen ist.
Die so erzeugte Sprache ist allerdings nur schwer verständlich. Besonders schlecht werden leise Laute (stimmlose Konsonanten) ausgegeben. Beispiele hierfür sind Worte wie "Hallo" oder "Vier", bei denen jeweils der erste Buchstabe fast vollständig verschluckt wird.
Bei der praktischen Anwendung stört auch der große Speicherbedarf von 2 bis 4 K Byte pro Wort sehr. Daher ist eine 1-Bit-Sprachsynthese wenig sinnvoll, obwohl sie etwa als akustische Rückmeldung eines erkannten Wortes durchaus wünschenswert wäre. (siehe 6.1)

5. Benutzung des ZX Spectrum

Meine allerersten Versuche mit der Sprachein- und -ausgabe habe ich auf einem Sinclair ZX Spectrum gemacht. Er besitzt einen Z80 und etwa 40 K Byte freien Speicher. Die Sprache habe ich über das eingebaute Kassetteninterface eingelesen und dann über den Lautsprecher ausgegeben, der auch schon eingebaut ist. Die ausgegebene Sprache war aber nur mit Mühe erkennbar.
Da ich mich über die schlechte Verständlichkeit wunderte und mich deren Ursache interessierte, habe ich diese Programme auf ein anderes Mikrocomputersystem übertragen. Der Grund hierfür war, daß mir für den ZX Spectrum im Gegensatz zu dem Mikrocomputersystem kein geeigneter Assembler und Monitor zur Verfügung standen. Für das Mikrocomputersystem mußte ich die Hardware zur Sprachein- und -ausgabe jedoch selber bauen. Ich habe sie in 3.2. und 4. beschrieben.
Nachdem die Spracherkennungsprogramme fertig waren, wollte ich sie auch auf den ZX Spectrum übertragen. Dazu mußte ich zuerst ein serielles Interface mit den dazugehörenden Programmen entwickeln. Mit diesem Interface erreiche ich eine übertragungsgeschwindigkeit von etwa 4,8 K Baud. Nun konnte ich die notwendigen Terminalprogramme schreiben, die ich für die Zeicheneinund -ausgabe auf dem ZX Spectrum benötigte. Sie sind etwa 1 K Byte lang und benutzen viele Unterprogramme des Spectrum-ROMs. Als diese Terminalprogramme ferig waren, konnte ich endlich die nur geringfügig geänderten Programme zur Spracherkennung übertragen.
Beim Testen der Programme stellte sich heraus, daß man das Kassetteninterface leider nicht anstelle des Schmitt-Triggers verwenden kann, da die maximale Dauer einer High-Periode hardwaremäßig auf etwa 0,5 ms begrenzt ist. Im übrigen verhält es sich aber etwa wie ein Schmitt-Trigger und benötigt Eingangsspannungen von > 4 Vss. So benutze ich jetzt meine in 3.2. beschriebene Schaltung, die ich über ein einfaches selbstgebautes Eingabeinterface, das aus drei Dioden und einem Tri-State-Treiber (74LS244) besteht, an den ZX Spectrum anschließe. Die Ausgabe über den Spectrum-Lautsprecher erwies sich als problemlos.
Ich bin jetzt also in der Lage, mit dem relativ weit verbreiteten Home-Computer Sinclair ZX Spectrum und nur etwas zusätzlicher Hardware, Spracherkennung mit einer für viele Anwendungen sicher akzeptabelen Erkennungsicherheit zu betreiben. Die Benutzung dieser Programme von BASIC aus erwies sich als nicht schwierig. (siehe 6.2.)

6. Anwendung der Spracherkennung

Nachdem ich das Spracherkennungssystem fertiggestellt hatte, wollte ich es auch praktisch anwenden. Bei solchen Anwendungen verwendet man häufig eine Kommandosprache. Zur Erhöhung der Erkennungssicherheit ist es sinnvoll, die Syntax der Kommandosprache in die Spracherkennung mit einzubeziehen.

6.1. Vorüberlegungen und Syntax-Konzept

Um die Verwendbarkeit in der Praxis zu untersuchen, habe ich als mögliche Anwendung eine sprachgesteuerte Gerätesteuerung entworfen. Mit ihr kann man mehrere Geräte ein- und ausschalten bzw. sie in verschiedene Stufen schalten. (siehe 6.2.) Dieses Problem läßt sich am einfachsten mit Kommandos lösen, die nur aus einem Wort bestehen.
Bei dem von mir gewählten Beispiel (siehe 6.2. und Syntax-Diagramm auf S. 14) wären dann 19 Kommandoworte nötig. Die Erkennungssicherheit wäre aber nicht zufriedenstellend. So habe ich für diese Anwendung eine Kommandosprache entwickelt, bei der ich insgesamt 15 Wörter benötige. Ein vollständiges Kommando besteht dabei aus 2 bis 5 Wörtern. Diese Kommandosprache hat eine feste Syntax, es sind also nur bestimmte Wortfolgen zugelassen. So kann ich die Anzahl der jeweils sinnvollen Wörter bei der Erkennung auf maximal 7 reduzieren. Das ist weniger als die Hälfte der bei den 1-Wort-Kommandos möglichen Wörter. Dies kann ich bei der Erkennung dadurch ausnutzen, daß ich dort immer nur diese Wörter zulasse. So verringere ich den die Erkennungsicherheit beeinflussenden Wortschatz auf durchschnittlich 4 Wörter. Zusätzlich habe ich die Möglichkeit realisiert, ein falsch erkanntes Wort zu korrigieren ("Nochmal"). Die Sprachsteuerung findet jetzt also in einem Dialog mit dem Computer statt.
Im einfachsten Fall kann die Rückmeldung des erkannten Wortes auf dem Bildschirm geschehen. Wesentlich interessanter wäre natürlich eine akustische Rückmeldung. Diese ist jedoch mit dem Prinzip der 1-Bit-Sprachsynthese nicht vernünftig möglich. (siehe 4.)
Beim praktischen Experimentieren hat sich nun herausgestellt, daß man, während die Sprachsteuerung eingeschaltet ist, nur Kommandos sprechen darf, da man sonst ungewollte Befehle auslösen würde. Da so die Sprachsteuerung praktisch unbrauchbar wird, habe ich eine Möglichkeit eingebaut, sie über ein spezielles Kommando ("Einschlafen") in einen Wartezustand zu versetzen. Aus diesem Wartezustand kommt man nur heraus, wenn man zwei bestimme Wörter hintereinander spricht. Damit auch das nicht aus Versehen geschieht, wende ich hier das Syntax-Konzept nicht an. Ich lasse beim "Aufwachen" also den gesamten Wortschatz zu.

6.2. Steuerung von Geräten

Ein sinnvolle Anwendung ist die oben schon kurz beschriebene Steuerung von Geräten. In meinem Beispiel kann ich 6 Geräte ein- und ausschalten und bei einem siebten zusätzlich noch zwischen 3 verschiedenen Stufen wählen. Um die Syntax der hierbei verwendeten Kommandosprache, deren Syntax-Diagramm sich auf S. 14 befindet, einfach zu implementieren, habe ich hierfür einen Kommandointerpreter geschrieben. Diesem Interpreter liegt die Syntax in Form einer Tabelle vor, so daß ich auch ohne Schwierigkeiten eine andere Kommandosprache implementieren könnte. Ein vollständiges Kommando in meiner Sprache lautet z.B.: "Licht Eins Intensitaet Zwei OK".

Diesen Kommandointerpreter habe ich auf dem Mikrocomputersystem in Assembler geschrieben. Den Zustand der gesteuerten Geräte kann ich an 8 Leuchtdioden auf einem Ausgabeinterface ablesen. An die Ausgänge dieses Interfaces könnte ich einfach Relais anschließen, und so die Geräte steuern. Auf dem Bildschirm werden dabei jeweils die erkannten Wörter eines Kommandos in einer Zeile dargestellt, so daß man kontrollieren kann, ob auch das richtige Wort erkannt wurde.
Ein Programm für diese Gerätesteuerung habe ich auch auf dem ZX Spectrum geschrieben. Ich habe dabei BASIC verwendet, um zu zeigen, wie einfach die Programme meines Spracherkennungssystemen von BASIC aus zu benutzen sind. Bei diesem Programm werden die Zustände der Geräte übersichtlich auf dem Bildschirm dargestellt. Zusätzlich habe ich alle Wörter, die man jeweils sprechen darf, auf dem Bildschirm dargestellt. So kann ich also in einem menügesteuerten Dialog mit dem Computer arbeiten. (siehe Fotos S. 14)

6.3. Abschließende Betrachtung der Anwendungen

Die Gerätesteuerung über Sprache hat gezeigt, daß man ein solches einfaches Spracherkennungssystem, wie ich es in dieser Arbeit beschrieben habe, in der Praxis durchaus verwenden könnte. Dadurch, daß ich die Syntax der Kommandosprache, mit der ich die Geräte steuere, mit in die Erkennung einbeziehe, kann ich die Erkennungssicherheit gegenüber einem System mit 1-Wort-Kommandos erhöhen. Problematisch bei einer solchen praktischen Anwendung wäre allerdings wieder die Befestigung des Mikrofons am Kopf. (siehe 3.2.1.) Man könnte es statt dessen in der Hand halten, sollte dabei aber beachten, daß man es nicht genau vor den Mund sondern etwas seitlich davon hält.
Mit einer solchen Sprachsteuerung könnte man eventuell Behinderten die Bedienung von Geräten erleichtern. Bei solchen Anwendungen wäre jedoch eine akustische Rückmeldung der erkannten Worte sinnvoll.

Jedoch dürfen bei allen Anwendungen eines Sprache erkennenden Computers die damit verbundenen Gefahren nicht übersehen werden.

7. Literaturverzeichnis

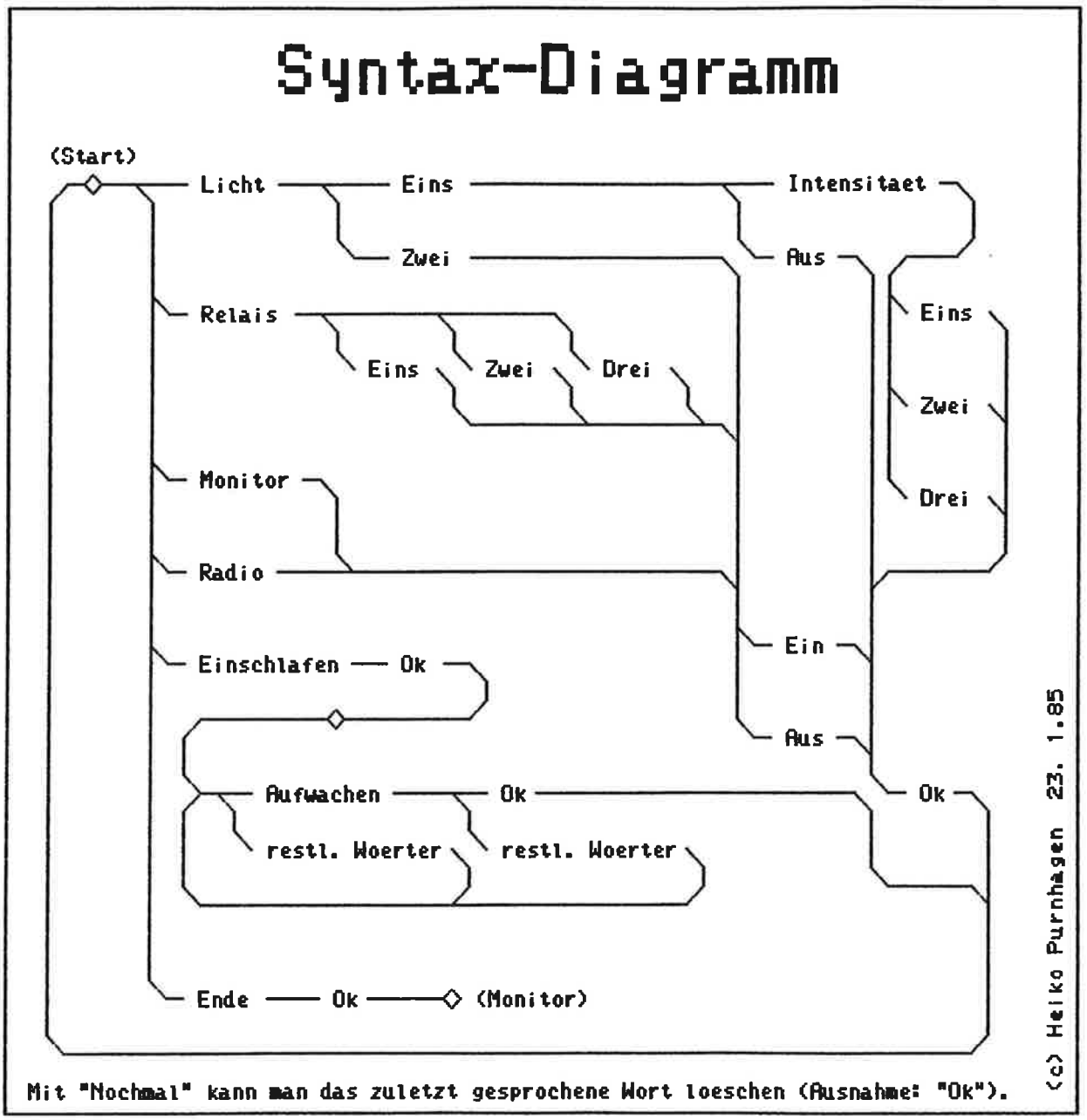
1) Datenblatt: ICL 7611 DCPA, Intersil, 1979

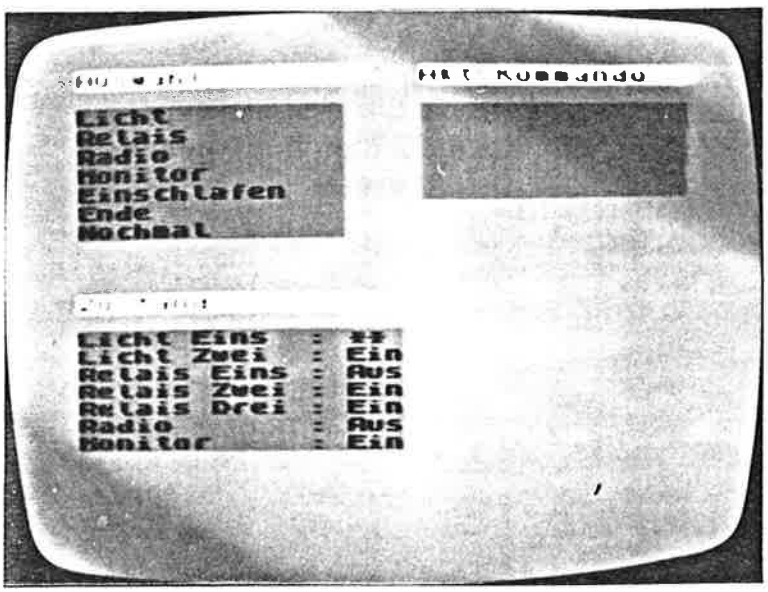
2) Beschreibungen des benutzten Mikrocomputersystems

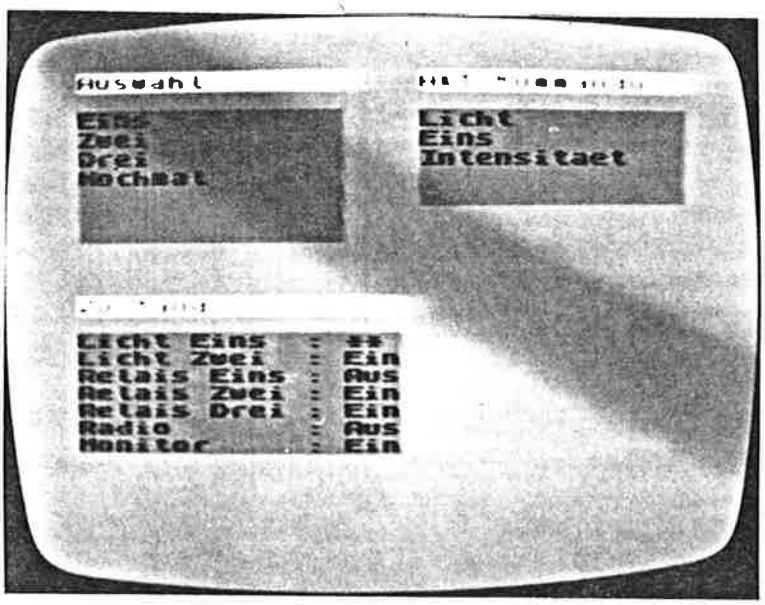
3) Beschreibungen und ROM-Listing des ZX Spectrums

4) JUFO-Arbeit: "Entwicklung eines Mikrocomputersystems mit Hard- und Software zur Spracherkennung und Sprachsynthese", Dirk Graudenz, 1984

5) Aufsatz: "Voice recognition", M. H. Kuhn, European Electronics, Nr. 6, 1981 und Nr. 1, 1982


Syntax-Diagramm
(Start)
Licht
Eins
Zwei
Relais
Eins
Zwei
Drei
Monitor
Radio
Einschlafen
Ok
Aufwachen
Ok
restl. Woerter
restl. Woerter
Ende
Ok
(Monitor)
Intensitaet
Aus
Eins
Zwei
Drei
Ein
Aus
Ok
(c) Heiko Purnhagen 23. 1.85
Mit "Nochmal" kann man das zuletzt gesprochene Wort loeschen (Ausnahme: "Ok").


Licht
Relais
Radio
Monitor
Einschlafen
Ende
Nochmal
Licht Eins : ##
Licht Zwei : Ein
Relais Eins : Aus
Relais Zwei : Ein
Relais Drei : Ein
Radio : Aus
Monitor : Ein


Auswahl
Eins
Zwei
Drei
Nochmal
Licht
Eins
Intensitaet
Licht Eins : ##
Licht Zwei : Ein
Relais Eins : Aus
Relais Zwei : Ein
Relais Drei : Ein
Radio : Aus
Monitor : Ein