INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Remote control of a robotic arm through speaker-dependent, isolated-word speech recognition

Atkinson, William Thomas, M.S.

Rice University, 1991
REMOTE CONTROL OF A ROBOTIC ARM THROUGH SPEAKER-DEPENDENT, ISOLATED-WORD SPEECH RECOGNITION

by

WILLIAM THOMAS ATKINSON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE MASTER OF SCIENCE

APPROVED, THESIS COMMITTEE

John B. Cheatham Jr., Professor
Department of Mechanical Engineering and Materials Science
Director

Angelo Miele, Professor
Department of Mechanical Engineering and Materials Science

Ian D. Walker, Assistant Professor
Department of Electrical and Computer Engineering

Houston, Texas
February, 1991
Abstract

REMOTE CONTROL OF A ROBOTIC ARM THROUGH SPEAKER-DEPENDENT, ISOLATED-WORD SPEECH RECOGNITION

William Thomas Atkinson

The focus of this thesis is the remote control of a six degree-of-freedom mid-range industrial robot arm through the use of speech recognition. The robot arm is controlled through speaker-dependent, isolated-word speech recognition. Template matching techniques are used to interpret verbal input and make the robot arm perform a desired task. The user inputs one- or two-word commands to interact verbally with the robot. The verbal input is matched against a vocabulary of known commands for recognition. Once the command is interpreted, the corresponding subroutine is run and the robot arm performs the task. The movement of the arm, to include the manipulation of its gripper, is accomplished in discrete increments. The program is designed to allow for multiple users and it can be modified to perform robotic control for other tasks.
Acknowledgements

I would like to express my sincere appreciation to Dr. John B. Cheatham, Jr. for his helpful advice and patience in directing my research. I am also deeply indebted to Sarmad Adnan, Jack Chen, Kevin Magee, John Norwood, and Peter Weiland who spent many hours helping me refine concepts and debug program code. I want to give a special thanks to Christopher E. Schuster for being a good friend and for encouraging me throughout my research, especially on those days when nothing seemed to go right. Finally, I wish to thank my wife Beverly, daughter Nicole, and son Justin who always encouraged and supported me and without whom this research and thesis would not have been possible.

I would like to acknowledge the support obtained from NASA/Johnson Space Center through the Research Institute for Computing and Information Systems (RICIS), Contract: 9-16, and NASA/JSC Grant NAG-9-372.

I also wish to thank the U.S. Army for its support and the opportunity to attend Rice University.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td><strong>Chapter</strong></td>
<td></td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Background</td>
<td>5</td>
</tr>
<tr>
<td>Speech Recognition System Considerations</td>
<td>5</td>
</tr>
<tr>
<td>Isolated-Word Recognition</td>
<td>7</td>
</tr>
<tr>
<td>Equipment</td>
<td>11</td>
</tr>
<tr>
<td>3. Development of the Speech Recognition System</td>
<td>15</td>
</tr>
<tr>
<td>Speech Recognition Software</td>
<td>16</td>
</tr>
<tr>
<td>Robot Control Software</td>
<td>19</td>
</tr>
<tr>
<td>4. Implementation of the Speech Recognition System</td>
<td>23</td>
</tr>
<tr>
<td>Set-up and Start-up</td>
<td>23</td>
</tr>
<tr>
<td>Running the Speech Recognition Program</td>
<td>25</td>
</tr>
<tr>
<td>5. Tests and Results</td>
<td>32</td>
</tr>
<tr>
<td>Recognition Tests</td>
<td>32</td>
</tr>
<tr>
<td>Control Tests</td>
<td>33</td>
</tr>
<tr>
<td>Other Robot-Control Tests</td>
<td>35</td>
</tr>
<tr>
<td>6. Conclusions</td>
<td>36</td>
</tr>
<tr>
<td>7. Areas For Future Work</td>
<td>38</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>40</td>
</tr>
<tr>
<td><strong>Appendixes</strong></td>
<td></td>
</tr>
<tr>
<td>A. spkrdep.c Source Code Listing</td>
<td>43</td>
</tr>
<tr>
<td>B. Subroutine robot_sim</td>
<td>58</td>
</tr>
<tr>
<td>C. rowbot.c Source Code Listing</td>
<td>62</td>
</tr>
<tr>
<td>D. spkrdep.h Header File</td>
<td>70</td>
</tr>
<tr>
<td>E. global.h Header File</td>
<td>71</td>
</tr>
<tr>
<td>F. utili.h Header File</td>
<td>72</td>
</tr>
<tr>
<td>G. commander.bat Code Listing</td>
<td>75</td>
</tr>
</tbody>
</table>
H. slave.c Source Code Listing .......................... 76
I. rtx3.c Source Code Listing .......................... 78
J. rtx.h Header File .................................... 91
K. Diagram of the RTX Robot ............................ 93
Tables

Table | Description | Page
---- | ----------- | ----
1. | List of Vocabulary Words for the System | 15
Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Block Diagram of the Speech Recognition System</td>
<td>4</td>
</tr>
<tr>
<td>2.</td>
<td>Template Matching System</td>
<td>8</td>
</tr>
<tr>
<td>3.</td>
<td>Effects of Sampling Rates in an Analog Waveform</td>
<td>9</td>
</tr>
<tr>
<td>4.</td>
<td>Template Matching Operations</td>
<td>10</td>
</tr>
<tr>
<td>5.</td>
<td>Speech Recognition System Communication Connections</td>
<td>23</td>
</tr>
<tr>
<td>6.</td>
<td>Flowchart for spkrdep.c</td>
<td>26</td>
</tr>
<tr>
<td>7.</td>
<td>Recognition of the Word &quot;Down&quot;</td>
<td>30</td>
</tr>
<tr>
<td>8.</td>
<td>Switching Among Multiple Users</td>
<td>31</td>
</tr>
<tr>
<td>K1.</td>
<td>Diagram of the RTX Robot</td>
<td>94</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

In the world of computers and robotics, the ultimate user-friendly system would be one with complete voice communications abilities. Consider the advantages of such a system: freedom to conduct hands-free and eyes-free operations, no need for the user to divert his attention from the main task, no special skills or knowledge needed by the user. Already in the world around us, we can see evidence of systems that use speech synthesis, speech recognition, or both: automobiles that tell us when our fuel level is low or when an intruder has entered a security zone around the exterior of the car, vending machines and cash registers that tell us how much change we will receive, baby dolls that respond to a human voice, even Artificial Intelligence systems that control household appliances and light switches or are used for air traffic control in major metropolitan airports. Imagine the areas where voice communications with machines would be beneficially applicable:

Household appliances could be controlled through verbal commands from the owner. For example, the user could turn on the television set, change channels, and control the volume, without having to touch the television or use a handheld remote control. There are many other possibilities, such as control of light switches, thermostats, telephone answering machines, coffee makers, microwave ovens, and so forth. Additionally, the user gains more voice-control as more home appliances are centrally controlled [1].

Similar applications could be employed in offices. Also, voice-control would be beneficial in word-processing and data-based applications. Imagine an executive being able to dictate a letter directly to a word processor that immediately types the letter and reads it back at the user's request. Think of the convenience that a store manager would have by being able to verbally record or check the day's business transactions [1, 4, 30].

Industry could use such a system to keep track of inventories without having a clerk spend hours typing data into a computer, to replace the need for keyboard input for computer-controlled equipment, or to enable inspectors on a production
line to record faults with an item while still holding the item for further inspection [1].

Verbal interface between man and machine in transportation is another area of application. A pilot must monitor the instruments and dials in the cockpit, while at the same time he is using his hands and feet to control the flight of the aircraft. Some of the more routine functions, such as the raising and lowering of landing gear, could be controlled by verbal commands. Speech recognition can also be used to assist air traffic controllers by providing commonly requested information to pilots [6]. A voice-controlled system could also benefit the driver of a motor vehicle. The driver could operate the windshield wipers or turn signals without having to release the steering wheel or divert his attention from the traffic around him.

Consider the possible applications in the area of education. Machines with voice interaction capabilities could be used to help teach spelling, arithmetic, foreign languages, and reading skills. This would be particularly beneficial to those who are illiterate, because it would provide them with a relatively fast method of learning while removing any fear of the keyboard [28, 32].

The handicapped could also benefit from speech recognition systems. Someone with paralyzed limbs could verbally control household appliances, as discussed earlier, as well as mobile robots or robot arms designed to add to the comfort of the user [18].

Voice communication systems could also be applied to the medical field, to military operations, and even to our space program. The possibilities seem endless and the potential unlimited.

At Rice University, the Mechanical Engineering Robotics Group uses a Texas Instruments (TI) Speech Application Toolkit to introduce speech synthesis and recognition to Introductory and Advanced Robotics students. The objective of this thesis is to develop speech recognition software that uses this TI-Speech system for the remote control of a robot. The research for this thesis was divided into phases. The first was the development of code to allow a user to create, train, refine, and test a vocabulary that could be recognized by the computer. This code had no immediate application except to demonstrate the speech recognition capabilities of the TI-Speech system and to provide a foundation for robot-control subprograms. The second phase of the research consisted of writing the subprogram "robo_sim," and associated code, to enable the user to manipulate the six-
degree-of-freedom mid-range industrial robot arm (RTX). This software shows a practical application of the overall speech recognition code; it can be replaced with software written to control a different robot for different tasks.

The following characteristics had to be considered: speaker-dependence or speaker-independence and isolated-word recognition or continuous speech understanding. A speaker-dependent, isolated-word recognition system was selected for a number of reasons. It is a relatively accurate system that is easily implemented with the hardware available. Although the computer will only respond to one user per vocabulary, this characteristic also serves as a safety factor that prevents the robot arm from improperly reacting to noise or to another person’s voice and performing an unwanted act.

The system used to implement the software consists of two IBM PC/XT microcomputers, the TI-Speech hardware, a microphone and speaker, and the RTX robot and its software. The RTX is typically controlled through keyboard commands; this system uses speech commands entered through a microphone to control the RTX. Figure 1 depicts a block diagram of this voice recognition system. After the user utters the speech command, the input is processed by the software on an IBM PC that has the TI-Speech board. Because the software used to control the RTX is incompatible with TI-Speech hardware, a second IBM PC is used to transport the processed input to interface with the RTX via RS-232 serial ports. This input is then processed by software used by the RTX and the desired task is performed. Notice that a speaker is also used for some voice synthesis operations (i.e., output), but the use of the speaker is not directly related to the task of controlling the robot arm. Therefore, references will be made to some speech synthesis processes that are used for audio-feedback from the computer, but this thesis will concentrate only on the speech recognition operations.
Figure 1

Block Diagram of the Voice Recognition System
CHAPTER 2
Background

As in any endeavor, before designing a voice-communications system between man and machine, it is important to first analyze the task at hand. While a more complex system may appear to be more elegant and intellectual, it may not be the optimal system based upon its application and task specifications. For example, why create a system capable of recognizing several hundred words when just a few words would be adequate to do the job? The larger the vocabulary, the more costly the system. This cost can be measured in the amount of memory needed to store the vocabulary and its corresponding code, time required for the system to run the software and respond to the input, and man-hours needed to write a more complex code.

In addition to the size of the vocabulary, it is necessary to keep the following parameters in mind: speaker dependence/independence, isolated-word versus connected-speech, and recognition versus understanding [23, pg. 170].

Speech Recognition System Considerations

Who will use the system will influence whether it is more prudent to design a speaker-dependent or a speaker-independent system. For example, in an operation in which a handicapped person or an astronaut controls a tele-operated robot, there are only a few users. But some systems, like a voice-controlled automatic teller at a bank, would have many users. A speaker-dependent system is better suited for operations with few users. This system requires that each user train the system to recognize each word that is
available in the vocabulary. Each word must then be refined to compensate for noise and differences in the utterance of the same word by the same person; these differences may be the result of a number of things, such as changes in intonation or pitch. Training the system for each user is time-consuming, but speaker-dependent systems are frequently used because they are simple, practical, and more economical, and they are very accurate in recognizing words trained by the user. But, as stated earlier, there are times when a speaker-independent system is more appropriate. A speaker-independent system does not require training and will respond to the voice input of anyone. While this system is more convenient for the user, it requires more complex software and is limited to a relatively smaller vocabulary due to the greater amount of memory that must be used for the computer code to run it. Ultimately, it is up to the designer to determine which system is most appropriate for the specified task.

Another consideration is whether the system will be required to recognize isolated words or understand connected words. As the name implies, an isolated-word recognition system recognizes single word utterances from the user. A connected-word understanding system receives an utterance, recognizes the individual words, and uses Artificial Intelligence (AI) techniques to "connect" the words into a meaningful phrase or sentence. This is second-nature for humans, but it is complicated for machines. We often take for granted what we must do naturally in order to understand what other people say. We must be able to determine if a change in sound or a pause is a new syllable for a word or a break between words. Additionally, we must be able to recognize a word even when it is pronounced differently because of the words around it or its context in a sentence. These requirements make connected-word understanding a very challenging undertaking.
Additionally, it is important to distinguish between recognition and understanding. Recognition implies comparing voice input with trained words until an acceptable match is found. Speech understanding, on the other hand, implies that there must be some knowledge of sentence structures or language patterns and that this knowledge must be employed to interpret the meaning of the voice input. Therefore, AI techniques are more often used for speech understanding than they are for speech recognition.

Isolated-Word Recognition

As stated in Chapter 1, there were a number of reasons why a speaker-dependent, isolated-word speech recognition system was selected, such as accuracy and ease of implementation with the TI hardware. The major driving force in the decision to use this type of system was an analysis of the objective to control the six degree-of-freedom robot arm. The number of people who would use the system would be limited. The users would include the Rice University Robotics Research Group and various Introductory and Advanced Robotics laboratory groups. This small number of potential users made it more practical to develop a speaker-dependent system. A vocabulary of only a few isolated words would be sufficient to control the RTX, therefore there was no need to develop a system that would use connected words to interpret the meaning of commands. Additionally, since the system was developed with instructional purposes in mind, it was desirable to have a system that would require/allow students to train and test their own speech recognition systems. These specifications indicated that a speaker-dependent, isolated-word speech recognition system would be the most practical system. Therefore, a more detailed look at speaker-dependent, isolated-word systems is in order.

System Design [23]. As stated earlier, speaker-dependent systems are very accurate. They can recognize large vocabularies (over one hundred words) with 91 to 99 percent
accuracy. Speaker-dependent, isolated-word recognition systems work by trying to match voice input patterns with already trained patterns that are stored in the computer's memory. These stored patterns are also known as templates. This system is also referred to as template matching. Figure 2 shows a block diagram of a typical template matching system.

![Figure 2: Template Matching System](image)

A microphone is used as the input device. Although it may seem to be a minor component of the system, it is important that the microphone be able to remove as much environmental noise from the input signal as possible. After the user speaks into the microphone, the signal is amplified for easier analysis and is filtered to further reduce noise. Since human speech usually ranges from 150 Hz to 3600 Hz, the amplifier should produce a corresponding frequency range. The signal is analyzed by the sample/hold, to distinguish the speech waveform's characteristics. The greater the sampling rate, the better the quality of the signal sent to the A/D converter. Figure 3 shows the effect of different sampling rates on an analog waveform; note how much more closely the "high" rate resembles the actual input waveform. To catch all the unique characteristics of a waveform, it must be sampled 30,000 times per second. However, this would require too much computer
memory. Therefore, the sampling rate must be less than 30,000 times per second, but it must still be large enough to output a fairly accurate waveform. A good rule-of-thumb is to use a sampling rate that is at least twice the highest frequency of the amplified signal, in this case $2 \times 3600 \text{ Hz} = 7200 \text{ Hz}$. Typically, sampling rates of 8000 to 12,000 samples per second are used; the system used in this research collects 8000 8-bit samples per second.

![Graphs showing effect of sampling rates on an analog waveform.](image)

**Figure 3**

**Effect of Sampling Rates on an Analog Waveform**

Since the human speech waveform is analog, it must be converted to digital form before it is processed by the system software. This is done by the A/D converter. Experience has
shown that an 8-bit converter produces a signal of good quality without requiring a large amount of computer memory.

The template matching system has two distinct phases: training and recognition. Figure 4 shows a block diagram of the template matching operations.

![Block Diagram]

**Figure 4**
Template Matching Operations

**Training Mode** [23]. In any speaker-dependent system, the user must train the system. The user must speak each vocabulary word, using a microphone as the input device. Each
word is entered more than once, usually 3 - 5 times, for refinement to increase the accuracy of the recognition properties. The software analyzes the acoustic characteristics of each word, generates a template for the word, and stores the pattern in the RAM. These stored patterns are used for matching with input patterns during the recognition mode.

**Recognition Mode [23].** As in the training mode, the input signal must be acoustically analyzed in the recognition mode. This code is stored in matrix form for comparison against the trained, or reference, pattern codes. The unknown patterns are then compared against the reference patterns, element-by-element. Once a match has been found, the computer will respond with an appropriate response. This response could be in the form of verbal feedback. An example of this could be a voice-operated-computer spelling tutor that says, "That's right," when the user gives a correct answer or, "Try again," when an incorrect answer is given. Other responses could be the lowering of a thermostat, printing-out a store room inventory, or, as in the case of this research, moving a robot arm.

The template matching can be done in one of two manners. In the first method, the unknown template is compared against all reference templates. The best-match is then considered to be the correct reference template. In the second method, the unknown pattern is systematically compared against the reference patterns until a "match" is found. Once a match has been made, within error constraints, the input is considered to be recognized and the search is discontinued. The latter method is used in this research.

**Equipment**

The two primary pieces of equipment that are of concern to this research are the RTX robot and the TI-Speech system. Detailed information can be found in the manuals.
that are part of the equipment packages, but it is helpful to have a general understanding of each.

**RTX Robot [25, 26, 27].** The RTX is a mid-range industrial robot. A picture of the RTX robot is found in Appendix K. The RTX has six degrees of freedom in a cylindrical configuration. The joints can be independently controlled and each is equipped with a DC servo which is made-up of a DC motor and an optical encoder. The gripper is also equipped with a DC servo. The robot may be controlled through an IBM PC or compatible computer. The robot is connected to the controlling computer through an RS-232 serial link. The software supplied with the robot allows the user to program in FORTH, PASCAL, or C. The independent manipulation of each joint can be used to accomplish simple robotic tasks involving "pick-and-place" routines and other similar activities. In order to use the robot in world space, inverse kinematic routines have been written using the C language. Thus any programs that utilizes these inverse kinematic routines, like the ones used in this research, must be written in the C language. The user is able to control the robot arm by controlling the position of the end effector along the x-, y-, and z-axes; the roll, pitch, and yaw of the wrist; and the open/close position of the gripper. In this research, the user is not concerned with the roll, pitch, and yaw of the wrist; the wrist is set such that it can approach and grasp an object from the top-down. The code used to manipulate the RTX for this research, allows the user to control movement along the axes (x, y, z) and the open/close status of the gripper.

**TI-Speech System [24].** The TI-Speech system's hardware and software give the user the ability to write speech synthesis and speech recognition programs. It requires DOS software (either PC-DOS or MS-DOS) in order to be implemented. The following
languages can be used with this speech system: C, Quick BASIC, or MS-MACRO
Assembler.

The hardware includes the TI-Speech board, which provides a signal processor, an
A/D and D/A converter, associated memory, and input/output interface circuits; an input
device (microphone); and an output device (speaker). A headset can be used as the
input/output device. Also, an optional telephone interface board can be installed on the TI-
Speech board for telephone-management applications.

The TI-Speech Application Toolkit consists of the following software: diagnostics,
algorithms, and DSR programs; run-time software; example programs; and TI-Speech
utilities. The TI-Speech system requires DOS software (version 2 or later), a compiler or
interpreter for one of the supported computer languages, and DOS LINK utility.

The algorithms provided by the packaged software implement the capabilities of the
TI-Speech board. Before performing a speech processing function, the user must
download the appropriate algorithm. Generally speaking, the eleven algorithms can be
divided into four areas:

1) The calibration algorithm determines the sensitivity of a speech device by
   analyzing the speech signal. It recommends values to ensure reliable performance
   of the speech processing algorithms.

2) Speech synthesis algorithms allow the computer to provide speech output.
   These algorithms can fall into one of two categories: natural analysis/synthesis
   which provides the ability to record and playback speech messages (like a tape
   recorder) or artificial constructive/synthesis in the form of text-to-speech processes.
   Text-to-speech enables the computer to generate speech from English text or from
   symbols that represent various speech sounds.

3) Four speech recognition algorithms provide for the recognition of speaker-
   dependent, isolated words; speaker-dependent, connected-speech; speaker-
   dependent, natural language; or speaker-independent digits (0 - 9, yes, no).
4) The telephone algorithm monitors the phone line during telephone-management applications. The user can use the phone (and the touchtone "keyboard") to remotely control a robot.

When using the speaker-dependent, isolated-word speech recognition algorithms, the user can create a vocabulary of up to fifty individual templates. A template can consist of individual words or phrases. If more than fifty templates are needed, it is possible to develop a library of more than one vocabulary and then create a code to allow the user to verbally switch between the vocabularies. Since only a few templates (twelve) were needed to control the RTX for this project, it was not necessary to develop multiple vocabularies.
CHAPTER 3
Development of the Speech Recognition System

As stated earlier, the goal of this research project is to develop speaker-dependent, isolated-word speech recognition software that employs the TI-Speech system for the remote control of the RTX. For this task, it was important to determine what commands would be needed to control the robot and to choose short, distinct words or phrases that would be used for these commands. It was key to select words that were distinct, so that the computer would not be confused by words that sound alike (e.g., "quick" and "quit"). The twelve words that were chosen for use in this research are given in Table 1. The action for the robot is based upon the recognition of the individual templates in the vocabulary.

<table>
<thead>
<tr>
<th>TEMPLATE</th>
<th>WORD</th>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>UP</td>
<td>MOVE +Z</td>
</tr>
<tr>
<td>001</td>
<td>DOWN</td>
<td>MOVE -Z</td>
</tr>
<tr>
<td>002</td>
<td>RIGHT</td>
<td>MOVE -Y</td>
</tr>
<tr>
<td>003</td>
<td>LEFT</td>
<td>MOVE +Y</td>
</tr>
<tr>
<td>004</td>
<td>FRONT</td>
<td>MOVE +X</td>
</tr>
<tr>
<td>005</td>
<td>BACK</td>
<td>MOVE -X</td>
</tr>
<tr>
<td>006</td>
<td>OPEN</td>
<td>OPEN GRIPPER</td>
</tr>
<tr>
<td>007</td>
<td>CLOSE</td>
<td>CLOSE GRIPPER</td>
</tr>
<tr>
<td>008</td>
<td>FASTER</td>
<td>STEPSIZE = 22 mm</td>
</tr>
<tr>
<td>009</td>
<td>SLOWER</td>
<td>STEPSIZE = 3 mm</td>
</tr>
<tr>
<td>010</td>
<td>QUIT</td>
<td>EXIT SPEECH SYSTEM</td>
</tr>
<tr>
<td>011</td>
<td>SWITCH</td>
<td>CHANGE USER</td>
</tr>
</tbody>
</table>

Table 1
List of Vocabulary Words for the System
In order to write the software for speech-recognition control of the RTX, the code was developed in two phases. The first phase was concerned with the speech recognition activities, while the second phase dealt with using the speech-recognition code to control the RTX. (Note that all code is written C. This is because C is the only language commonly supported by the TI-Speech system and the RTX software.)

**Speech Recognition Software**

This phase of software development was further divided into two phases. Phase I was the development of the program `rowbot.c` (Appendix C). This program performs run-time routines, such as identification of the input and output device(s), calibration, and loading of algorithms. Phase II was the development of the program `spkrdep.c` (Appendix A and Appendix B). This code allows the user to employ the speech recognition capabilities of the TI-Speech system. The subroutines of `spkrdep.c` perform operations directly related to the training and recognition of speech templates.

**Run-time routines (Appendix C).** Some of the more important subroutines in `rowbot.c` are listed below.

```c
int LoadAlgorithm (filename)
char *filename;
```

The subroutine loads a speech algorithm from the disk onto the speech board RAM. An application must use this subroutine before the system can use text-to-speech, synthesis, analysis, recognition, or verification routines.
int sError (message, error)

char *message;

int error;

The subroutine returns a hexadecimal error code, if there is an error. The meaning of the numerical codes are found in Appendix C of the TI-Speech Application Toolkit manual.

int SetInput ( )

The subroutine performs calibration for the input channel. This determines the amplification of the input signal.

int SetOutput ( )

The subroutine examines, or adjusts, the audio output level (i.e., volume) of the speech system.

int SetSpeechIO ( )

The subroutine identifies the input and output channels (i.e., microphone and speaker or headset). The subroutine also uses the subroutines SetInput and SetOutput.

int StartSpeech ( )

The subroutine checks to insure that tispeech.com has been installed. The TI-Speech system cannot be used, unless tispeech.com has been installed first.

int StopSpeech (exit_code)

int exit_code;

The subroutine exits the speech system and returns to DOS.
Speech recognition routines (Appendix A). The program spkrdep.c is the heart of this whole research project and contains code that employs rowbot.c for the application of speech recognition code. Some of the more important subroutines in spkrdep.c are listed below.

```c
void abort ( )
```
The subroutine aborts the program by keystroke.

```c
void creat_word (first_time, key_abort, templa_id, frames, name)  
int key_abort, *templa_id, *frames, *first_time;
char *name;
```
The subroutine creates the initial templates of the vocabulary. Internally, it uses the routine format enroll (board, *first_time, key_abort, templa_id, frames, &status).

```c
void init_vocabs ( )
```
The subroutine initializes the content of the templates.

```c
int recog ( )
```
The subroutine is used for recognition activities. Internally, it uses the routine format recognize (0, first_time, valid, &temp_id, ret_parms, &status).

```c
void refine_voice (tmpl_tot)
```
```c
int tmpl_tot;
```
The subroutine refines already existing templates. Internally, it uses the routine format update (0, one, 1, 1, &temp_list, error, &status). The subroutine also prints to screen the refining error after every refinement.
void save_em (temp_id, phrase_num)

int temp_id, phrase_num;

The subroutine saves the templates after they have been refined. Internally, it uses the
routine format save_template (0, 0, temp_id, &temp, &length[phrase_num],
&status).

void train ( )

The subroutine trains the vocabulary. As part of the training operations, the subroutine
calls upon the subroutines creat_word and refine_voice.

void up_date ( tmpl_total)

int tmpl_total;

The subroutine refines the existing templates. It is invoked outside of the training mode
and is used to perform updates over a period of time. The subroutine calls upon the
subroutine refine_voice.

Robot Control Software

The robot control software actually falls into three categories. The first category is
the group of subroutines in spkrdep.c that move the robot arm and control the gripper. The
second category provides the code that performs the communication operations between the
two IBM PC microcomputers and the RTX via RS-232 serial ports. And the third category
is the inverse kinematics routines that were written by Sarmad Adnan.
**Robot control routines (Appendix B).** The robot-control subroutines in spkrdep.c are listed below.

`void backup ( )`

The subroutine moves the RTX arm in the negative x-direction.

`void closegrip ( )`

The subroutine closes the RTX gripper.

`void down ( )`

The subroutine moves the RTX arm in the negative z-direction.

`void forward ( )`

The subroutine moves the RTX arm in the positive x-direction.

`void left ( )`

The subroutine moves the RTX arm in the positive y-direction.

`void move_robot (x, y, z, g)`

`int x, y, z, g;`

The subroutine sends the new location of the end effector (i.e., x, y, z) and the status of the gripper (i.e., open or close) to the RTX. It also prints the data to screen. The subroutine is also used to open communications between the two IBM PC microcomputers.

`void opengrip ( )`

The subroutine opens the RTX gripper.
void right ( )

The subroutine moves the RTX arm in the negative y-direction.

int robot_sim (x_start, y_start, z_start, g_start)
int x_start, y_start, z_start, g_start;

The subroutine controls the movement of the RTX arm in the x-, y-, and z-directions and the open/close status of the gripper. This is the heart of the robot control software.

void stepdown ( )

The subroutine sets the stepsize for arm movement at 3 millimeters.

void stepup ( )

The subroutine sets the stepsize for arm movement at 22 millimeters.

void up ( )

The subroutine moves the RTX arm in the positive z-direction.

Equipment communications routines. The code used for communications between the IBM computers and the RTX is listed below.

void move_robot (x, y, z, g)
int x, y, z, g;

The subroutine contains the lines of code that send the processed information (i.e., new position of arm and status of gripper) from the first IBM PC microcomputer to the second one. (Also see Appendix B.)
commander.bat
The routine sets the communications mode for the signal leaving the first IBM PC microcomputer. The signal is sent using Procomm with the following settings: mode com1 1200, n, 8, 1, p. This routine is also used as a start-up routine for the first IBM PC. (Also see Appendix G.)

slave.c
The program sets the communications mode for the signal received by the second IBM PC from the first PC and for the signal sent to the RTX from the second PC. It also prints to screen the current position of the arm and status of the gripper and any error messages that originate due to problems with the inverse kinematics of the RTX. (Also Appendix H.)

Inverse kinematics routines. The inverse kinematics software for the RTX is provided in Appendix I and Appendix J.
CHAPTER 4

Implementation of the Speech Recognition System

Before implementing the speech recognition system, the programs must be compiled and linked, the equipment must be set-up, and the start-up functions must be invoked. At this point in time, the appropriate programs and library have been compiled and linked; the library used, speechms.lib, is the MS library that is part of the TI-Speech software package.

Set-up and Start-up

Equipment Set-up. As shown in Figure 1, this voice recognition system requires that the two IBM PC microcomputers and the RTX robot be connected via RS-232 serial ports. Figure 5 shows how the communication connections are made.

Figure 5
Speech Recognition System Communication Connections
Serial port 1 of the IBM PC with the TI-Speech board is connected to serial port 2 of the second IBM PC. After the signal is input to the first microcomputer, through the microphone, and is processed by the speech recognition software, the command for the new configuration of the robot and its gripper is sent to the second IBM PC via this first RS-232 connection. This signal is then sent to the robot through the second RS-232 connection, from serial port 1 of the second PC to the RS-232 serial port of the RTX. The RTX then makes the appropriate adjustments, based upon the command that is sent.

After these communication connections have been made and the microphone and speaker have been connected to the first IBM PC, the equipment can be turned on and the system put into operation.

**Start-up Routines.** There are four start-up routines that must be invoked: commander, start, init, and slave. The command **commander** is used on the first IBM PC. This command sets the parameters needed for serial port 1 to be the outgoing communications line, changes to the appropriate directory (i.e., "cd tispeech"), and calls upon TI-Speech functions that install the device service routine and the text-to-speech runtime routine and perform a series of preliminary self-tests. The remaining start-up routines are all typed in to the second IBM PC.

The command **start** insures that the computer and RTX can communicate with each other and that the RTX library is installed. Before using the command **init**, the user must make sure that the motor power is on. This command initializes the robot arm; the RTX runs its motors, driving the arm components against their mechanical stops. The arm will conclude this check by going to an initialized position, with the arm straight out and the gripper horizontal (as depicted in the diagrams in Appendix K). The command **slave** sets
the parameters needed for serial port 2 to receive the incoming signals from the first IBM
PC. When the second IBM PC is ready, "Speech Slave Ready for Commands" will be
printed to screen. (Note that in order to invoke these three start-up routines, it is assumed
that the user is in the correct directory; "bill" in this case.)

**Running the Speech Recognition Program**

After the preliminary equipment set-up and the start-up routines are completed, the
user may start the speaker-dependent, isolated-word speech recognition program. This is
done by typing-in the command **spkrdep**. When the speech recognition program is
invoked, it performs calibration functions and the template-matching operations of training
and recognition. Figure 6 shows the flowchart for the operations for the speech
recognition program, spkrdep.c.

Before running any template-matching routines, the program prompts the user for
information so that it can identify the input/output device and perform calibration operations
(e.g., set the volume for the speaker for any speech synthesis output). The program then
checks to see if a vocabulary already exists. If a vocabulary exists, the program gives the
user the option of going directly to the recognition mode or refining the vocabulary before
beginning the recognition operations. If a vocabulary does not exist, the program
automatically goes to the training mode. After the vocabulary has been created and refined
once, the program allows the user to choose between performing another refinement or
going into the recognition mode.
Figure 6

Flowchart for spkrdep.c
Training Mode. Once the program has gone to the training mode, the first thing that it must do is create the templates for the user's vocabulary. The computer will prompt the user to say each of the words from Table 1 just once. This is done to establish a temporary reference template for each individual word.

After these reference templates have been stored in the computer's memory, the program will begin the refining routine. At the start of the refinement of each word, the computer prompts the user with the question, "How many times do you want to update? (0-10)." Generally speaking, 3 - 5 times is good. The computer prints to screen the refining error, following each utterance of a word. This error indicates the relative closeness of the updated utterances to the matched word. A positive integer close to 0 indicates a close match, while a high value (e.g., fifty or greater) indicates a poor match. If the refining error is minus 1, the speech recognition system was not able to make any match and the updated utterance goes unrecognized. After the user has updated the template the number of times chosen, the templates are stored as the reference templates for the remainder of the program.

It is important to note at this time that the user must create and update the vocabulary in a relatively noise-free environment. The recognition system will pick-up background noise during the training phase. It cannot identify the speaker's voice apart from the unwanted noise, so it saves the noise as part of the reference template. This can lead to problems when trying to update the vocabulary or when running the recognition operations. Likewise, it is important to conduct recognition operations in the same noise-free environment. Background noise will cause the input signal to change, regardless of how well the user enunciates a word. This will, most often, result in an unrecognized utterance.
The ideal method for refining a template takes place over a period of hours or days. The reason for this is that people have a tendency to rush the utterance of a word when repeating that same word over-and-over again in a short period of time. This causes the speech signal to vary from the pattern it would have if the word were spoken more naturally. While some variance is expected and accepted, this unnatural change in pronunciation often produces a template that is not close enough for recognition purposes. If the user were able to leave the system for a period of time and return to update the vocabulary, the utterances would most probably be more natural. Any differences in the speech signals would be minor and would not hinder recognition operations.

This program allows for template refinement over a period of time. This refinement is done outside the training mode. The user may update his vocabulary immediately after training it. This option is given to the user at this early point in the program, in case the user is not happy with the refining error of a certain template and wants to update it as soon as possible. The update option is also available at the beginning of subsequent uses of the program. This allows the user to leave the system and return at a later time to refine the templates. The user has the choice of updating one or more templates or creating an entirely new vocabulary. If the user chooses to update the vocabulary, the system initiates the refinement procedures. If the user wants to replace the current vocabulary, the program exits the system to allow the user to delete his vocabulary. After this vocabulary has been deleted, the user may reenter the system and start training a new vocabulary.

Following the update option, the program goes directly into the recognition mode.
Recognition Mode. Once training has been completed, the user may begin the recognition operations. In this program, all the words in the vocabulary except "Quit" and "Switch" are used to remotely control the RTX robot. Therefore, when the recognition mode begins, the program automatically goes to the subroutine robot_sim. At this time, the first IBM PC will print to screen,

\[ x \ 450 \ y \ 10 \ z \ 900 \ g \ 0 \]
Robot Commander Ready

and the second IBM will respond with ":: 450 : 10 : 900 : 0 ::". This indicates that the "commander" PC has sent the start position to the "slave" PC, and that the signal has been sent to the RTX. Upon receiving this signal the RTX will move to the position where: x=450, y=10, z=900, and g=0 (i.e., the gripper is closed). The gripper is no longer horizontal in its orientation, but is now pointed at the ground so that it may approach an object from the top-down. Also, the initial stepsize is set at 3 millimeters.

As the user speaks the commands into the microphone, the system performs recognition operations. It searches for a match for the input (unknown) template. If a match is found, the program runs the corresponding subroutine and moves the RTX accordingly. If no match is found, "Not Recognized" is printed to screen on the monitor of the commander PC. When a match is made, both monitors display the new values for x, y, z, and g. The monitor of the slave PC always displays the last successful move; in other words, if the match is unsuccessful, the robot arm does not move and the data displayed on the screen represents the current status of the RTX.
Figure 7
Recognition of the Word "Down"

Figure 7 depicts a successful search for the command "Down." The user says, "Down." The recognition subroutine systematically compares the input template against the reference templates. The comparison with Template 000 results in no match. The search continues. When the unknown template is compared with Template 001, the two templates are similar enough to result in a match. Since this reference template corresponds with the subroutine "down", the search is ended. The program responds by moving the robot arm down one stepsizes and updating both monitors.

The system is also designed to allow for multiple users. By saying, "Switch," a user allows another user to assume verbal control of the RTX. This command informs the system that the current user is leaving the system. The program switches to the next user and checks to see if a vocabulary exists. If it does, the user is given the option of updating
the vocabulary or going into the recognition mode. If there is no vocabulary, the system
goes to the training mode to create a vocabulary for the new user.

Switches between users occur in a cyclic manner. For example, assume that there
are four users. User 1 switches to User 2, User 2 to User 3, User 3 to User 4, and User 4
back to User 1. Figure 8 show the flow of switches from user-to-user. The program can
be modified to allow for any number of users.

![Diagram](image)

**Figure 8**
Switching Among Multiple Users

The speech recognition program will continue to run, until the user indicates that he
wants to exit the system in order to create a new vocabulary, gives the command "Quit," or
aborts the program with a keystroke.
Chapter 5

Tests and Results

Recall that this research was conducted in two phases. The first phase was the development of the speech recognition program and the second was the development of the robot-control software. Also recall that the robot-control software can be replaced with compatible software to control different robots for different tasks. Based upon these characteristics, it was necessary to conduct tests on these operations.

Recognition Tests

When the program spkrdep.c was first written and tested, the subroutine robot_sim did not do any type of robot control. Instead, the program was written so that it would respond to recognized commands by displaying a response on the monitor. Originally, the responses were to print to screen the command given (i.e., "up", "down", "right", etc.). When this proved to be successful, the code was modified and the computer responded by printing out the new values for x, y, z, and g for the RTX, based upon the most recent command given.

These tests proved to be successful and they naturally progressed to the development of the robot-control code. There were some problems that arose in this phase of the program development. The two most significant problems encountered were the refining techniques and the limitation to just one user. Additional code was written to eliminate these problems.
The initial program required that the user perform all refinements to the vocabulary immediately after creating the templates. There was no capability to update the vocabulary at a later time. If the user wanted to update just one or two templates, he had to delete the entire vocabulary. This resulted in the erasure of "good" templates along with the "bad" ones. Even though this vocabulary consists of only twelve words, this method of refinement was unnecessarily time-consuming and, at times, very frustrating. The subroutine up_date was added to solve this problem.

If a new user wanted to control the system through the use of the original program, the current user's vocabulary had to be removed or renamed. The second user could return control of the RTX to the first user by exiting the system, removing the new user's vocabulary, restoring the original user's vocabulary, and restarting the program. This was tedious and it required particular care on the part of the users to ensure that established vocabularies were moved, not destroyed. The command "Switch" was added, along with its accompanying code. This not only allowed multiple users to control the system but it also allowed the users to save their vocabularies without having to move them for protection.

**Control Tests**

The natural progression after the recognition phase was the modification of the code so that the program would control the RTX robot. This was the main goal of the entire research project. This phase was successfully completed. There was a problem of incompatibility between the RTX software with the TI-Speech board. A second IBM PC was incorporated into the system to overcome this problem.
In the first phase of control testing, only one person used the speech recognition system. The user was able to easily move the robot arm along the x-, y-, and z-axes; open and close the gripper; and increase and decrease the stepsize. In the next phase, other users were enlisted to try to create their own vocabulary and control the RTX. This proved to be equally successful and it was encouraging to see the system respond properly to different voices. This was another reason for the development of the "switch" code. Among those who tested the system were another male graduate student, his wife, and my four-year old son. It was particularly satisfying to see the speech system respond to my son's voice. (He had one command as part of another user's vocabulary. His lone command was "Quit.") This showed that the system is responsive to different voices, regardless of the user's gender or age.

Another test of the system was the implementation of the pick-and-place capabilities of the robot. A block was set upon a table in front of the RTX and the user verbally controlled the movement of the arm and gripper. The arm moved in such a manner that the gripper approached the block from the top down, picked-up the block, and placed it on another point on the table. All users were able to successfully complete this task. The only difficulty with this test was trying to find a good view of the operations. In order to get a better view of the gripper and block, the user had to stand close to the robot. This problem was not a result of any system shortcomings, but rather a result of the desired task. Just as in any experiment that deals with grasping, the user must find a way to view the operations from different angles. But this "problem" served to demonstrate one of the advantages of a speech-recognition system. The user was able to move to the robot for better observation and still maintain control of the robot arm. If the user had been confined to controlling the RTX through keyboard commands, he would have had to move back-and-forth between the computer terminal and the robot to conduct this test.
Other Robot-Control Tests

Tests were conducted to examine the capability to modify this program for robot control for other tasks. The first test incorporated a program which simulated a robot moving in a known environment with a square and two triangles as obstacles [21, pp. 207-238]. This test was unsuccessful due to an interrupt routine in the program that made it incompatible with the TI-Speech system. Therefore, a different program, with compatible software was tested. This program was a simple navigation program that simulated moving a robot, represented by "#", through a known environment free of obstacles; this environment was represented by the computer screen. The movements were discrete. There were only five commands: north, south, east, west, and quit. This program was successfully tied into the recognition routine and the user was able to move the "robot" across the screen. The code for this routine is not included in this report, but the success of this test is a simple example of the validity of the claim that this program can be modified to control other robots for other tasks. (This is qualified by the requirement that the robot-control software is compatible with the TI-Speech software and hardware.)
Chapter 6
Conclusions

The speaker-dependent, isolated-word speech recognition system presented in this thesis represents the integration of speech recognition software and robot-control software to allow for the remote-control of a robot through the spoken word. It sets the foundation for control of the robot arm in a system where no keyboard commands or hand-held teaching pendants are needed. This gives the user an advantage, because he is able to control the robot while using his hands to perform other tasks. It also makes attempting to control the RTX more attractive to new users, because the user does not necessarily have to be familiar with the RTX and its inverse kinematics. Rather, the user can experience robot control by using something that he is very familiar with: verbal commands.

While this research project has proven to be successful in controlling the RTX through speech recognition, it is not without its shortcomings. The system does not adhere to the earlier definition of an ultimately user-friendly system because it does not have complete voice communications abilities. To be "friendly", the system should provide audio feedback to the user. In other words, the code that causes the computer to print to screen should be replaced with some type of speech synthesis routines. This feedback can be through the playback of a recorded message or through text-to-speech processes. This system uses some speech synthesis. Speech synthesis welcomes the user when the program is started up and informs the user when the system is ended by an exit or abort routine. But, that is the extent of the speech synthesis used in this research project. However, this lack of speech synthesis did not hinder the accomplishment of the goal of
this project: control of the robot arm. It would have served to make the system more attractive, but it was not a necessary quality.

In summary, this speaker-dependent, isolated-word speech recognition system is successful when it is applied to the control of the RTX robot. The user can move the arm and control the gripper well enough to be able to perform simple pick-and-place operations. Additionally, the system is designed to permit multiple users and the software can be modified to control other robots for other tasks. This second characteristic is very important, because it enables the user to use the basic skeleton of the recognition system to carry on in future endeavors in other areas of robotic control.
Chapter 7

Areas for Future Work

The work already done on the development of this speech recognition system opens many possibilities for future efforts to expand this project and provides the potential for other developments. The TI-Speech system contains other algorithms that have not been utilized in this project. It would be beneficial if the independent speech recognition of the digits 0 - 9 and the words "yes" and "no" were incorporated into this system. An example of how this could be applied would be independent speech login capabilities. Each user could have a numerical password. When the system starts up, it would use a speech synthesis routine to ask for the user's number. The user could identify himself through speaking his number, which the system would recognize regardless of who was speaking. Upon recognition of the password, the computer would open the correct vocabulary for follow-on speaker-dependent input.

The user could try to use the language recognition algorithm to replace the isolated-word recognition algorithm. This program would require that the user define the grammar and vocabulary for the language needed in the application. The grammar would consist of a set of rules that would specify the sequences of words that constituted acceptable sentences. This would allow the user to use continuous speech, instead of isolated words, because the words in the vocabulary would be recognized if they were used in the proper sequence. The vocabulary would be limited to words needed for the application.
In the area of extensions to the RTX control program, more routines could be added to allow for more precise control. This would require a larger vocabulary, but remember that this research project used only eleven of fifty templates possible in a vocabulary. Routines could be added to give the user control of the roll, pitch, and yaw of the wrist. Also, pre-taught positions could be added, so that the user could send the end effector directly to a desired position, instead of having to move the gripper there in discrete steps.

Another possible use of this speech recognition system has already been discussed: the incorporation of other compatible robot-control programs. The system could be modified to provide verbal control of robot navigation systems, of multi-fingered grippers, or mobile robots. Two other potential uses at Rice University deal with projects currently underway. One would be to use the program as part of the lab periods for the Introductory and Advanced Robotics classes to introduce the students to the applications of speech synthesis and speech recognition. The other would incorporate this system with the mobile omni-directional robot in the Mechanical Engineering Robotics research [2, 8]. Currently a joystick is used to control the movement of the robot base and the movement of the arm mounted on the base. This requires a great deal of concentration and coordination on the part of the user. If the user could control the arm through verbal input, his hands would be free to use solely for controlling the movement of the base.

These are only a few of the improvements and extensions of this project that could be considered for future projects. Ultimately, the application and potential of this speech recognition system is limited only by our imaginations.
Bibliography


Appendix A

spkrdep.c Source Code Listing

This is the source code for spkrdep.c, minus the subroutine robot_sim. The program spkrdep.c is the heart of the speaker-dependent, isolated-word speech recognition system. The portion of the code that is in this appendix is used for the speech recognition operations: training and recognition. The subroutine robot_sim is used to control the RTX robot and it can be found in Appendix B.

/*
 Robotic Control through Speaker_Dependent, Isolated_Word Speech Recognition

 Program spkrdep.c

 By William T. Atkinson
 29 January 1991

 The focus of this research is the remote control of a robot through the use of speech recognition. This project uses a speaker-dependent, isolated-word program that allows the user to control a 6 degree-of-freedom mid-range industrial robot (through the subprogram 'Robot_Sim'). The bulk of this program is designed such that any other compatible C-language program that is written to control a robot (regardless of whether the focus is on navigation, gripper control, arm movement, etc.) can be added (ie can replace 'Robot_Sim'), minor modifications that are needed can be made, and a new vocabulary created; so that the robot control can be done through the use of verbal input. In other words, the basic shell, or skeleton, of this program can be used in conjunction with other compatible robot control programs to allow the user the freedom of verbal commands to run the control program.
 */

#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdlib.h>
#include <dos.h>
#include <malloc.h>
#include "global.h"
#include "utili.h"
#include "spkrdep.h"
int usr=1;
char *tmpl[50]; /* The global vocabulary consists of 50 templates */
char *words[50];
char store[80];
int length[50];
int valid[4]={-1,-1,-1,-1};
int status;

/*--------------------------------------------------------------------------*/

main()
{
    void train();
    void tx_to_sph();
    void up_date();
    int recog();
    void init_vocabs();
    void reset_channel();

    /* The following subroutines are peculiar to the control of the robot arm. */
    int robot_sim(int, int, int, int);
    void move_robot(int, int, int, int);
    void cls();
    void position();
    void up();
    void down();
    void right();
    void left();
    void forward();
    void backup();
    void opengrip();
    void closegrip();
    void stepup();
    void stepdown();

    int response;
    int i;
    int x;
    int in_channel;
    int out_channel;
    int status;
    int ttype;
    int vol;
    int defgain;
    int spchbuf;
    int tmpl_total;

    init_vocabs(); /* Initialize the template */
/*
  1. Clear Screen
  2. Start Speech and initial parameters for speech board
  3. Initialize the input/output, the volume, and the gain
 */

for (i=1; i<25; i++) printf("\n");   /* 1. Clear Screen */
StartSpeech();    /* 2. Start Speech */
SetSpeechIO();    /* 3. Initialize I/O, volume, & gain */

/*
Two methods are made available to welcome the user to this particular program:

1. Text_to_Speech operation to fetch "howdy.voc" and "purpose.voc" files and speak them,

2. Play_File operation to fetch pre-recorded "welcome.msg" and "intro.msg" files and play/speak them.

The first method requires less memory space to store the messages but produces a metallic/mechanical sounding "voice."

The second method requires less programming skills on the part of the programmer and produces a more natural sounding "voice."

*/

/*
Text-to-speech welcome is not used in this particular program, but it is available.

tx_to_sph(f,"howdy.voc");
tx_to_sph(f,"purpose.voc");
*/

/* This is an example of the Play_file synthesis operations. */

SetOutput();

play_file(board,TTONEx_opt,KEYBD_opt,"welcome.msg",&status);  
play_file(board,TTONEx_opt,KEYBD_opt,"intro.msg",&status);  
play_file(board,TTONEx_opt,KEYBD_opt,"begin.msg",&status);

for (i=1; i<25; i++) printf("\n");   /* Pause */
/*
   Reset the input channel for "Microphone" and turn off the output channel; the
output channel is not needed in the training mode, refining mode, and/or recognition mode
to establish an application.
*/

in_channel=4; /* input channel = Microphone */
out_channel=0; /* output channel = off */
reset_channel(in_channel,out_channel);

LoadAlgorithm("LWRIIC.IMG");
ttype=0;

new_speaker: /* WHEN SWITCHING TO NEW USER, GO HERE. */

/*
   Check to see if the robotic world vocabulary exists.
   If it does, give the user the option of updating the vocabulary or going to the
   recognition mode.
   If it does not, go the training mode.
*/

if(user==1) /* Identify User 1 */
{
    printf("User1\n");
    usr=0;
}
else
{
    printf("User2\n"); /* Identify User 2 */
    usr=1;
}

if(usr==0) /* Check for User 1's vocabulary */
{
    if(load_vocab(board,"usernum1.voc",spchbuf,ttype,&status))
    {
        if(status==0xD1)
        {
            /* Print the return value of load_vocab, if needed, for debugging. */

            printf("There is no vocabulary. Go to the training mode.\n");

            train(); /* Train the vocabulary */
        }
    }
else sError("Load_vocab",status);

}
if(usr==1) /* Check for User 2's vocabulary */
{
    if(load_vocab(board,"usernum2.voc",spchbuf,type,&status))
    {
        if(status==0xD1)
        {
            /* Print the return value of load_vocab, if needed, for debugging. */
            printf("There is no vocabulary. Go to the training mode.\n");
            train(); /* Train the vocabulary */
        }
    }
    else sError("Load_vocab",status);
}

/*
 * The vocabulary has been successfully established.
 * Give the user the option of updating the vocabulary or going to the recognition mode.
 */
printf("The vocabulary exists. \n");
printf("Do you want to update the vocabulary \n");
printf("or do you want to go to the recognition mode. \n");
printf("Type '1' for update or '0' for recognition.\n");
scanf("%d",response):
/*
 * NOTE: The total number of templates for robot_sim is 12. The first template is
 * '0', so the total number sent to up_date is 11.
 */
if(response==1) up_date(robo_num-1);

/* Go to robot_sim to control the RTX robot. */
printf("\n");
printf("Now it is time to run the robot simulator.\n");
if (robot_sim(450, 10, 900, 0)==1)
{
    goto new_speaker;
}
/* StopSpeech and go back to DOS */

reset_channel(4,0);
tx_to_sph(f, "bye.voc");
for (i=1; i<25; i++) printf("\n");
StopSpeech();
exit(0);

/* Subroutine train is used to train and refine the vocabulary */

void train()
{

    void creat_word();
    void abort(),save_em();
    void refine_voice();
    int i,first_time,key_abort,templa_id,frames,temp_id;
    int tmpl_total,spchbuf,status;

    /* Initialize the new vocabulary for the first word by setting up the flag "first_time". The same vocabulary is used for all subsequent words, so turn off the "first_time" flag after the first word. */

    first_time=1;
    key_abort=0;

    /* Initialize the body */

    /* NOTE: These three "for" loops could be simplified by using a double "for" loop calling the same subroutine. This is just a note for extension. */

    /* The data structure of "body","actions",and "direction" can also be written in a "structure" form when extended. */

    templa_id=0;
    for(i=0;i<robo_num;i++)
    {
        if(kbhit()) abort(); /* Invoke the abort function to exit the program while debugging. */
        else
            
        words[templa_id]=usrfun[i];
        creat_word(&first_time,key_abort,&temp_id,&frames,usrfun[templa_id]);
        templa_id++;
/* If there is an error, check first_time. */

} /*

Because the first template is the "0" template, the total template number must be
subtracted by one.
*/

tmpl_total=--templa_id;

/* Refine the template in the same vocabulary. */

refine_voice(tmpl_total);

/* Save the vocabulary. */

if(usr==0)
    if(save_vocab(0,"usernum1.voc",spchbuf,0,0,&status))
        sError("Save_vocab",status);
if(usr==1)
    if(save_vocab(0,"usernum2.voc",spchbuf,0,0,&status))
        sError("Save_vocab",status);

/* Subroutine abort is used to abort the program by keystroke. */

void abort()
{
    getch();
    printf("Program aborted by pressing a keystroke \n\n");
    StopSpeech();
}

/* Subroutine save_em is used to save templates of the vocabulary. */

void save_em(temp_id,phrase_num)
int temp_id,phrase_num;
{
    int temp;
    int status;
    length[phrase_num] = 0;
    save_template(0,0,temp_id,&temp,&length[phrase_num],&status);
if(status != 3)
{
    /* Error 3 is expected, because the length is 0.
     The call (save_template()) returns correct length. */
    sError("Save_template",status);
}

tmplt[phrase_num] = malloc(length[phrase_num]*2); /* allocate storage */
if(save_template(0,0,temp_id,tmplt[phrase_num],&length[phrase_num],&status))
{
    /* Save the template now with the correct length. */
    sError("Save_template",status);
}

/*====================================================================*/

/* Subroutine creat_word is used to create the initial templates of the vocabulary. */

void creat_word(first_time,key_abort,templa_id,frames,name)
int key_abort,*templa_id,*frames,*first_time;
char *name;
{
    int status;
    printf("Please say the following keyword \"%s\"\n",name);
    /* If the return value of "enroll" is zero then enroll is successful */
    if(enroll(board,*first_time,key_abort,templa_id,frames,&status))
        sError("Enroll",status);
    else
    {
        *first_time = 0;
    }
}

/*====================================================================*/

/* Subroutine refine_voice is used to refine the templates. */

void refine_voice(tmpl_tot)
int tmpl_tot;
{
    void save_em();
    int i=0,spchbuf,temp_list,error[3],j,times,init,one;
    int status;
    /* Save the templates before refining. */
    for(i=0;i<tmpl_tot;i++) save_em(i,i);
    printf("\nPlease refine the keywords \n");
    for(i=0;i<tmpl_tot;i++)
    {
        if(kbhit() != 0) abort();
    }
Each time, a single template is loaded from the computer memory to the vocabulary area of the speech processor. Therefore, when the template in the vocabulary area is updated, there is only one template in this area; the vocabulary area is cleared every time the "load_template" subroutine is called (the second parameter in this routine is 1). The template I.D. of the updating is always 0.

```c
if(load_template(0,1,0,1,tmpl[i],&length[i],&status))
  sError("load_template",status);
printf("\n");
printf("The current word is ",usrfunc[i]);
printf("How many times do you want to update? (0-10) ");
scanf("%d", &times);
printf("\n");
temp_list=0;
one =1;
if(times 1= 0) printf("Please say the following word: ",usrfunc[i]);
for(j=0;j<times;j++)
{
  /* Abort the program to save time when debugging. */
  if(kbhit()) abort();

  /* Update the vocabulary. */
  if(update(0,one,1,1,&temp_list,error,&status))
    
    if(status==0xD0)
      
      getch();

    else if(status != 0x13) sError("Update",status);
  }

  one =0;
  printf("The %d refining has the error = %d\n",j,error[0]);
}

/* Save the updated template. NOTE: The template I.D. is always zero. */
save_em(0,i);
}
init = 1;
for(i = 0; i<=tmpl_tot; i++)
{
  if(load_template(0,init,0,1,tmpl[i],&length[i],&status))
    sError("load_template",status);
  init = 0;
}
```
/* Subroutine up_date is used to give the user the opportunity to refine an already existing vocabulary. This refinement is done outside of the training mode. */

void up_date(tmpl_total)
int tmpl_total;
{
    void refine_voice(tmpl_total);
    void save_em();

    int answer;
    int spchbuf;
    int i;
    int status;

    printf("\n");
    printf("Do you want to create a new vocabulary (0)\n");
    printf("or refine the current templates (1)? \n");
    scanf("%d",&answer);

    if(answer==0)
    {
        printf("You will have to delete the vocabulary and start over. \n");

        reset_channel(4,0);
        tx_to_sph(f,"bye.voc");
        for(i=1;i<25;i++) printf("\n");
        StopSpeech();
        exit(0);
    }

    printf("tmpl_total is %d\n", tmpl_total);
    refine_voice(tmpl_total);

    if(usr==0)
        if(save_vocab(0,"usernum1.voc",spchbuf,0,0,&status))
            sError("Save_vocab",status);

    if(usr==1)
        if(save_vocab(0,"usernum2.voc",spchbuf,0,0,&status))
            sError("Save_vocab",status);  
}
int recog()
{
    int first_time,temp_id,ret_parms[10];
    int x,spchbuf,status;
    int keeper;
    /* void sp_status(); */

    first_time = 1;
    do
    {
        if(kbhit() !=0) abort();
        recognize(0,first_time,valid,&temp_id,ret_parms,&status);
        first_time = 0;

        if(status == 0x13) printf("Not recognized.\n");

        if(status == 0)
        {
            switch(temp_id)
            {
                case 0 :  up();
                        break;
                case 1 :  down();
                        break;
                case 2 :  right();
                        break;
                case 3 :  left();
                        break;
                case 4 :  forward();
                        break;
                case 5 :  backup();
                        break;
                case 6 :  opengrip();
                        break;
                case 7 :  closegrip();
                        break;
                case 8 :  stepup();
                        break;
                case 9 :  stepdown();
                        break;
                case 10 : printf("Exit requested\n");
                            keeper=0
                            break;
                case 11 : printf("Switch requested\n");
                            keeper=1
                            break;
            }  
        }
    } while (kbhit() !=0);
while((temp_id != 10) && (temp_id != 11)) || (status==0x13);

if(halt_speech(0,&status)) sError("Halt_Speech",status);

return (keeper);
}

void tx_to_sph(tx_or_f,name)
char tx_or_f;
char *name;
{
    int status,keyopt;
    LoadAlgorithm("TTS.IMG");
    SetOutput();
    if(tx_or_f == 'f')
        speak_file(board,ttone,keyopt,name,&status);
    else
        speak_text(board,ttone,keyopt,name,strlen(name),&status);
}

void reset_channel(in,out)
in int in,out;
{
    int vol,status,defgain;
    /* printf("In = %d Out = %d\n",in,out); */
    vol=5;
    defgain=5;
    volume(0,SET,&vol,&status);
    gain(0,SET,&defgain,&status);
    channels(0,SET,&in,&out,&status);
}
Subroutine sp_status is used to show the status of the speech board. This subroutine has the following properties:
1. Shows the algorithm downloaded to the speech board,
2. Shows the parameters of speech status,
3. Is useful for debugging.

```c
void sp_status()
{
    int code,mode,status,bready,bread;
    speech_status(board,&code,&mode,&bready,&bread,&status);
    if(status)
        sError("Status",status);
    else
    {
        /* Property #1: Algorithm downloaded. */
        switch(code)
        {
        case (0):
            printf("No algorithm downloaded.");
            break;
        case (0x12):
            printf("9600 bps Analysis algorithm downloaded.");
            break;
        case (0x22):
            printf("9600 bps Synthesis algorithm downloaded.");
            break;
        case (0x28):
            printf("Text-to-Speech algorithm downloaded.");
            break;
        case (0x30):
            printf("2400 bps Synthesis/Analysis algorithm downloaded.");
            break;
        case (0x34):
            printf("16000 bps Synthesis/Analysis algorithm downloaded.");
            break;
        case (0x36):
            printf("32k bps Synthesis/Analysis algorithm downloaded.");
            break;
        case (0x60):
            printf("2400 bps Synthesis/Recognition algorithm downloaded.");
            break;
        case (0x80):
            printf("Calibrate algorithm downloaded.");
            break;
        default:
            printf("Algorithm downloaded, code = %2x", code);
        }
    }
```
/* Property #2: Parameters of speech status. */
switch(mode)
{
    case (0):
    printf("Ready to download speech algorithm.\n");
    break;
    case (1):
    printf("Nothing has been initialized.\n");
    break;
    case (2):
    printf("Analysis initialized.\n");
    break;
    case (3):
    printf("Analysis active.\n");
    break;
    case (4):
    printf("Synthesis initialized.\n");
    break;
    case (5):
    printf("Synthesis active.\n");
    break;
    case (6):
    printf("Recognition initialized.\n");
    break;
    case (7):
    printf("Recognition active.\n");
    break;
    default:
        printf("Mode = %x.\n",mode);
        
        }  

        if (bready && (mode==3))
            printf("Analysis result buffer is available.\n");
        if (bread)
            printf("Synthesis buffer has been read.\n");

    }

    /**************************************************************************
    
    */ Subroutine init_vocabs is used to initialize the content templates. */

    void init_vocabs()
    {
        int i;
        robo.num=robo_num;
        robo.template=usrfun;
        robo.response=output;
        user.num=user_num;
        user.template=usname;
    }
user.response=ansusr;
user.index=1;
}

/*************************************************************************/

/* Go to Appendix B. */
Appendix B

Subroutine robot_sim

The subroutine robot_sim is part of the program spkrdep.c (Appendix A). The subroutine robot_sim is used to control the RTX robot. This subroutine also sets the parameters to open the lines of communications from the serial port of the first IBM PC microcomputer. The portion of the code that is in Appendix A is used for the speech recognition operations: training and recognition.

/*-----------------------------------ROBOT SIMULATOR-----------------------------------*/

/*

Robot Simulator

The following code is used to control the movement of the RTX arm in the x-, y-, and z-direction and the open/close status of the gripper.
*/

int x_curr, stepsizen=3, y_curn; /* current x,y pos of robot */
int z_curn, g_curn; /* current z and gripper */

int robot_sim(x_start, y_start, z_start, g_start)
int x_start, y_start, z_start, g_start;
{

/* Start position of robot and start status of gripper */
x_curn=450;
y_curn=10;
z_curn=900;
g_curn=0;

move_robot(x_curn, y_curn, z_curn, g_curn);
printf("\n");
printf("Robot Commander Ready\n");
if (recog()==1) /* recog()==1 when "switch has been entered. */
return (1);
printf("Exiting As Commanded\n");
return (0);
}
/*/ Subroutine move_robot sends the new location of the end effector and status of the gripper to the RTX. It also prints this data to screen. */

void move_robot(x, y, z, g)
int x, y, z, g;
{
FILE *fp;

fp=fopen("com1", "w");

printf("\n");

printf("%d %d %d %d\n", x, y, z, g);

fprintf(fp, "%d %d %d %d\n", x, y, z, g);

fclose(fp);
}

/*/ Subroutine cls is used to clear the screen. */

void cls()
{
int i;
for (i=1; i<25; i++) printf("\n");
}

/*/ Subroutine up is used to move the RTX in the +Z-direction. */

void up()
{
int x,y,z,g;

z=z_curr+stepsize;
z_curr=z;

move_robot(x_curr,y_curr, z_curr, g_curr);
}
/* Subroutine down is used to move the RTX in the -Z-direction. */

void down()
{
    int x,y,z,g;
    z=z_curr-stepsize;
    z_curr=z;
    move_robot(x_curr,y_curr,z_curr,g_curr);
}

/* Subroutine right is used to move the RTX in the -Y-direction. */

void right()
{
    int x,y,z,g;
    y=y_curr-stepsize;
    y_curr=y;
    move_robot(x_curr,y_curr,z_curr,g_curr);
}

/* Subroutine left is used to move the RTX in the +Y-direction. */

void left()
{
    int x,y,z,g;
    y=y_curr+stepsize;
    y_curr=y;
    move_robot(x_curr,y_curr,z_curr,g_curr);
}

/* Subroutine forward is used to move the RTX in the +X-direction. */

void forward()
{
    int x,y,z,g;
    x=x_curr+stepsize;
    x_curr=x;
    move_robot(x_curr,y_curr,z_curr,g_curr);
}
/* Subroutine backup is used to move the RTX in the -X-direction. */

void backup()
{
    int x,y,z,g;
    x=x_curr-stepsizen;
    curr=x;
    move_robot(x_curr,y_curr, z_curr, g_curr);
}

/* Subroutine opengrip is used to open the RTX gripper. */

void opengrip()
{
    g_curr=50;
    move_robot(x_curr,y_curr, z_curr, g_curr);
}

/* Subroutine closegrip is used to close the RTX gripper. */

void closegrip()
{
    g_curr=0;
    move_robot(x_curr,y_curr, z_curr, g_curr);
}

/* Subroutine stepup sets the stepsizen to 22 mm for movement in x, y, z. */

void stepup()
{
    stepsizen=22;
}

/* Subroutine stepdown sets the stepsizen to 3 mm for movement in x, y, z. */

void stepdown()
{
    stepsizen=3;
}
Appendix C

crbot.c Source Code Listing

This is the source code for crbot.c. The program performs the run-time routines necessary for the using the TI-Speech system. It must be linked with the speech recognition program, spkrdep.c.

/*
 Program crbot.c

 By William T. Atkinson
 21 January 1991

 This program is designed to be used with spkrdep.c to control the RTX robot. Compile and link it with spkrdep.c and speechms.lib.

*/

#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#include <dos.h>
#include <malloc.h>
#include "global.h"
#include "utili.h"

int dsrblk[17];
int ioOff = 0;
int ioVol = 5;
int ioDevice = 0;
int ioGain = 5;

/*****************************/

/* Subroutine StartSpeech checks to insure that tispeech.com has been installed. */

int StartSpeech()
{
    int system,dsr,status,brdtype;

    /* printf("\nChecking for speech board to use: ");
        for (board=0; board <= 13; board++)
            */
{  
printf(" %d", board);

	/* speech_config(board,&system,&dsr,&brdtype,&status);

	/* if (status == 0) break;

	}

	if(status == 0xE2) printf("nPlease install TISPEECH.COM
");
	if (status) sError("speech_config", status);
	printf("nUsing TI-Speech board number %1d
", board);
	/* Reset (HARD) the speech board */
	dsrblk[0] = (char) board + (0x0A << 8);

call_dsr(dsrblk);
}

/*------------------------------------------------------------------------------------------*/

/* Subroutine StopSpeech is used to exit the speech system and return to DOS. */

int StopSpeech(exit_code)
int exit_code;
{
    printf("nCleaning up ...");
    gain(board,SET,&ioOff,&status); /* reset to no i/o channels */
    channels(board,SET,&ioOff,&ioOff,&status);
    printf("n returning to DOSn"); /* inform user */
    exit(exit_code);
}

/*------------------------------------------------------------------------------------------*/

/* Subroutine sError returns a hexadecimal error code in case of an error.
   The meaning of the error code is in Appendix C of the TI-Speech Application Toolkit
   manual. */

int sError(message,error)
char *message;
int error;
{
    printf("%s error: %4x (hex)n",message,error);
    StopSpeech(-1);
    return(-1);
}
/* Subroutine LoadAlgorithm loads a speech algorithm from the disk onto the speech
board RAM. An application must use this subroutine before the system can use text-to-
speech, synthesis, analysis, recognition, or verification routines. */

int LoadAlgorithm(filename) /* Load TI-Speech algorithm if not downloaded */
char *filename;
{
  static struct algs
  {
    char *algnme;
    int algcde;
  }

  algorithms[] = { "SYNANAL.IMG", 0x30,
                   "SYN96.IMG", 0x22,
                   "ANA96.IMG", 0x12,
                   "SYNANA16.IMG", 0x34,
                   "ADPCM.IMG", 0x36,
                   "TTS.IMG", 0x28,
                   "LWR.IMG", 0x60,
                   "CALIB.IMG", 0x80,
                   "LWRIC.IMG", 0x42,
                   "VENROLL.IMG", 0x64,
                   "LR2000.IMG", 0x46,
                   "VERIFY.IMG", 0x48,
                   "SIDR.IMG", 0x4A,
                   

  int i, status, code, mode, aready, sready, alg_found, lcode;
char datname[20];
char *spchbuf;
for (alg_found=0, i=0; algorithms[i].algcode != 0; i++)
{
  if (strcmp(filename, algorithms[i].algnme) == 0)
  {
    alg_found = 1;
    lcode = algorithms[i].algcode;
    break;
  }
}

/* see if filename is valid */
if(alg_found == 0)
{
  printf("%s not in programs algorithm table ", filename);
  sError("LoadAlgorithm", 0);
}
/* see if algorithm is already there */
if (speech_status(board,&code,&mode,&aready,&sready,&status))
    sError("Speech_status",status);

if (lcode == code) return(0);

/* special case for synthesis */
if ( ( (code==0x60) || (code==0x64) ) && (lcode == 0x30) ) return(0);

spchbuf = (char *) malloc(8200);
if (spchbuf == NULL) sError("malloc spchbuf ", 0);

/*
   The majority of this check deals the future extension to include connected-speech,
   continuous speech, and/or speaker-independent digit recognition.
*/

if (load_speech(board,filename,spchbuf,&status) == 0)
{
    if (strcmp(filename,"SIDR.IMG") == 0)  strcpy(datname,"SIDR.DAT");
    else if (strcmp(filename,"VERIFY.IMG") == 0)  strcpy(datname,"VERIFY.DAT");
    else if (strcmp(filename,"LR2000.IMG") == 0)  strcpy(datname,"LR2000.DAT");
    else if (strcmp(datname,""));
    if(strcmp(datname,"") != 0) load_speech(board,filename,spchbuf,&status);
}

free(spchbuf);

if(status)
{
    printf("%s ",filename);
    sError("Load_speech",status);
}

/*************************************************************************/

/* Subroutine SetOutput examines or adjusts the audio output level (volume) of the speech
system. */

int SetOutput()
{
    int status;
    char tone;
    if (gain (board,SET,&ioOff,&status)) sError("Gain",status);
    if (volume (board,SET,&ioVol,&status)) sError("Volume",status);
    if (channels(board,SET,&ioOff,&ioDevice,&status)) sError("Channels",status);
    if (ioDevice == 1)
{  
    if(phone_mode(board, APL_MODE, &status)) sError("Phone Mode",status);  
    tone = '\'';  
    do  
    {  
        read_tone(board, &tone, &status);  
    }  
    while (tone != '\''); /* ignore errors if phone brd not installed */
}

/*============================================================================*/

/* Subroutine SetInput examines or sets the gain value of the input channel. The gain  
   determines the amplification of the input signal. */

int SetInput()  
{  
    int status;  
    char tone;  

    if (ioDevice !=1)  
    {  
        if (volume(board,SET,&ioOff,&status)) sError("Volume",status);  
        if (gain(board,SET,&ioGain,&status)) sError("Gain",status);  
    }  
    if (channels(board,SET,&ioDevice,ioOff,&status)) sError("Channels",status);  

    tone = '\'';  
    do { read_tone(board,&tone,&status);  
    } while (tone != '\''); /* ignore errors if phone brd not installed */
}

/*============================================================================*/

/* Subroutine SetSpeechIO selects the input and output channels. */

int SetSpeechIO()  
{  
    void readABORT();  
    int status;  
    char c;  
    c = 'M';  
    printf("nSelect device: M(icrophone/Speaker H(eadset");  
    printf(" ["%c]; ",c);  
    GetLine(c',&c);

switch(c)
{
    /*
     * case 'L':
     *    if(phone==0) break;
     *    ioDevice = 1;  * Telephone input *
     *    ioGain = 0;   * Turn off the microphone gain *
     *    SetOutput();  * Return to the caller *
     *    return(0);
     */
    case 'M':
    case 'S':
        default: ioDevice = 4;  /* Microphone input and speaker output */
        break;
    case 'H': ioDevice = 8;  /* Headset input and output */
        break;
}

    /* Speak a voice file so user can press F3/F4 to set output volume */
    LoadAlgorithm("SYNANAL.IMG");
    SetOutput();
    printf("\nSelect volume: F3 Louder  F4 Softer  O(k C(an't hear anything\n");
    while(1)
    {
        play_file(board,TTONE_opt,KEYBD_opt,"SETVOL.VOC",&status);
        if (status == KEYHIT) break;
        if (status) sError("Play_file",status);
    }

    c = getch();
    if (toupper(c) == 'C')
    {
        printf("\nMake sure your speaker or headset is correctly installed\n");
    }

    /* Get user selected volume and prompt for confirmation */
    if (volume(board,GET,&ioVol,&status)) sError("Volume",status);
    printf("Recommended volume [%d] -", ioVol);
    GetLine(d', ioVol);
    if(ioVol>15)
    {
        printf("\nVolume out of range. Using 15.\n");
        ioVol=15;
    }

    /* Speak a voice prompt, then listen to get a recommended input gain */
    do
    {
        SetOutput();
        play_file(board,TTONE_opt,KEYBD_opt,"SETGAIN.VOC",&status);
        if (status == KEYHIT) readABORT();
        else if (status) sError("Play_file", status);
        printf("press RETURN and read the following sentence:\n");
    ...
readABORT();
LoadAlgorithm("CALIB.IMG");
SetInput();
printf("This is the way I will talk to the speech board during...
\n\n");
calibrate(board,8,&ioGain,&status);
if (status == KEYHT) readABORT();
else if (status) sError("Calibrate", status);

/* Get user set gain and prompt for confirmation */
c='N';
if (ioGain>=12)
{
    printf("High gain value detected. Recalibrate? - [N] -");
    GetLine('c',&c);
}

LoadAlgorithm("SYNANAL.IMG");
}

while (c == 'Y');
printf("Recommended gain [%d] -",ioGain);
GetLine('d',&ioGain);
if(ioGain>15)
{
    printf("nGain out of range. Using 8.n");
ioGain=8;
}

/* Subroutine readABORT */

void readABORT()
{
    int c;
    c = getch();
    if(c == 0) getch();
}

/* Subroutine GetLine */

int GetLine(c,v)
char c,*v;
{
    char *p,inbuff[82];
    fflush(stdout);

}
p=inbuf;
while((*(p=getchar())!="\n") if (*p!='\b') p++; else if(p>inbuf) p--;
*p="\0";
if (p==inbuf) return;
switch(c)
{
    case 'c': *v = toupper(inbuf[0]);
      break;
    case 'd': sscanf(inbuf, "%d", v);
      break;
    case 's': strcpy(v,inbuf);
      break;
}
}
Appendix D

spkrdep.h Header File

The file spkrdep.h is used to establish the vocabulary structure for the speech recognition program. It stores the words that are used for training (input) and the actions that correspond to each word (output). After this vocabulary is trained, it is stored in memory.

```c
struct vocab {
    int num;
    char **template;
    char **response;
    char buf[8200];
    int index;
};

struct vocab user,robo;

char *usrfun[12] =
{"up", "down", "right", "left", "front", "back", "open", "close", "faster", "slower", "quit", "switch"};

char *output[12] =
{"up", "down", "right", "left", "forward", "backup", "opengrip", "closegrip", "stepup", "stepdown", "exit", "change"};

char *username[2] = {"Bill", "New_user"};

char *globalusr[2] = {"Bill", "New_user"};

char *ansusr[2] = {"Hi Bill.", "Hi Newbee."};

int robo_num = 12, user_num = 1;
```
Appendix E

global.h Header File

The code for global.h defines the values for variables that are called by various routines and functions within the TI-Speech system.

#define APL_MODE 1
#define board 0
#define FALSE 0
#define GET 0
#define KEYHIT 0xD0  /* return status for keyboard abort */
#define TTONEABORT 0xC1  /* return status for touch tone abort */
#define kybd_opt 1  /* enable function keys and keyboard abort */
#define PLAY 1
#define RECORD 0
#define SET 1
#define STN_MODE 0
#define SYN_ACTIVE 5  /* synthesis is active flag */
#define TRUE 1
#define tone FALSE  /* do not abort on touch tone */
#define NO_KB_ABORT FALSE
#define TTONE_opt 1  /* TRUE*/
#define KEYBD_opt 1  /* TRUE*/
Appendix F
utili.h Header File

This file, utili.h, serves as the utility file for the TI-Speech system programs.

extern int CLS();
extern int Connected();
extern int EnableVocab();
extern int FlushTones();
extern int Get_Line();
extern char *get_mem();
extern int InitTPK();
extern int LoadAlgorithm();
extern int SetDevice();
extern int SetGain();
extern int SetInput();
extern int SetIO();
extern int SetOutput();
extern int SetParms();
extern int SetVolume();
extern int sError();
extern int StartSpeech();
extern int StopSpeech();

/process Global variables/

extern char filename[82];
extern char algo[16];
extern char strings[82];
extern char TTSstring[82];
extern char TTSfile[82];
extern char spchbuf[8200];
extern char clearscreen[5];

extern unsigned int valid_ids[4];

extern int active;
extern int brdtype;
extern int code;
extern int init;
extern int inn;
extern int io;
extern int iDevice,iOff; /* input/output device selection */
extern int iGain;
extern int iVol;
extern int keyopt;
extern int line_silence;
extern int mode;
extern int off;
extern int out;
extern int rings;
extern int ring_count;
extern int status;
extern int tpkchan;
extern int vocab;
extern int voice_silence;

extern char indev;
extern char ioChannel;
extern char out_chan;

/ *----- general procedure declarations -----*/

/* extern char *malloc(); */
extern char *strcpy();
extern char *strlen();
extern int khit();

/ * Speech runtime routine declarations */

extern int answer_phone();
extern int calibrate();
extern int call_dsr();
extern int channels();
extern int connect_phone();
extern int dial_phone();
extern int enroll();
extern int gain();
extern int hangup_phone();
extern int halt_speech();
extern int load_speech();
extern int load_template();
extern int phone_mode();
extern int phone_status();
extern int play_file();
extern int play_masync();
extern int play_mem();
extern int play_tone();
extern int playrec_parms();
extern int read_tone();
extern int recognize();
extern int record_msg();
extern int save_template();
extern int speak_file();
extern int speak_text();
extern int speech_status();
extern int tpk_active();
extern int tpk_change();
extern int tpk_disable();
extern int tpk_enable();
extern int tpk_vocab();
extern int update();
extern int volume();
Appendix G

commander.bat Code Listing

This routine, commander.bat, sets the communications mode for the signal leaving the IBM PC with the TI-Speech board (i.e., the "commander" PC). It is also used to run the start-up routines for the TI-Speech system.

mode com1 1200,n,8,1,p
cd tispeech
tispeech
tts
Appendix H

slave.c Source Code Listing

The program slave.c sets the communications mode for the signal received by the second IBM PC (the "slave" PC) and for the signal sent to the RTX from the slave PC. It also prints to screen the current position of the robot arm and the status of the gripper.

/* Program slave.c */

#include <stdio.h>
#include <gf.h>
#include <asiports.h>

extern int ok;

main()
{
    int status, count;
    unsigned mode = ASINOUT\IBINARY\INORMALRX;
    char str[600];
    int value, x, y, z, gr;
    int rval = 0;

    if((status = asiopen(COM2, mode, 5000, 5000, 1200, P_NONE, 1, 8, ON, ON)) != ASSUCCESS)
    {
        printf("com port init failed\n");
        exit(status);
    }

    value = arm_prepare();
    if(value)
    {
        printf("-1\n:Library not installed %d :\n", value);
        exit(0);
    }

    printf("0\n:Speech Slave Ready for Commands\n");
while(!kbhit())
{
    for(count=0; count<80; count++)
    {
        while( getrxcnt(COM2) < 1 )
            if(kbhit())
                exit(0);

        str[count] = asigetc(COM2);

        if (str[count]==13)
        {
            str[count]=0;
            break;
        }
    }
}

sscanf(str, "%d%d%d%d", &x, &y, &z, &gr);
printf(":: %d:%d:%d ::\n", x, y, z, gr);
rv all=arm_move_cartesian(x, y, z, 0, -90, 0, gr, 10, 1);
if(rval)
{
    printf("\ERROR IN KINEMATICS [%x]\n", rval);
    break;
}
}
Appendix I

rtx3.c Source Code Listing

This is the source code used to perform the inverse kinematics needed to run the RTX robot. It was written by Sarmad Adnan and is reprinted with his permission.

/*

RTX inverse kinematics solution
(high level part of the set to)
(use with rtx.lib & rtx.h)

for the
MicroSoft C3.00 Compiler

COPYRIGHT october 23rd 1987 by
SARMAD ADNAN
284-84-1489
(713) 630-9334

RICE UNIVERSITY
Houston, TX 77030

arm_move_cartesian is the routine that the user should call to move the RTX in world coordinates the routine arm_prepare() must be called once at the start of a program in order to initialize the arm, the parameters to arm_move_cartesian are all integers, the origin lies at the base of the vertical track. x axis directly away from the RTX the y axis to the left and the zee upwards, orientations of the gripper is defined follows, roll is the angle in degrees that the gripper makes around its axis (positive clockwise),pitch is the inclination angle of the gripper from the horizontal (negative downward), yaw is the angle the whole wrist assembly makes with the x axis (negative clockwise) grp is the gripper opening (0 closed). the pwx, pwy and pwz define coordinates of the wrist point.
LIMITS
0 < pwz < 940 m.m
0 < pwx^2+pwy^2 < 508^2 m.m
-110 < yaw < 110 degrees
0 < grp < 85 m.m
-98 < pitch < 4 degrees
-132 < roll < 180 degrees
0 < speed < 160

care must be taken in moving downwards on
the zee axis considerable overshoot occurs
raising the possibility of the arm hitting
the end-stop or floor at high speed. "ok"
is the global variable defined for error-
detection by the lower level routines this
may be checked by the user to determine
the current error status of the arm these
error codes are listed in the RTX manuals.
errors may also however occur due to the
specification of impossible or singular
configurations. or positions outside the
working envelope these errors are flagged
by the integer returned by this function
error_code = arm_move_cartesian( );

ERRORS CODES
error_code bit 0  zee coordinate error
error_code bit 1  shoulder error
error_code bit 2  elbow error
error_code bit 3  roll angle error
error_code bit 4  pitch angle error
error_code bit 5  yaw angle error
error_code bit 6  gripper range error
error_code bit 7  initialization error

(warning: this program prefers a lefty
configuration of the RTX, however if a
point isn't accessable in this way then it
automatically tries to access the point in
a righty configuration, this may a concern
in your applications.)

*/

#include <math.h>
#define ABS(u) ( (u<0) ? (-(u)) : (u) )

extern int parm[8];
extern int ok ;
int arm_prepare()
{
    int ret_code=0;

    ok = 0;
    arm_init_comms(1,0,&ok);
    arm_reload_pids(&ok);
    arm_stop(3,&ok);
    if(ok != 0)
        ret_code = 128;
    return(ret_code);
}

int arm_move_cartesian(pwx,pwy,pwz,roll,pitch,yaw,grp,speed,precision)
int pwx,pwy,pwz,roll,pitch,yaw,grp,speed,precision;
{
    int limit,th2,th3,z,w1,w2,y,arm_solve_inverse(),arm_move_joints();
    int gr, arm_calc_grip(),maxspeed = 160;

    if(ok == 0)
    {
        th2=pwx;
        th3=pwy;
        z =pwz;
        w1=roll;
        w2 =pitch;
        y =yaw;
        gr = grp;

        limit = arm_solve_inverse(&th2,&th3,&z,&w1,&w2,&y);
        limit = limit + arm_calc_grip(&gr);
        if(limit == 0)
        {
            if((speed>maxspeed)|| (speed<0))
                speed = maxspeed;
            arm_move_joints(th2,th3,z,w1,w2,y,gr,speed);
            arm_wait_if_busy(precision);
        }
    }
    return(limit);
}
int arm_solve_inverse(th2, th3, z, w1, w2, y)
int *th2, *th3, *z, *w1, *w2, *y;
{
    int temp, temp1;
    double phi, yy, xx, r, theta, theta2, theta3, pitch, roll, yaw, hold;
    int retval = 0;
    if((*th2==0)&&(0==0)) /* the atan2 function gives */
        /* th3 = 1;               /* a domain error at x=y=0 */
        xx = *th2;
        yy = *th3;
        r = xx * xx + yy * yy;
        r = sqrt(r);
        if(r > 507.0) /* out of reach */
            return(8);
        temp = *z * 3.7495 - 3555; /* 3.7495 is zed scale */
        if((temp > 0) || (temp < -3554)) /* zed not possible */
            retval = 1;
        phi = atan2(yy, xx);
        theta = acos(r / 507); /* 507 = 2 * 253.5 link length */
        theta2 = phi + theta; /* lefty configuration by default */
        theta3 = -2 * theta;
        if((theta2 > 1.5707) || (theta2 < -1.5707)) /* if lefty not possible*/
            theta2 = phi - theta; /* try righty configuration */
        theta3 = 2 * theta;
        if((theta2 > 1.5707) || (theta2 < -1.5707)) /* if righty not possible*/
            retval = retval + 2; /* send error code */
    }
    if((theta3 > 2.635) || (theta2 < -3.1415)) /* cant bend elbow there */
        retval = retval + 4;
    if( (*w2 > 4) || (*w2 < -98) )
        retval = retval + 16;
    if( (*w1 > 181) || (*w1 < -132) )
        retval = retval + 8;
    pitch = (double)(w1 + w2) / 0.07415; /* roll and pitch are calculated*/
    roll = (double)(w2 - w1) / 0.07415; /* as a function of the w1 & w2 */
    theta = 558.05 * ((double)*y/57.29577) - theta2 - theta3 / 2;
    yaw = (theta3 * 279.056) + 1071;
    if( (theta > yaw) || (theta < (yaw - 2072)) )
        retval = retval + 32;
if(retval == 0)
{
  *th2 = theta2 * 1674.183;
  *th3 = theta3 * 837.168;
  *z  = temp;
  *w1  = roll;
  *w2  = pitch;
  *y  = theta;
}
return(retval);

int arm_calc_grip(gr)
int *gr;
{
  int ret = 0;
  double temp1;
  temp1 = *gr;
  if((*gr < 0) || (*gr > 85))
    ret = 64;
  temp1 = sqrt(0.00342 + 0.0000428 * temp1) - 0.0584;
  *gr = temp1;
  return(ret);
}

int arm_move_joints(a,b,cc,dd,e,f,g,h)
int a,b,cc,dd,e,f,g,h ;
{
  int status,motor ;
  int limit,th2,th3,z,w1,w2,y;
  int th2o,th3o,z0,w1o,w2o,y0,gro;
  int pwxsp,pwysp,pwzsp,pw1sp,pw2sp,pysp,pgosp;
  int t[8],s[8],c[8],count,dmax,d[8],r[8];
  int i,j,sum,temp;
  double store;
  r[1]=a; r[0]=b; r[2]=cc;
  r[6]=g; r[7]=h;
if(ok != 0)
    return(0xFF00);

arm_current_encoders(&c[1],&c[0],&c[2],&c[3],&c[4],&c[5],&c[6]);

for(i=0;i<=6;++i)
    d[i] = ABS(r[i]-c[i]);

d[2]+=d[2];
dmax=1;
for(i=0;i<=6;++i)
    if( d[i] > dmax ) dmax=d[i];

for(i=0;i<=6;++i)
    {
    store=r[7];
    store/=dmax;
    store+=d[i];
    s[i] = ABS(store);
    s[i] = (s[i]==0) ? (1) : (s[i]) ;
    }

for(i=0;i<=6;++i)
    {
    arm_write(i,4,s[i],&ok);
    arm_write(i,3,r[i],&ok);
    }
arm_go(1,0x1555 ,&ok);

int arm_wait_if_busy(precision)
int precision;

{
    int motor,sum=1 ;
    int ce[7] ;
    int cd[7] ;
    int status = 1;

    if(precision==0)
    {
    }
else if(precision==1)
    {
    while((status & 1) != 0)
        arm_general_status(&status,&ok) ;
    arm_stop(0,&ok);
    }
else
    {
    while((status & 1) != 0)
        arm_general_status(&status,&ok) ;
while(sum!=0)
{
    for(motor=0;motor<=6;++motor)
    {
        arm_read(motor,0,&ce[motor],&ok);
        arm_read(motor,8,&cd[motor],&ok);
        ce[motor] &= 0x3FFF; /*kill 2 high bit*/
        sum+=((ABS(cd[motor])-ABS(ce[motor])) < 0) ? (1) : (0);
    }
    arm_stop(0,&ok);
}

int arm_move_wristpoint(pwx,pwy,pwz,speed,precision)
int pwx,pwy,pwz,speed,precision ;
{
    int limit,th2,th3,z,arm_solve_inverse1(),arm_move_joints1();
    int maxspeed = 160 ;

    if(ok == 0)
    {
        th2=pwx;
        th3=pwy;
        z =pwz;

        limit = arm_solve_inverse1(&th2,&th3,&z);
        if(limit == 0)
        {
            if((speed>maxspeed)||(speed<0))
            {
                speed = maxspeed ;
                arm_move_joints1(th2,th3,z,speed);
                arm_wait_if_busy1(precision);
            }
        }
        return(limit) ;
    }

    int arm_solve_inverse1(th2,th3,z)
int *th2,*th3,*z;
{
    int temp, temp1 ;
    double phi,yy,xx,r,theta,theta2,theta3,pitch,roll,yaw,hold ;
    int retval = 0 ;
}
if(*th2==0) & & (*th3==0)) /*the atan2 function gives */
  *th3=1; /*a domain error at x=y=0 */
x = *th2;
xx = xx;
**
y = *th3;
**
r = xx * xx + yy * yy;
**
r = sqrt(r);
**

if(r > 507.0) /*out of reach */
  return(8);
  temp = *z * 3.7495 - 3555; /*3.7495 is zed scale */
  if((temp > 0) || (temp < -3554)) /*zed not possible */
    retval = 1;
  phi = atan2(yy,xx);
  theta = acos(r / 507); /*507 = 2 * 253.5 link length */
  theta2 = phi + theta; /*lefty configuration by default */
  theta3 = -2 * theta;
  if((theta2 > 1.5707 || (theta2 < -1.5707)) /*if lefty not possible*/
    {
      theta2 = phi - theta; /*try righty configuration */
      theta3 = 2 * theta;
      if((theta2 > 1.5707) || (theta2 < -1.5707)) /*if righty not possible*/
        retval = retval + 2; /*send error code*/
    }
  if((theta3 > 2.635) || (theta2 < -3.1415)) /*cant bend elbow there */
    retval = retval + 4;
}

if(retval == 0)
{
  *th2 = theta2 * 1674.183;
  *th3 = theta3 * 837.168;
  *z = temp;
}
return(retval);
}

int arm_move_joints1(th2,th3,z,speed)
int th2,th3,z,speed ;
{
  int status,motor ;
  arm_stop(3,&ok);
  if(ok == 0)
    {
      for(motor=0;motor<=2;++motor)
        arm_write(motor,4,speed,&ok);
    }
arm_write(0,3,th3,&ok);       /*feeding encoder counts into */
arm_write(1,3,th2,&ok);       /*individual motors */
arm_write(2,3,z ,&ok);
arm_go(1,0x15 ,&ok);         /*elbow,shoulder, zed motors go*/
}

int arm_wait_if_busy1(precision)
int precision;
{
int motor ;
int current_error = 100 ;
int current_deadband = 0 ;
int status0, status1, status2, status = 1 ;

if(precision==0)
    {} else if(precision==1)
    {
        while((status & 1 ) != 0 )
        {
            arm_motor_status(0,&status0,&ok ) ;
            arm_motor_status(1,&status1,&ok ) ;
            arm_motor_status(2,&status2,&ok ) ;
            status=( 0x01 & (status0 || status1 || status2));
        }
        arm_stop(0,&ok ) ;
    }
else
    {
        while((status & 1 ) != 0 )
        {
            arm_motor_status(0,&status0,&ok ) ;
            arm_motor_status(1,&status1,&ok ) ;
            arm_motor_status(2,&status2,&ok ) ;
            status=( 0x01 & (status0 || status1 || status2));
        }
        for(motor=0;motor<=2;++motor)
        {
            while((current_error>current_deadband) || (ok != 0 ) )
            {
                arm_read(motor,0,&current_error,&ok ) ;
                arm_read(motor,8,&current_deadband,&ok ) ;
                current_error = (0x3FFF & current_error ) ; /*kill 2 high bit*/
            }
            arm_stop(0,&ok ) ;
        }
    }
int arm_move_wrist(roll, pitch, yaw, speed, precision)
int roll, pitch, yaw, speed, precision;
{
    int limit, th2, th3, z, w1, w2, y, arm_solve_inverse2(), arm_move_joints2();
    int maxspeed = 160;

    if(ok == 0)
    {
        w1 = roll;
        w2 = pitch;
        y = yaw;
        limit = arm_solve_inverse2(&w1, &w2, &y);
        if(limit == 0)
        {
            if((speed > maxspeed) || (speed < 0))
            {
                speed = maxspeed;
                arm_move_joints2(w1, w2, y, speed);
                arm_wait_if_busy2(precision);
            }
        }
        return(limit);
    }
}

int arm_solve_inverse2(w1, w2, y)
int *w1, *w2, *y;
{
    int temp, temp1, th2, th3;
    double phi, yy, xx, x, theta, theta2, theta3, pitch, roll, yaw, hold;
    int retval = 0;

    arm_read(0, 1, &th3, &ok);
    arm_read(1, 1, &th2, &ok);
    theta2 = (double)th2 * 0.00059725;
    theta3 = (double)th3 * 0.00119450;

    xx = (253.5*(1+cos(theta3))*cos(theta2));
    yy = (253.5*(1+cos(theta3))*sin(theta2));
    r = xx * xx + yy * yy;
    r = sqrt(r);
    if((yy == 0) && (xx == 0)) yy = 1;
    phi = atan2(yy, xx);

    theta = acos(r / 507);  /* 507 = 2 * 253.5 link length */
    if(*w2 > 4) || (*w2 < -98)
        retval = retval + 16;
if( (*w1 > 181) || (*w1 < -132) )
    retval = retval + 8;
pitch = (double)(*w1 + *w2) / .07415; /**< roll and pitch are calculated*/
roll = (double)(*w2 - *w1) / .07415; /**< as a function of the w1 & w2 */
theta = 558.05 * (((double)*y/57.29577) - theta2 - theta3/2);
yaw = (theta3 * 279.056) + 1071;

if( (theta > yaw) || (theta < (yaw -2072)) )
    retval = retval + 32;

if(retval == 0)
    {
        *w1 = roll;
        *w2 = pitch;
        *y = theta;
    }
return(retval);


int arm_move_joints2(w1,w2,y,speed)
int w1,w2,y,speed ;
{
    int status,motor ;

    arm_stop(3,&ok);
    if(ok == 0)
    {
        for(motor=3;motor<=5;++motor)
            arm_write(motor,4,speed,&ok);

        arm_write(3,3,w1,&ok); /**<feeding encoder counts into */
        arm_write(4,3,w2,&ok); /**<indivisual motors */
        arm_write(5,3,y,&ok);
        arm_go(1,0x540 ,&ok); /**<roll,pitch,yaw motors go*/
    }
}

int arm_wait_if_busy2(precision)
int precision;
{
    int motor;
    int current_error = 100;
    int current_deadband = 0;
    int status3, status4, status5, status = 1;

if(precision == 0)
{
}
else if(precision == 1)
{
    while((status & 1) != 0)
    {
        arm_motor_status(3, &status3, &ok);
        arm_motor_status(4, &status4, &ok);
        arm_motor_status(5, &status5, &ok);
        status = ((status3 | status4 | status5));
    }
    arm_stop(0, &ok);
}
else
{
    while((status & 1) != 0)
    {
        arm_motor_status(3, &status3, &ok);
        arm_motor_status(4, &status4, &ok);
        arm_motor_status(5, &status5, &ok);
        status = ((status3 | status4 | status5));
    }

    for(motor = 3; motor <= 5; ++motor)
    {
        while((current_error > current_deadband) || (ok != 0))
        {
            arm_read(motor, 0, &current_error, &ok);
            arm_read(motor, 8, &current_deadband, &ok);
            current_error = (0x3FFF & current_error);/* kill 2 high bit*/
        }
        arm_stop(0, &ok);
    }
}

int arm_move_grip(grp, speed, precision)
int grp, speed, precision;
{
    int limit, arm_move_joints();
    int gr, arm_calc_grip(), maxspeed = 160;

    if(ok == 0)
    {
        gr = grp;
limit = arm_calc_grip(&gr);
if(limit == 0)
{
    if((speed>maxspeed)||(speed<0))
        speed = maxspeed;
    arm_move_joints3(gr,speed);
    arm_wait_if_busy(precision);
}
return(limit);

int arm_move_joints3(g,speed)
int g,speed ;
{
    int status ;
    arm_stop(3,&ok) ;
    if(ok == 0)
    {
        arm_write(6,4,speed,&ok) ;
        arm_write(6,3,g ,&ok);
        arm_go(1.0x1000 ,&ok); /*all motors go, also gripper*/
    }
}

int arm_current_encoders(th2o,th3o,zo,w1o,w2o,yo,g)
int *th2o,*th3o,*zo,*w1o,*w2o,*yo,*g;
{
    int arm_read();
    int th2,th3,z,w1,w2,y,gr;
    arm_read(0,1,&th3,&ok);
    arm_read(1,1,&th2,&ok);
    arm_read(2,1,&z ,&ok);
    arm_read(3,1,&w1 ,&ok);
    arm_read(4,1,&w2 ,&ok);
    arm_read(5,1,&y ,&ok);
    arm_read(6,1,&gr ,&ok);

    *th2o = th2 ;
    *th3o = th3 ;
    *zo = z ;
    *w1o = w1 ;
    *w2o = w2 ;
    *yo = y ;
    *g = gr ;
}
Appendix J

rtx.h Header File

This header file is used with the program rtx3.c to perform the inverse kinematics needed to run the RTX robot. It was written by Sarmad Adnan and is reprinted with his permission.

/* include file for the function declarations for the C Language link to the RTX's RAM resident library. The library must be active before any calls may be made to it through this suite of functions

COPYRIGHT 17th OCT 1987 SARMAD ADNAN

author = sarmad adnan
1st edit = 17th Oct 1987
2nd edit = 20th Oct 1987
*/

extern rtx_int(int);

/*constants */
#define on -1
#define off  0
#define no_error 0
#define passing 0
#define theoretical 1
#define accurate 2

/*motor names */
#define elbow 0
#define shoulder 1
#define zed 2
#define wrist1 3
#define wrist2 4
#define yaw 5
#define grip 6
#define zedown 7

/*toggle names */
#define toggle_off 0
#define toggle_on 1
/*motor_mode names */
#define position_mode 0
#define force_mode 1
#define absolute_mode 2
#define relative_mode 3
#define user_input 4
#define user_output 5

/*@stop_mode names */
#define dead_stop 0
#define ramp_stop 1
#define free_stop 2
#define free_off 3

/*@go_mode names */
#define manual 0
#define numeric 1

/*@data_code names */
#define error_data 0
#define current_position_data 1
#define error_limit_data 2
#define new_position_data 3
#define speed_data 4
#define kp_data 5
#define ki_data 6
#define kd_data 7
#define dead_band_data 8
#define offset_data 9
#define max_force_data 10
#define current_force_data 11
#define acceleration_time_data 12
#define user_ram_data 13
#define user_io_data 14
#define actual_position_data 15

/*@soak names */
#define init 0
#define soak_on 1
#define init_soak 2
#define soak_off 3

/*@global variables */
int parm[8];
int ok ;
int lib_check = 0 ;
Appendix K

Diagram of the RTX Robot

Figure K1, Diagram of the RTX Robot, is on the next page. It shows three different views of the RTX robot. The diagram was drawn by Sarmad Adnan and is reprinted with his permission.
Figure K1

Diagram of the RTX Robot