Towards Voice-Guided Robotic Manipulator Jogging

Sebastian van Delden and Benjamin Overcash
Division of Mathematics and Computer Science
University of South Carolina Upstate
Spartanburg, SC 29316
{svandelden, beovercash}@uscupstate.edu

Abstract *— Initial steps toward designing and implementing a voice-guided approach to robotic manipulator jogging are outlined in this paper. Even though robotic teach pendants continue to become more light weight and easy to use, they will always require an operator to have at least one hand occupied by the teach pendant during the development of an automation task. We are developing an intuitive hands-free approach to manipulator jogging and application development that translates English voice commands into manipulator movements and program statements. The system is being implemented and tested on a Stäubli RX60 manipulator and freely available voice recognition packages.*

Keywords *— Speech Recognition, Robotics, Automation, Natural Language Processing.*

I. Introduction

The use of robotic manipulators in factory automation is commonplace. "Automate or Evaporate" is the saying in the manufacturing world. Industrial robotic systems improve productivity by increasing throughput and enhancing the quality of manufactured goods. It is amazing that robotic automation has existed since the early 1960s - long before the age of personal computers, the internet and email. Industrial robotic tasks include welding, material handling, and product assembly.

Input sensors are often utilized to bring flexibility to and enhance the capabilities of a robotic system. For example, camera systems can assist robots in performing quality control by visual inspection, or guiding a manipulator's end-effector to a desired pose. Pressure and proximity sensors are also commonly use to determine the location of a part, a tool or the end-effector. Audio input sensors, however, have so far rarely been used in robotic automation.

Even though robotic teach pendants continue to become more light weight and easy to use, they will always require an operator to have at least one hand occupied by the teach pendant during the development of an automation task. We are developing an intuitive hands-free approach to manipulator jogging and application development that translates English voice commands into manipulator movements and program statements.

Even though there exists a vast array of robotics journals, conferences and workshops, and thousands of published works on numerous aspects of robotic automation, the literature contains very few contributions on voice-guided manipulator jogging. [1] describes some initial work on a voice guided system which was later integrated into a hand gesture recognition system [2] so that the robot could be guided using both audio and visual commands. [3] is most closely related to this work and provides several details on that particular implementation. There are also some recent articles in medical literature on commercial voice assisted robots used in laparoscopic [4] and endoscopic [5] surgeries. Some of the reasons for the lack of voice guided manipulator technology include:

- Original industrial systems were not design to incorporate voice technology into their controller.
    - This however is rapidly changing as very sophisticated robotics programming environments are emerging, for example: Stäubli's *Robotics Studio* software and VAL3 programming language, FANUC's *Proficy*, and ABB's *Robot Application Builder*.
- Controllers and robotic environments are typically noisy
    - This is an unavoidable problem, however, modern microphones and voice recognition technology do an adequate job in filtering out background noise.
- Readily available voice recognition packages are still relatively new and not completely reliable.
    - However, several free systems are currently available that do an adequate job recognizing commands in a closed domain once properly trained by their user.

Voice recognition software can be somewhat unreliable when used across an open domain by numerous users for which it as not trained. However, because this system will be use for factory automation tasks by one operator (or perhaps a small group of operators), very accurate voice recognition can be achieved on this closed domain if the system is properly trained.

In this paper, we examine how an audio input sensor can enhance robotic automation tasks by describe the initial steps towards designing and implementing a voice guided system for jogging a robotic manipulator. In particular, we provide a detailed overview, including an informal qualitative and quantitative analysis, of the two speech recognition packages that were used in our experiments.

The remainder of this paper is organized as follows. Section II provides a detailed overview of the two speech recognition software packages that were used in our experiments. Section III describes the components of the initial working system that we have implemented including a portion of the initial grammar, and Section IV concludes the paper which a summary of current and future work.

II. Speech Recognition Packages

We have experimented with two speech recognition packages: Sphinx [6] and MSAPI [7]. As part of the contribution of this work we provide specific details on each system and report on the PROs and CONs including ease-of-use and initial accuracy that we experienced with each package.

A Sphinx

Sphinx is based on the Java Speech API developed with Sun. JSAPI was released on October 26th, 1998. It is not in a 'finished' state however it is provided with some support for the benefit of third party developers hoping to add speech recognition to their Java programs.

Sphinx provides wrapper classes for the functions within the JSAPI which are commonly used within speech recognition software. These include the microphone, the recognizer, the dictionary, and the linguistic and acoustic models. It uses XML formatted configuration files for predefining properties of the different parts of the program. The Sphinx system was developed on a Linux platform and therefore any training systems are also implemented on Linux. The classes are included as jar files compiled under Apache Ant Builder. The grammar files are formatted using JSGF or Java Speech API Grammar Format. The Grammar is rule base and supports regular expression capabilities and the importing of external grammar files. Alternate input files for recognition include linguistic files with n-gram support.

Our preliminary implementations were derived from demo examples included with the download and used the Java Speech API in its original form. The second implementation used the Sphinx library to produce the same effects. By using the Sphinx library, we were given easy functionality of an acoustic analyzer and a dictionary wrapper as well as several other components that could easily be managed via the XML configuration file. All of these libraries however use several methods that require an extensive amount of memory, larger than that allocated to a Java program by default and the -Xmx<amount>m switch was needed to increase the default heap size.

The initial Sphinx implementations functioned by loading linguistic files that maintained n-gram information on specified sentences and commands common to the robotic manipulation. The results were quite good and the linguistic file was reworked into a JSGF grammar. Initial tests worked well, however, with the addition of new, and particularly long commands, the recognition quickly broke down. Even with careful wording, certain commands have a very low success rate of recognition. This problem was resolved by manipulating the internal settings for the recognizer within the XML configuration file. In particular, we altered:

- The probability of predicting a false positive.
- The probability of needing to insert a word to get a match.
- The probability of needing to insert a syllable to get a match.

By increasing the probability of getting a match, the system would match more frequently, however would produce odd responses when you spoke something that was not at all a command. False positive responses dropped significantly by decreasing the probability of word insertion and increasing the probability of syllable insertion. This worked quite well with the exception of number recognition. Firstly, numbers such as 'four' and 'two' have a number of words that sound similar, such as 'for' and 'to' and 'too.' So the grammar had to accept any form to be equivalent to the number in order to recognize it. Secondly, numbers such as 'four' and 'five' sound alike when spoken quickly. Similar problems exist between 'three' and 'eight' and 'two' and 'ten'.

With still more tweaking, the program responded correctly about 70 to 90% of the time and was therefore adequate enough for connecting with the robot. The connection, a simple socket connection, carried V+ formatted commands to the CS7B controller which was executing a program which looped the DOS command. For this reason, the Java program required an ad hoc method for breaking down the input string from the user and formatting it to V+. The V+ method used to move the robot was DRIVE which allows for the rotation of a joint by a certain degree, positive or negative, at a specified speed, which we keep at 10% of the robot's fastest speed which is recognized as a safe training speed.

The commands that we successfully implemented are the DRIVE command, the CLOSEI command and the OPENI command. The ad hoc V+ translation method was quite robust however and no simple solutions were offered.

Overall, the Sphinx-4 Speech Library is very robust. It is fairly easy to implement however requires much extra memory. The recognition abilities are fair but improvement would require much time and a much further explanation of the workings of the libraries. While a training program could be implemented, the official training program suggested by Sphinx is Linux based and would not be an intuitive method for a real-world Windows implementation due to lack of simplicity. The grammar formatting is however quite easy to follow and understand.

B MSAPI

The Microsoft Speech API was first released around 1995, and was supported on Windows 95. This version included low-level Direct Speech Recognition and Direct Text To Speech APIs which applications could use to directly control engines, as well as simplified 'higher-level' Voice Command and Voice Talk APIs. It has since had continued support within the Windows OS. It is designed for implementation among Visual Basic, C++, and more recently C#. The more current versions of Windows contain a speech configuration manager in the control panel which allows you to customize the default text to speech synthesis voice as well as to create recognition profiles. Microsoft has its own built in training program which is well developed.

Microsoft SAPI uses XML formatted grammars. Grammar functionality includes rule defining, property name and value specifications, optional segments, and lists as well as rule referencing.

The MSAPI is built into most Windows systems by default for text-to-speech and voice recognition support. For this reason, it would be a convenient choice because computers would not be required to download any additional software. The Microsoft SAPI also has significant support via MSDN and the SDK comes with its own well sorted and collected set of help files and examples. It was far easier to implement an initial system when compared to the Sphinx API. The Microsoft SAPI is designed for C++ and Visual Basic implementation but can be run from anything with OLE capabilities.

Surprisingly the examples were rather in-depth implementations and we had to examine them very carefully before identifying the common components needed to produce a simple program. The recognizer supports both static and dynamic grammars which can be created in memory or loaded via file or dll. The grammars are formatted in XML and support rule creation, rule referencing, optional sections, and lists. The most import feature being the property name and value support for phrases. This allows the recognized text to hold an extra identifying value. The recognizer is event driven and requires few commands to set up. Within the recognition event it is possible as well to access the property values, which in the grammar we made equal to the V+ command equivalent to the action. For example, rotating a join corresponds to the DRIVE command, one, two and three correspond to 1, 2 and 3, and open / close gripper commands correspond to OPENI and CLOSEI.

The recognition provided response of similar quality of the Sphinx API at first. However, using Microsoft's built-in speech configuration dialog control panel, we simply adjusted the recognizer to produce slower but more accurate results and a vast improvement ensued.

Winsock was introduced into the program to function in the same way as the Java implementation and results were about 95-97% accuracy. We then added dynamic rule support to the grammar being used and provided support for adding variables corresponding to the V+ HERE command which remembers a location. The program interface also lists the variables created. The variable names, being outside of the grammar, do not always end up being exactly what was intended, so keeping track of how the computer recorded them is important. With the commands stored in memory within a virtual grammar rule, they can then be easily recalled and recognized in combination of saying 'Go to' or something of that nature in order to execute the MOVE command in V+. This worked with large success and would provide support for complex movements and much expansion into saving variables for future loading/use, defining methods at run-time for batch execution, and defining frames at run-time as well.

Overall, the MSAPI implementation is quite successful. There are little extra files required for successful execution of the program and there is much support both within the package and on MSDN however most of it is in C++. The grammar design is rather confusing and would not be easily modified, however, its complexity does provide support for many powerful features. The MSAPI is also far less taxing on memory.

III. Initial Working System

An initial working system has been implemented and a screenshot of the system is shown in Figure 1. The three windows are as follows:

- The V+ Voice Command Interface program which controls the recognition and transmission of commands.
- The Debug Window, which is a child process of the V+ Voice Command Program, monitors the V+ Voice Command Program's variables, and displays the text being received by the CS7B controller.
- The Tera-term Window displaying the CS7B controller terminal.

The first two program windows are explained in some more detail below while the third is simply the Tera-term software and needs no further explanation.

A V+ Voice Command Interface

The V+ Voice Command Interface is implemented in Microsoft Visual Basic 6.0 for simplicity and lack of overhead. The menu system has components for loading grammar files (*.XML), specifying the socket to monitor for connections, and toggling the display of the debug window.

The upper left text field displays speech recognized by the MSAPI recognition event. If the recognized speech also matches the grammar, then a listing of applicable rules follows. Each rule displays the name of the rule as specified in the loaded grammar. After the rules, the properties are shown. Properties are specified in the grammar as a defined property name and given a value dependant on the phrase matched. The screenshot shows the property value of “JOINT” and its matched phrase’s value is “4”. The program iterates through the properties and concatenates them into a string which represents the command to be transmitted to the controller.

The text field to the right displays variables. When the phrase matching the command “HERE” is recognized, the grammar is set to match any single word phrase the follows. This word is then dynamically added to the grammar that is loaded and the word is added to a variable list to help the user of the program to keep track of stored variables. This is also important because if the recognizer misunderstands what you say and saves the variable as a slightly different word that you expected, you can easily see the difference. For example, the command “set position alpha” may issue the command “HERE although” due to a misrecognition of the work “alpha.” This is a result of the word “alpha” not being part of the grammar, however now that the word “although” has been added, it can be recognized with much higher precision when referring to it later.

The text boxes to the bottom left display status information, including which client is connected and the port currently being used. The status of the connection and send/receive process is also displayed. The last box displays the command being sent to the CS7B controller.

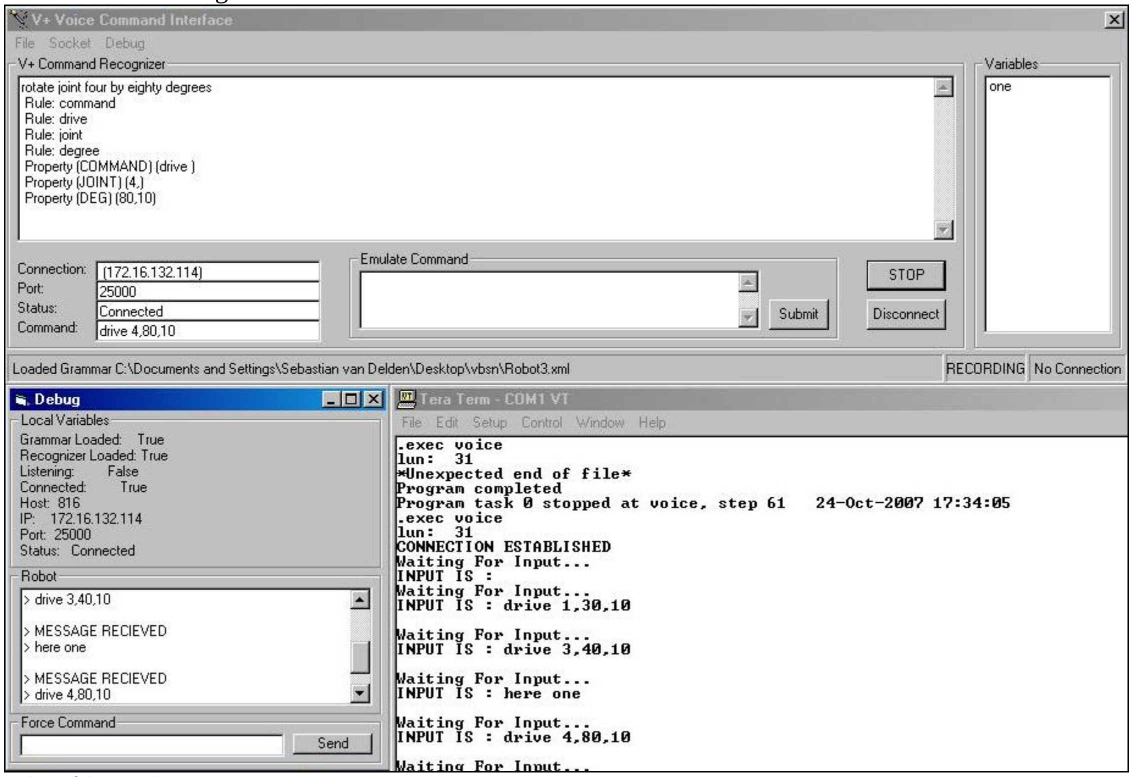
The text field in the bottom-middle of the window displays a to-be-completed feature that allows you to enter a string which the program will emulate as speech for the recognizer to break down.

The two command buttons toggle the starting and stopping of the recognizer and the connection / disconnection of the socket. And lastly, the status bar at the bottom shows the loaded grammar, recognition status, and socket connectivity status.

B Debug Window

The upper part of the debug window displays a listing of the variables and their values. The boolean values include: if the grammar is loaded, if the recognizer is loaded, if the socket is listening, and if it is connected. The other variables displayed include client value, client IP address, port value, and the status. All of the variables belong to debug’s parent window and are referenced publicly.

The bottom text field displays the information being received from the program running on the CS7B controller. This includes notification of connection, and echo of the command send, and notification of command received.



```
rotate joint four by eighty degrees
 Rule: command
 Rule: drive
 Rule: joint
 Rule: degree
 Property (COMMAND) (drive )
 Property (JOINT) (4,)
 Property (DEG) (80,10)
```

```
one
```

```
Connection: (172.16.132.114)
Port:       25000
Status:     Connected
Command:    drive 4,80,10
```

```
Loaded Grammar C:\Documents and Settings\Sebastian van Delden\Desktop\vbsn\Robot3.xml   RECORDING   No Connection
```

```
Grammar Loaded:   True
Recognizer Loaded: True
Listening:        False
Connected:        True
Host: 816
IP:   172.16.132.114
Port: 25000
Status: Connected
```

```
> drive 3,40,10

> MESSAGE RECIEVED
> here one

> MESSAGE RECIEVED
> drive 4,80,10
```

```
.exec voice
lun:  31
*Unexpected end of file*
Program completed
Program task 0 stopped at voice, step 61   24-Oct-2007 17:34:05
.exec voice
lun:  31
CONNECTION ESTABLISHED
Waiting For Input...
INPUT IS :
Waiting For Input...
INPUT IS : drive 1,30,10

Waiting For Input...
INPUT IS : drive 3,40,10

Waiting For Input...
INPUT IS : here one

Waiting For Input...
INPUT IS : drive 4,80,10

Waiting For Input...
```

Fig. 1 Screenshot of the initial working system.

The last text field provides a way of sending a typed command directly through the socket meant for debug purposes of realignment, etc. The commands allowed are only those executable by the DOS V+ command. Any other command will result in the V+ program incorrectly terminating.

C Initial Grammar

Figure 2 outlines a portion of the grammar that we have implemented in XML format which is recognized by MSAPI. The opening definition of the grammar is specified with <GRAMMAR>…</GRAMMAR>.

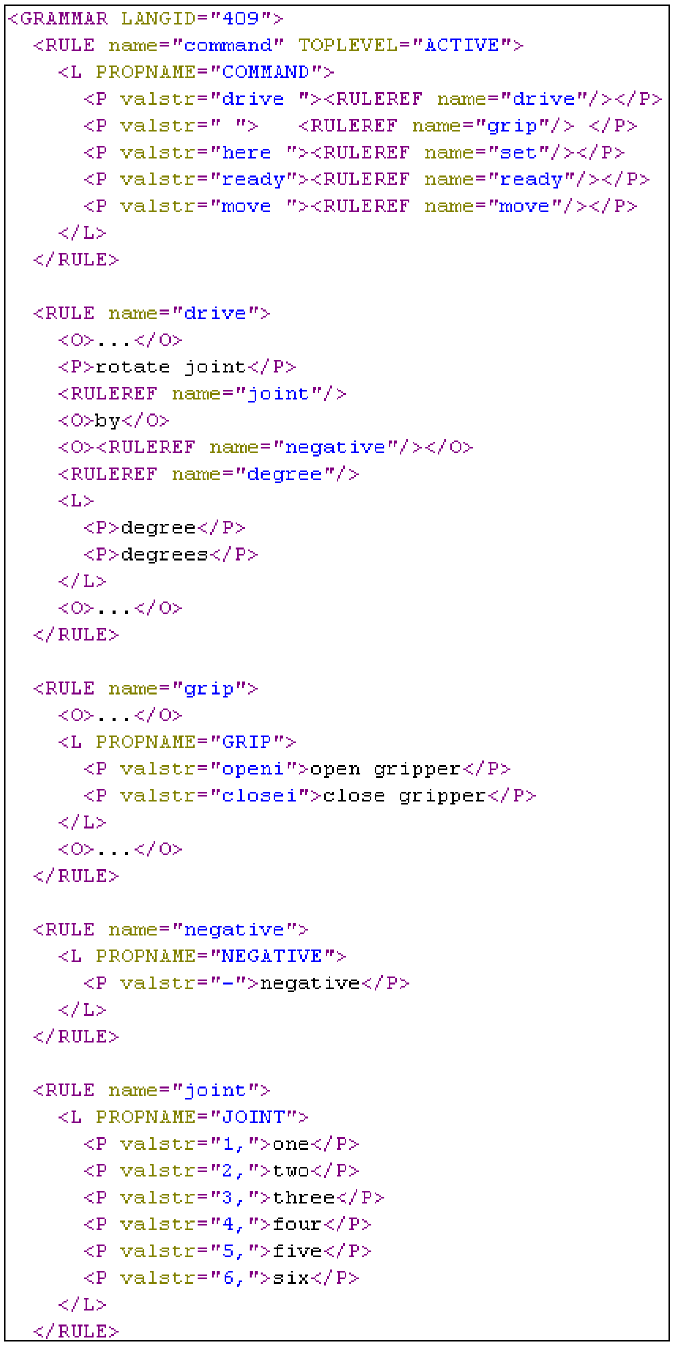
The components of <GRAMMAR> are <RULES>. Rules are given names and within them are the specifications of what defines a recognized rule. Having a TOPLEVEL of ACTIVE specifies that a rule is the default entry point of the grammar. Other optional components are the DYNAMIC value which when set to true allows for dynamic appendance to the rule.

Inside rules, multiple <PHRASE> properties are specified. A phrase contains the spoken words to be recognized. If all phrases are matched, then the rule is successfully matched and grammar is recognized. Phrases, shortened to <P> can be given values that are returned to the property specified by a parent entity. Values can be specified with "VAL" as a number or "STRVAL" as a string. A phrase consisting of any spoken text can be defined with "..." but cannot be referenced from the recognition event although using <DICTATION> will reference a word which can be further referenced. This latter feature is affective for implementing dynamic grammars.

To make the rules more accessible, three components may be implemented:

- A rule reference defined by <RULEREF> may be implemented which acts as a direct link to the rule specified.
- The <OPT> or <O> component shows an optional phrase for recognition.
- A <LIST> or <L> component allows for a list of phrases where only one of which may be matched. The list can also be given a property name to be matched with a property value of one of its possible phrase matches.

The three components can be combined in many ways to form very powerful grammars. Many more details on XML tags are used by MSAPI can be found here [9].



```
<GRAMMAR LANGID="409">
  <RULE name="command" TOPLEVEL="ACTIVE">
    <L PROPNAME="COMMAND">
      <P valstr="drive "><RULEREF name="drive"/></P>
      <P valstr=" ">   <RULEREF name="grip"/> </P>
      <P valstr="here "><RULEREF name="set"/></P>
      <P valstr="ready"><RULEREF name="ready"/></P>
      <P valstr="move "><RULEREF name="move"/></P>
    </L>
  </RULE>

  <RULE name="drive">
    <O>...</O>
    <P>rotate joint</P>
    <RULEREF name="joint"/>
    <O>by</O>
    <O><RULEREF name="negative"/></O>
    <RULEREF name="degree"/>
    <L>
      <P>degree</P>
      <P>degrees</P>
    </L>
    <O>...</O>
  </RULE>

  <RULE name="grip">
    <O>...</O>
    <L PROPNAME="GRIP">
      <P valstr="openi">open gripper</P>
      <P valstr="closei">close gripper</P>
    </L>
    <O>...</O>
  </RULE>

  <RULE name="negative">
    <L PROPNAME="NEGATIVE">
      <P valstr="-">negative</P>
    </L>
  </RULE>

  <RULE name="joint">
    <L PROPNAME="JOINT">
      <P valstr="1,">one</P>
      <P valstr="2,">two</P>
      <P valstr="3,">three</P>
      <P valstr="4,">four</P>
      <P valstr="5,">five</P>
      <P valstr="6,">six</P>
    </L>
  </RULE>
```

Fig. 2. A portion of the system's initial grammar in XML format which is recognized by MSAPI.

D Online Demo

We have captured a short video demo of system which includes sound. A screen shot of the demo is shown in Figure 3 and the link to the video is also included.

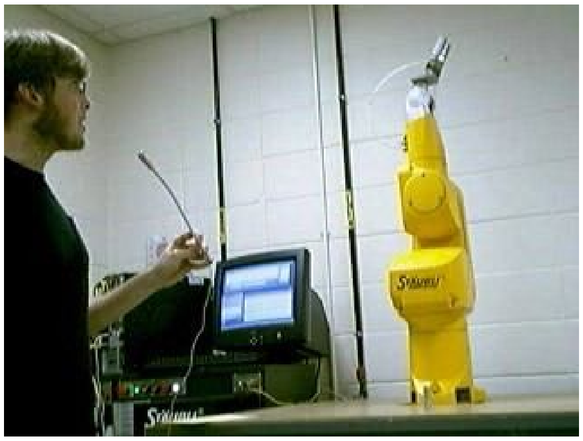


Fig 3. User interacts with our voice guided robotic system. To view video demo (with sound) please go to: http://faculty.uscupstate.edu/svandelden/VoiceJogging1.wmv

The system in this video was not yet trained on the user's specific voice. With a careful examination of the video demo, you will notice that the system incorrectly recognized "fifty degrees" as "eighty degrees". Also, one of the instructions had to be repeated twice before the system recognized it. Otherwise, the system recognized all of the other commands perfectly, performing joint rotations, training points, and opening/closing the gripper. Also, the hand held microphone is being replaced by a wireless headset to achieve a truly hands-free training environment.

IV. Conclusions AND Future Work

Initial results are very encouraging and indicate to us that implementing an accurate voice-guided system for jogging a robotic manipulator can be attained using currently available speech recognition APIs. Overall, we feel that the MSAPI speech recognition package was easier to integrate into the system and gave us better initial results when compared to the Sphinx system.

This initial work is currently being expanded in several ways:

- We are experimenting with an additional C++ based voice recognition system [8] and will be comparing it to the Sphinx and MSAPI systems.
- We are currently working on a completed formal grammar that encompasses all of the commands needed to design any automation task on the Stäubli RX series of machines which uses the V+ system.
- Once a completed grammar is finalized, we will be performing a more detailed and formal quantitative and qualitative analysis to determine if the voice-guided system improved operator productivity during application development.
- We will also be adapting the current system to a Stäubli RS20 manipulator which uses Stäubli's most current programming environment and VAL3 language. We will extend the current system to the VAL3 language and, in the process, work to develop a general system that is easily adapted to new iterations of robotic programming languages.

Acknowledgment

We would like to express our sincere thanks to the Stäubli Corporation for making this research possible by generously donating six RX60 manipulators and a RS20 manipulator to our institution.

We would also like to thank USC Upstate's Research Advisory Council for their partial funding of this work.

References

[1] T. Yoshidome, N. Kawarazaki, and K. Nishihara. "A Robot Operation By A Voice Instruction Including A Quantitative Expression," *In Proc. of the 5TH Franco-Japanese Congress & 3RD European-Asian Congress On Mechatronics*, pp 123-126, 2001.

[2] P. Norberto. "Robot-by-voice: Experiments on commanding an industrial robot using the human voice," *Industrial Robot Journal*, vol. 32(6), pp 505-511, 2005.

[3] N. Kawarazaki, T. Yoshidome, and K. Nishihara, "An Assistive Robot System Using Gesture and Voice Instructions," In Proc of the *2nd Cambridge Workshop on Universal Access and Assistive Technology* (incorporating the 5th Cambridge Workshop on Rehabilition Robotics), 2004.

[4] S. Shew, D. Ostelie, and G. Holcomb III. "Robotic Telescopic Assistance in Pediatric Laparoscopic Surgery," *Pediatric Endosurgery & Innovative Techniques*, vol. 7(3), 2003.

[5] C. Nathan, V. Chakradeo, K. Malhotra, H. D'Agostino, and R. Patwardhan. "The Voice-Controlled Robotic Assist Scope Holder AESOP for the Endoscopic Approach to the Sella," *Skull Base*, vol. 12, pp-123-132, 2006.

[6] Sphinx-4 Speech Recognizer written in Java. Collaboration between the Sphinx group at Carnegie Mellon University, Sun Microsystem Laboratories, Mitsubishi Electic Research Labs, And Hewlett Packard, contributions from University of CA at Santa Cruz, and MIT. http://cmusphinx.sourceforge.net/sphinx4/

[7] Microsoft Speech Application Programming Interface (API) and SDK, Microsoft Corporation, http://www.microsoft.com/speech.

[8] C. Becchetti. "Speech recognition : theory and C++ implementation," *Wiley publishers*, 1999.

[9] Grammar Format Tags for The Microsoft Speech API, MSDN Library. http://msdn2.microsoft.com/en-us/library/ms723634.aspx