Development of a Voice-Controlled Human-Robot Interface

Warat Khaewratana
wxk9242@rit.edu

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
Thesis

Development of a Voice-Controlled Human-Robot Interface

Warat Khaewratana

May 13, 2016

Department of Manufacturing and Mechanical Engineering Technology
College of Applied Science and Technology
Rochester Institute of Technology

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Manufacturing and Mechanical Systems Integration
Committee Approval:

S. Manian Ramkumar

Committee Member/Interim Department Chair/Thesis Advisor

Date

James Lee

Committee Member/Program Director for Graduate Studies and Research

Date

Alan Raisanen

Committee Member

Date
ABSTRACT

The goal of this thesis is to develop a voice-controlled human-robot interface (HRI) which allows a person to control and communicate with a robot. Dragon NaturallySpeaking, a commercially available automatic speech recognition engine, was chosen for the development of the proposed HRI. In order to achieve the goal, the Dragon software is used to create custom commands (or macros) which must satisfy the tasks of (a) directly controlling the robot with voice, (b) writing a robot program with voice, and (c) developing a HRI which allows the human and robot to communicate with each other using speech. The key is to generate keystrokes upon recognizing the speech and three types of macro including step-by-step, macro recorder, and advanced scripting. Experiment was conducted in three phases to test the functionality of the developed macros in accomplishing all three tasks. The result showed that advanced scripting macro is the only type of macro that works. It is also the most suitable for the task because it is quick and easy to create and can be used to develop flexible and natural voice command. Since the output of macro is a series of keystrokes, which forms a syntax for the robot program, macros developed by the Dragon software can be used to communicate with virtually any robots by making an adjustment on the output keystroke.

Keywords:

advanced scripting, human-robot interface, macro, speech recognition, voice command, voice-controlled
# Table of Contents

1. **Introduction**  ......................................................... 1  
   1.1 Background .......................................................................................................................... 1  
   1.2 Research Aim and Approach ............................................................................................... 4  
   1.3 Thesis Layout....................................................................................................................... 6  

2. **Related Work** ......................................................... 7  
   2.1 Development of Speech Recognition System........................................................................ 7  
   2.2 Development of Acoustic Model using Hidden Markov Model Toolkit............................... 9  
   2.3 Applications of Voice-controlled HRI.................................................................................... 13  
       2.3.1 Personal Robot Applications ..................................................................................... 13  
       2.3.2 Industrial Robot Applications ................................................................................... 20  
       2.3.3 Humanoid Robot and Android Applications ............................................................ 25  

3. **Concept & Methodology** ............................................. 28  
   3.1 Concepts............................................................................................................................. 28  
   3.2 Methodology....................................................................................................................... 30  
       3.2.1 Step-by-Step .............................................................................................................. 31  
       3.2.2 Macro Recorder ........................................................................................................ 33  
       3.2.3 Advanced Scripting ................................................................................................... 35  

4. **Experiment** ................................................................. 38  
   4.1 Direct Control Phase .......................................................................................................... 38  
   4.2 Program Creation Phase ..................................................................................................... 44  
   4.3 Human-Robot Interaction Phase ........................................................................................ 45  

5. **Results & Conclusion** ................................................ 48  
   5.1 Direct Control Phase .......................................................................................................... 48  
   5.2 Program Creation Phase ..................................................................................................... 52  
   5.3 Human-Robot Interaction Phase ........................................................................................ 55  
   5.4 Conclusion ......................................................................................................................... 59  
   5.5 Future Work ....................................................................................................................... 60  

**Acknowledgements** ..................................................... 61  

**References** ................................................................. 62
Appendix

A. Advanced Scripting Macro Scripts
   A.1 Direct Control .................................................................68
   A.2 Program Creation ...........................................................72
   A.3 Human-Robot Interaction .............................................79

B. V+ Programming Scripts .................................82
List of Figures

2.1 Development processes of acoustic model .................................................................10
2.2 (a) Three-state left-right topology of phoneme and (b) Modified silence model .......12
2.3 Framework for hybrid HRI utilizing combination of speech recognition and hand gesture recognition ..........................................................................................................................14
2.4 Vision system detecting specific color and directional movement of objects in the background as commanded .........................................................................................................................16
2.5 IBL system converting natural speech to robot program............................................19
2.6 (a) ASR interface application controlling two industrial robots: a pick-and-place robot and a welding robot, and (b) an operator commanding the welding robot using a headset microphone ..........................................................................................................................21
2.7 Sound feedback system indicating state of the robot................................................26
2.8 Demonstration of an android developed using the RMDB.......................................26
3.1 Example of how to create a step-by-step macro ...........................................................32
3.2 Example of how to create a macro-recorder macro ....................................................34
3.3 Example of how to create an advanced scripting macro .............................................37
4.1 AdeptThreeXL robot work cell environment for the experiment which includes the robot arm, three golf ball feeders, golf balls, and proximity sensors .................................................................39
4.2 Golf ball feeder positions (a) Feeder 1 Output, (b) Feeder 2 Output, and (c) Feeder 3 Output ........................................................................................................................................39
4.3 A point location “fl0” is recorded (taught) upon successful recognition of voice command “teach point feeder one output” .........................................................................................................................41
4.4 V+ language code “do move f2i” gets produced and passed into robot interface to move the robot upon successful recognition of voice command “go feeder two input” .................42
4.5 Voice command “drive joint X for Y at Z” is designed such that three parameters are adjustable based on what the operator says .............................................................................................................42
4.6 A macro for voice command “do move point1” created using (a) advanced scripting, (b) step-by-step, and (c) macro recorder .........................................................................................................................43
4.7 Macro allowing the operator to say one of the items in the list, and all items lead to the same result which is a conversion from speech to numerical input ......................................................................46
4.8 Sample V+ programming script requesting an input from the operator to select an action for the robot to perform .........................................................................................................................46
4.9 Sample V+ programming script for pick-and-place operation which requests three inputs from the operator: two being any point locations and one being any integers of choice ..........................47
5.1 Illustration of variables used in calculation of Recognition Speed Rate .......................51
5.2 Advanced scripting programming to write (a) basic code with only sequence of keystrokes, (b) intermediate code that takes at least one variables which depend on spoken words, and (c) advance code whose desired written form and spoken form are not the same ..........................................................54
5.3 Example of advance code whose written form (actual V+ code) is different from its spoken form (voice command for writing the code)..........................................................54
5.4 Successful run without bad prompt inputs leads the program to loop back for more actions .....................................................................................................................57
5.5 Proximity sensor tells the program to send a message to the operator when the feeder runs out of golf ball, cancel the remaining pick-and-place operations, and loop back ............57
5.6 Bad point locations do not abort the program but instead cause it to switch to executing the first block and, after the action is over, loop back .................................................................58
5.7 Bad prompt input for number of golf ball (e.g. anything else but numerical number), on the other hand, causes the program to abort ......................................................................58
B.1 Scripts of the V+ language program including a main program and four subprograms....82
## List of Tables

4.1 List of voice commands for Direct Control Task and corresponding keystrokes which form a line of V+ language code..................................................................................................................41

5.1 Data collected for Recognition Speed Rate calculation for (a) short commands and (b) long commands..........................................................................................................................51
Chapter 1
Introduction

1.1 Background

Rapidly increasing development of technology as well as the advancing knowledge in the field of robotics, throughout the century, have led the world to become a place where robots sharing living space with humans is no longer a mere thought in science fiction. Nowadays, robots can be seen in public places such as schools and universities, hospitals, amusement park, etc. as well as within households that are financially affluent enough to purchase a personal robot. Consequently, interaction between humans and robots becomes another important factor in such society. In order for a robot to serve its purpose to the fullest, an operator should be able to interact or control the robot in the easiest and most natural manner. Thus, human-robot interface (HRI) becomes an active research topic for many researchers in the field.

In many industries where automated machines and robots are used in manufacturing, machining, material handling, or the combination of the three, human workers operate the robots in different ways. The robots could be controlled manually using a mouse, a keyboard, a joystick, or a teach pendant like they traditionally were (Poncela & Gallardo-Estrella, 2015). As technology advances, HRI has become easier and more time-efficient. Nowadays, technology for HRI includes the use of speech (Pires, 2005), face movement (Bergasa, Mazo, Gardel, Barea, & Boquete, 2000), eye gaze direction (Matsumoto, Ino, & Ogasawara, 2001), gloves (Harada, Sato, & Mori, 2000), and electromyography (EMG) signal from muscular activity (Kim, Kim, Kim, Son, & Lee, 2006). While these HRIs possess various advantages, which make each of them superior to one another, they also have their own limitations. For example, voice-controlled HRI
may be inappropriate in noisy condition and environment where loud noise is prohibited such as hospital (Kuno, Murashima, Shimada, & Shirai, 2000); people tend to move their face and eye gaze unconsciously which could cause misleading commands (Nishimori, Saitoh, & Konishi, 2007); and the user who operates robots using muscular activity sensors such as EMG not only needs trained skills but also carries mental burden from using the device (Nishimori et al., 2007).

Of all the mentioned human-robot interfaces, voice-controlled interface is the one humans feel most comfortable with. That is because voice is the most natural means of communication among human beings as opposed to body language, gesture, face expression, and eye contact (Ferre, Macias-Guarasa, Aracil, & Barrientos, 1998). Through the means of speaking, people can convey messages including thoughts, opinions, and needs to each other very easily. One limitation to the statement is that it only holds true when the conversation partners are able to understand the spoken language. For example, speech in foreign language will fail to convey the meaning of its delivered message if listeners do not understand that particular language. Since computer, using today’s technology, can be programmed so that it can understand speech in various languages such as Japanese (Lee, Kawahara, & Shikano, 2001), English (Lee & Kawahara, 2009), Chinese (Yang, Iwano, & Furui, 2008), Korean (Kim, Jung, & Chung, 2004), Thai (Jongtaveesapatorn, Wutiwiwatchai, Iwano, & Furui, 2008), French (Illina, Fohr, Mella, & Cerisara, 2004), Estonian (Alumäe, 2004), and Slovenian (Rotovnik, Maucec, Horvat, & Kacic, 2002), voice-controlled HRI is an appropriate option. Moreover, voice-controlled HRI is suitable for jobs that require human operator the need to use both hands while operating the robots such as the case of a humanoid robot assisting a human operator in carrying an external wall panel (Yokoyama et al., 2003).
Many researchers and professionals in the field of robotics have conducted various studies to find ways to command robots using voice communication. Control of robots will become much easier if all it takes is to have an operator speak directly to a robot the detailed instruction for tasks that must be performed. This particular advantage of the voice-controlled system provides even users who are not specialized in the field of robotics or do not possess knowledge and experience in programming an opportunity to control a robot without difficulty (Iba, Paredis, & Khosla, 2005). Although still undergoing experimentation and not yet robust enough for commercial usage, voice-controlled system has realized a possibility of controlling robots through means of speaking. Voice-controlled system is a system which utilizes automatic speech recognition (ASR) to convert human operator’s spoken words to written texts or syntaxes that can be understood by robots. With this system, an operator can communicate with a robot by saying set of commands, and the robot will execute the command to accomplish the task.

Up until now, automatic speech recognition has been used in many applications in various fields such as entertainment industry: interactive tour-guide robot (Drygajlo, Prodanov, Ramel, Meisser, & Siegwart, 2003); medical field: voice-controlled intelligent wheelchair for elders and disabled individuals (Nishimori et al., 2007) and robot-assisted rehabilitation system (Barkana, Das, Wang, Groomes, & Sarkar, 2011); and robotics: teleoperation of mobile robot (Poncela & Gallardo-Estrella, 2015). The ASR is also being implemented in the development of humanoid robots such as Honda ASIMO (Nakadai, Nakajima, Ince, & Hasegawa, 2010) and NAO robot (Aldebaran, n.d.). These developments have shown the potential to create a world where humans and robots live and interact with each other in everyday life.
1.2 Research Aim and Approach

The aim of this research is to develop a human-robot interface for controlling an industrial robot using voice command. The HRI should virtually replace the need for keyboard and mouse to control the robot, and thus the ASR is used. Poncela and Gallardo-Estrella (2015), using the Hidden Markov Model Toolkit (HTK), developed their ASR system based on the Spanish language model taken from the ASR engine Sphinx. This approach requires knowledge of how to use the HTK, and for the ASR to work, it also requires the phonetically balanced training sentences to be prepared, parameterized, and then used to train the acoustic model whose process consists of 5 steps of data manipulation (Poncela & Gallardo-Estrella, 2015). Another approach is to use a commercially available ASR engine. Pires (2005), for his ASR system, used Microsoft Speech Engine and Microsoft Speech Application Programming Interface (SAPI). This approach requires the ASR engine to first convert an operator’s spoken command into text form and then feed the string of text, along with recognition confirmation event, to the SAPI. The SAPI then takes the inputs as prompts and produces the corresponding programming scripts for the robot’s controller interface. Lastly, the robot interface executes the task program according to the scripts which moves the physical robot and completes the intended task.

The approach used in this paper is to utilize a commercially available ASR engine which has the application programming interface already built into it in order to reduce the complexity of the proposed HRI. The whole system consists of two main components: (a) the ASR engine with built-in application interface and (b) a software program interface for controlling the robot. The chosen ASR engine for this research is Dragon NaturallySpeaking Professional 12 (Dragon Systems, Inc., 1998). The professional version of the Dragon software allows users to develop custom voice commands (which will be referred to as macros from this point on) for virtually
any applications including text-based applications such as Microsoft Word and non-text-based applications that do not allow text editing features (Bendewald, 2007). The software also comes with text-to-speech function which allows it to provide spoken feedback back to the user.

The robot used in the experiment is the AdeptThree-XL whose software program interface is Adept Windows (Adept Technology, Inc., 1997). The Adept Windows can be used to control the robot in two ways: (a) direct control of the robot in monitor mode and (b) execution of the written program script in program mode. With Dragon NaturallySpeaking Professional, it is possible to control the robot in both ways using voice. When the Dragon software hears and recognizes an operator’s speech (word, phrase, or sentence), it converts the speech from audio signal to text form. It then searches through its command database for a command or macro whose keyword for activation matches the input text. Once it finds the match, it executes the command which could result in a sequence of keystrokes, an output of texts and/or pictures, or open/close a computer application. In order to control the Adept robot, user must create macros which, when activated, will output a keystroke sequence which spells out a syntax understandable by the robot and then passes it to the robot controller for execution. If the Dragon software is to be used to control different robot, for instance, it can be done simply by changing the output syntax to suit the robot controller’s programming language. Under the same methodology, different robots using different programming languages can all be controlled by voice command with Dragon NaturallySpeaking Professional software.
1.3 Thesis Layout

The following is the breakdown of the layout of this thesis paper which includes:

- **Chapter 2**
  
  Related works about the development of speech recognition system using HTK and HRI applications in controlling robots are discussed.

- **Chapter 3**
  
  Concepts for tasks to complete in order to fulfill the goal of interfacing Dragon NaturallySpeaking Professional with Adept Windows and methodology used are described in detail.

- **Chapter 4**
  
  Experiment regarding the actual development of the voice-controlled HRI and the test run is mentioned in detail.

- **Chapter 5**
  
  The final chapter presents analysis of results of the experiment, conclusion of the thesis, and possible future work.
Chapter 2

Related Work

The concept of interaction between human and robot has been studied extensively throughout the past decades. Researchers have strived for the human-robot interface that allows users to communicate with robots in the manner as easy and natural for humans as possible. As a result, HRIs for controlling robots based on various means of communication have been developed. Among those means is speech, which is the main focus of this paper. The following will be discussions of (a) the development of speech recognition system using Hidden Markov Model Toolkit and (b) the application in controlling wide area of robots including personal robots, industrial robots, and humanoid robots and androids.

2.1 Development of Speech Recognition System

Before discussing about the processes of how to develop an ASR system, it is important to have a general idea of how speech is recognized and converted to text form by the system. An ASR engine needs a voice decoder software (also known as speech recognition parser) for speech recognition process. Mainly, a decoder software needs two models: a language model and acoustic model, for it to be able to recognize speech (VoxForge, 2006). The language model can be further broken down into word lexicon and grammar (Poncela & Gallardo-Estrella, 2015). Word lexicon is the software’s database which contains large amount of words that can be recognized by the ASR system (VoxForge, 2006). Grammar contains sets of word combinations predefined by the rules and restrictions of grammar of any particular languages (VoxForge,
2006). For example, word lexicon and grammar used in an English voice decoder software are different from those used in a Japanese model.

An acoustic model contains sets of distinct sounds (called phonemes) that make up each of the word predefined in the language model (VoxForge, 2006). For example, the word “call” has phonemes of "k", "ao", and "l", and the word “young” has phonemes of "y", "ah", and "ng" (VoxForge, 2006). The user must train the decoder software each distinct sound that will combine to create words, phrases, or sentences intended to be recognized during the operation of the robot. The training process requires the user to record sample words, phrases, or sentences prepared beforehand into the system (Poncela & Gallardo-Estrella, 2015). Once the system recognizes the sample sounds, it can relate to similar sounds in the future. This process is, in a way, similar to teaching a robot arm each position required in order to accomplish a certain task.

After a user provides the ASR engine some input speech, the decoder software reads the data as a combination of distinct sounds (VoxForge, 2006). All of the distinct sounds that the software can recognize must be predefined by the trained acoustic model. Once the software finishes reading the whole word, phrase, or sentence, it then searches for similar words or word combinations predefined and stored in the word lexicon and grammar. If it finds a match, it then returns the result in a form of text. The text becomes an output and can be used as code syntax or a trigger of the command for controlling robots.
2.2 Development of Acoustic Model using Hidden Markov Model Toolkit

Acoustic model can be created using Hidden Markov Model Toolkit. HTK is a toolkit which is widely used to develop acoustic model for various applications such as speech recognition, speech synthesis, and character recognition based on Hidden Markov Model or HMM (Poncela & Gallardo-Estrella, 2015). Development of an acoustic model can ultimately be divided into two phases: data preparation and training. Furthermore, the training phase is divided into 5 steps. The procedure for the whole process is illustrated in Figure 2.1 (Poncela & Gallardo-Estrella, 2015). As shown in the figure, training sentences are necessary input for the whole process. The training sentences must also meet the minimum requirements for the HTK to be able to compile the sentences into an acoustic model, thus the design for the training sentences are an additional yet crucial step (VoxForge, 2006). The HTK requires training sentences to have good phonetic balance and coverage (Young et al., 2006). In order to achieve the requirements, the sentences must contain the minimum of 30 - 40 sentences with each sentence consisting of 8 - 10 words and have 3 - 5 repeating occurrences of each phoneme (VoxForge, 2006). Poncela and Gallardo-Estrella (2015), in their work, prepared 72 training sentences with the total of 802 words.
Once the phonetically balanced training sentences are prepared, they are ready for the data preparation phase. The sentences are recorded and converted from the audio signal into sequence of feature vectors, which are then parameterized using either Mel frequency cepstral coefficients (MFCC) or linear prediction coefficients (LPC) (Poncela & Gallardo-Estrella, 2015). The parameterized data then enter the training phase. The first step is to generate a set of flat start monophonemes using HTK. To do so, the connection of the phoneme pronunciation must first be defined by HMM in one way. Poncela and Gallardo-Estrella (2015) went with the common assumption that pronunciation of a phoneme goes through three phases sequentially – the transition-in from the previous phoneme occurs, then the pure pronunciation of a phoneme occurs, and finally transition-out to the next phoneme occurs. Figure 2.2a illustrates the three-state left-right topology of phoneme pronunciation as previously described (Poncela & Gallardo-Estrella, 2015). The topology applies to both phoneme of sound and silence model which is a model for long pauses at the end of a sentence. Step 2 is to modify the silence model in order to obtain short pause model which is a model for pauses between words. Figure 2.2b illustrates the modified silence model featuring both short and long pauses (Poncela & Gallardo-Estrella, 2015). Step 3 is to realign the training data for words that have multiple pronunciation, for example the word “the” can be pronounced as “duh” and “dee” depending on the grammatical context. Step 4 is to copy and re-estimate sets of monophoneme HMMs in order to obtain context-dependent triphoneme HMMs. Finally, Step 5 is to determine the states within the triphoneme sets and tie them to create an acoustic model (Poncela & Gallardo-Estrella, 2015).
Figure 2.2: (a) Three-state left-right topology of phoneme and (b) Modified silence model. Adapted from “Command-based voice teleoperation of a mobile robot via a human-robot interface,” by A. Poncela and L. Gallardo-Estrella, 2015, *Robotica*, 33, p. 9. Copyright 2014 by the Cambridge University Press. Adapted with permission.
2.3 Applications of Voice-controlled HRI

This section discusses works of various researchers who applied ASR system into the development of robots. While the field of voice-controlled robotics application is vast, it can be categorized mainly into personal robots, industrial robots, and humanoid robots and androids.

2.3.1 Personal Robot Applications

Studies on HRI potentially lead to large number of applications in robot control. Personal robot is one of the areas of the application. Personal robot is a robot design to provide support or attend to an individual for miscellaneous purposes. Because of that, the main target customer for this type of robot includes people who are not robotics expert. Therefore, it is crucial that the HRI for the application is user-friendly, meaning it is easy for anyone to understand and use.

Iba, Paredis, and Khosla (2005) used the combination of speech recognition and hand gesture recognition to develop the hybrid HRI to control a personal vacuum-cleaning robot (Figure 2.3). In their work, SPHINX-II was chosen for the ASR engine (Iba, Paredis, & Khosla, 2005). The speech recognition part allows the user to give the robot symbolic commands (i.e. names, confirmations, and program statements), and the hand gesture recognition part allows the user to provide parameters such as speed, angles, and positions that corresponds of the given command (Iba, Paredis, & Khosla, 2005). The combination of the two recognition systems makes the HRI an intuitive, user-friendly one that gives the user “the ability to provide interactive feedback to coach the robot throughout the programming process” (Iba et al., 2005, p. 86).
Nishimori, Saitoh, and Konishi (2007) developed the intelligent wheelchair with voice-controlled HRI. Because the wheelchair is operated by voice commands, even elderly or physically handicapped users can maneuver it easily. In their work, a voice decoder software *Julian* was used for the speech recognition system to recognize voice commands in Japanese including nine reaction commands (e.g. run forward and backward, stop, and turn) and five verification commands (Nishimori et al., 2007). The result of the experiment returned successful recognition rates of 98.3% for reaction commands and 97.0% for verification commands (Nishimori et al., 2007).

Inamura, Inaba, and Inoue (1998) proposed the interactive sensing system which allows the robot to find a human in a complex background using the combination of speech and vision system. In the demonstration, the robot tried to find the operator using vision system to detect specific features of the operator such as color of clothes or directional movement (Figure 2.4). The operator must tell the robot which features to detect in order for the robot to find the target human. The robot successfully found the target, which showed that it can recognize speech and was able to perform the task accordingly (Inamura, Inaba, & Inoue, 1998).
Figure 2.4: Vision system detecting specific color and directional movement of objects in the background as commanded. Adapted from “Finding human based on the interactive sensing,” by T. Inamura, M. Inaba, and H. Inoue, 1998, Intelligent Autonomous Systems, p. 92. Copyright 1998 by the IOS Press. Adapted with permission.
Poncela and Gallardo-Estrella (2015) developed a teleoperation HRI for controlling a mobile robot using voice commands in Spanish. The robot was tested for command recognition rate and performance on navigation task. Their original developed Spanish acoustic model was shown to have 98.8% word recognition rate and above 95% command success rate, which is the indicator of the number of correctly recognized sentences over the total number of sentences (Poncela and Gallardo-Estrella, 2015).

Personal robot is not limited to utility robot but it also includes robot which is built solely for entertainment purpose. In order for a robot to fully provide user a sense of entertainment, a joystick controller is probably not enough. Fujita and Kitano (1998) developed an interactive personal robot called MUTANT, a pet-type robot whose purpose is to give the user entertainment through interaction by gesture and voice. By interacting with the user, MUTANT is able to perform several behaviors and activities such as chasing after the ball, shaking hands with the user, and expressing its emotion through action (Fujita & Kitano, 1998). The advanced interactive technology is a key feature which differentiates MUTANT from a regular remote-controlled animal-type robot.

The ideal voice-controlled HRI should also allow the recognition of the natural language, that is, the speeches recognizable by a robot should not be only those that exactly match the grammar constrain. Unlike industrial robots whose main job is to recognize concise commands and perform the tasks with maximum efficiency, personal robots and humanoid robots for entertainment are created to provide comfort to users during the interaction. In such situation, the users should not be restrained by the grammar constrain when they make conversation with their robots – they should feel like talking to a real person or a real pet. With that concept in mind, Lauria, Bugmann, Kyriacou, and Klein (2002) implemented the Instruction-Based Learning (IBL)
system in their design of the ASR system that is capable of recognize even voice commands not exactly matching the grammar constrain. The robot used to illustrate the work listened to voice command in the form of natural speech, and the IBL system allowed the robot to either (a) find and perform the understandable command which mostly resembles the speech input or (b) generate a tag name for new command and program codes necessary to execute it (Figure 2.5).

The ASR system analyzes an input command containing words unnecessary for the recognition and translate it to a program code understandable by the robot (Lauria, Bugmann, Kyriacou, & Klein, 2002). This system provides users the ability to give the robot a command in the most natural speaking manner, and the robot still recognizes and follows it.
2.3.2 Industrial Robot Applications

Since industrial robots are usually operated by specialized employees in an industry for the sole purpose of working such as manufacturing, machining, and material handling, it may not seem to be necessary to apply the voice-controlled feature to the HRI to make the robot amusingly interactive; and it might seem like a teach pendant is enough to control the robots accurately. However, by applying voice-controlled HRI to industrial robots, the manufacturing process can be improved in terms of autonomy, efficiency, and agility (Pires, 2005). The benefits become much clearer when an operator must control multiple robots at the same time. Voice-controlled HRI, in this situation, eliminates the need for the operator to physically move between robot stations in order to shift the control. All robots, integrated into a single ASR system, are able to receive a spoken command at once, and only the robots configured to response to that specific command become active and perform the task. This area of application, therefore, becomes another research subject for HRI.

Pires (2005), whose work illustrates the previous points, reported the results of operating industrial robots using voice commands via personal computer equipped with a sound board and a headset microphone. The system consisted of two robots: a pick-and-place robot and a welding robot, and both robots were operated by a single ASR system (Figure 2.6). Once initialized, both robots would receive the same command input from an operator. But of course, a command is meant to control only one of the robots, which are assigned for different tasks. In order to prevent the confusion, the grammar for the ASR system was developed to include a key word to explicitly indicate which robot is to respond to the particular voice input. Pires (2005) specified the speech recognition grammar for any commands that they must include the word “robot”, followed by a tag word “one” or “two” indicating which robot is to respond to the command, and
Figure 2.6: (a) ASR interface application controlling two industrial robots: a pick-and-place robot and a welding robot, and (b) an operator commanding the welding robot using a headset microphone. Adapted from “Robot-by-voice: Experiments on commanding an industrial robot using the human voice,” by J. N. Pires, 2005, Industrial Robot: An International Journal, 32(6), p. 510. Copyright 2005 by the Emerald Group Publishing Limited. Adapted with permission.
then the keywords for task to perform and corresponding parameter. For example, the command is “robot one home”. While both robots receive the same command at the same time, only the robot pre-assigned by a tag number “one” will respond to the command and, thus, move to home position; the other robot will not recognize the command and remain alert for the next input (Pires, 2005).

The speech recognition engine used in the system was the Microsoft Speech Engine. In addition, the Microsoft Speech SDK 5.1, a software development kit developed by Microsoft for the purpose of developing speech-recognition integrated applications, was used as well (Pires, 2005). The contents of the Microsoft SDK package include the Microsoft Speech Engine, a text-to-speech engine, the MS SAPI which is a set of programming scripts providing connection between the speech recognition engine and an application, and other application development tools (Microsoft Corporation, 2016). In his work, Pires (2005) used the grammar builder included in the SAPI to introduce the developed grammar rule for commanding the industrial robots to the language model. Basically, a prewritten set of codes provided in the SAPI adds the new rule to the grammar so the robots understand commands that follow that rule.

Later on, Pires, Veiga, and Araújo (2009) proposed the Programming-by-demonstration (PbD) system which improved the task of programming, increasing the efficiency in the coworker scenario for small to medium-sized enterprises (SMEs). The main idea of the PbD system is to allow an operator to teach a robot to perform a task without a need to manually type in the robot programming codes for each task (Pires, Veiga, & Araújo, 2009). The robot is taught how to move its arm by having the operator physically guide the arm through the desired movement, and the code is to be written, line by line, using voice commands (Pires et al., 2009). The PbD system allows even a user with beginner-level in robot programming to be able to
operate a robot because the code is written by the user interface application and not by the operating person. This system also provides the operator an ability to teach and program the robot for new tasks in the middle of the work process, which is a common setting in the human-robot coworker scenario (Pires et al., 2009). The function of the ASR system used in this work is similar to the previous work with the only difference being the output of the ASR. In the previous work, the recognized voice command results in the robot physically performing the task corresponding to the preprogrammed command right away, whereas in the later work, the recognized command writes the code for the task to be interpreted by the robot at a later time. The development of the grammar for the ASR system in both works is practically the same with a slight change in syntax.

In other works, Zhang, Von Collani, and Knoll (1999) applied ASR to control two robot arms in assembly jobs. In addition to the ASR system, force-torque sensors and cameras are also installed to the robot arms, making the HRI multimodal. The goal was to interact with the robot arms and control them, mainly by voice, to perform tasks using somewhat loosely said command, meaning that the command can be under-specified, incomplete, or context-dependent (Zhang, Von Collani, & Knoll, 1999). In such case, the robot relies on additional information regarding the target object, current assembly state, etc. sent from the force-torque sensors and vision system. Myers, Pritchard, and Brown (2001) used combination of voice and force in automatic programming of their robot – they generated programming codes without manually writing it. The system, however, requires the code database to be manually written for new task. Afterwards, the user can simply run the robot using voice command and create new codes with new parameters for the similar tasks by physically guiding the robot through each required subtasks with desired movement and speed (Myers, Pritchard, & Brown, 2001). Throughout the
guiding process, the robot can be commanded to collect the data for the movement velocity and force/moment as it goes through the motion. The collected data become new parameters, and the robot can then mimic the same move with the updated parameters (Myers et al., 2001). This work is similar to that of Pires et al. (2009) in the aspect of teaching robot how to move by physically guiding it. However, rather than generating the new programming code line by line using voice, it simply updates new parameters to the existing code database.
2.3.3 Humanoid Robot and Android Applications

In recent years, many researchers in the field have been working to realize a concept which, a century ago, was regarded only as science fiction. The ultimate goal is to create humanoid robots and androids that are capable of interpreting natural language and engaging in a conversation with humans. While the day that the goal will be fulfilled may not come anytime soon, the development has taken significant progress. Yokoyama et al. (2003) developed a humanoid robot named HRP-2P that demonstrated a capability of working hand-to-hand with human operators in a construction work. Through the voice-controlled HRI with additional feature of state-indication sound feedback system (Figure 2.7), the humanoid robot successfully help the operator carrying large-sized external wall panel and mounting the panel to the building (Yokoyama et al., 2003). After finished saying a command to the robot, the operator listens to one of the four different sounds – each of which indicates the state of the robot and, ultimately, informs the operator whether the command is successfully received and passed on to the command processing unit (Yokoyama et al., 2003). This feature is useful because it reassures the operator that the robot is working on executing a command and therefore prevents the operator from repeating the command with the misunderstanding that the robot might have failed to receive the first command.

Kondo (2013) proposed the motion planning method called Reconfigurable Motion Database (RMDB), which was used to develop an android making it capable of engaging in a conversation with a group of people. A voice decoder software Julius was used for the ASR system, allowing a recognition of conversation in Japanese. As illustrated in Figure 2.8, with the proposed HRI system, the android can engage in various conversations, adjust its gesture based on the location of its conversation partner or target object, and be interrupted by a new speaker in
Figure 2.7: Sound feedback system indicating state of the robot. Reprinted from “Cooperative works by a human and a humanoid robot,” by K. Yokoyama et al., 2003, *Robotics and Automation*, 3, p. 2989. Copyright 2003 by the Institute of Electrical and Electronics Engineers. Reprinted with permission.

Figure 2.8: Demonstration of an android developed using the RMDB. Adapted from “Construction of Reconfigurable Motion Database for real-time Human-Robot Interaction,” by Y. Kondo, 2013, *NAISTAR*, p. 4. Copyright 2002-2015 by the DuraSpace. Adapted with permission.
real-time (Kondo, 2013). These works show that, today, humanoid robots can work alongside humans, and androids can engage in a conversation with people, making them seem rather human.
Chapter 3
Concept & Methodology

3.1 Concepts

The goal is to develop an HRI which uses Dragon NaturallySpeaking software to control an industrial robot (AdeptThree-XL in this case) through the robot’s software program interface (Adept Windows). Control of the Adept robot available in Adept Windows is based on programming method. Like any other robots, programming method can be categorized into three types: online programming, offline programming, and hybrid programming which is the combination of online programming and offline programming.

- **Online Programming**
  This is a method of writing a line of code which is executed immediately to move the robot, teach point location, or activate or deactivate the robot’s end effector.

- **Offline Programming**
  This method involves writing out the whole sequence of tasks to be completed first and then execute it altogether.

- **Hybrid Programming**
  This method is the combination in which the point locations are taught in online programming and sequence of tasks are written in offline programming. This method usually involves prompting an operator for confirmation or selection of task execution as well as providing feedback to the operator.
Based on these concepts, the mission of controlling the Adept robot via Adept Windows can also be broken down into three tasks in total: direct control, program creation, and human-robot interaction. The goal of this research will be achieved once the voice command is proven to be usable in accomplishing all three tasks.

- **Direct Control**
  
The task requires the use of voice command to perform an online programming to control the robot. Dragon NaturallySpeaking should recognize an operator’s voice command, return a corresponding line of Adept programming code, and execute it with one spoken word, phrase, or sentence.

- **Program Creation**
  
The task requires the use of voice command to write an Adept language programming script including sequences of tasks for the robot to complete. The operator should also be able to edit the script and navigate through the program without touching a keyboard or a mouse.

- **Human-Robot Interaction**
  
The task requires the use of voice command to interact with the robot. When the robot prompts the operator for confirmation or selection of task to execute, the operator should be able to answer the prompt using speech. Following the operator’s reply, the robot should also provide feedback in a form of written text or speech. Dragon NaturallySpeaking Professional comes with text-to-speech function which allows the generation of voice feedback.
3.2 Methodology

Using the robot’s software program interface, the robot is traditionally controlled by an operator manually typing in lines of codes to the computer. The codes are then executed at the interface level, and the robot gets driven according to the program in the executed codes. This research aims to replace typing process with voice command. With an ASR engine, speech can be converted into text form upon successful recognition and fed to the robot interface application. However, problem occurs when some of the robot software program interfaces, including Adept Windows, do not recognize and accept written texts. Instead, they accept a sequence of direct keystrokes on the keyboard which forms an imitation of text. This problem prevents the use of regular speech-to-text dictation function in many ASR engines.

The solution to the problem is to convert speech into a sequence of keystrokes which forms the desired text instead of actual written text and then pass the output of the keystrokes to the robot interface application. The proposed HRI using Dragon NaturallySpeaking Professional allows the operator to develop macros which perform the function as described upon recognition of the voice command. Four types of macros are available in Dragon NaturallySpeaking Professional, although three of them: step-by-step, macro recorder, and advanced scripting allow speech-to-keystroke conversion (Bendewald, 2007).
3.2.1 Step-by-Step

This type of macro performs a series of tasks one at a time and in top-to-bottom order or, as the name suggests, step by step. Available functions include sending a keystroke, sending a group of keystrokes, typing text, running an application, toggling microphone state (on/off), and pausing the macro for a specified period of time. Each step is created by simply selecting one of the functions and specifying its parameters (i.e. keystrokes, text, or pause time). Sending keystrokes and pausing functions are useful for this particular HRI. The pause function (known as “Wait”) helps when the computer or an application needs some time before it becomes ready for the next step. The macro may fail if all steps, one right after another, are executed too soon before the computer or the application can catch up. Therefore, it is important to pay attention to the activation order for all steps as well as necessary pause between each step. Step-by-step is easy to develop. While the downside is that it cannot be used to perform complex functions, it is enough for many tasks.
Figure 3.1: Example of how to create a step-by-step macro
3.2.2 Macro Recorder

This type of macro allows the user a possibility to record and replay the mouse movement, mouse click, and keystrokes in real-time (Bendewald, 2007). The advantages of macro recorder are that it can be (a) created to perform any tasks simply by recording the process steps required to complete the task for the first time and (b) replayed to perform the task again at a later time. The disadvantage is the lack of flexibility in reusing the macro. For the replay to work successfully, the user has to make sure that the current computer environment is compatible, for instance, the target file is in the same location on the desktop as it was when the macro was first recorded. The safest thing is to adjust the environment to be exactly the same before replaying the task. Without careful attention, macro recorder will be likely to fail.
Figure 3.2: Example of how to create a macro-recorder macro
3.2.3 Advanced Scripting

This type of macro is the most complex to develop because it requires basic knowledge of how to work with text-based programming language. The upside is that it is the most powerful tool to use to create macro for tasks at any levels of complexity. The available functions particularly useful to achieve the goal of this research includes the following.

- **SendKeys**

  This function is the same as sending keystrokes function previously described under Step-by-Step. In addition, SendKeys allows the user to output a long sequence of keystrokes with a single line of programming code. By creating an advanced scripting macro with multiple lines of code, it is possible to use one voice command to fill a page of document with text properly formatted and aligned. For this particular voice-controlled HRI application, one line of SendKeys can write out a program to move a robot and execute it immediately after.

- **List Variable**

  List variable function creates a variable which can assume one of many values indicated in the list upon creating a macro. This is useful to create a macro that takes different parameter depending on the situation. For example, a macro to move one of the joints of a robot arm can be executed by saying “move joint X” where X is the joint number. Another use of list variable is when different spoken words are used to execute the same command. For example, a command to input a value of 21 to the robot controller is designed to be executed by saying (a) “My name is David”, (b) “I am David”, or (c) “This is David”. In this case, the variable is any words preceding the word “David”, and
the list must contain at least those three groups of words. When creating a list variable macro, the user must either select a predefined list containing all desired values or create a new list with the values. A macro can make use of multiple list variables. For example, a macro named “move joint A for B at C” where A is the joint number, B indicates the degree of joint rotation, and C determines the speed of rotation.

- **HeardWord**

  HeardWord is an extremely useful function which executes another existing command or custom macro upon recognition of the macro it resides in. Basically, it allows another commands or macros to be nested in a macro without a need of rewriting programming scripts for them.

- **Wait**

  This function is the same as Wait function previously described under Step-by-Step. The function as a programming script is followed by a number representing wait time in milliseconds (Dragon Systems, Inc., 1998). In Dragon NaturallySpeaking Professional 12, however, the unit for Wait has been changed to seconds instead, which makes the use of the function less confusing to programmers.
Figure 3.3: Example of how to create an advanced scripting macro
Chapter 4

Experiment

The experiment is divided into three phases. Each phase contributes to one of the three tasks described in Chapter 3. Throughout this chapter, two different programming languages are presented. One is a language similar to Microsoft Visual Basic for macro creation using Dragon NaturallySpeaking (Bendewald, 2007). The other is V+ language program for Adept Technology industrial robot (Adept Technology, Inc., 1998). Figures 4.1 and 4.2 illustrate the robot work cell environment consisting of the robot arm, three golf ball feeders, golf balls, and proximity sensors attached at the output end of each feeder.

4.1 Direct Control Phase

This phase concerns the use of Dragon NaturallySpeaking Professional to develop macros which produce a line of V+ language programming code and execute it upon recognition of the operator’s voice command. The output of a macro is a sequence of keystrokes imitating syntaxes which forms a line of code followed by a keystroke for the enter key to execute the code. This function gives the operator a direct control of the robot with voice command without a need to write a program by hand.

In this phase, a set of macros, one for each line of V+ language code, is developed (Table 4.1). Upon successful development of macros, they will be tested for their functionality, determining whether the macros work or not. The failed macros will then be examined for causes of error and, if possible, fixed. The macros will also be used to test for accuracy and speed of voice recognition in the Dragon software. The accuracy will be determined by Command
Figure 4.1: AdeptThreeXL robot work cell environment for the experiment which includes the robot arm, three golf ball feeders, golf balls, and proximity sensors

Figure 4.2: Golf ball feeder positions (a) Feeder 1 Output, (b) Feeder 2 Output, and (c) Feeder 3 Output
Recognition Rate (CRR) which is a ratio of voice commands successfully recognized to the total number of spoken commands (Equation 1). Large value of CRR indicates that the Dragon software has high accuracy in voice command recognition. The ideal value is 100%, indicating perfect recognition. The recognition speed will be determined by Recognition Speed Rate (RSR) which is an amount of time taken to naturally speak a command comparing against amount of time from the start of the utterance to successful macro execution (Equation 2). The difference between two parameter values should be close to zero, indicating that the macro is recognized and executed right after the operator finishes saying the command. Therefore, large value of RSR is preferred. In order to collect reasonable amount of data for CRR and RSR, eight randomly selected participants will join in the experiment and get to control the robot using voice command. In addition to collecting quantitative data, a survey will also be given out to those participants in order to collect qualitative data regarding the functionality of the proposed voice-controlled HRI. Furthermore, each individual macro is created using all three types of macro, namely step-by-step, macro recorder, and advanced scripting, to demonstrate the differences in the creation process as well as the functionality (Figure 4.6). The underlying assumption is that a macro whose structure is complex will have slow command execution speed, whereas a macro which can be created in a couple of simple steps will be executed at high speed.

\[
\text{CRR} = \frac{\text{Recognized Voice Commands}}{\text{Total Spoken Voice Commands}} \quad (1)
\]

\[
\text{RSR} = \frac{\text{Time Period of Utterance}}{\text{Total Time Period from Utterance to Execution}} \quad (2)
\]
Table 4.1: List of voice commands for Direct Control Task and corresponding keystrokes which form a line of V+ language code

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Advanced Scripting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Power</td>
<td>SendKeys &quot;e n (space) p o (enter)&quot;</td>
</tr>
<tr>
<td>Disable Power</td>
<td>SendKeys &quot;d i s (space) p o (enter)&quot;</td>
</tr>
<tr>
<td>Calibrate [robot]</td>
<td>SendKeys &quot;c a l (enter) y (enter)&quot;</td>
</tr>
<tr>
<td>Create Program [xxx]</td>
<td>SendKeys &quot;s e (space) + (ListVar1) + &quot;(enter) y (enter)&quot;</td>
</tr>
<tr>
<td>Edit [Program] [xxx]</td>
<td>SendKeys &quot;s e (space) + (ListVar1) + &quot;(enter) i&quot;</td>
</tr>
<tr>
<td>Teach Point [xxx]</td>
<td>SendKeys &quot;h e r e (space) + (ListVar1) + &quot;(enter) (enter)&quot;</td>
</tr>
<tr>
<td>Go [xxx]</td>
<td>SendKeys &quot;d o (space) m o v e (space) + (ListVar1) + &quot;(enter)&quot;</td>
</tr>
<tr>
<td>Drive Joint (#) For (#) At (#)</td>
<td>SendKeys &quot;d o (space) d r i v e (space) + (ListVar1) + &quot;,&quot; + (ListVar2) + &quot;,&quot; + (ListVar3)+ &quot;(enter)&quot;</td>
</tr>
<tr>
<td>Where are you now</td>
<td>SendKeys &quot;w h e r e (enter)&quot;</td>
</tr>
<tr>
<td>Where is (xxx)</td>
<td>SendKeys &quot;p o i n t (space) + (ListVar1) + &quot;(enter)&quot;</td>
</tr>
<tr>
<td>Vacuum</td>
<td>End Effector On</td>
</tr>
<tr>
<td>Vacuum</td>
<td>End Effector Off</td>
</tr>
<tr>
<td>Load Program [xxx]</td>
<td>SendKeys &quot;l o a d (space) + (ListVar1) + &quot;(enter)&quot;</td>
</tr>
<tr>
<td>Run</td>
<td>Execute [Program] [xxx]</td>
</tr>
<tr>
<td>Abort</td>
<td>SendKeys &quot;a b o r t (enter)&quot;</td>
</tr>
<tr>
<td>Stop</td>
<td>SendKeys &quot;d i s (space) p o (enter)&quot;</td>
</tr>
<tr>
<td>Close Adept Window</td>
<td>SendKeys &quot;% f x&quot;, Wait 1, SendKeys &quot;[enter]&quot;</td>
</tr>
</tbody>
</table>

Figure 4.3: A point location “f1o” is recorded (taught) upon successful recognition of voice command “teach point feeder one output”
Figure 4.4: V+ language code “do move f2i” gets produced and passed into robot interface to move the robot upon successful recognition of voice command “go feeder two input”

Figure 4.5: Voice command “drive joint X for Y at Z” is designed such that three parameters are adjustable based on what the operator says
Figure 4.6: A macro for voice command “do move point1” created using (a) advanced scripting, (b) step-by-step, and (c) macro recorder
4.2 Program Creation Phase

This phase concerns the design of macros which use series of voice commands to write a V+ language programming script including simple programs for controlling the robot. In program mode, the operator writes a program for the robot by simply saying what needs to be written without touching a keyboard or a mouse. The operator should also be able to edit the script and navigate through the program using only voice commands. In this phase, and also in the following phase, all macros are created using Advanced Scripting method to allow the highest flexibility in writing lines of V+ code, especially one that takes variable as parameter. After the necessary macros are developed, they will be tested for their functionality. For data collection purpose, the same group of participants from the previous phase will get to try writing codes using voice command. A survey will be given out to those participants in order to collect qualitative data regarding the efficiency, in other words, how easier and faster to write program codes with voice comparing to manually typing them by hands.
4.3 Human-Robot Interaction Phase

This phase concerns the development of macros designed to interact with the robot using speech. All macros in this phase are created using Advanced Scripting method to allow the flexibility in the operator’s choice of words used in voice command or reply message (Figure 4.7). For example, the operator can say either “My name is David” or “I am David” and the end result will be the same. A prewritten V+ programming script will be executed to activate the interaction function (Figure 4.8). Throughout the interaction, the robot asks the operator to provide inputs. Some inputs determine which action the robot will take next, while some inputs specify parameters necessary for the robot to proceed with its intended action. The operator then gives the robot the requested input by saying it to the robot. Depending on the type of input being requested at that point of time, which is predetermined in the programming script, the speech is converted to either string variable or numerical variable that can be understood by the robot. For the experiment and data collection, the same group of participants from the previous phases will talk to the robot and request it to perform some actions such as moving around, performing pick-and-place the specified number of golf balls between two designated locations, or having a small back-and-forth conversation with the robot. A survey will be given out to those participants in order to collect qualitative data concerning how well the developed macros work as well as how natural or human-like the robot is during the interaction process. In order to promote the human-like conversation, the text-to-speech function available in Dragon NaturallySpeaking Professional which makes a voice feedback from the robot possible is utilized in this phase.
Figure 4.7: Macro allowing the operator to say one of the items in the list, and all items lead to the same result which is a conversion from speech to numerical input.

Figure 4.8: Sample V+ programming script requesting an input from the operator to select an action for the robot to perform.
Figure 4.9: Sample V+ programming script for pick-and-place operation which requests three inputs from the operator: two being any point locations and one being any integers of choice
Chapter 5

Results & Conclusion

This section discusses the results of the experiment from each phase and analysis of the collected data, both qualitative and quantitative. Conclusion of the thesis and possible future work are presented at the end of the section.

5.1 Direct Control Phase

After trying out all three types of a macro, it was determined that only advanced scripting macro works for the proposed voice-controlled HRI. Sufficient knowledge in programming makes development of this type of macro quick and easy. Furthermore, advanced scripting allows one macro to be used with large number of voice commands whose only difference is parameter or name variation. For example, a “do move” command for point1, point2, and point3 can be created with one advanced scripting macro “do move <point_name>”. This advantage helps reduce the total number of macros created tremendously and, as a result, save a lot of time. The other two types of macro cannot be used for the HRI. Step-by-step macro, even though having the same “Send Keys” function as advanced scripting macro, fails to pass a group of keystrokes altogether into Adept Windows. Its “Send Keystroke” function, which sends only one keystroke at a time, also fails. Macro recorder simply fails in Adept Windows. It does not record keystrokes typed in the application’s window at all. An attempt to record the desired keystrokes from another application such as Notepad and Microsoft Word and then replay it in Adept Windows was made and ended up with failure. Possible cause of failure in the other two macro types is that the built-in macro developing function of Dragon NaturallySpeaking Professional is
not designed to be compatible with non-universal robot interface programs such as Adept Windows, so neither macro recorder nor step-by-step works. Advanced scripting, on the other hand, deals with development of macro at a programming level, which allows macros to be created and used in virtually all programs compatible with a computer.

Due to step-by-step and macro recorder not being feasible for the HRI, only advanced scripting macro is used for the development, implementation, and data collection purpose. Command Recognition Rate (CRR) is determined to be approximately 85%. This number comes from the fact that throughout the experiment, approximately 1,020 commands out of 1,200 spoken commands were successfully recognized.

\[
CRR = \frac{1020}{1200} = 0.85 = 85\%
\]

Many of the failed commands were misheard and interpreted as other commands. Some commands either failed to recognize the spoken value for parameters or misheard and took in incorrect values instead. Voice commands that were not recognized at all were, in fact, the minority group of the failed recognition. CRR can be improved by training the Dragon software. As the Dragon software gets used to the user’s pronunciation of words in command name, voice commands tend to fail less. Each of the 8 participants was required to do 10-minute voice training prior to participating in the experiment, and the result was the CRR of 85%. If the participant underwent additional trainings, it would be possible to increase the CRR.

Recognition Speed Rate (RSR) was measured by saying a command and timing the duration between the spoken words were uttered and the command recognized. Tables 5.1a and 5.1b show the recorded data in terms of Time1, Diff, and Time2 for short commands (e.g. Go
Point1) and long commands (e.g. Teach Point Feeder Three Output) respectively. *Time1* indicates the amount of time taken to finish saying a voice command; *Diff* indicates the amount of time between the end of *Time1* and the moment when the spoken command gets recognized; and *Time2* is the summation of *Time1* and *Diff*, indicating the total time from start of the utterance of the command and recognition of the command (Figure 5.1). RSR is the ratio of *Time1* to *Time2*.

\[
RSR = \frac{Time1}{Time2} = \frac{Time1}{(Time1 + Diff)} = \frac{0.7}{(0.7 + 2.0)} = 0.259 = 25.9\%
\]

The average RSR came out to be 23.03% for short commands and 34.12% for long commands. These results show that the Dragon software recognizes and executes long commands faster than short commands. This is because long commands contain more words for the Dragon software to use as a hint to guess the context of the operator’s speech. As a result, the Dragon software is able to interpret and, therefore, recognize long command quicker and more accurately than short commands. Both short and long commands, however, showed low percentage of RSR, indicating that it took quite a while after the operator finishes saying a command for it to be recognized and executed. Possible causes of the low RSR include noisy environment during the experiment, low processing speed of the computer used in the HRI, and lack of enough voice training. Therefore, RSR, just like CRR, can also be improved by providing additional voice training within the software.
Table 5.1: Data collected for Recognition Speed Rate calculation for (a) short commands and (b) long commands

<table>
<thead>
<tr>
<th></th>
<th>Time1(s)</th>
<th>Diff(s)</th>
<th>Time2(s)</th>
<th>RSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0.7</td>
<td>2.0</td>
<td>2.7</td>
<td>0.259</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.2</td>
<td>2.8</td>
<td>0.214</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.1</td>
<td>2.7</td>
<td>0.222</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.7</td>
<td>2.3</td>
<td>0.261</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>1.7</td>
<td>2.5</td>
<td>0.320</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>1.8</td>
<td>2.5</td>
<td>0.280</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.0</td>
<td>2.7</td>
<td>0.259</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>1.7</td>
<td>2.4</td>
<td>0.292</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>1.8</td>
<td>2.5</td>
<td>0.280</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1.7</td>
<td>2.2</td>
<td>0.227</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.3</td>
<td>3.6</td>
<td>0.194</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.3</td>
<td>3.0</td>
<td>0.233</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2.5</td>
<td>3.0</td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>3.8</td>
<td>4.5</td>
<td>0.156</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.2</td>
<td>2.8</td>
<td>0.214</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.3</td>
<td>3.5</td>
<td>0.200</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.9</td>
<td>3.5</td>
<td>0.200</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>1.8</td>
<td>2.5</td>
<td>0.280</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2.1</td>
<td>2.6</td>
<td>0.192</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.0</td>
<td>2.7</td>
<td>0.293</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.0</td>
<td>2.7</td>
<td>0.259</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>2.9</td>
<td>3.7</td>
<td>0.216</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.8</td>
<td>3.4</td>
<td>0.176</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.2</td>
<td>2.9</td>
<td>0.241</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.3</td>
<td>3.4</td>
<td>0.176</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>2.5</td>
<td>3.3</td>
<td>0.242</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>2.1</td>
<td>2.6</td>
<td>0.192</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>2.3</td>
<td>3.0</td>
<td>0.233</td>
</tr>
</tbody>
</table>

AVERAGE 0.2303

<table>
<thead>
<tr>
<th></th>
<th>Time1(s)</th>
<th>Diff(s)</th>
<th>Time2(s)</th>
<th>RSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>0.9</td>
<td>1.5</td>
<td>2.4</td>
<td>0.375</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>1.8</td>
<td>2.7</td>
<td>0.333</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>1.6</td>
<td>2.7</td>
<td>0.407</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>1.7</td>
<td>2.8</td>
<td>0.393</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>1.6</td>
<td>2.7</td>
<td>0.407</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>3.3</td>
<td>4.2</td>
<td>0.214</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>3.0</td>
<td>3.9</td>
<td>0.231</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>3.5</td>
<td>4.7</td>
<td>0.255</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>2.5</td>
<td>3.6</td>
<td>0.306</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.3</td>
<td>3.3</td>
<td>0.256</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>2.2</td>
<td>4.2</td>
<td>0.476</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.1</td>
<td>3.1</td>
<td>0.323</td>
</tr>
<tr>
<td></td>
<td>1.7</td>
<td>2.0</td>
<td>3.7</td>
<td>0.459</td>
</tr>
<tr>
<td></td>
<td>2.1</td>
<td>2.8</td>
<td>4.3</td>
<td>0.429</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>2.9</td>
<td>3.1</td>
<td>0.365</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>1.9</td>
<td>2.8</td>
<td>0.321</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.8</td>
<td>3.6</td>
<td>0.278</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>1.6</td>
<td>2.6</td>
<td>0.385</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>2.2</td>
<td>3.1</td>
<td>0.250</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>2.4</td>
<td>3.3</td>
<td>0.273</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.2</td>
<td>3.2</td>
<td>0.315</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>2.7</td>
<td>4.1</td>
<td>0.341</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>2.3</td>
<td>3.4</td>
<td>0.324</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>2.5</td>
<td>4.5</td>
<td>0.444</td>
</tr>
</tbody>
</table>

AVERAGE 0.3412

Table 5.1: Data collected for Recognition Speed Rate calculation for (a) short commands and (b) long commands

Figure 5.1: Illustration of variables used in calculation of Recognition Speed Rate
5.2 Program Creation Phase

Advanced scripting macro makes it possible to write any V+ language code with one spoken word, phrase, or sentence. Short codes like “Break” and “Abort” can have the voice command be just one word and the macro simply spelling the word out. For codes which utilizes abbreviation in written form for simplicity in typing process, for example f3i representing feeder three input, when it comes to saying it out loud, it might be more comfortable to say the whole thing out rather than saying “ef-three-ai”. In such a case, highly flexible advanced scripting macro is a great solution. With only a few lines of script, it allows the operator to say the code in its actual spoken form in order to write it out in its own written form (Figure 5.2). Expanding from the same method even further, it is possible to create one macro which covers many voice commands and whose command names and output keystrokes can be completely unrelated in terms of written form and spoken form. For example, a macro is developed such that it will produce keystrokes of “a-p-p-r-o” when it hears “approach”, “get close”, and “go near”, and it will produce “d-e-p-a-r-t” upon recognizing “depart”, “leave”, and “back out”. This provides the operator more freedom in choosing command name as well as ability to program by speaking naturally.

The survey collected from 8 participants in the experiment showed positive impression on using voice command to write program codes and navigate around the program pages and lines. Many respondents mentioned how easy and fast it was to edit a line or a block of program simply by saying a word. They also mentioned how accurate and quick to write a long line of code which includes multiple parameters without making any typos. Some participants also provided a suggestion to improve the macros to allow even more convenience in program editing, for example, to add commands for copying and cutting multiple lines of program by saying the
command once. This can be done easily by going back to the original commands and adding a variable for a number of lines to their name. Despite all the positive comments on writing and editing program with voice, there are times when manually typing is faster. Therefore, the most effective way to write a program is to use a combination of both voice and hands.
Figure 5.2: Advanced scripting programming to write (a) basic code with only sequence of keystrokes, (b) intermediate code that takes at least one variables which depend on spoken words, and (c) advance code whose desired written form and spoken form are not the same

Figure 5.3: Example of advance code whose written form (actual V+ code) is different from its spoken form (voice command for writing the code)
5.3 Human-Robot Interaction Phase

The V+ programming script successfully allows the operator to select the actions at will for the robot to perform. Many sections of the script include prompts which request inputs from the operator. Some of them are numerical values, while the others are abbreviated point location names. The advanced scripting macro allows the operator to provide the necessary inputs by speaking to the robot. The important point is that the macro lets the operator answer the prompt questions in a natural way rather than spelling out the abbreviated name or number. For example, when being asked, “What can I do for you today?” the operator can say, “pick and place.” The macro, upon successful recognition, changes the spoken phrase into a number corresponding to that of the program block in the script. The macro then passes the number into the program, causing the program to jump to the corresponding program block and, as a result, activate the subprogram of choice. This makes the interaction sound more like natural conversation than replying to the robot by saying “three” in order to activate program block #3 (which is the pick-and-place subprogram in this case). The macro also utilizes the “TTSPlayString” function which provides voice feedback to the operator in addition to the text feedback provided separately by the Adept Windows.

The program, especially the pick-and-place subprogram, was tested for bugs and failures. Throughout several rounds of test run, bad inputs were intentionally used to determine the consequence of the failed program. The only critical one is the input for number of golf balls – bad input for this prompt causes the program to abort. If the inputs for pick-and-place locations do not follow the restriction of the program, the program’s first block will be executed instead. This allows for second chance without the need to re-run the whole program again. This bug was intentionally left alone because it is actually helpful when the command recognition
misinterprets the speech and, as a result, feed in incorrect texts, which can occur rather often. The pick-and-place environment also utilizes proximity sensors to detect the presence of golf balls in each feeder. This allows for more complexity in programming. Thus, the program was designed such that if the sensor detects no golf ball remaining in a feeder, it will stop any further pick-and-place operations occurring on that particular feeder and loop back for selection of different action.

The survey collected from the 8 participants showed strong interest in capability to talk to and command the robot by speaking in a natural manner. Pick-and-place part of the program that allows an operator to move golf balls from one location to another simply by saying three parameters received extra attention. Many comments stated that it is much easier and faster to perform the pick-and-place operation via the interaction program than commanding line-by-line as done in the direct control phase of the experiment. Many participants also found the voice feedback from the robot rather amusing, especially when they had the small conversation with the robot. The overall rating proved the success of the human-robot interaction with voice.
Figure 5.4: Successful run without bad prompt inputs leads the program to loop back for more actions

Figure 5.5: Proximity sensor tells the program to send a message to the operator when the feeder runs out of golf ball, cancel the remaining pick-and-place operations, and loop back
Figure 5.6: Bad point locations do not abort the program but instead cause it to switch to executing the first block and, after the action is over, loop back.

Figure 5.7: Bad prompt input for number of golf ball (e.g. anything else but numerical number), on the other hand, causes the program to abort.
5.4 Conclusion

Dragon NaturallySpeaking Professional 12 can be used as an automatic speech recognition software to develop macros which successfully convert a user’s voice command into a sequence of keystrokes. This shows that the use of the Dragon software makes it possible to replace a keyboard and a mouse with voice for robot control and program development tasks. The Dragon’s text-to-speech function, along with V+ language programming script, allows for successful development of the proposed voice-controlled human-robot interface. In the case of communicating with AdeptThree-XL robot, advanced scripting is the only macro type successfully passing the keystrokes into Adept Windows software interface while step-by-step and macro recorder fail to do the job. The strong points of the advanced scripting macro are that the macro can be created quickly and easily as well as used to develop flexible voice commands. The results of the experiment on the Dragon's efficiency in voice recognition came out as CRR = 85%, RSR = 23.03% for short commands, and 34.12% for long commands. The CRR indicated moderately accurate voice command recognition while the two RSR indicated long period of time between when the user finished saying a command and the successful recognition and execution of command. Under the same methodology, the Dragon software can be used to control virtually any other robots using different programming language simply by adjusting the output keystrokes to suit the target language.
5.5 Future Work

Future work for this research will mainly focus on applying the same methodology of using Dragon NaturallySpeaking in voice communication with a robot to another robots which use different programming languages. The robot program can also be further developed for more complexity in order for the robot to understand all variables required to perform a task from only one sentence of voice command. For example, by simply saying “get me an orange juice,” the robot should know what to do and how to do. More complex robot program also expands the range of possible applications, for example commanding a robot arm by voice to play a board game like chess for user with disability. It is also important to make this voice-controlled HRI applicable in various languages other than English such as Japanese.
Acknowledgements

This section is dedicated to all individuals who have helped me reach this point of success. Firstly, I would like to show my gratitude to Dr. S. Manian Ramkumar, my thesis adviser who worked with me through the whole processes of planning, researching, and writing the thesis as well as supported in funding for the Dragon NaturallySpeaking software. I would also like to thank Yanelys Alvarado for working with me in scheduling each appointment with Dr. Ramkumar.

Next, I would like to give my thanks to Noah Peterham for supporting me in obtaining and installing the Dragon software, for providing various technical supports, and for his sincere attitude. My thank also goes to Sun Woo Ji for teaching me all about the Adept robot to the point that I can work with it myself, providing me useful resources regarding the robot programming software, and being a very welcoming assistant. I would like to thank Professor Larry Kiser for helping me out during the research process by guiding me directions to the useful resources and providing helpful advices in the programming aspect. I would also like to thank 8 participants for their friendly cooperation in the experiment and helpful feedbacks.

I would like to give my special thanks to all of my friends and colleagues: Daniel Appiah-Mensah, Keagan Turner, Christopher Rhoades, Hiroki Tanaka, Andrew Choi, Kristina Chalker, Thanyathon Padungsong, Nuttapat Suntharapa, and Jakarin Chanpapatpol, who went through this hardship together and provided supports and encouragement along the way. Lastly, yet the most important individuals, I would like to express my thanks from the bottom of my heart to my loving and caring parents who are always there for me and provide me supports and encouragement. Without them, I would not arrive at this point of success. Truly thanks.
References


doi:10.1109/ICPR.2000.903007


doi:10.1016/S0921-8890(02)00166-5


doi:10.1109/ROMAN.2001.981912


doi:10.1109/ROBOT.2001.933255


Appendix

A. Advanced Scripting Macro Scripts

This section lists all macros that are used in the experiment and their corresponding advanced scripting programming codes which indicate the tasks performed upon command recognition. The macros are divided into three groups: Group 1 macros are used in Direct Control, Group 2 macros are used in Program Creation, Group 3 macros are used in Human-Robot Interaction. Macros will be shown in the following format.

Command Name

Sub Main
  List of Tasks
End Sub

<List Variable> : All words included in the List Variable
A.1 Direct Control

<exerun> <progname>

Sub Main
    SendKeys "exe " + ListVar2 + "[enter]"
    TTIPlayString "hello! may i have your name please?"
End Sub

<exerun>: execute / run
<progname>: test1 / test2 / test3

<tool> off

Sub Main
    SendKeys "sig -1[enter]"
End Sub

<tool>: end effector / vacuum

<tool> on

Sub Main
    SendKeys "sig 1[enter]"
End Sub

<tool>: end effector / vacuum

abort

Sub Main
    SendKeys "abort[enter]"
End Sub

calibrate

Sub Main
    TTIPlayString "give me a minute"
    SendKeys "cal[enter]y[enter]"
End Sub
disable power

Sub Main
  TTSPayString "the power is down"
  SendKeys "dis po{enter}"
End Sub

drive joint <1to3> for <distrange> at <1to100>

Sub Main
  TTSPayString "begin driving in 2 seconds"
  Wait 4
  SendKeys "do drive " + ListVar1
  SendKeys "," + ListVar2
  SendKeys "," + ListVar3
  SendKeys "{enter}"
End Sub

<1to3> : 1 - 3  
<distrange> : -100 - 100
<1to100> : 1 - 100

enable power

Sub Main
  TTSPayString "the power button is ready"
  SendKeys "en po{enter}"
End Sub

go <feeder>

Sub Main
  slash_posn = InStr (ListVar1, ")")
  word_val$ = Mid$(ListVar1, 1, slash_posn -1)
  spoken$ = Mid$(ListVar1, slash_posn +1)
  TTSPayString "heading to " + spoken$
  Wait 2.5
  SendKeys "do move " + word_val$ + "{enter}"
End Sub

<feeder> : f1\ feeder one input / f1o\ feeder one output / f2\ feeder two input / 
f2o\ feeder two output / f3\ feeder three input / f3o\ feeder three output / safe\ safe point
go <locname>

```vbnet
Sub Main
    TTSPlayString "heading to " + ListVar1
    Wait 2
    SendKeys "do move " + ListVar1 + "{enter}"
End Sub
```

<locname> : home / point1 / point2 / point3 / point4 / point5 / point6 / point7 / point8 / point9 / safe

---

speed <1to100>

```vbnet
Sub Main
    SendKeys "speed " + ListVar1 + "{enter}"
End Sub
```

<1to100> : 1 – 100

---

stop now

```vbnet
Sub Main
    SendKeys "dis po{enter}"  
    TTSPlayString "all actions have been canceled"
End Sub
```

---

teach point <feeder>

```vbnet
Sub Main
    slash_posn = InStr (ListVar1, "\")
    word_val$ = Mid$ (ListVar1, 1, slash_posn -1)
    spoken$ = Mid$ (ListVar1, slash_posn +1)
    TTSPlayString spoken$ + ", target locked"
    SendKeys "here " + word_val$ + "{enter}{enter}"
End Sub
```

<feeder> : f1\feeder one input / f1o\feeder one output / f2i\feeder two input / f2o\feeder two output / f3i\feeder three input / f3o\feeder three output / safe\safe point
teach point `<locname>`

```plaintext
Sub Main
    TTSPlayString ListVar1 + ", target locked"
    SendKeys "here " + ListVar1 + "{enter}{enter}"
End Sub
```

`<locname>` : home / point1 / point2 / point3 / point4 / point5 / point6 / point7 / point8 / point9 / safe

where are you now

```plaintext
Sub Main
    TTSPlayString "here's the coordinate"
    SendKeys "where{enter}"
End Sub
```

where is `<feeder>`

```plaintext
Sub Main
    slash_posn = InStr (ListVar1, "\")
    word_val$ = Mid$ (ListVar1, 1, slash_posn -1)
    spoken$ = Mid$ (ListVar1, slash_posn +1)
    TTSPlayString "here's the coordinate for " + spoken$
    SendKeys "point " + word_val$ + "{enter}{enter}"
End Sub
```

`<feeder>` : f1i\feeder one input / f1o\feeder one output / f2i\feeder two input /
               f2o\feeder two output / f3i\feeder three input / f3o\feeder three output / safe\safe point

where is `<locname>`

```plaintext
Sub Main
    TTSPlayString "here's the coordinate for " + ListVar1
    SendKeys "point " + ListVar1 + "{enter}{enter}"
End Sub
```

`<locname>` : home / point1 / point2 / point3 / point4 / point5 / point6 / point7 / point8 / point9 / safe
A.2 Program Creation

<monitor>

    Sub Main
    SendKeys "{F4}"
    End Sub

<monitor> : exit program / monitor mode / teach mode

---------------------------

<subprog> sub program

    Sub Main
    SendKeys "{F3}"
    End Sub

<subprog> : load / open

---------------------------

approach <feeder> <1to100>

    Sub Main
    slash_posn = InStr (ListVar1, "\")
    word_val$ = Mid$ (ListVar1, 1, slash_posn -1)
    SendKeys "appro " + word_val$ + ", " + ListVar2 + "{enter}"
    End Sub

<feeder> : f1\feeder one input / f1o\feeder one output / f2i\feeder two input /
          f2o\feeder two output / f3i\feeder three input / f3o\feeder three output / safe\safe point
<1to100> : 1 - 100

---------------------------

approach <locname> <1to100>

    Sub Main
    SendKeys "appro " + ListVar1 + ", " + ListVar2 + "{enter}"
    End Sub

<locname> : home / point1 / point2 / point3 / point4 / point5 / point6 / point7 / point8 / point9 /
           safe
<1to100> : 1 – 100

---------------------------

begin of line

    Sub Main
    SendKeys "{esc},i"
    End Sub

---------------------------
bottom page

```vbnet
Sub Main
    SendKeys "{esc}\bi"
End Sub
```
break

```vbnet
Sub Main
    SendKeys "break{enter}" 
End Sub
```
clear <1to100> buffer

```vbnet
Sub Main
    SendKeys "{esc}"
    Wait 0.1
    SendKeys ListVar1 
    Wait 0.1
    SendKeys "{esc}ki"
End Sub
```
<1to100>: 1 – 100

copy line

```vbnet
Sub Main
    SendKeys "{F9}"
End Sub
```
create program <proname>

```vbnet
Sub Main
    SendKeys "see " + ListVar1 + "{enter}"
    Wait 1
    SendKeys "y{enter}"
    Wait 1
    SendKeys "i"
End Sub
```
<proname>: test1 / test2 / test3
cut line

```vbnet
Sub Main
    SendKeys "+{F9}"
End Sub
```

delay <1to100>

```vbnet
Sub Main
    SendKeys "delay " + ListVar1 + "{enter}"
End Sub
```

<1to100> : 1 – 100

delete buffer

```vbnet
Sub Main
    SendKeys "{esc}^ki"
End Sub
```

delete line

```vbnet
Sub Main
    SendKeys "{esc}^di"
End Sub
```

depart <1to100>

```vbnet
Sub Main
    SendKeys "depart " + ListVar1 + "{enter}"
End Sub
```

<1to100> : 1 – 100

down <1to100>

```vbnet
Sub Main
    SendKeys "{down} + ListVar1 + ""
End Sub
```

<1to100> : 1 – 100
edit <progname>

    Sub Main
        SendKeys "see " + ListVar1 + "{enter}"
        Wait 1
        SendKeys "i"
    End Sub

<progname> : test1 / test2 / test3

end of line

    Sub Main
        SendKeys "{esc}.i"
    End Sub

insert mode

    Sub Main
        SendKeys "i"
    End Sub

left <1to100>

    Sub Main
        SendKeys "{left } + ListVar1 + "}"
    End Sub

<1to100> : 1 – 100

left word <1to100>

    Sub Main
        SendKeys "{esc}"
        Wait 0.1
        i=0
        Do Until i=ListVar1
            SendKeys "{esc}"
            Wait 0.1
            SendKeys "{tab}"
            Wait 0.1
            i=i+1
        Loop
        SendKeys "i"
    End Sub

<1to100> : 1 – 100
move <feeder>

 Sub Main
     slash_posn = InStr (ListVar1, "\")
     word_val$ = Mid$(ListVar1, 1, slash_posn -1)
     SendKeys "move " + word_val$ + "{enter}"
 End Sub

<feeder> : f1\feeder one input / f1o\feeder one output / f2\feeder two input /
           f2o\feeder two output / f3\feeder three input / f3o\feeder three output / safe\safe point

move <locname>

 Sub Main
     SendKeys "move " + ListVar1 + "{enter}"
 End Sub

<locname> : home / point1 / point2 / point3 / point4 / point5 / point6 / point7 / point8 / point9 / safe

move to line <1to100>

 Sub Main
     SendKeys "{esc}" + ListVar1 + "li"
 End Sub

<1to100> : 1 – 100

paste

 Sub Main
     SendKeys "{esc}"+"ai"
 End Sub

paste all

 Sub Main
     SendKeys "+[F10]"
 End Sub

read mode

 Sub Main
     SendKeys "{esc}"
 End Sub
repeat paste

    Sub Main
    SendKeys "{esc}-ai"
End Sub

repeat search

    Sub Main
    SendKeys "{F8}"    ' \nEnd Sub

right <1to100>

    Sub Main
    SendKeys "\{right " + ListVar1 + "}"
End Sub
<1to100> : 1 – 100

right word <1to100>

    Sub Main
    SendKeys "\{esc}\{tab " + ListVar1 + "}i"
End Sub
<1to100> : 1 – 100

search

    Sub Main
    SendKeys "\{home}\{F7}"
End Sub

search replace

    Sub Main
    SendKeys "\{home}+{F7}"
End Sub
sig 1

    Sub Main
    SendKeys "sig 1{enter}"  
    End Sub

---

sig minus 1

    Sub Main
    SendKeys "sig -1{enter}"  
    End Sub

---

speed <1to100> always

    Sub Main
    SendKeys "speed " + ListVar1 + " always{enter}"  
    End Sub

<1to100> : 1 – 100

---

toggle program

    Sub Main
    SendKeys "+{F3}"  
    End Sub

---

top page

    Sub Main
    SendKeys "{esc}^ti"  
    End Sub

---

undo

    Sub Main
    SendKeys "{F6}"  
    End Sub

---

up <1to100>

    Sub Main
    SendKeys "{up } + ListVar1 + "}"  
    End Sub

<1to100> : 1 – 100
A.3 Human-Robot Interaction

<username>

Sub Main
    TTSPayString "welcome back " + ListVar1 + ", what can i do for you today?"
    SendKeys ListVar1 + "{enter}"
End Sub

<username> : Master / Pomm

<reply1>

Sub Main
    SendKeys "2{enter}"
    Wait 29
    TTSPayString "i hope you like my moves, what else can i do for you today?"
End Sub

<reply1> : move around / show some moves

<reply2>

Sub Main
    TTSPayString "please tell me where to pick up a golf ball.", "/s180"
    SendKeys "3{enter}"
End Sub

<reply2> : move golf ball / pick and place

<reply2.1>

Sub Main
    slash_posn = InStr (ListVar1, "\")
    word_val$ = Mid$ (ListVar1, 1, slash_posn -1)
    spoken$ = Mid$ (ListVar1, slash_posn +1)
    TTSPayString spoken$ + ", roger that"
    Wait 3
    SendKeys word_val$ + "{enter}"
    TTSPayString "please tell me where to place a golf ball.", "/s180"
End Sub

<reply2.1> : f1o\feeder one output / f2o\feeder two output / f3o\feeder three output
<reply2.2>

Sub Main
    slash_posn = InStr (ListVar1, "\")
    word_val$ = Mid$(ListVar1, 1, slash_posn -1)
    spoken$ = Mid$(ListVar1, slash_posn +1)
    TTSPlayString spoken$ + ", roger that"
    Wait 3
    SendKeys word_val$ + "{enter}"
    TTSPlayString "how many golf balls would you like me to move?", "/s180"
End Sub

<reply2.2> : f1i\feeder one input / f2i\feeder two input / f3i\feeder three input

<reply2.3>

    Sub Main
        TTSPlayString "alright, here we go!"
        Wait 2.5
        SendKeys ListVar1 + "{enter}"
    End Sub

<reply2.3> : 1 - 6

<reply3>

    Sub Main
        SendKeys "4{enter}"
        TTSPlayString "how are you today?"
    End Sub

<reply3> : let's talk / talk to me

<reply3.1>

    Sub Main
        TTSPlayString "i'm glad you're doing well"
        SendKeys "11{enter}"
        Wait 3
        TTSPlayString "what else can i do for you today?"
    End Sub

<reply3.1> : doing good / i'm fine thanks
<reply3.2>

Sub Main
    TTSPlayString "i'll do something to make you happy"
    SendKeys "12(enter)"
    Wait 2.5
    HeardWord "open", "pcmeranian"
    Wait 3
    TTSPlayString "what else can i do for you today?"
End Sub

<reply3.2> : not too good / ok I guess

<reply4>

Sub Main
    SendKeys "5(enter)"
    TTSPlayString "see you next time then"
End Sub

<reply4> : no thanks / that's fine
B. V+ Programming Scripts

This section contains scripts of the whole V+ language program which is executed in the Human-Robot Interaction phase of the experiment. The scripts include a main program and four subprograms.

```plaintext
.PROGRAM test1()

MOVE safe
SPEED 50
CALL userinfo($userinfo)

1
PROMPT "What can I do for you today? ", choice
CASE choice OF
    VALUE 2:
    GOTO 2
    VALUE 3:
    GOTO 3
    VALUE 4:
    GOTO 4
    VALUE 5:
    ABORT
END

2
CALL movearound()
GOTO 1

3
CALL mover()
GOTO 1

4
CALL talktome($userinfo)
GOTO 1

```

```plaintext
.PROGRAM userinfo($username)

PROMPT "Hello! May I have your name please? ", $username
IF $username == "" THEN
    $username = "Master"
ELSE
END

TYPE "Welcome back "+$username+"! "
```
.PROGRAM movearound()

1
   MOVE point1
   BREAK
   APPRO f1o, 50
   DELAY 1
   MOVE point1
   BREAK

2
   MOVE point2
   BREAK
   APPRO f2o, 50
   DELAY 1
   MOVE point2
   BREAK

3
   MOVE point3
   BREAK
   APPRO f3o, 50
   DELAY 1
   MOVE point3
   BREAK

4
   MOVE safe
   BREAK
   MOVE f1i
   DELAY 1
   MOVE safe
   BREAK
   MOVE f2i
   DELAY 1
   MOVE safe
   BREAK
   MOVE f3i
   DELAY 1
   MOVE safe
.PROGRAM mover()

0

PROMPT "Please tell me where to pick up a golf ball. ", $fromloc
PROMPT "Please tell me where to place a golf ball. ", $toloc
PROMPT "How many golf balls would you like me to move? ", ballnum

IF $fromloc == "f10" THEN
    IF $toloc == "f11" GOTO 1
    IF $toloc == "f21" GOTO 2
    IF $toloc == "f31" GOTO 3
END

IF $fromloc == "f20" THEN
    IF $toloc == "f11" GOTO 4
    IF $toloc == "f21" GOTO 5
    IF $toloc == "f31" GOTO 6
END

IF $fromloc == "f30" THEN
    IF $toloc == "f11" GOTO 7
    IF $toloc == "f21" GOTO 8
    IF $toloc == "f31" GOTO 9
END

1

FOR i = 1 TO ballnum
    IF $(SIG(1001)) == -1 THEN
        APPRO f1o, 100
        BREAK
        MOVE f1o
        SIGNAL 1
        DELAY 1
        DEPART 100
        BREAK
        MOVE safe
        BREAK
        MOVE f1i
        BREAK
        SIGNAL -1
        DELAY 1
        MOVE safe
        BREAK
    ELSE
        TYPE "There is no golf ball left here."
        GOTO 10
    END
END

END
END
GOTO 10

2
FOR i = 1 TO ballnum
   IF (SIG(1001) == -1) THEN
       APPRO f10, 100
       BREAK
       MOVE f10
       SIGNAL 1
       DELAY 1
       DEPART 100
       BREAK
       MOVE safe
       BREAK
       MOV f2i
       BREAK
       SIGNAL -1
       DELAY 1
       MOVE safe
       BREAK
   ELSE
       TYPE "There is no golf ball left here."
       GOTO 10
   END
END
GOTO 10

3
FOR i = 1 TO ballnum
   IF (SIG(1001) == -1) THEN
       APPRO f10, 100
       BREAK
       MOVE f10
       SIGNAL 1
       DELAY 1
       DEPART 100
       BREAK
       MOVE safe
       BREAK
       MOV f3i
       BREAK
SIGNAL -1
DELAY 1
MOVE safe
BREAK
ELSE
    TYPE "There is no golf ball left here."
    GOTO 10
END
END
GOTO 10

4 FOR i = 1 TO ballnum
    IF (SIG(1002) == -1) THEN
        APPRO f2o, 100
        BREAK
        MOVE f2o
        SIGNAL 1
        DELAY 1
        DEPART 100
        BREAK
        MOVE safe
        BREAK
        MOVE f1i
        BREAK
        MOVE f1i
        SIGNAL -1
        DELAY 1
        MOVE safe
        BREAK
    ELSE
        TYPE "There is no golf ball left here."
        GOTO 10
    END
END
GOTO 10

5 FOR i = 1 TO ballnum
IF (SIG(1002) == -1) THEN
    APPRO f2o, 100
    BREAK
    MOVE f2o
    SIGNAL 1
    DELAY 1
    DEPART 100
    BREAK
    MOVE safe
    BREAK
    MOVE f2i
    BREAK
    SIGNAL -1
    DELAY 1
    MOVE safe
    BREAK
ELSE
    TYPE "There is no golf ball left here."
    GOTO 10
END
GOTO 10

FOR i = 1 TO ballnum
    IF (SIG(1002) == -1) THEN
        APPRO f2o, 100
        BREAK
        MOVE f2o
        SIGNAL 1
        DELAY 1
        DEPART 100
        BREAK
        MOVE safe
        BREAK
        MOVE f3i
        BREAK
        SIGNAL -1
        DELAY 1
        MOVE safe
        BREAK
ELSE
  TYPE "There is no golf ball left here."
  GOTO 10
END
GOTO 10

7
FOR i = 1 TO ballnum
  IF (SIG(1003) == -1) THEN
    APPRO f3o, 200
    BREAK
    MOVE f3o
    SIGNAL 1
    DELAY 1
    DEPART 200
    BREAK
    MOVE safe
    BREAK
    MOVE fli
    BREAK
    SIGNAL -1
    DELAY 1
    MOVE safe
    BREAK
  ELSE
    TYPE "There is no golf ball left here."
    GOTO 10
  END
END
GOTO 10

8
FOR i = 1 TO ballnum
  IF (SIG(1003) == -1) THEN
    APPRO f3o, 200
    BREAK
    MOVE f3o
    SIGNAL 1
    DELAY 1
DEPART 200
BREAK
MOVE safe
BREAK
MOVE f2i
BREAK
SIGNAL -1
DELAY 1
MOVE safe
BREAK
ELSE
   TYPE "There is no golf ball left here."
   GOTO 10
END
END
GOTO 10
9
FOR i = 1 TO ballnum
   IF (SIG(1003) == -1) THEN
      APPRO f3o, 200
      BREAK
      MOVE f3o
      SIGNAL 1
      DELAY 1
      DEPART 200
      BREAK
      MOVE safe
      BREAK
      MOVE f3i
      BREAK
      SIGNAL -1
      DELAY 1
      MOVE safe
      BREAK
   ELSE
      TYPE "There is no golf ball left here."
      GOTO 10
   END
END
END
GOTO 10
10
DELAY 1

----- mover ----------------- Step 238 of 238 ----- Command mode ---------

.PROGRAM talktome($username)
  PROMPT "How are you today, "+$username+"? ", answer1
  CASE answer1 OF
    VALUE 11:
      TYPE "I am glad you are doing well, "+$username+"."
    VALUE 12:
      TYPE "I will do something to make you happy, "+$username+"."
  END
END
DELAY 2