Gesture and Speech Controlled Robotics
Summer Project Report

Yvan Seth
yvan@seth.id.au

April 22, 2003
Contents

1 Introduction ............................................. 2

2 Research ................................................ 3
   2.1 Speech Recognition ............................. 3
      2.1.1 Brief History ............................ 3
      2.1.2 Available Development Solutions ....... 3
      2.1.3 ViaVoice .................................. 4
      2.1.4 Sphinx .................................... 5
   2.2 Gesture Recognition ............................ 5
   2.3 Combining the Technologies .................. 6

3 Design .................................................. 7
   3.1 Important Considerations ..................... 7
   3.2 High Level Design .............................. 7
   3.3 Examining the Components .................... 7
      3.3.1 The Robots .............................. 7
      3.3.2 Robot Control ........................... 7
      3.3.3 Speech Input ............................. 8

4 Implementation ....................................... 9
   4.1 Technology Decisions ......................... 9
      4.1.1 Platform .................................. 9
      4.1.2 Speech Recognition ....................... 9
      4.1.3 Mindstorms ............................... 9
      4.1.4 Miscellaneous ........................... 10
   4.2 The Robots .................................... 10
      4.2.1 Design .................................. 10
      4.2.2 Code .................................... 11
   4.3 MStorms Control ............................... 11
   4.4 Voice Client ..................................... 13

5 Concluding Remarks ................................. 16
   5.1 Improvements .................................. 16
   5.2 Extensions .................................... 16

A Code Examples ....................................... 18
   A.1 Robot Skeleton Code ........................... 18
   A.2 Sample Robot Actions ......................... 19

B Configuration Files ................................. 21
   B.1 Voice Client Configuration .................. 21
   B.2 Robot Control Configuration ................ 22

C mstorns Setup ...................................... 24
   C.1 Linux Configuration ........................... 24
   C.2 Software Installation ......................... 24
      C.2.1 Quick Install ............................. 24
      C.2.2 Manual Install ........................... 25
   C.3 Setting Up ..................................... 26
1 Introduction

The speech and gesture controlled robotics project was began in December 2002 with the initial goal of exploring relevant technologies and building a framework for further exploration of control systems. We’re interested in user control systems for all kinds of applications using relatively natural interaction in a non-ideal environment. In such a system any particular user should be able to successfully issue commands even while there may be background noise from conversation or other sources. It is believed that integrating gesture recognition and speech recognition would be a step towards the realisation of such a system.

This report covers the work done in building a speech recognition based control system for simple robots. This is a starting point for the exploration of a multi-modal approach to robot control, further work should now be done to explore the combination of speech and gesture recognition. The system implemented puts a layer of abstraction between the speech processing code and the robot control making it easier to develop alternative command input systems.
2 Research

2.1 Speech Recognition

2.1.1 Brief History

Academic interest in speech recognition dates back to the late 1940s when the US government funded research into the field. The aim was to build a system capable of processing Russian communications, this project was a complete failure and gave us the famous example of “the spirit is willing but the flesh is weak” being translated into the Russian for “the vodka is strong but the meat is disgusting”. This was just the start though, research into speech recognition at institutions like CMU and MIT continued. Slowly progress continued toward the goal of continuous speech recognition, aided by huge increases in computer processing power.

Through the decades since the late 40s much progress was made. In the mid 90s speech recognition was being used for user interactive phone systems. In the late 90s speech recognition software began to become available on supermarket shelves though it wasn't much use, today those same products are much improved. These days good continuous speech recognition systems can downloaded for use by anyone. It is fair to say that the technology of speech recognition is a mature one.

2.1.2 Available Development Solutions

There are quite a few speech recognition systems available to developers these days. These systems span a wide range of target platforms and uses, most of them are proprietary and several come at a high cost, a very small number can be used or experimented with for no charge. Some examples of the solutions available can be found in the table on page 4. The best approaches for real-time continuous speech recognition involve expensive hardware speech processors such as Intel’s Dialogic, however the software-only solutions can provide close to real-time recognition on a modern desktop computer.

As many of the solutions available are not accessible due to their high cost research was only carried out for IBM's ViaVoice and CMU's Sphinx, both of which can be used freely (A trial version of ViaVoice may be used free of charge for educational purposes).
<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Platforms</th>
<th>Description</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dialogic¹</td>
<td>Intel</td>
<td>Windows, Linux</td>
<td>A combined hardware/software solution aimed at telephone based interactive systems, used by several commercial recognition systems.</td>
<td>$600-$1500</td>
</tr>
<tr>
<td>CASSI Speech Recognition²</td>
<td>Conversay</td>
<td>Windows, Linux, PDAs</td>
<td>Designed to be compact with mobile device and Internet applications in mind, focus on a distributed approach.</td>
<td>Available By Request.</td>
</tr>
<tr>
<td>Dragon Naturally Speaking³</td>
<td>ScanSoft</td>
<td>Windows</td>
<td>Available on store shelves this is a Windows application for both Control and Dictation. There are development tools available for building your own software around the recognition engine.</td>
<td>Available By Request.</td>
</tr>
<tr>
<td>ViaVoice⁴</td>
<td>IBM</td>
<td>Windows, Mac, Linux(?)</td>
<td>Known as a brilliant system to develop with under a variety of platforms (though IBM's Linux support for ViaVoice seems to have recently been dropped).</td>
<td>$359 for consumer product, trial available.</td>
</tr>
<tr>
<td>Sphinx⁵</td>
<td>CMU</td>
<td>Any (available as source)</td>
<td>Sphinx is a product of Speech Recognition research at CMU, it is very accurate and configurable. Comes with command-line tools and libraries are available for C and Java.</td>
<td>$FREE</td>
</tr>
</tbody>
</table>

Table 1: Voice Recognition Solutions

### 2.1.3 ViaVoice

ViaVoice is widely accepted as “the speech recognition package of choice” it is a commercial product from IBM and is commonly used on the Windows platform for dictation. IBM has also released a publicly available SDK to encourage the development of speech recognition enabled software, however in order to use such software permanently a user would need to purchase the ViaVoice engine.

---

⁵ [http://www.speech.cs.cmu.edu/sphinx/](http://www.speech.cs.cmu.edu/sphinx/)
ViaVoice would be a prime candidate for use in this project as it has a good reputation for recognition accuracy. However I had trouble finding the Linux SDK on the ViaVoice website, it appeared to have ceased to exist. I have recently discovered that IBM has withdrawn ViaVoice Dictation for Linux and with it the SDK.

2.1.4 Sphinx

There are three versions of Sphinx available, I chose to examine Sphinx-2 which is the current ‘stable’ version and is optimised for continuous speech recognition. Sphinx is the result DARPA\(^{10}\) funded research that started several decades ago (back when it was known as ARPA\(^{11}\)), to this day this research continues and CMU has done a large amount of work in the area. Quoting from their website:

The Sphinx Group at Carnegie Mellon University is committed to releasing the long-time, DARPA-funded Sphinx projects widely, in order to stimulate the creation of speech-using tools and applications, and to advance the state of the art both directly in speech recognition, as well as in related areas including dialog systems and speech synthesis.

Sphinx has many features, the list below shows some of the most useful of them in the context of this project.

- Highly Configurable.
- Software provided for construction your own acoustic models.
- Software provided to build your own specialised vocabularies.
- Can provide multiple possible interpretations of an utterance.
- Full source code available.
- BSD Style license (you can do anything at all with Sphinx so long as you include the Sphinx copyright notice).
- Comes with enough usage examples to get you started.
- Comes with a good acoustic model based on hundreds of hours of input (unfortunately this has been built from American news broadcasts).

2.2 Gesture Recognition

Natural gesture recognition technologies are much younger than speech recognition, at this time there is much active research in the field and little in the way of publicly available implementations. Recognition using special gloves and similar devices has been around for some time now\(^{12}\), however we're

\(^{10}\text{Defence Advanced Research Projects Agency}\)
\(^{11}\text{Advanced Research Projects Agency}\)
\(^{12}\text{Similar to speech recognition, research into glove-type gesture recognition was funded by ARPA and began in the mid 60s.}\)
interested in systems that do not inconvenience the user. Natural gesture recognition involves a computer system interpreting the body movements of a person based on video input, modern processor speeds and video technology makes this approach realistic. Several approaches to interpreting gestures are being explored within the academic community; using methods ranging from pure statistical modelling to neural networks.

Since at this stage the technology is developmental, seeking open and available implementations of gesture recognition systems proved unsuccessful. The best programming resource I found in relation to gesture recognition was the Intel Open Computer Vision Library\textsuperscript{13} which provides a library of tools which would be of use in developing a gesture recognition system.

Practical exploration of gesture recognition is left as a task for further study.

2.3 Combining the Technologies

There has long been some interest in combining speech and gesture recognition for HCI, some of the earliest successful work in the area is that of \cite{?} in which a system was developed that combines speech and pointing to interact with a large computer screen. More recent analysis, such as \cite{?}, has moved into the areas of natural language. In this case there are no specific gesture commands, instead the natural gesticulation of a subject is observed and correlated to their speech with the aim of deriving statistical data which can be used to improve speech recognition.

\footnote{http://www.intel.com/research/mrl/research/opencv/}
3 Design

3.1 Important Considerations

The most important outcome of this project is that it should be easy for someone working with HCI for robot control to interface with Mindstorms robots. The design of this system was approached with this in mind, the aim was to hide all the low level parts of the system behind a relatively high level interface.

In order to have a system that is as extensible as possible it was decided that there should be a network communications interface. This will allow the core processing to be separated from the robot control, for example the control unit could be run on an on-board handheld computer with 802.11b support while the speech processing occurs on a high end desktop PC.

Another issue that needs to be considered is that in the future people may wish to build different robots, when they do it should be easy to get them working with the system.

3.2 High Level Design

The final design is quite modular, there is the Robot’s on-board logic and the robot control program which receives commands via the network. Part of this project was to build a sample implementation using speech recognition to control robots, this forms a third module for this implementation. Figure 3 shows a diagram of these three main components and how they interrelate.

3.3 Examining the Components

3.3.1 The Robots

The most important design consideration for the robots is that it should be easy to interface new robots with the system. This involves two separate items, the first is covered below in the Robot Control section and it is that configuring the control system for different robots should be simple. The second issue is that writing the code for the robot’s on-board logic should be simple, this is covered by choosing a language that is easy to understand and by providing a well commented example as well as a skeleton which can be filled out for arbitrary robot actions. Part of the design decision here involved choosing the best implementation option for the robot code, I decided that the LeJOS\textsuperscript{14} system best matched the required properties, there is more about this in the Implementation section of this report.

3.3.2 Robot Control

The robot control module is intended to create an abstraction between the low level communications with the robots and the command interface. This module handles the driving of the serial IR transmitter and the construction and sending of packets to the robot RCX units, this is the sort of thing we do not want to have to think about when we’re working on the high level control interface.

\textsuperscript{14}http://lejos.sf.net/
As well as simply being able to send data to the robots the robot control module needs an accessible interface providing a method allowing other software to send control messages to the robots. We also don't want the user interface software to have to worry about opcodes for actions and argument formatting, so what we really require is a higher level command set. Such a command set is specified in a configuration file that the control module takes as a command line parameter, an example of such a configuration file can be found in Appendix B.2.

It was decided that the software interface to the robot control module should be network based. The program should simply accept robot control directives as specified in the robot configuration file, translate them to instructions and transmit them to the robot. The return side of the protocol simply consists of a single integer, a return value of 200 is to be interpreted as meaning that the command received correctly translated into an instruction. There are only three error codes, 401 means that the command is completely indecipherable, 402 means that a required argument is missing, 403 means an argument has been given where none is expected and a 404 means that an argument value is outside of its acceptable range. These numbers were only chosen because they're similar to HTTP return codes.

The reason for specifying a network interface to the robot control module is that it would allow the control software to run on a handheld computer with a wireless network card. The handheld device could be rigged to communicate directly with the RCX and be mounted onboard a robot. This would allow maximum maneuverability and range of the robot and also add the potential for onboard logic capable of doing something more interesting that what can be achieved with the RCX alone. The handheld could be capable of driving the robot in a certain manner, in response to physical stimuli for example, when no instructions are being received from elsewhere. This also means that all the CPU intensive work like speech and gesture recognition can run on a desktop machine and this recognition software can send directives to the robot via the network.

3.3.3 Speech Input

The design for the speech input module is very simple at this stage as this is intended to just be an example implementation. I decided that configurability of the input grammar was an important point for this module and the design is based on the structure of a configuration file, an example of which can be seen in Appendix B.1. It should be easy for someone extending this system or interfacing the system with new robots to specify new language features and alter others, the configuration file is designed to make this as easy as possible.

All robot control should be done through a single instance of the speech input module so some method is required to determine which robot is being addressed. I believe that the best way to handle this is to use the concept of 'modes', so the speech recognition module will have special voice commands which are the assigned names of the robots, when one of these names is head it will switch into the required mode. The best thing about this is that it means that the same syntax can be used for different robots, the modules will parse input based on the grammar for the currently selected robot. I think that one simple ideal incorporation of gesture recognition would be to implement this modedness using gaze tracking.
4 Implementation

The final system that came out of this project enables users to control Lego robots using only their voice. This section explains in detail the construction of the system and implementation decisions made.

4.1 Technology Decisions

4.1.1 Platform

The first decision made was what Operating System to work with, my decision was to use Linux as a development platform. The main factor was personal preference, I’ve worked with Unix-like platforms for years and find it provides a much more efficient development environment than Windows. Using a machine with a 2.4 kernel is essential if you want to use the USB IR Towers (part of the Mindstorms system) by preference I was working under Debian/GNU\textsuperscript{15} Linux, however I have also had the system running on a RedHat\textsuperscript{16} Linux machine for demonstration purposes. A detailed explanation of system configuration can be found in Appendix A. Although there seem to be more options for speech recognition solutions available under Windows almost all of those are non-free so they cease to be a consideration for this project.

4.1.2 Speech Recognition

The choice of which speech recognition engine to use had to be made between ViaVoice and Sphinx, since there are no longer new versions of ViaVoice available for Linux I chose Sphinx for speech recognition. I used Sphinx-2, which is optimised for continuous speech recognition and can be downloaded by following the links from http://www.speech.cs.uu.edu.au/sphinx/. Sphinx was compiled without any problems on the three target Linux machines that I worked with.

4.1.3 Mindstorms

I decided not to use the standard firmware available on the Lego Mindsorums RCX as the programming interface is not very friendly if you’re dealing with it directly, as it provided a non-obvious programming interface. I chose to use LeJOS\textsuperscript{17}, which is a fork of the Java TinyVM project which implements a Java VM especially for the Lego RCX. The main advantage of using LeJOS is that the code in Java is really simple and easy to read, this means that there will be little difficulty in writing action code for a new robot. Appendix B contains an action code skeleton complete with comments explaining how to fill in functions to make a robot compatible with this system.

\textsuperscript{15}http://www.debian.org/
\textsuperscript{16}http://www.redhat.com/
\textsuperscript{17}http://lejos.sf.net/
4.1.4 Miscellaneous

Another important decision the languages to use for implementing the PC-side modules. The module that communicates with the RCX via the IR Transmitter hardware was written in C as this was the most logical choice for dealing with such a simple low-level task. The speech recognition module was written in Perl, the Perl program executes the sphinx-continuous program and then interprets any text that it generates. The decision to use Perl was made because it gave the fastest development time, Perl is very good for text processing. However any future system that also has to deal with gesture recognition data should probably be written in a language like C or C++ for the sake of maximising runtime performance.

4.2 The Robots

4.2.1 Design

After much trial and error two final robot designs were decided upon, they are shown in Figure 4.2.1. Gatherer is loosely based on a robot found on the web named Pevs\textsuperscript{18}, while Sorter is similar in design to an arm featured on the top of the Mindsorms V2.0 box. Gatherer is built to move around and pick up small objects (Lego bricks), it is driven differentially (each drive wheel is controlled by a separate motor) and uses two rotation sensors to gauge distance covered as well as turning angle. Gatherer uses a third motor to open and close the grabber on the front, two chained touch sensors are used to tell when the grabbing mechanism is either fully open or fully closed, elastic bands provide the closing force.

Sorter is a stationary arm that can turn through about 200 degrees, though it must turn in coarse increments as it uses a light sensor in combination with marks on the base board to use it’s turning. One motor drives the turning, while two at the rear top (rigged to the same power line) move the arm up and down. A third motor is used to open the hand, which is pulled closed by an elastic band when the motor is disengaged. A touch sensor is used to tell when the hand is fully open and

\textsuperscript{18}http://www.seattlerobotics.org/encoder/200108/using_a.pid.html
other is used for the full-left rotation position. The full-left rotation sensor is used to find the starting position when the robot is first switched on and also when the 'HOME' command is issued.

### 4.2.2 Code

The code for the robots is written in Java and is very simple. Initially it sets up a loop polling for IR input, when input is received it tries to decode it. The Mindstorms RCX's are limited to receiving data sequences that match one of its standard IR codes, so I can't just arbitrarily make up my own codes. I have chosen to use one particular code "Set motor power" (opcode 13\(^{19}\)) which has 3 one byte arguments. I use the first argument as an opcode and the last two form an optional argument to the opcode. This means that all codes received by the robots are prefixed with a byte containing 13, this is ignored. A sequence of if/else if statements decodes the received data based on the opcode byte and performs the appropriate action (or no action if the opcode has no match, each robot has about 10 actions and there are 256 possible opcodes).

The main advantage of using LeJOS is that the code required to make the robots perform actions is really simple and the code skeleton for a robot is easy to understand (See Appendix B).

In this particular implementation the opcodes for the two robots are designed in such a way that they may be used as a single robot, this was done by constraining the robots respective opcode sets to non-overlapping ranges. The reason for doing this is that I had to control both robots with a single IR transmitter (because there is only one serial transmitter and I didn't have time to integrate USB functionality into the code transmission module) and overlapping codes would have caused problems, telling one robot to turn right, for example, could end up making the other go forwards.

### 4.3 MStorms Control

The high level design of this system is optimised for the situation where robots have an onboard computer with 802.11b support, this way the onboard computer could have a direct IR connection to the RCX which is shielded from the outside world. The robot control program implements the very simple network protocol outlined in section 3.3.2, the implementation is that of a threaded server written in C.

This MStorms Control program structure is shown in Figure 4.3 this is a network-server style of program which contains two constant threads of execution. At program startup a thread is spawned which monitors the state of a command queue after spawning this thread the main thread sits in a socket accept loop waiting for client connections. The command queue is a thread safe FIFO containing IR command structs, if the thread monitoring this queue sees anything waiting it will pop off the first queue item and send the appropriate data to the serial port and thus the IR tower. The socket accept thread spawns a connection handler thread whenever a socket connection is made, this connection handler thread reads data from the remote and attempts to parse this based upon the commands specified in the command file. If there are any problems with the read data an error code is returned, otherwise a command struct will be created and pushed onto the command queue. The

\(^{19}\)http://graphics.stanford.edu/~kokoa/rcx/opcodes.html
network behaviour of this connection handler is outlined in section 3.3.2 and a sample configuration file specifying commands is in Appendix B.2.

In the initial implementation of this system we are controlling our robots from a standard desktop PC this means that we have to use stationary IR transmitters, there are a couple of issues caused by this. The first is that if there are multiple robots then transmissions to one are likely to interfere with transmissions to the other. The solution to that was to merge the two robots into a single robot from the system’s perspective. As we can have 256 possible opcodes and the robots only need about ten each this posed no problem.

The second problem is that the IR communication system really requires line-of-sight between the IR port on the RCX and the IR transmission tower and fluorescent light causing degradation of the signal. The solution to this was to hack the serial version of the Mindstorms IR tower and add 4 more IR LEDs to it which can be placed around the ‘play’ zone to give 360 degree coverage. The modified IR transmitter can be seen in Figure 4.3.
4.4 Voice Client

The speech recognition module of my system is a wrapper to the `sphinx2-continuous` program. This part of the system is written using the Perl language, Perl was chosen because it's advantages in text processing would lead to a reduced development time. The cost of using Perl is that the runtime will be slower, but it turns out that the processing runtime is negligible compared to the time it takes sphinx2 to process audio data. I would recommend however that a system built to also handle gesture recognition data should probably be written in a compile language as the components would probably be easier to fit together that way. Sphinx2 provides a standard library interface that can easily be accessed from a C or C++ program.

This module works by executing the sphinx2-continuous program which prints to standard output its interpretation of any utterances heard. Values for the myriad command-line arguments for the sphinx2-continuous program are specified in the program code and have been tuned to suit the conditions the system typically works under. The Perl program captures all output from sphinx and parses it based on a syntax definition in a file which it takes as a parameter. There are a small number of compulsory parameters to the `voiceclient` program, here's what the program tells you if you give it the argument -h:

```
VoiceClient v0.1
--------------

Usage: voiceclient -c robotconfig [ -l languagemodel | -d ]

-c robotconfig
    Specify the file containing the robot data.

-l languagemodel
    This should be a directory containing a valid Sphinx2 language model. If no
    language model is specified one with a huge vocabulary will be used by default.

-d
    Causes voiceclient to generate a file containing the full range of phrases
    generated from the robotconfig syntax. This file can be given to
    http://www.speech.cs.cmu.edu/tools/lmtool.html to generate a language model that covers
    exactly the vocabulary and other model data for your robot syntax.

-h
    The program will print this help message and then exit.
```

The -c argument is mandatory unless you're just calling up the help screen. 'robotconfig' should point to a valid configuration file of the same form as the one included in Appendix B.1. This
configuration file specifies what robot modes exist and where the controllers for each robot are located (IP and port combination) it then specifies the voice syntax for command translation. The best overview of the format and syntax file would be to have a look at the included sample which covers all the features that are available for specifying commands. This configuration file is read and a series of hashes are built based on the syntax specification, the configuration file will either be successfully parsed or the program will exit informing the user of an error in the configuration file.

Unless the -l argument is given the speech control module will use a language model which covers most of the English language, it is recommended that you don’t use this default if possible. With such a huge vocabulary the continuous speech recognition engine is much more likely to make mistakes, for this reason the -d argument has been provided this argument causes the program to print to standard output a set of strings generated from the command syntax. This output can be sent to a file and used to generate a language model specifically for your syntax, using such a model gives much better recognition results. The path to your custom model should be provided after the -l.

The perl program that is executed to begin processing speech is called voiceclient, as its name suggests it is a network client program. As it parses Sphinx’s interpretation of spoken utterances any proper robot commands it hears are converted to the correct command format and sent to an instance of the robotcontrol server (see previous section). voiceclient has two distinct phases of operation split into “setup” and “processing”. The “setup” phase is entered immediately after starting voiceclient the first thing which occurs in this phase is the parsing of the syntax configuration file (Appendix B.1) and building of the data-structures used in parsing speech. The second phase is the startup of the sphinx2-continuous program, which causes a slight pause as it initialises it’s own internal data structures.

As already mentioned the sphinx2-continuous program takes a huge number of command line parameters (36 usually). In my voiceclient implementation I have left most of these as they were set in a sample Sphinx2 program but have made some small adjustments to the parameters, there is a setting (which is actually a combination of options) called “beamwidth” altering this moves between fast but less accurate recognition and more accurate but slower recognition. I changed the “beamwidth” to give faster recognition (at the cost of some accuracy) in order to improve the responsiveness of the system. The exact specification for the arguments is best covered by examining the voiceclient code and reading the CMU Sphinx2 documentation. The command line argument reference explains the purpose of the myriad options, using these options I provide the paths to my own language model and provide paths for the caching of all heard utterance (in the hope of building up enough material for us to create our own acoustic model). There is a section in the reference that describes the “beamwidth” options, the sample Sphinx2 program sphinx2-demo provides to sets of possible options, “narrow” for speed and “wide” for accuracy. I have configured the options in voiceclient to be between these two settings, though more to the “narrow” side.

Once an utterance is translated to text by sphinx2-continuous the output is processed by a parsing function in the voiceclient program. This parses the text against a grammar given in the robot configuration file, when a line parses successfully the corresponding command is sent via the

\footnote{http://www.speech.cs.cmu.edu/sphinx/doc/sphinx2.html#sec_cmdline}
network to a `mstomscontrol` server. This command is a single alphanumeric string which is given with each grammar line in the configuration file, the command may have a single numeric argument which must also be specified in the file. The specification of the argument is in the form of a numeric range which will be parsed as a spoken numeric sequence. The server will send back a code (detailed in section 3.3.2) indicating that the command has been received and also if the command was correct for the server it was sent to, this simple interaction is shown in Figure 4.4.
5 Concluding Remarks

At the end of this phase of the project a speech recognition based control system has successfully been built. This system can quite effectively demonstrate the problems with purely speech driven control and begs the exploration of methods to refine the recognition process. Working on the based provided by this project a focus can now be made on this sort of refinement.

5.1 Improvements

The voice recognition itself could be hugely improved by building our own acoustic model based on local voices. Further improvement could be gained out of using our own acoustic model by modeling with whole-word phenoms, this is possible because our vocabulary is quite simple.

I feel that having one configuration file at the mstormscontrol end which maps alpha numeric codes to opcodes and another at the voiceclient end which maps speech patterns to alpha numeric codes is not neat. It would probably be preferable to put all configuration into a single file and have both programs use the same configuration. It may even be a good idea to remove the alpha numeric codes all together and have the voiceclient send the raw opcodes to mstormscontrol, this pretty much simplifies mstormscontrol so that it is just a network relay point for opcodes.

At the moment the voiceclient code does not handle multiple IP target modes, all the networking has been simplified to a single remote IP and port for all modes provided on the command line. Fixing this is just a matter of writing the code to process the modes fully.

5.2 Extensions

There are a few options for extending this system in the realm of gesture recognition technologies. The main one, and that which is a long term goal of this work, is to augment the speech recognition with gesture based input. The main goal of a gesture augmented approach would be for the gesture input to be as natural as possible. Forcing a user to learn a defined gesture command set would reduce the effectiveness of the system as a user-interface.

Another possible interesting gesture based extension involved eye-tracking. The system in its current incarnation uses different “modes”, the current mode specifies what command set you are using and thus how the parser treats input. These modes are switched between using vocal commands. A far better way to do this would be to decide which mode the system is in based on the gaze direction if the user. If you have distinct objects to control then the users voice commands could be interpreted appropriately for the object [s]he is looking at.

When this system was designed there was some speculation toward having an on-board handheld computer on each robot, thus support for this sort of functionality was kept in mind. In fact it is this sort of potential application that was the whole reason for implementing a network layer. A huge problem with the current system is the limited range of the IR communications, especially under fluorescent lighting. The solution to this is to have the IR transmitter as close as possible to the RCX at all times and one way to do this is to put a small computer on the robot that is running the mstormscontrol program. This computer would require a network interface and most handhelds
these days can be fitted with standard 802.11b wireless cards. This would allow the robots to wander anywhere they can within an 802.11b network and accessible to the speech control software (or other future software). The bandwidth provided by 802.11b is large enough to transmit low-fidelity motion video, this means that a camera could be attached to mobile robots greatly extending their usefulness.

One other way to solve the IR range problem would be to devise a small radio device for the robots. This device would be tuned to a different frequency for each robot (so the frequency becomes equivalent to a net IP) and would have a simple circuit to convert radio signals into IR pulses. This would be far more cost effective than having an on-board handheld PC, but you lose the potential for local logic (on the handheld) to provide some sort of autonomous functionality, the handheld could also be interfaced with a much wider array of sensors than what is available just on an RCX. I do think that the main design bottleneck is the RCX itself, replacing with completely with a handheld PC, some custom control circuitry and on-board battery power would be the most ideal solution.

An idea that was suggested during the development of the system was a better feedback system, at the moment feedback is sent via text on the machine running voiceclient. This is far from intuitive, some other feedback mechanism must be implemented. The logical one that was suggested is voice feedback, there are several programs available that can convert text to speech. As an added feature this speech could be transmitted to the robot so that the feedback comes from the actual device that you are controlling. This could be achieved by sending the audio data over either simple FM radio or an 802.11b connection. Using FM radio, as for the radio control suggestion above a special circuit could be designed that can pick up FM radio and send the signal to a small speaker. This circuit could be tuned so that each robot has a different pick-up frequency, the voiceclient machine would know the frequencies for each device and transmit the text-to-speech processed feedback audio appropriately using simple FM transmission hardware that can have its transmit frequency changed by the host PC. The 802.11b solution, which really means TCP/IP networking, would simply be to send the audio data across the network link and have the handheld PC on the robot play it as it arrives.
A Code Examples

Sample code as appropriate.

A.1 Robot Skeleton Code

The following segment of code outlines a skeleton that can be used to develop the control code for new robots. The action code would go in the appropriate if/else if blocks for each opcode, to create more actions you just add in another else if that handles the new opcode.

import jsx.platform.rcx.*;
import jsx.robotics.*;

/* This code provides a skeleton for implementing robot actions for PC
 * controlled Lego Mindstorms robots.
 * 
 * Author: Yvan Seth Project: http://www.it.usyd.edu.au/~mstorms/
 * Date: 29th Jan 2003
 */

public class Skeleton
{

    /* This 'main' method for the robot is where all the incoming commands
     * are processed, most of the code is in this one method.
     */
    public static void main (String[] arg)
    {
        LCD.showNumber (0000);
        // buffer space used for incoming data
        byte[] rPacket = new byte[10];
        for (;;)
        {
            if (Serial.isPacketAvailable())
            {
                Serial.readPacket (rPacket);
                rPacket[0] = (byte) ""(rPacket[0]);
                Serial.sendPacket (rPacket, 0, 1);
                rPacket[0] = (byte) ""(rPacket[0]);

                int rOpCode = rPacket[0] & 0xF7;
                if( rOpCode == 0x13 )
                    {
                        if( rPacket[1] == 0x00 )
                        {
                            // what to do for opcode 00
                        }
                        else if( rPacket[1] == 0x01 )
                        {
                            // what to do for opcode 01
                            // if 01 has an integer
                            // argument the call below
                            // could be used to retrieve
                            // it.
                            int dist = combineBytes( rPacket[2], rPacket[3] );
                        }
                    }
                else if( rPacket[1] == 0x05 )
A.2 Sample Robot Actions

Here are a couple of examples of fully implemented robot actions. The first example is the “Open Grabber” command for the Gatherer robot.

The following line goes at the top of the main function to initialise the sensor used to detect when the jaw is fully open.
// activate sensor 2 as a boolean sensor
Sensor.S2.setTypeAndMode( 1, 0x20 );

And next is the actual else if block that processes the opcode for “Open Grabber”.

else if( rPacket[1] == 0x07 )
{
    // "grab" = open jaws, move forward 4 cm, close jaws, move backward 4 cm.
    Motor.B.forward();

    // open
    ready = false;
    while( true )
    {
        if( Sensor.S2.readBooleanValue() == false )
            ready = true;
        if( Sensor.S2.readBooleanValue() == true && ready )
            break;
    }
    Motor.B.flt();
    open = true;

    // forward
    rNav.travel( 4 );

    // close
    Motor.B.backward();
    ready = false;
    while( true )
    {
        if( Sensor.S2.readBooleanValue() == false )
        {
            ready = true;
            if( Sensor.S2.readBooleanValue() == true && ready )
                break;
        }
    }
    Motor.B.flt();
    open = false;
}
B Configuration Files

B.1 Voice Client Configuration

The implementation of the voiceclient is explained in Section 4.4, below is an example of a configuration file specifying a robot syntax. This configuration file is that one which was used for the project demonstration comments within the file explain the format.

# This is a configuration file for the 'sorter/gatherer' robot system this is a
# two robot system which acts (from the software perspective) as a single robot
# with two addressing modes.

# This defines the 'Modes' of operation for this robot
Modes  SORTER,GATHERER,IGNORE

# the word that causes program exit followed by the code to send to the robot
# control on exit (if any) and the speech patter to trigger an exit.
EXIT
  EXIT

# the codes that can be sent to the 'SORTER' robot, the indent is ESSENTIAL for
# defining a block, the block starts with the mode the commands are valid in,
# there is a special block ALL which is for commands that can be sent in any
# mode. Each entry starts with the code to send to the robot-control server,
# the complexities of these codes are explained as we go through them, on the
# next line (one extra indent, essential) is the syntax for the command.
ALL
  DNCE
    DANCE

# mode code  speech pattern
SORTER
  SACK
    ack
      # ack is a 'special', it is sent when we switch to this robot mode
HOME
  HOME | GOTO HOME
    # two very simple entries, for these the code HOME is sent
    # whenever just 'HOME' or 'GOTO HOME' is heard. The pipe (|)
    # character indicates an OR
CUP1
  GOTO CUP ONE
CUP2
  GOTO CUP TWO
PKUP
  PICK UP | GRAB
PTDN
  PUT DOWN
RELS
  DROP
ENDE
  GOTO END | END
POSN 1-7
  GOTO 1-7
    # more interesting, the 1-7 means that the code to send has an
    # argument in the range of 1 to seven, and then in the voice
    # syntax it means the same thing, so this covers the
B.2 Robot Control Configuration

The configuration file for a robot control node simply lists the set of acceptable commands and their robot opcode mappings. The “opcodes” as I call them are the raw data signals (with optional 2 byte argument) that are sent to the robot via the IR communications channel. The configuration file below shows the command to opcode mapping for the Sorter-Gatherer robot combination.

```plaintext
# translation of, say GOTO THREE to POSN 3
CLK 1-10
[GO] RIGHT 1-10
CCLK 1-10
[GO] LEFT 1-10
    # another new feature on these lines, the words in between [ # and ] are optional.
GATHERER
GACK
    ack
FWRD 1-100
[GO] FORWARD 1-100 [CENTIMETRES]
FWRD 1-1000
[GO] FORWARD 1-10x10 METERS
    # the robot doesn’t have a special command for moving by
    # meters, that would be silly, instead the speech processor is
    # capable of doing the conversion... if there is an ’x’ factor
    # in the variable then the values is multiplied by the
    # following value. The code parameter must use the real value
    # range. (this is possibly not the best way to do this, maybe
    # the code arguments should be single numbers that specify
    # which variable in the sentence to replace it with (there
    # could be more than one variable!
BKWD 1-100
[GO] BACKWARD 1-100 [CENTIMETRES]
BKWD 1-1000
[GO] BACKWARD 1-10x10
LEFT 1-360
[TURN] LEFT 1-360 [DEGREES]
RIGHT 1-360
[TURN] RIGHT 1-360 [DEGREES]
OPEN
OPEN
CLOSE
CLOSE
GRAB
GRAB
DROP
DROP

# that’s all folks. Notice that there is no block for the ’IGNORE’ mode, this
# is simply because that mode (as appropriately named) is a mode in which no
# interpretation of the speech is attempted, it’ll switch into other modes when
# it hears their names.
```
B CONFIGURATION FILES

# lines that begin with # are ignored as are lines containing only whitespace

# Gatherer
EXIT 00
FWRD 01 1000
BKWD 02 1000
LEFT 03 360
RIGHT 04 360
OPEN 05
CLOS 06
GRAB 07
DROP 08
DNCE 09
ACKN 0a

# Sorter
HALT 20
HOME 21
CUP1 22
CUP2 23
CLK 24 20
CCLK 25 20
PKUP 26
PTDN 27
RELS 28
ENDE 29
POSN 2a 10
C  mstorms Setup

This appendix will cover the appropriate setup of a Linux system to run the mstorms robot control software, the setup of the mstorms robots demonstration and execution of the software.

C.1 Linux Configuration

Before installing the mstorms software you must create a user account to install and run the system, I suggest using the user name mstorms but any user name will suffice. You should ensure that your target Linux system has the following things before continuing:

- The gnu C compiler (gcc) and associated development packages installed (configure, make, etc.).
- perl 5.6 or later.
- A recent version of the Java Development Kit.
- Correctly configured microphone input to the primary dsp device.
- User for the mstorms package must be able to read from /dev/dsp, have full access to USB devices and serial ports.

You will also require the following:

- A microphone.
- The Lego Mindstorms robots, Sorter mat and a collection of blocks (items to gather and sort).
- A USB Lego tower.
- The adapted serial Lego tower.

The mstorms package has been successfully installed and run on Debian (Release = Woody, Kernel = 2.4.19) and RedHat (Release = 8.0, Kernel = 2.4.18).

C.2 Software Installation

C.2.1 Quick Install

The mstorms CDROM contains all the software you need to run the mstorms system. There is a quick-installation script located in the root directory of the CDROM which you can execute as the mstorms user to perform a software install. To do this log into the target machine as mstorms mount the CDROM and execute the install.sh script located in the root of the CDROM, for example if the CD was mounted at /cdrom you would execute /cdrom/install.sh in your shell.

This will create the following directory structure in the mstorms home directory:

/home/mstorms/
    sphinx2-0.4/
    lejos_2_1_0/
control/
voiceclient/
robotcode/

There is some possibility that this quick installation will not work, so the next section covers full manual installation of the system.

C.2.2 Manual Install

This section covers the full installation process, compiling the Sphinx and LeJOS packaged rather than using pre-compiled binaries. The following software needs to be installed on your system:

- CMU Sphinx 2
- LeJOS
- mstorms Control
- mstorms Voice Client
- mstorms Robot Code

All this software can be obtained from the mstorms CDROM, you should first install the “CMU Sphinx 2” and “LeJOS” software packages, you can copy the installation files to the mstorms home directory from that CDROM, the files are sphinx.tar.gz and lejos.tar.gz. Once copied to your directory these files should be gunzipped and untarred giving you the directories lejos_2_1_0 and sphinx2-0.4, in each of these directories contains the respective packages source trees, follow the instructions in the README and the INSTALL files in these directories to compile the packages. The configure script for CMU Sphinx should be given the argument --prefix=$HOME so that the make install step installs in in your own home directory. Compile LeJOS and let the binaries remain in the lejos_2_1_0 directory, you should also compile the kernel module located in lejos_2_1_0/tools/linux/legousbstower. There is a README in the legousbstower directory, follow the instructions and make sure the kernel module is loaded whenever you are going to use the mstorms system (I added it to the /lib/modules/<kernelversion>/kernel/drivers/usb/ directory on the systems I used and made sure the module was loaded automatically at boot time).

Once you have compiled the above packages you’ll need to add the following lines to your users .bashrc (or add the equivalent lines to the configuration for your shell of choice):

```bash
# set so that lejos uses the usb IR tower
export RCXTTY=usb

# add the sphinx and lejos systems to the users path
export PATH=$HOME/usr/bin:/home/mstorms/lejos_2_1_0/bin/:$PATH
```

Once this is done you should close your shell and open a new one. You should now copy the mstorms software to the user directory, do this with the following command:
cp /cdrom/mstorms/* ~

This will copy three directory trees to the come directory, each of these directories contains a different component of the mstorms system. The control directory contains some C code files, to compile these simply enter the directory and type the make command, this code should compile without problems on any Linux machine. You may now also wish to enter the robotcode directory and execute lejosc *.java to compile the robot code files for Sorter and Gatherer.

Assuming all the above has proceeded correctly you should now have a correctly installed and configured copy of the mstorms system. If are any problems you could try contacting mstorms@it.usyd.edu.au for support.

C.3 Setting Up

Before running a session you should make sure you have all of the following:

- Robots, Sorter and Gatherer (with fresh batteries).
- Computer with software installed.
- Microphone.
- One USB IR tower.
- The modified serial IR tower.
- The mat for Sorter.
- A selection of items to gather.

The first thing to do is to position the three transmitter nodes of the modified serial Lego IR tower. These should be placed around the play area in a rough equilateral triangle, at a height of about 50 centimetres off the play surface. The modified IR tower should be plugged into a serial port on the control computer, a USB IR tower should be plugged into a USB port (used to flash and download software to the robots) and a microphone should be plugged into the appropriate audio input. Figure 5 shows the general arrangement of the hardware components.

Choose a position for the Sorter robot and connect it to the power, turn it on and place a Lego USB IR tower close to the IR port on Sorter’s RCX. On the computer that the USB tower is plugged into enter the lejosfwd1 command, this will download the LeJOS firmware to the RCX when the program exits the transfer is complete. Then enter the $HOME/robotcode/ directory and execute lejos Sorter (you may have to run lejosc Sorter.java to compile the code first. If the transmission connection is established correctly then the lejos program on the computer will display percentage transfer figures, the Sorter code is fully transferred to the RCX when the lejos program exits.

Next the Gatherer robot will probably have to be flashed, carry out exactly the same process for this as you did for the Sorter robot but execute lejos Gatherer instead of lejos Sorter. It is very important that while doing this for Gatherer you have the USB IR Tower facing well away from the
IR Port on Sorter’s RCX, otherwise you may semi-flash Sorter at the same time which would mean repeating the process for Sorter again.

Now that the robots are initialised and the serial IR transmitter nodes are distributed around the play area you can start the control program, to do this you can just type `$HOME/control/mstormsdaemon` this will produce some initialisation output in the terminal. Now in another terminal you can start `$HOME/voicelient/mstormssvc` again you will see some status information as the program initialises, when it is ready you will see *Listening*, printed on the screen.

The system is now ready for use. Simply issue vocal commands that match the grammar in Appendix B.1 into the microphone. Try contacting mstorms@it.usyd.edu.au if you experience any difficulties in setting up and using the *mstorms* software.