Voice-Radio-Controlled Remote Car (VRCRC)

Members:

Don Carlo Dy
Jeffrey Irlanda
Rio Ong
Zharlou Panganiban
Alexis Ryan Razon
Armelle Remedio
Christian Ryan Santos
Hans Sevilla
Ron Earl Tablatin
Jaime Tayamora
Sherdon Niño Uy

Adviser:

Tristan H. Calasanz

Abstract

This project involves two parts. The first is the interfacing of the remote-controlled car to the personal computer. The second one is the analysis and recognition of human voice. To interface the car to the PC, the parallel port is utilized for the said purpose. Switching circuits and opto-isolators were also implemented to make the PC be able to control the car. In implementing the speech analysis, linear predictive coding(LPC) and fast-fourier transform(FFT) were used as a foundation to understand the theory and background study of voices. After this, the idea of threshold triggering was used. This means that there are threshold voltages that must be triggered to differentiate among words. These words are the ones that make up the commands that the car can recognize. All these ideas for voice analysis were implemented through a software solution, thus the use of MATLAB was needed. The MATLAB program is responsible to make the whole system work. The software was able to communicate with the hardware by using C-language and this was compiled in MATLAB as MEX-files(MATLAB Executable) in order for MATLAB to recognize the code.

Introduction

Computer interfacing is a very important and useful knowledge. The reason is due to the reason that the computeris treated as the "wonder machine" at present. Many technological application may be achieved or implemented when using the personal computer. The interfacing also makes it possible to make different popular electronic device be able to communicate to each other or capable of controlling one another.

To be able to learn in utilizing the computer for interfacing external devices, the group decided to tinker with a remote-controlled toy car(RC). The RC will be controlled through the PC. Also, to make matters interesting, the computer will capture voice as its input and that input will decipher what the user wishes the RC to do.

This project may seem to be a useless effort because it simply involves the controlling of a toy. In a way, it is but this is a stepping stone to a much complicated or highly needed application in the future. What this project present is the possibility to make it easy for anyone to control any device by just speaking in human speech and the computer will take all the difficulty of the task.

The Project

In implementing the project, the group was divided into two in order to distribute the work. One group will make the RC be interfaced to the computer's parallel port. The other would deal with speech and voice analysis and manipulate it to produce a desirable output recognizable to the RC so it may respond to the user's command.

For the implementation of the interface, the solution used involved hardware design. To attain the goal, the following were used or built:

- An RC car

- Switching circuit
- Opto-isolators

To be able to do voice analysis and use the data as input, a software solution was chosen. The following discussions detail the implementation and the work that were done to reach the goal of the group.

Hardware Implementation

Zharlou T. Panganiban

The Hardware part of this project consists of the parallel port interfacing and the transmitter interfacing. The first part is concerned with setting up the needed hardware precautionary measures for safe and proper parallel port interfacing. The schematic diagram below shows the circuit used for safeguarding the parallel port from feedback signals coming from the interfaced device.

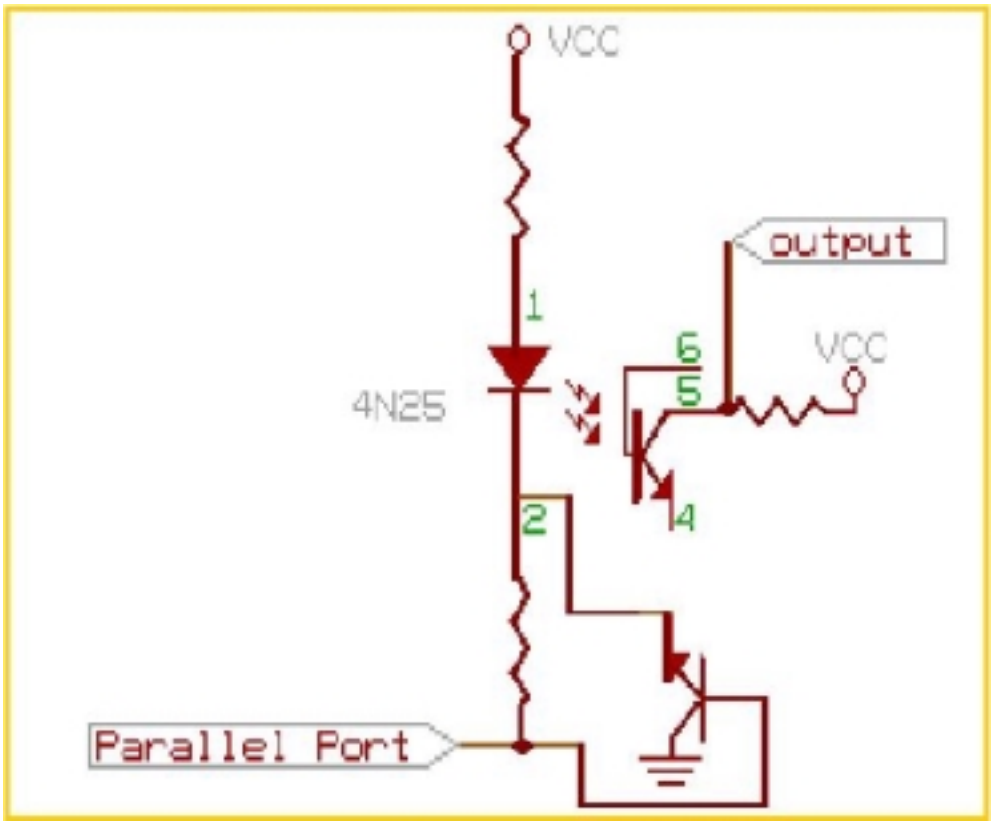


Figure 1. Interface circuit for one pin in the parallel port.

Optoisolators are used to guard the port by isolating it from the external device. In the schematic diagram, it is shown that optoisolators are mainly composed of an LED and a photodiode. If there is an input in the Vcc node and no output in the parallel port, the LED is biased, and it gives off light. This light induces a current in the phototransistor. It will then

create a voltage drop, and the voltage read at the output is at ground level. If there is an output voltage in the parallel port, then the LED is not biased. No current will be induced in the phototransistor and thus, the voltage read at the output is the Vcc node.

This circuit is duplicated four times since there are four pins used for this project. This corresponds to the four basic commands used: GO, BACK, LEFT and RIGHT. No more pin is assigned to the STOP command, since the four pins are used for this specific command; the four pins are reset once this command is executed. The orcad-pcb design for the whole interface is shown below.

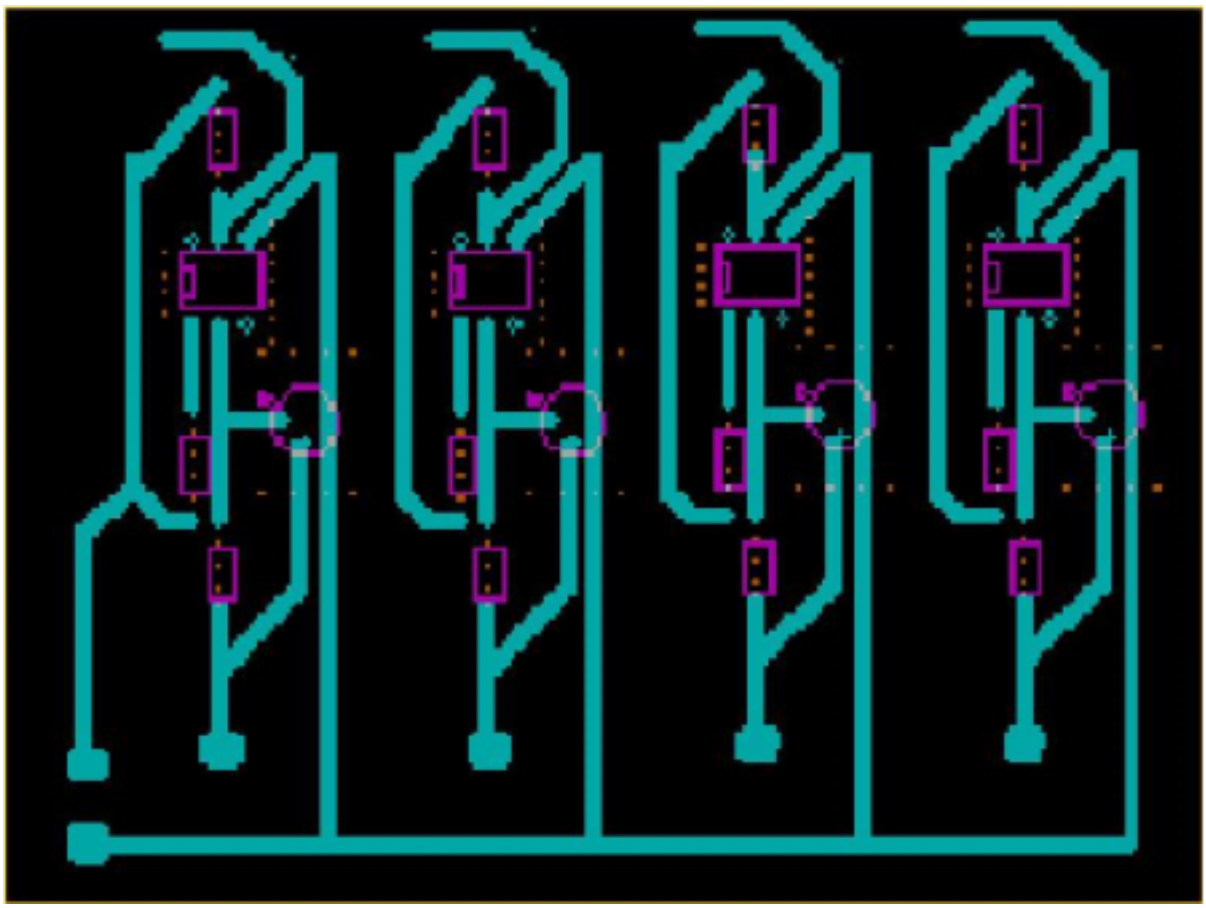


Figure 2. ORCAD-PCB design for the parallel port interface

INTERFACING THE TRANSMITTER TO THE PARALLEL PORT

Ong, Rio
Dy, Don Carlo

Since we have already interfaced the software to the hardware by being able to control the output of the parallel port and isolating the computer from the outside using optoisolators, we are

now in the position to interface the transmitter. This will be the last stage of the project design implementation. As was discussed, the first four data lines of the computer (pins 2 – 5) was used to interface the hardware to the software. Each of the four pins correspond to each of the commands. These commands are: “Go”, “Back”, “Left” and “Right”. Pin 2 corresponds to the “go” command, pin 3 to “back” command, pin 4 to the “left” command and pin 5 to the “right” command.

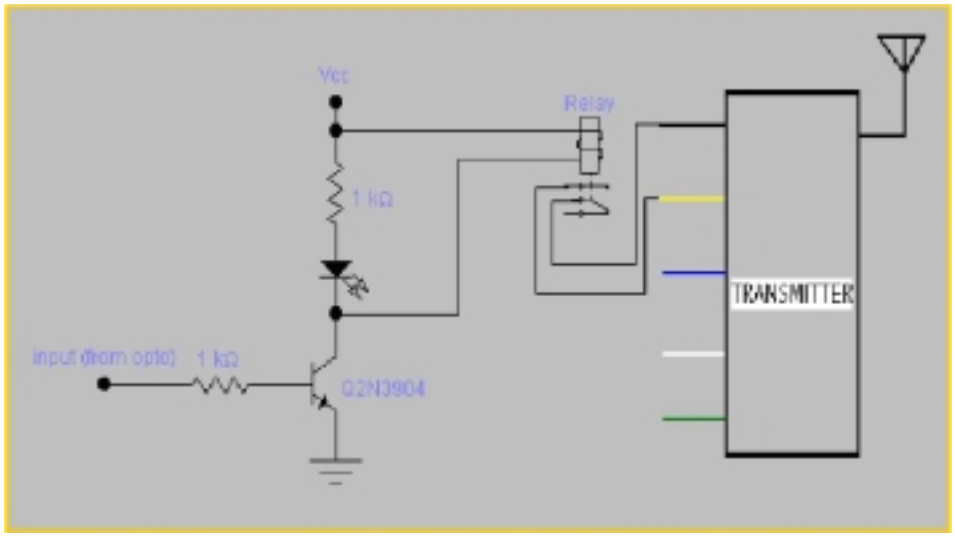
The transmitter and receiver to be used were included upon purchasing the Radio-Controlled Car. What was done, was to remove the transmitter circuit from the remote and re-simulate everything in the hand-held remote. These includes, 9 volts supply which we did by tapping 9 volts from a dry-cell. Another is the connection of the switches and wires. We simplified the transmitter circuit by removing the 2 joy sticks in the hand-held remote and replace it by supplying 9 volts to a specific input. The diagram of the simplified transmitter circuit appears below. Looking on the right side of the picture on the box indicated by “transmitter”, one can notice the 5 wires of the transmitter. The black line is the 9-volt supply. The yellow wire corresponds to the “go” input line, blue to the “back” input line, white to the “left” input line, and green to the “right” input line. By connecting each of these input lines to the 9-volt supply (black wire), the specific command is transmitted, received by the remote car, and the command is executed. For example, if the yellow wire is fed with 9 volts, the car will go forward. In the case of transmitting two commands (e.g., forward and right), 9 volts is fed into the yellow and green wire. So the problem now is how to connect the 9-volt supply wire to each of the input lines by using only TTL output from the optoisolators connected to the parallel port of the PC.

This could easily be implemented by using relay-switches, and transistors. For this discussion, only one of the commands is discussed since the others share the same hardware implementation and discussion as this.

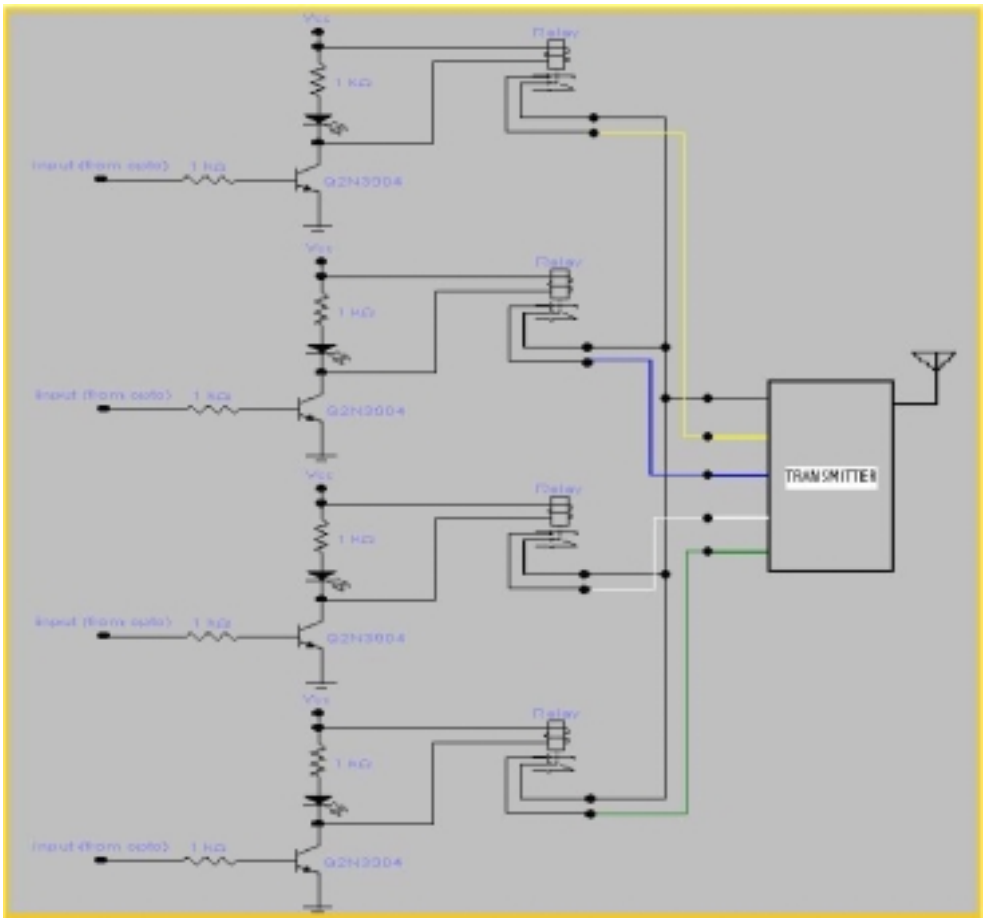
Referring to the diagram below, the input of the circuit is coming from the TTL output (pin 2 of parallel port) of the optoisolators. And the transistor (Q2N3904) serves as a switch. If the transistor is saturated by applying TTL high logic into the its base, the emitter of the transistor is then immediately switched to ground. This results to a complete circuit of the relay switch causing a flow of current across its coil. This then induces a magnetic field to the mechanical switch of the relay and closing the normally open pins (opening the normally closed) of the relay. This then connects the black and yellow wire of the transmitter and thus sending a “go” signal to the car.

If for example, a low logic is outputted by the optoisolator, the transistor is not saturated. The transistor’s emitter retains its no connection to the ground. This implies that the circuit of the relay switch is not complete and current would not flow through the coil. There will be no magnetic induction to trigger the mechanical switch from switching. The normally open pins of the switch is retained and 9-volts is not supplied to the “go” input line (yellow). Hence no signal is transmitted.

The LED and 1K resistor are placed for purposes of displaying the status of the input signal. If a high logic is inputted, this LED will light up. Otherwise, this remains off.



As for the other commands, the same components are needed: relay switch, transistor, resistors and LED. The same connections are made for the other commands. The diagram of the hardware is shown below.



As can be seen, each of the input lines of the transmitter (yellow, blue, white, and green wires) are connected to the black 9-volt supply if the corresponding relay switch and transistor are triggered. In short, as was discussed above, each of the input lines are connected to the black wire when a high logic is inputted to the base of the transistor causing a transmission of a particular command.

Recording voices and finding the exact word to be used as the input signal.

Jaime Tayamora

Recording sample voices that will be used as input for the PC-controlled car seems a very easy task at the start. Equipment that can record voice is easily available, much more, persuading an individual to make his or her voice the sample to be used is not a very difficult task at hand. It was obvious at the moment that almost all of us wanted to say the words 'go,' 'left,' 'right,' 'back' or 'stop' and see the car moved.

The recording process proceeded without much difficulty, except for certain factors, which are quite inevitable especially for a surrounding that we had in the laboratory. Since other groups were also finishing their projects and were all hurrying to see the results of their jobs, technically, the signal-to-noise ratio is very low and this affected the input for quite a few sampling.

The device used is a PC microphone, with a small plastic stand and handle, connected to the soundcard at the back of the CPU. This would later on serve as the sole connection of the controlling person to the RC (remote-controlled car). The words intended to be used to control the car are 'go,' for moving forward; 'left,' for turning left; 'right,' for turning right; 'back,' for moving backward and 'stop,' for making the RC halt. At least ten of each of these words were recorded via Windows Sound Recorder and stored as a *.wav* file. These voice files will then be processed further using MATLAB in order to be analyzed.

Consequently, the recording process would have ended promptly after the sample voices are saved if not for some serious problems encountered after the initial samples were analyzed. Through many various processes done in the MATLAB, it was found that not all of the original words could be used to control the RC. Only the words 'go,' 'left' and 'stop' remained and substitutes for the other command words have to be formulated.

Eventually, instead of using the words right and back, only the letter 'r' is uttered for 'right,' and the letter 'b,' pronounced with a high pitch, is used for 'back.' Another set of the new commands were recorded and then integrated in the already running program in the MATLAB.

Speech Signal Analysis

Hans D. Sevilla

The speech signal has been studied for various reasons and applications for many years. Some studies broke down the speech signal into its smallest portions, called phonemes. But to describe the speech signal in terms of its general characteristics, the traditional vocoders (voice coders) have classified the input speech signal either as voiced or unvoiced. The voiced speech segment is characterized by a high-energy content and periodicity while the unvoiced part is known by its relative appearance which is more of a random noise with no periodicity. Sometimes, there are parts of speech that are a mixture of the two and they are called transition regions. In the Voice recognition part of our project, it is necessary to remove the noise, which appears like near-zero signals. Removing this near-zero segments of the input speech signal requires MatLab functions such as WavChop and the Butterworth filter, which are discussed later in the report.

In speech coding schemes, the frequency domain representation of the speech signal is necessary. For this purpose, the short-time Fourier transform is very useful. This transform is very important especially in examining features of the speech signal not obvious in the time domain representation. The short-time Fourier transform is defined by:

$$S_k(e^{jw}) = \sum_{n=-\infty}^{\infty} w(k-n)s(n)e^{-jwn}$$

where w(k - n) is a real window sequence used to isolate the portion of interest or the portion of the input signal that will be analyzed at a particular index, k. In the analysis of speech signals, the shape and length of the window affects the frequency representation of speech. In this case, several windowing techniques have been formulated.

Windowing determines the portion of the speech signal that is to be analyzed by zeroing out the signal outside the region of interest. Windowing techniques include the Rectangular, Bartlett, Hamming, Hanning, Blackman, and Kaiser. The most commonly used are the Rectangular and the Hamming methods because of their extremely opposite features. Their formulae are shown below:

Rectangular:

$$w(n) = 1; 0 \le n \le N-1 \qquad w(n) = 0; otherwise$$

Hamming:

$$w(n) = 0.54 - 0.46\cos(2\pi \frac{n}{N-1}) \quad ;0 \le n \le N\text{-}1$$

$$w(n) = 0 \quad ;\text{otherwise}$$

Due to these equations, the Rectangular window have greater frequency resolution but has very high undesirable frequency leakage. On the other hand, the Hamming window has lesser frequency resolution but has very low leakage. Thus, rectangular windows are not recommended for speech spectral analysis. So in our project, the windowing technique used is the Hamming.

After a speech spectrum is windowed through Hamming, which is also a function in MatLab, the resulting window can now be passed to the fast Fourier transform. After the FFT has been applied on the speech signal, speech signal analysis can now proceed with Linear Predictive Coding or other methods like the Pitch Prediction and the Threshold Method.

LPC – Linear Predictive Coding

Alexis Ryan Razon

One of the most powerful speech analysis methods is that of Linear Predictive coding or LPC analysis as it is commonly referred to. In LPC analysis the short-term correlations between speech samples (formants) are modelled and removed by a very efficient short order filter. Another equally powerful and related method is pitch prediction. In pitch prediction, the long-term correlations of speech samples are modelled.

Before parameters can be extracted from a speech signal, it is necessary to have a theoretical model for the analysis. In speech processing, the source-filter model of speech production is generally used as a means of analysis. The time varying filter definition is affected by the combined spectral contributions of the glottal flow, the vocal tract and the radiation of the lips.

$$H(z) = \frac{S(z)}{X(z)} = \frac{G}{A(z)} \quad \text{where} \quad A(z) = 1 - \sum_{j=1}^{p} a_j z^{-j}$$

This equation can be transformed into the sampled time domain:

$$s(n) = Gx(n) + \sum_{j=1}^{p} a_j s(n-j)$$

The above equation is well known as the LPC difference equation, which states that the value of the present output s(n), may be determined by summing the weighted present input Gx(n), and a weighted sum of the past output samples. Hence in LPC, the analysis of a speech signal is simplified because the result only gives parameters represented by a_j, which are in short coefficients of the different corresponding segments. Moreover, LPC is already defined in MatLab.

In our project, LPC analysis was first implemented. However, due to the similarity of the coefficients and the number of data elements for the different commands, the group decided to implement another speech analysis method, which is the Threshold Method. Fortunately, the experience with LPC analysis made the Threshold Method a lot easier to use.

Threshold Values

Armelle Reca Remedio

To distinguish the commands, threshold values are used. The threshold is the minimum value that can be reached per command. The process of obtaining the values is by selection. The person with a relatively stable voice is set as the basis for the threshold.

The advantage of the program is that of its simplicity. But since speech recognition is obviously very complex, the approach is limited. A specific speech pattern of a person and a set

of specific spectra are used thus limiting the voice recognition to certain people who can reach the threshold values.
A problem will also occur if the person is not feeling well especially if sick since the equilibrium of the person is altered. Voice change also depending on the time of day thus making the recognition difficult to obtain.

For the Modem Connection

In the hardware part of the project, the commands (go, back, left, right, and stop) are connected to the parallel port of the CPU. By accessing the specific pins in the ports using C programming (outport and inport functions of the language), the status of the voice-controlled-remote-car can be send via a modem.

MEX-file Interfacing

Jeffrey Irlanda
Ron Earl B. Tablatin

The main objective of computer interfacing is to allow the user to receive and/or send data out to the real world using the input/output ports. One of the most popular ways to interface is to use the parallel port. For our project, out goal is to control a remote controlled car using the computer via the parallel port. Voice commands will be issued by the user, interpreted by the computer, and then sent to the remote control of the car via the parallel port, which in turn sends the command to the car itself.

The programming language to be used in the project is crucial, since it must have both the capability for complex mathematics (for analyzing the voice) and the capability of controlling the ports (for data transmission). But the software need not be programmed using one language alone, since compatible programs can be made to communicate to each other, one will do the voice recognition and the other will control the parallel port. The first plan was to use Matlab for the voice recognition and Visual Basic for the control of the parallel port. But it proved to be very difficult for the two programs to communicate pass data to each other, so another approach must be done.

Matlab has a built-in utility called MEX. MEX is used to convert FORTRAN or C-code to MATLAB Executable ("MEX") format. MEX-files are dynamically linked subroutines that can be called from within MATLAB as regular MATLAB functions. With this, a simple C program that sends data through the parallel port was made, and then compiled using the MEX utility. A DLL was created, and this in turn is the one called from within Matlab. The choice of embedding assembly language in the C program instead of using the built-in function outport/outportb was because outport/outportb won't compile to MEX format. The reason is still unknown, but the most likely culprit is the compiler itself.

Listed below are the C++ source codes that were compiled to MEX-file format:

Punta0.cpp:

```
#include <stdlib.h>
#include "mex.h"
```

```
void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x00
        mov dx,0x378
        out dx,al
}
}
```

Punta1.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x01
        mov dx,0x378
        out dx,al
}
}
```

Punta2.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x02
        mov dx,0x378
        out dx,al
}
}
```

Punta4.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x04
        mov dx,0x378
        out dx,al
}
}
```

Punta5.cpp:

```
#include <stdlib.h>
```

```
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x05
        mov dx,0x378
        out dx,al
}
}
```

Punta6.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x06
        mov dx,0x378
        out dx,al
}
}
```

Punta8.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x08
        mov dx,0x378
        out dx,al
}
}
```

Punta9.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x09
        mov dx,0x378
        out dx,al
}
}
```

Punta10.cpp:

```
#include <stdlib.h>
#include "mex.h"

void mexFunction ( int nlhs, mxArray *plhs[ ], int nrhs, const mxArray *prhs[ ] ){
asm { mov al,0x0A
        mov dx,0x378
        out dx,al
}
}
```

Basic MATLAB Functions and Final Program in MATLAB

Christian Ryan Santos
Sherdon Niño Uy

The library of MATLAB offers many useful tools to manipulate audio data. Aside from making matrix treatment easy for the user, audio data would also be a breeze to deal with. The following are commands which may be used in MATLAB readily.

- Daqrecord
- Wavrecord
- Wavread
- Wavwrite
- Butter
- Wavplay
- Find

The Daqrecord records any audio signal picked up from the microphone of the computer. This recording is resident to MATLAB only and cannot be recognized in other application softwares. This is used for data acquisition and returning the audio data in matrix form.

The Wavrecord is similar to Daqrecord except that it uses Windows standard audio recorder to acquire data. This opens the Windows recorder and records the sound from the microphone. The recorded sound is recognized as a data bearing a .wav extension. Wavplay simply opens a .wav file specified by the user and plays it to the PC speakers.

For the function Wavread, MATLAB will open a specified .wav file and read its contents. The user may declare how much bytes in the file will be read. The bytes read will be transformed to a matrix format. The Wavwrite creates a .wav file in the current directory. With this function, the user may write a .wav file to the disk at the specified folder. The filename is to be provided by the user. This is used for saving wave files into the disk.

To create filters, the Butter is used to create filters which follow the Butterworth polynomials. This basically filters out noise in order to obtain the desired audio data. The Find

function is for searching undesired values in the data matrix acquired and discarding them. Essentially, it would search a matrix for all non-zero values and create a new matrix where MATLAB will place the non-zero values.

The basic MATLAB functions were used to do the following programs that made communication with the parallel port possible. The main functions are the following:

- Wavchop
- Jfilt
- PC

Wavchop was used as the function which opens a wavfile, removes the noise of the recorded voice, and saves the filtered file to another. This process was necessary in order to obtain all the necessary data of the voiced without the unwanted ones. This takes out the white noise and silence from the voice sample taken. To place the voice sample to the bandwidth where human voice is located, the Jfilt function was used. This function uses a butterworth bandpass filter to get rid of unvoiced speech and to reject received signals beyond human voice. It is also in this part where fast-fourier transform is implemented to reproduce the signal in a format much easier to analyze as compared to its time domain.

The main program that runs all these functions, including the MEX-files, is the PC.m. This program implements the threshold triggering in order to decipher from “left”, “right”, “go”, “back”, and “stop”. This program records a voice command from the user and saves the file to disk. It then commences to call the Wavchop and Jfilt functions to filter the voice sample. After that, it would proceed to compare which threshold is triggered by the sampled voice. When it has finished with the voice analysis, it would then call on the corresponding MEX-file which contains the command that determines which data port must be turned high. The table below lists the commands that the RC can do and the corresponding commands that the user must say.

Action	Command	Triggering Value
Left	"Left"	20
Right	"R"	145
Forward	"Go"	300
Reverse	"B" (in high pitched voice)	700
Stop	"Stop"	80

The following is the source code for voice analysis.

Jfilt.m:

```
function [r1,r2,r3]=jfilt(wfile)
% jfilt.m
% Speech Recognition of YES / NO
%  Filter for raw .wav files
% Copyright 1997 Jonathan Cline
%
%       This program is free software; you can redistribute it and/or
%       modify it under the terms of the GNU General Public License
```

```
%       as published by the Free Software Foundation; either version 2
%       of the License, or (at your option) any later version.
%
%       This program is distributed in the hope that it will be useful,
%       but WITHOUT ANY WARRANTY; without even the implied warranty of
%       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%       GNU General Public License for more details.
%
%       To obtain a copy of the GNU General Public License
%       write to the Free Software
%       Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%       02111-1307, USA,
%       or download it at http://www.gnu.org/copyleft/gpl.html
%
% version 10:44AM  6/13/1997

% These thresholds were experimentally determined
%  mileage may vary with climate
NO_THRESHOLD=4.5e4;
YES_THRESHOLD=2.5e4;

GO_T = 300; %370
BACK_T = 700; %650
STOP_T = 80;
RIGHT_T = 145; %270
LEFT_T = 20;



[w,Fs]=wavread(wfile);
if max(w) > 0
   w = w./(max(w));
end
N=8192;

%w=w-mean(w);   % don't do this (remove 'DC'); tweaks poles/zeroes all up

W=fft(w,N);
wmag=abs(W);
%wmag=wmag/max(wmag);

[range]=0:2*pi/N:(N-1)/N*2*pi;

% Create filter - butterworth bandpass. get only between 0.3 and 0.8

wn = [0.3 0.8];
[zeros,poles,K] = butter(5,wn, 'bandpass');
[num,den] = zp2tf(zeros, poles, 1);



[H,F] = freqz(num, den, N, Fs);
Hm = abs(H);


% Convolve magnitude of sound with magnitude of filter
```

```
out = wmag .* Hm;
% return values for use when called
% r2 is the filename of the input wav file
% r1 is the magic value which differentiates the sounds
%       ie, the maximum value of the output
r2=wfile;
r1=max(out);

% r3 is the match assumed for the sound file

if (r1 > BACK_T)
   r3 = 'Back';
elseif(r1 > GO_T)
   r3 = 'Go';
elseif (r1 > RIGHT_T)
   r3= 'Right';
elseif (r1 > STOP_T)
   r3 = 'Stop';
elseif (r1 > LEFT_T)
   r3 = 'Left';
else r3 = 'Indistinguishable';
end

% end matlab source
```

Wavchop.m:

```
function wavchop(wname,wnewname)
% Extract meaningful part of wav-file recorded via microphone
%   ie, chops off 'audio space' from beginning & end
% Copyright 1997 Jonathan Cline
%
%       This program is free software; you can redistribute it and/or
%       modify it under the terms of the GNU General Public License
%       as published by the Free Software Foundation; either version 2
%       of the License, or (at your option) any later version.
%
%       This program is distributed in the hope that it will be useful,
%       but WITHOUT ANY WARRANTY; without even the implied warranty of
%       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%       GNU General Public License for more details.
%
%       To obtain a copy of the GNU General Public License
%       write to the Free Software
%       Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
%       02111-1307, USA,
%       or download it at http://www.gnu.org/copyleft/gpl.html.
%
% Example usage:  wavchop('myfile', 'newfile')
% version 2:00PM  6/3/1997

if nargin~=2
        error('WAVCHOP takes two arguments, ');
   error('which are the names of the .WAV files');
end
```

```
if isempty(findstr(wname,'.')) %==[]
        wname=[wname,'.wav'];
end

if isempty(findstr(wnewname,'.')) %==[]
        wnewname=[wnewname,'.wav'];
end

[wsave,Fs,fmt]=wavread(wname);

w=wsave;

% Normalize
wmean=mean(w);
%w=w-wmean; Don't do this
wmax=max(abs(w));
%w=w/wmax;  Or this either

% Extract signal, ie, remove near-0 parts at beginning & end
% actually trims parts @ beg & end less than 15% of absolute max value
wstart=0;
for k=1:size(w)
   if abs(w(k))>wmean+15*wmax/100 & wstart==0
      wstart=k;
   end;
end
wstop=size(w);
for k=size(w):-1:1
        if abs(w(k))>wmean+15*wmax/100 & wstop==size(w)
                wstop=k;
        end
end;

wnew=w(wstart:wstop);

wavwrite(wnew,Fs,wnewname);

% end matlab source
```

PC.m:

```
%PC - Matlab program for controlling a remote controlled car using voice
commands
function pc()

%Threshold values for specific commands
GO_T = 300;       %370
BACK_T = 700;     %850; %650
STOP_T = 80;
RIGHT_T = 145;    %270
LEFT_T = 20;

%some constants
GO = 1;
BACK = -1;
STOP = 0;
```

```
RIGHT = 2;
LEFT = -2;
STRAIGHT = 0;
currentFB = 0;
currentRL = 0;
Fs = 11025;    %sampling rate
r = 100;          %counter
punta0;           %stop as initial condition
for  s = 1:r
   pause;         %wait for user
   stream = wavrecord(1*Fs,Fs,'double');  %get command from user

   wavwrite(stream,Fs,'stream.wav');      %write it to file
   wavchop('stream.wav', 'stream.wav');   %get rid of unwanted data (delete
parts 15% below the average data)
   [a,b,c]=jfilt('stream.wav');           %make sense out of the data
   fprintf('%d\t%s\t%s\n',a,b,c);         %print result

         if (a > BACK_T)                  %control the pc
            if (currentRL == LEFT & currentFB == STOP)
               punta6;
               currentRL = LEFT;
               currentFB = BACK;
            elseif (currentRL == RIGHT & currentFB == STOP)
               punta10;
               currentRL = RIGHT;
               currentFB = BACK;
            else
               punta2;
               currentRL = STRAIGHT;
               currentFB = BACK;
            end
         elseif(a > GO_T)
            if (currentRL == LEFT & currentFB == STOP)
               punta5;
               currentRL = LEFT;
               currentFB = GO;
            elseif (currentRL == RIGHT & currentFB == STOP)
               punta9;
               currentRL = RIGHT;
               currentFB = GO;
            else
               punta1;
               currentRL = STRAIGHT;
               currentFB = GO;
            end
         elseif (a > RIGHT_T)
            if (currentFB == GO)
               punta9;
               currentFB = GO;
               currentRL = RIGHT;
            elseif (currentFB == BACK)
               punta10;
               currentFB = BACK;
               currentRL = RIGHT;
            else
               punta8;
```

```
            currentFB = STOP;
            currentRL = RIGHT;
         end

      elseif (a > STOP_T)
         if (currentFB == GO)
            punta2;
         elseif (currentFB == BACK)
            punta1;
         end
         pause(1);
         punta0;
         currentRL = STRAIGHT;
         currentFB = STOP;
      elseif (a > LEFT_T)
         if (currentFB == GO)
            punta5;
            currentFB = GO;
            currentRL = LEFT;
         elseif (currentFB == BACK)
            punta6;
            currentFB = BACK;
            currentRL = LEFT;
         else
            punta4;
            currentFB = STOP;
            currentRL = LEFT;
         end
      else
         punta0;
         currentFB = STOP;
         currentRL = STRAIGHT;
      end
   s = s + 1;
end
```

Conclusion

The project was able to attain the desired goals that were set at the beginning. This project has its limitations though. Since the voice commands are threshold triggered, any sound that could trigger that threshold could be acknowledged by the computer as a command and send a controlling data to the RC. The execution time is also not very fast but may be tolerated. This is brought about by calling C++ program into the MATLAB software instead of vice-versa. Also, in order to use the software, a MATLAB program must be installed to the user's computer and run it. It may not run as a stand-alone program and requires MATLAB to execute it because MATLAB cannot compile a .m file into an executable one. These aspects needs improvement to make the system more efficient. All in all, the group was still able to deliver the project in a working and complete form.