Voice Recognition in Automobiles

Sarbjeet Singh
M.Tech. CSE(2st Year)
Sri Sai College of Engg. And Technology, Pathankot

Sukhvinder Singh
M.Tech. CSE(2st Year)
Sri Sai College of Engg. And Technology, Pathankot

Mandeep Kour
M.Tech. CSE(2st Year)
Sri Sai College of Engg. And Technology, Pathankot

Sonia Manhas
M.Tech. CSE(2st Year)
Sri Sai College of Engg. And Technology, Pathankot

ABSTRACT

To create a car controlled by voice of humans is a innovative concept. In this paper we use the concept of speech recognition algorithm and algorithms that will worn on for the command of the users. The switching concept is used initially, the remote is provided with the button, when that button is pressed after that the speech recognition process starts. Then after user will command for opening window , the speech recognition system will process accordingly and the respective window will open. Accordingly the other commands will be processed.

Keywords: Speaker Independent Speech Recognition, Dragon Naturally Speaking Software.

1. INTRODUCTION

In this paper we introduced a new concept of voice recognition in car which uses the concept of speech recognition algorithm. The electrical and mechanical domains are used. The digital image processing is also used. Voice recognition is coming to remote control and car navigation system .The user will command through microphone installed in the remote control of car. The signal are commanded in analogue form which needs to be converted into digital form. The car is installed with the large database which consist of vocabulary[1] , that compose of all keywords used for commanding the car. The system is installed with fully computer system, the size of a voice-recognition program's effective vocabulary is directly related to the random access memory capacity of the computer in which it is installed [2]. The car is installed with special hardware that is display, which display the all the available commands and the instructions to the users to make the system user friendly .If users will input the incorrect commands the display will generate error message and provide the most related commands to the user available in the system vocabulary and keywords on display to the users. Automatic Speech Recognition (ASR) is a model of voice recognition designed for dictation .This model is installed in the car for dictation. our concept is totally based on the concept on artificial intelligence and robotics. The paper is organised as- section 2 describes general information about the system, section 3 describes how the system works, section 4 describes how speech recognition works, in section 5 process of transformation of pcm digital audio is presented, section 6 describes spoken phenomenon, in section 7 we describes how to reduce computation and increase accuracy. Section 8 presents context free grammers. Section 9 and 10 describes continuous dictation and adaptation respectively. Atlast in section 11 the conclusion is given.

2. GENERAL INFORMATION

When used in conjunction with the Multi Function Steering Wheel (available on many recent models), you can also operate all principal functions and accessories. You can access phone functions, including recalling stored numbers and dialing, operate Navigation System functions, or take notes through the built-in memo function.

Currently, the size of the non-speaker-dependent vocabulary includes around 30 words, including numbers and commands. Spoken sequences of commands of up to five words and columns of numbers can be recognized with a high degree of accuracy[3].

You can create a telephone book with up to 40 numbers. Dialing is then simply a matter of speaking a name. Other normal telephone functions, such as repeat dialing and call hang-up are also voice activated[4].

3. HOW IT WORKS

Voice recognition uses a neural net to "learn" to recognize your voice. As you speak, the voice recognition software remembers the way you say each word. This customization allows voice recognition, even though everyone speaks with varying accents and inflection.

The voice commands you use in your car are chosen from a fixed vocabulary and are passed on to the car telephone or navigation system via the telephone interface. The system gives acoustic feedback on everything recognized.

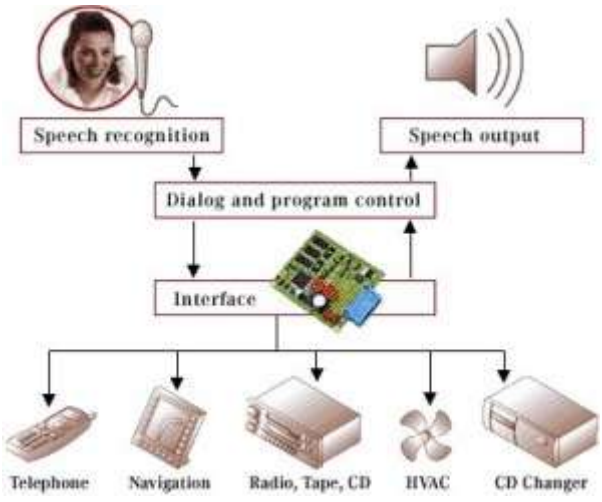

Figure1-How the system works.

The system requires no lengthy voice recognition protocol and responds to a simple series of set voice commands that are not sensitive to the accent or dialect of the speaker. The voice control is a finite speech dialog system, which follows a predefined structure. Faulty operation or error recognition can easily be corrected by simply repeating the desired command. The voice recognizer is resistant to stationary environmental noise.

4. HOW SPEECH RECOGNITION WORKS

You might have already used speech recognition in products, and maybe even incorporated it into your own application, but you still don't know how it works. This document will give you a technical overview of speech recognition so you can understand how it works, and better understand some of the capabilities and limitations of the technology.

Speech recognition fundamentally functions as a pipeline that converts PCM (Pulse Code Modulation) digital audio from a sound card into recognized speech. The elements of the pipeline are:

1. Transform the PCM digital audio into a better acoustic representation
2. Apply a "grammar" so the speech recognizer knows what phonemes to expect. A grammar could be anything from a context-free grammar to full-blown English.
3. Figure out which phonemes are spoken.
4. Convert the phonemes into words.

I'll cover each of these steps individually

5. TRANSFORM THE PCM DIGITAL AUDIO

The first element of the pipeline converts digital audio coming from the sound card into a format that's more representative of what a person hears. The digital audio is a stream of amplitudes, sampled at about 16,000 times per second. If you visualize the incoming data, it looks just like the output of an oscilloscope. It's a wavy line that periodically repeats while the user is speaking.[5] While in this form, the data isn't useful to speech recognition because it's too difficult to identify any patterns that correlate to what was actually said.

To make pattern recognition easier, the PCM digital audio is transformed into the "frequency domain." Transformations are done using a windowed fast-Fourier transform.[6] The output is similar to what a spectrograph produces. In frequency domain, you can identify the frequency components of a sound. From the frequency components, it's possible to approximate how the human ear perceives the sound.

The fast Fourier transform analyzes every 1/100th of a second and converts the audio data into the frequency domain. Each 1/100th of a second results is a graph of the amplitudes of frequency components, describing the sound heard for that 1/100th of a second. The speech recognizer has a database of several thousand such graphs (called a codebook) that identify different types of sounds the human voice can make. The sound is "identified" by matching it to its closest entry in the codebook, producing a number that describes the sound. This number is called the "feature number." (Actually, there are several feature numbers generated for every 1/100 the of a second but the process is easier to explain assuming only one.)

The input to the speech recognizer began as a stream of 16,000 PCM values per second. By using fast Fourier transforms and the codebook, it is boiled down into essential information, producing 100 feature numbers per second.

6. SPOKEN PHENOMENAS

I'm going to jump out of order here. To make the recognition process easier to understand, I'll first explain how the recognizer determines what phonemes were spoken and then explain the grammars.

In an ideal world, you could match each feature number to a phoneme. If a segment of audio resulted in feature #52, it could always mean that the user made an "h" sound. Feature #53 might be an "f" sound, etc. If this were true, it would be easy to figure out what phonemes the user spoke.

Unfortunately, this doesn't work because of a number of reasons:

- Every time a user speaks a word it sounds different. Users do not produce exactly the same sound for the same phoneme.
- The background noise from the microphone and user's office sometimes causes the recognizer to hear a different vector than it would have if the user was in a quiet room with a high quality microphone.
- The sound of a phoneme changes depending on what phonemes surround it. The "t" in "talk" sounds different than the "t" in "attack" and "mist".
- The sound produced by a phoneme changes from the beginning to the end of the phoneme, and is not constant. The beginning of a "t" will produce different feature numbers than the end of a "t".

The background noise and variability problems are solved by allowing a feature number to be used by more than just one phoneme, and using statistical models to figure out which phoneme is spoken. This can be done because a phoneme lasts for a relatively long time, 50 to 100 feature numbers, and it's likely that one or more sounds are predominant during that time. Hence, it's possible to predict what phoneme was spoken.

Actually, the approximation is a bit more complex than this. I'll explain by starting at the origin of the process.[7] For the speech recognition to learn how a phoneme sounds, a training tool is passed hundreds of recordings of the phoneme. It analyzes each 1/100 th of a second of these hundreds of recordings and produces a feature number. From these it learns statistics about how likely it is for a particular feature number to appear in a specific phoneme. Hence, for the phoneme "h", there might be a 55% chance of feature #52 appearing in any

1/100 th of a second, 30% chance of feature #189 showing up, and 15% chance of feature #53. Every 1/100 th of a second of an "f" sound might have a 10% chance of feature #52, 10% chance of feature #189, and 80% chance of feature #53[8].

The probability analysis done during training is used during recognition. The 6 feature numbers that are heard during recognition might be:

52, 52, 189, 53, 52, 52

The recognizer computes the probability of the sound being an "h" and the probability of it being any other phoneme, such as "f". The probability of "h" is:

80% * 80% * 30% * 15% * 80% * 80% = 1.84%

The probability of the sound being an "f" is:

10% * 10% * 10% * 80% * 10% * 10 % = 0.0008%

You can see that given the current data, "h" is a more likely candidate. (For those of you that are mathematical sticklers, you'll notice that the "probabilities" are no longer probabilities because they don't sum to one. From now on I'll call them "scores" since they're un-normalized probabilities.)

The speech recognizer needs to know when one phoneme ends and the next begins. Speech recognition engines use a mathematical technique called "Hidden Markov Models" (HMMs) that figure this out. This article won't get into the mathematics of how HMM's work, but here's an intuitive feel. Lets assume that the recognizer heard a word with an "h" phoneme followed by an "ee" phoneme. The "ee" phoneme has a 75% chance of producing feature #82 every 1/100 th of a second, 15% of chance feature #98, and a 10% chance of feature #52. Notice that feature #52 also appears in "h". If you saw a lineup of the data, it might look like this:

52, 52, 189, 53, 52, 52, 82, 52, 82, etc.

So where does the "h" end and the "ee" begin? From looking at the features you can see that the 52's are grouped at the beginning, and the 82's grouped at the end. The split occurs someplace in-between. Humans can eye-ball this. Computers use Hidden Markov Models.

By the way, the speech recognizer figures out when speech starts and stops because it has a "silence" phoneme, and each feature number has a probability of appearing in silence, just like any other phoneme.

Now our recognizer can recognize what phoneme was spoken if there's background noise or the user's voice had some variation. However, there's another problem. The sound of phonemes changes depending upon what phoneme came before and after. You can hear this with words such as "he" and "how". You don't speak a "h" followed by an "ee" or "ow", but the vowels intrude into the "h", so the "h" in "he" has a bit of "ee" in it, and the "h" in "how" as a bit of "ow" in it.

Speech recognition engines solve the problem by creating "tri-phones", which are phonemes in the context of surrounding phonemes. Thus, there's a tri-phone for "silence-h-ee" and one for "silence-h-ow". Since there are roughly 50 phonemes in English, you can calculate that there are 50*50*50 = 125,000 tri-phones. That's just too many for current PCs to deal with so similar sounding tri-phones are grouped together.

And now for our last issue. The sound of a phoneme is not constant. A "t" sound is silent at first, then produces a sudden burst high frequency of noise, which then fades to silence. Speech recognizers solve this by splitting each phoneme into several segments and generating a different senone for each segment. The recognizer figures out where each segment begins and ends in the same way it figures out where a phoneme begins and ends.

After all this work, the speech recognizer has all the mechanics necessary to recognize if a particular phoneme was spoken. An important question still needs answering: How does it determine which phoneme to look for?

A speech recognizer works by hypothesizing a number of different "states" at once. Each state contains a phoneme with a history of previous phonemes. The hypothesized state with the highest score is used as the final recognition result.

When the speech recognizer starts listening it has one hypothesized state. It assumes the user isn't speaking and that the recognizer is hearing the "silence" phoneme. Every 1/100 th of a second it hypothesizes that the user has started speaking, and adds a new state per phoneme, creating 50 new states, each with a score associated with it. After the first 1/100 th of a second the recognizer has 51 hypothesized states.[9]

In 1/100 th of a second, another feature number comes in. The scores of the existing states are recalculated with the new feature. Then, each phoneme has a chance of transitioning to yet another phoneme, so 51 * 50 = 2550 new states are created. The score of each state is the score of the first 1/100 th of a second times the score if the 2 nd 1/100 th of a second. After 2/100 ths of a second the recognizer has 2601 hypothesized states.

This same process is repeated every 1/100 th of a second. The score of each new hypothesis is the score of it's parent hypothesis times the score derived from the new 1/100 th of a second. In the end, the hypothesis with the best score is what's used as the recognition result[10].

Of course, a few optimizations are introduced.

If the score of a hypothesis is much lower than the highest score then the hypothesis is dropped. This is called pruning. The optimization is intuitively obvious. If the recognizer is millions of times more confident that it heard "h eh l oe" than "z z z z," then there's not much point in continuing the hypothesis that the recognizer heard, "z z z z". However, if too much is pruned then errors can be introduced since the recognizer might make a mistake about which phoneme was spoken.

Recognizers also optimize by not hypothesizing a transition to a new phoneme ever 1/100 th of a second. To do this though, the recognizer must limit what phonemes can follow other phonemes.

7. REDUCING COMPUTATION AND INCREASING ACCURACY

The speech recognizer can now identify what phonemes were spoken. Figuring out what words were spoken should be an easy task. If the user spoke the phonemes, "h eh l oe", then you know they spoke "hello". The recognizer should only have to do a comparison of all the phonemes against a lexicon of pronunciations.

It's not that simple.

1. The user might have pronounced "hello" as "h uh l oe", which might not be in the lexicon.
2. The recognizer may have made a mistake and recognized "hello" as "h uh l oe".

3. Where does one word end and another begin?
4. Even with all these optimizations, the speech recognition still requires too much CPU.

To reduce computation and increase accuracy, the recognizer restricts acceptable inputs from the user. On the whole, this isn't a bad assumption because:

- It's unlikely that the user will speak, "zwickangagang" since it's not a valid word.
- The user may limit him/her-self to a relatively small grammar. There are millions of words, but most people only use a few thousand of them a day, and they may need even fewer words to communicate to a computer.
- When people speak they have a specific grammar that they use. After all, users say, "Open the window," not "Window the open."
- Certain word sequences are more common than others. "New York City" is more common than "New York Aardvark."

8. CONTEXT FREE GRAMMERS

One of the techniques to reduce the computation and increase accuracy is called a "Context Free Grammar" (CFG). CFG's work by limiting the vocabulary and syntax structure of speech recognition to only those words and sentences that are applicable to the application's current state.

The speech recognition gets the phonemes for each word by looking the word up in a lexicon. If the word isn't in the lexicon then it predicts the pronunciation; See the "How Text-to-Speech Works" document for an explanation of pronunciation prediction. Some words have more than one pronunciation, such as "read" which can be pronounced like "reed" or "red". The recognizer basically treats one word with multiple pronunciations the same as two "words".

CFG's slightly change the hypothesis portion of speech recognition. Rather than hypothesizing the transition to all phonemes, the recognizer merely hypothesizes the transition to the next acceptable phonemes. From the initial "silence" phoneme the recognizer hypothesizes the "s" in send, "k" in "call", and "eh" in exit. If the recognizer hypothesizes phoneme transitions from the "s" phoneme, it will only hypothesis "eh", followed by "n", "d", "m", "ae", "l", etc.

You can see how this significantly reduces the computation. Instead of increasing the number of hypotheses by a factor of 50 each time, the number of hypotheses stay constant within a word, and only increase a little bit on word transitions. Given a normal amount of pruning, there are no more than about 10 hypotheses around at a time.[11]

When the user has finished speaking, the recognizer returns the hypothesis with the highest score, and the words that the user spoke are returned to the application.

9. CONTINUOUS DICTATION

Continuous dictation allows the user to speak anything he/she wants out of a large vocabulary[14]. This is more difficult than discrete dictation because the speech recognition engine doesn't easily know where one word ends and the next begins. For example: Speak out loud "recognize speech" and "wreck a nice beach" quickly; They both sound similar.[15]

Continuous dictation works similar to discrete dictation except the end of a word is not detected by silence. Rather, when a hypothesis reaches the end of a word in continuous dictation, it then produces thousands of new hypotheses and prunes those out. The language model probability helps to prune the hypothesis down a lot more in continuous dictation.[12]

Recognizers use a lot more optimizations to optimize processing and memory in continuous dictation systems. The article won't cover those here because their description doesn't help explain the underlying technology[16].

10. ADAPTATION

Speech recognition system "adapt" to the user's voice, vocabulary, and speaking style to improve accuracy. A system that has had time enough to adapt to an individual can have one fourth the error rate of a speaker independent system. Adaptation works because the speech recognition is often informed (directly or indirectly) by the user if it's recognition was correct, and if not, what the correct recognition is.

The recognizer can adapt to the speaker's voice and variations of phoneme pronunciations in a number of ways. First, it can gradually adapt the codebook vectors used to calculate the acoustic feature number. Second, it can adapt the probability that a feature number will appear in a phoneme. Both of these are done by weighted averaging[13].

The language model can also be adapted in a number of ways. The recognizer can learn new words, and slowly increase probabilities of word sequences so that commonly used word sequences are expected. Both these techniques are useful for learning names.

11. CONCLUSION

This was a high level overview of how speech recognition working in the cars. To use the voice concept is very complex process in automobiles because some applications are more complex to install and use .one can easily open the windows by using the concept of voice recognition and close as well. The other applications possible are , controlling the music system, commanding over the power windows ,steering locking .The voice recognition concept is very much innovative and sensitive concept in the field of automobiles and iti can be made more secure using the concept of finger print analysis process..

12. REFERENCES

[1] W. Stokoe, D. Casterline, and C. Croneberg, A Dictionary of American Sign Language on Linguistic Principles, Gallaudet College Press, Washington D.C., USA, 1965.

[2] S. Ong and S. Ranganath, "Automatic sign language analysis: A survey and the future beyond lexical

meaning," IEEE Trans. PAMI, vol. 27, no. 6, pp. 873–891, June 2005.

[3] T.S. Huang Y. Wu, "Vision-based gesture recognition: a review," in Gesture Workshop, Gif-sur-Yvette, France, Mar. 1999, vol. 1739 of LNCS, pp. 103–115.

[4] G. Yao, H. Yao, X. Liu, and F. Jiang, "Real time large vocabulary continuous sign language recognition based on op/viterbi algorithm," in Intl. Conf. Pattern Recognition, Hong Kong, Aug. 2006, vol. 3, pp. 312–315.

[5] C. Vogler and D. Metaxas, "A framework for recognizing the simultaneous aspects of american sign language," Computer Vision & Image Understanding, vol. 81, no. 3, pp. 358–384, Mar. 2001.

[6] R. Bowden, D. Windridge, T. Kadir, A. Zisserman, and M. Brady, "A linguistic feature vector for the visual interpretation of sign language," in European Conf. Computer Vision, 2004, vol. 1, pp. 390–401.

[7] S. B. Wang, A. Quattoni, Louis-Philippe Morency, David Demirdjian, and Trevor Darrell, "Hidden conditional random fields for gesture recognition," in Computer Vision & Pattern Recognition, New York, USA, June 2006, vol. 2, pp. 1521–1527.

[8] J. L¨o¨of, M. Bisani, C. Gollan, G. Heigold, B. Hoffmeister, C. Plahl, R. Schluter, and H. Ney, "The 2006 RWTH parliamentary speeches transcription system," in ICSLP, Pittsburgh, PA, USA, Sept. 2006.

[9] D. Keysers, T. Deselaers, C. Gollan, and H. Ney, "Deformation models for image recognition," IEEE Trans. PAMI, p. to appear, 2007.

[10] P. Dreuw, T. Deselaers, D. Rybach, D. Keysers, and H. Ney, "Tracking using dynamic programming for appearance-based sign language recognition," in IEEE Intl. Conf. on Automatic Face and Gesture Recognition, Southampton, Apr. 2006, pp. 293–298.

[11] C. Neidle, J. Kegl, D. MacLaughlin, B. Bahan, and R.G. Lee, The Syntax of American Sign Language, MIT Press, 1999.

[12] T. K¨olsch, D. Keysers, H. Ney, and R. Paredes, "Enhancements for local feature based image classification," in Intl. Conf. Pattern Recognition, Cambridge, UK, Aug. 2004, vol. 1, pp. 248–251.

[13] A. Zolnay, R. Schl¨uter, and H. Ney, "Acoustic feature combination for robust speech recognition," in ICASSP, Philadelphia, PA, Mar. 2005, vol. 1, pp. 457–460.

[14] D. Klakow and J. Peters, "Testing the correlation of word error rate and perplexity," Speech Communication, vol. 38, pp. 19–28, 2002.

[15] A. Agarwal and B. Triggs, "Recovering 3d human pose from monocular images," IEEE Trans. PAMI, vol. 28, no. 1, pp. 44–58, Jan. 2006.

[16] P. Dreuw, D. Stein, and H. Ney, "Enhancing a sign language translation system with vision-based features," in Intl. Workshop on Gesture in HCI and Simulation 2007, Lisbon, Portugal, May 2007, p. to appear.

[17] Dillon, T.W., & Norcio, A. F. (1997, October). User performance and acceptance of a speech-input interface in a health assessment task. International Journal of Human-Computer Studies, 47(4), 591-602.