Digital speech within 100 Hz bandwidth

Mike Lebo, N6IEF
3172 Pasternack Place
San Diego, CA 92123-3058
mike-lebo@ieee.org
858-278-5851

Objective

To modify and write code needed to convert analog voice into narrow band digital modulation.

Why do this?

The bandwidth of voice is about 2400 Hz. When speech could be reduced to 100 Hz, the gain would be 13.8 dB (24X). Processing gain by a computer is cost free. This project receives weak signals 10 dB (10X) below SSB (Single Side Band) noise floor of the radio.

Generating of the transmit phonemes

Since each person sounds different from another, it is clear that the computer must recognize the unique phonemes used by only that person while operating this software. The software must be able to teach itself the phonemes so that it can recognize that person's voice, which is done by reading words shown on the monitor into the microphone while holding down the space bar of the keyboard. A phoneme is to speech as the alphabet is to reading or writing.

The code used

The 45 phonemes are represented by a code made up of 1's and 0's. The code is similar to a court recorder typing out steno, which can be read back. All code groups start with 1 and end with two or more 0's. Since phonemes are grouped by the shape of the mouth, tongue and lips, the codes used in one group of phonemes should be as different as possible from other groups. Some phonemes are longer than others and they should have a longer code. Of the 53 codes, only 45 are used with eight as spares. This code is exactly the same Varicode used in PSK-31, (Phase Shift Keying with 31 Hz bandwidth).

100, 1100, 10100, 11100, 101100, 111100, 1010100, 1011100, 1101100, 1110100, 1111100, 10101100, 10110100, 10111100, 11010100, 11011100, 11101100, 11110100, 11111100, 101010100, 101011100, 101101100, 101110100, 101111100, 110101100, 110110100, 110111100, 111010100, 111011100, 111101100, 111110100, 111111100, 1010101100, 1010110100, 1010111100, 1011010100, 1011011100, 1011101100, 1011110100, 1011111100, 1101010100, 1101011100, 1101101100, 1101110100, 1101111100, 1110101100, 1110110100, 1110111100, 1111010100, 1111011100, 1111101100, 1111110100, 1111111100

As shown, the code is the fastest speed for each phoneme. By adding one or more extra 0's to any code, the length of that phoneme is stretched by increments of 1/100 of a second. This is very important because voice speed is constantly changing. The original 45 phonemes are expanded to many new phonemes.

The software summary

Voice received through the computer's microphone is converted into numbers, amplified to a constant level, converted into 16 bands of frequency, cut into three parallel 30 mS sections of time, compared in a two-stage process to a library of 45 phonemes that have been made by the operator of the radio, converted to a digital code, stretched to fit the operator's real speech, and sent to the radio in a way similar to QPSK-63 (Quadrature Phase Shift Keying with 63 Hz bandwidth) to be transmitted.

The modification of the WinPSK program

This software is modified from the QPSK-63 software. Moe Wheatley, ae4jy, has done an outstanding job on his open source WinPSK program and his documentation of the software. Please read the PSKCore.DLL (Dynamic-Link Library) Software Specification and Technical Guide at http://www.moetronix.com/ae4jy/winpsk.htm. The new QPSK-100 (Quadrature Phase Shift Keying with 100 Hz bandwidth) is a modification of the QPSK-63 software that is now being used over-the-air. It has a built-in error correcting code that corrects for one out of five digits being wrong. Before installing this QPSK-100 software, make sure your radio, interface and computer are working by testing the WinPSK program with PSK-31 over-the-air.

The transmit sequence

The transmit sequence starts with the pressing of the space bar on the computer keyboard and continues until the space bar is released. The computer speakers' D/A (Digital to Analog) converter is forced to zero. The AGC (Automatic Gain Control) is un-frozen.

The 400 mS synchronizing alternating series of ones and zeros is sent to the transmit section of the WinPSK program. This 100 Hz BPSK code is used by the other computers' receiver section of the WinPSK program to re-synchronize the 100 Hz clock. This insures that the receiver section of the WinPSK program is sampled in the middle of each code digit and is not sampled during the transitions.

The sampling 66,000 Hz clock starts the A/D (Analog to Digital) converter from the microphone input of the computer. Each clock cycle makes the A/D output a 16-digit signed number. Each number goes to the AGC (Automatic Gain Control) array and the AGC level **adjustor**.

The AGC is used to amplify the weak signal from the microphone to about 90% of the maximum value for the 16-digit signed number. This is done by TBD (To Be Determined) method. It will use the normal fast attack and slow decay, but it will be frozen when the space bar is not pressed.

Some of the numbers from the AGC level adjustor go to 32 FIR (Finite Impulse Response) low-pass filters. A FIR low-pass filter has a frequency F and a number of taps N and a sampling rate. The problem with filters is the time difference, DPD (Differential Propagation Delay), between the outputs of high frequency filters and the outputs of low frequency filters with the same input to both. The 17 F frequencies for the FIR filters are 8000 Hz, 6083 Hz, 4625 Hz, 3517 Hz, 2674 Hz, 2033 Hz, 1546 Hz, 1176 Hz, 894 Hz, 680 Hz, 517 Hz, 393 Hz, 299 Hz, 227 Hz, 173 Hz, 131 Hz, and 100 Hz.

A first order attempt to solve the DPD problem is to use different sampling frequencies for each group of two FIR filters. The numbers from the A/D are at a 66,000 Hz rate. When every fourth number is used, the new sampling rate is 16,500 Hz, or 66,000 Hz divided by 4 is 16,500 Hz. The 16 divide-by numbers are 4, 5, 7, 9, 12, 16, 21, 28, 36, 48, 63, 82, 110, 145, 190, and 251.

For example, the divided-by-4 sampling rate is used by the two highest frequency FIR low-pass filters, 8000 Hz and 6083 Hz. Both FIR low-pass filters need to have the same number of taps N to insure that their output numbers are available at the same time, or zero DPD. By subtracting the output numbers from these two FIR low-pass filters, new numbers are created at the same sampling rate. These numbers are approximately the instantaneous amplitude of the sound between the two frequencies. In the same way the other numbers are made by two FIR low-pass filters for each of the other 15 frequency bands, with each associated sampling rate. NOTE: Each set of two FIR low-pass filters has the same sampling rate, and taps N, and their DPD is zero, so their output numbers can be subtracted.

The DPD between frequency bands is not zero, but this doesn't matter because the numbers between frequency bands are never used together.

This complicated process is being done to change the time-amplitude energy of voice into the time-frequency energy of speech.

Some people say that there are 44 phonemes and one extra phoneme for no sound. Dividing the A/D sample clock rate of 66,000 Hz by 1980 makes the **phoneme sample interval**. This interval is 30 mS. After the start of the phoneme sample interval, the absolute values of the next 14 numbers from each of the 16 frequency bands are examined for the largest value. This is called the **peak search process**. Just before the end of the interval, say at count 1979 of 1980, the 16 peak numbers are put into the **phoneme sample array**. The phoneme sample array can be visualized as a blue transparency bar-graph with 16 vertical columns, but it actually is a 16 by 1 array of numbers. This process re-synchronizes the DPD problem to the original 66,000 Hz sample clock of the microphone input D/A.

In order not to miss a phoneme, the above procedure is **repeated in parallel**, two other times by starting at counts 660 and 1320 from the original 1 to 1980. This insures a new phoneme sample array every 10 mS. The 30 mS time interval is used to detect each of the 45 phonemes, even when the phoneme lasts longer. To reduce the chances of receiving part of one phoneme and part of another phoneme, a new set of 16 peak numbers is started every 660 numbers or 10 mS. Overlapping numbers insure that a phoneme is not missed.

One of three parallel **phoneme comparators** takes its phoneme sample array and compares it to one of 45 arrays of 16 numbers from the **phoneme library**, visualized as a yellow transparency bar-graph. By subtracting one array from the other array, visualized as overlapping the yellow and the blue transparencies, the differences are visualized as blue and yellow and the common part of the bar-graph is visualized as green. To amplify these 16 differences, they are multiplied by themselves to make them all positive numbers and these 16 positive numbers are added together to make the single **error number** for that comparison. In the same way, the next array of 16 numbers from the phoneme library is subtracted from the original phoneme sample array until all 45 arrays from the phoneme library are used. The phoneme code for the three smallest error numbers of the 45 possible error numbers is sent to the guesser along with their error numbers and code sizes from the phoneme library. Although this process takes some time, the output rate should be the same as the input rate of 30 mS. Since there are three peak detectors with three comparators staggered 10 mS apart, a phoneme code with its error number and code size is sent into the guesser every 10 mS. The **code size** is a number from three to ten, which is the number of ones and zeros in that phoneme code.

The **guesser** is used to determine what code should be sent to the output Q. The guesser is like a Q with three levels. Three phoneme codes and their error numbers enter the back of the guesser and work their way down to the front of the guesser. So there are always nine phoneme codes in the guesser. Whenever

three codes are entered, three other codes are removed. When there are three of the same phoneme codes in the guesser, the error number of that phoneme code in the front of the guesser is divided by three. When there are two of the same phoneme codes in the guesser, the error number of that phoneme code in the front of the guesser is divided by two. After the divides, the phoneme code and the code size of the smallest error number of the three in the front of the guesser is sent to the output Q. This happens every 10 mS.

The **output Q** is a buffer that is used to fix problems that happen when one phoneme transitions to another phoneme in our speech. The output Q is used to sort the phoneme codes into groups, like sorting cards into suits. When the phoneme code sent to the back of the output Q is the same as any of the two previous phoneme codes in the output Q, the new phoneme code is moved forward to that same phoneme code group.

One phoneme code is removed from the front of the output Q as each digit of the phoneme code is sent to the transmit part of the WinPSK program. But before a new phoneme code group is sent to the transmit part of the WinPSK program, the number of phoneme codes in that group is checked to see that they are more than the **minimum number** for that code size. When they are less than the minimum number, the group is removed from the output Q.

An **extra zero** is sent to the transmit part of the WinPSK program as each extra phoneme code beyond the phoneme code size is removed from the output Q. An example would be the phoneme code of 10100, which is different from 10100000 because the sound of the second code last 3/100 of a second longer. Although there only 45 fundamental phoneme codes, there are hundreds of extensions. No extra zeros are sent to the special phoneme code of 100, but the code could repeat when needed.

When the output Q does not contain enough of the phoneme codes, each digit of the code is still sent to the transmit part of the WinPSK program, but the output Q does not move to the next phoneme code until all the digits of that code are sent.

Code sizes (Minimum number) are 3 (2), 4 (2), 5 (3), 6 (4), 7 (4), 8 (5), 9 (6) and 10 (7).

At the start of each transmission sequence, when the space bar on the computer keyboard is pressed, the guesser and output Q are filled with a quantity of the code 100, the **special code** for no-sound, because the computer takes some time for the numbers from the microphone A/D to be processed. At the start of a transmission, these leading 100 special codes are removed from the output Q and the ones and zeros of the rest of the real phoneme codes are sent to the transmit part of the WinPSK program.

Each digit of the phoneme code is sent serially at a 10 mS rate. This is the same rate at which the error numbers enter the guesser and the same rate at which the audio code modulates the radio transmitter.

At the end of each transmission, the space bar on the computer keyboard is released, all 100 special codes on the back of the output Q are removed and the special **end code** of 1111111111 is sent to the output Q and then to the transmit part of the WinPSK program. This sets the squelch of the other computers' receiver section of the WinPSK program.

With today’s computers having 3 GHz clocks and quad processors, twelve billion operations can be done every second. Speech recognition software in 2004 did not have this computer power and did not work very well. In the event the guesser makes a mistake, our brains deal with the occasional anomalous sound from the computer's speaker. Words may sound mispronounced, but we should know what they mean.

This transmit sequence may look like speech recognition software, but it has two differences. First, speech-to-text software requires the ability to handle spelling and meaning. An example would be the homonyms “to,” “two,” and “too.” Most of the code for speech recognition software would not be used. Second, speech recognition software has no time limit from sound to text. The transmit sequence of this software requires a minimum fixed time delay.

The receiver sequence

The receiver sequence starts with the release of the space bar on the computer keyboard and continues until the space bar is pressed. The microphone A/D is forced to zero. The guesser is not allowed to send more codes to the output Q.

After the 400 mS BPSK signal re-synchronizes the 100 Hz clock and releases the squelch, the ones and zeros coming from the receive part of the WinPSK program are sent to the phoneme comparator. The first one after two consecutive zeros starts a new phoneme code. The first code of ones and zeros assumes a 100 special code for no-sound has been detected. Since the phoneme code is sent serially, each digit goes to the phoneme code library one at a time where half of the library is eliminated with each digit after the first one. When the next digit is received, half of the half of the library is eliminated and so on until two consecutive zeros are detected. That is when the phoneme code is found. Then four phoneme arrays (audio clips) are found from the phoneme library. The first phoneme array is called the **main array**. It is ((the code size – 2) X 10 mS) long and has ((the code size – 2) X 660) numbers. The next phoneme array is called the **zero array**. It is 10 mS long and has 660 numbers. The next phoneme array is called the **third array**. It is the same as the zero array, but each of the numbers is divided by three. The last phoneme array is called the **two-thirds array**. It is the same as the third array, but each of the numbers is multiplied by two.

Normally a .wav file would be used for an audio clip, but that won’t work for 10 mS to 80 mS sound clips with 660 to 5280 numbers in each array. A new way to send the numbers to the speaker D/A will be made by a TBD method.

When the first two consecutive zeros of the present phoneme code are detected, each of the numbers in the present third array and each of the numbers in the previous two-thirds array are added in the **first blender array**. Then each of the numbers in the present two-thirds array and each of the numbers in the previous third array are added in the **second blender array**. Then the first blender array is sent to the sound card D/A buffer of the computer, followed by second blender array, followed by the main array of the present phoneme code. When another zero is detected after the first two zeros of the present phoneme code, the zero array of the present phoneme code is sent to the sound card D/A buffer for each extra zero.

The two 10 mS blender arrays are used to ease the transition from one phoneme to the next phoneme when played on the computer's speaker.

Then the next detected phoneme code is sent to the sound card D/A buffer and so forth. The sampling rate for the D/A is 66,000 Hz because 66,000 Hz was used to make the original phoneme code arrays in the look-up library. Although this example uses one set of phoneme voice clips for each phoneme code, the computer contains 11 other sets of phoneme voice clips, which can be selected by the operator pressing one of the F1 through F12 keys on the computer keyboard.

Making the operator's phonemes sequence

Before doing the transmit sequence the phoneme library arrays must be known. This is a one-time only event, which must be done before the computer is connected to the radio. The operator says words into the microphone that are displayed on the computer monitor, while holding down the space bar on the keyboard.

The same microphone and A/D converter from the transmit section are used to make the numbers of the phoneme, which are then applied to the same FIR filters. After the start of the phoneme sample interval, the absolute value of the next 14 numbers from each of the 16 frequency bands are examined for the largest value. This is the same peak search process as in the transmit section. Just before the end of the interval, say at count 1979 of 1980, the 16 peak numbers are put into the phoneme sample array. The phoneme sample array becomes the library value for that phoneme. But this library value might be wrong. So the word should be repeated and averaged. When the change in the average is small, then there is enough information to use the array. This needs to be done for all 44 phonemes. The no-sound phoneme is the only exception. No testing is required. Any DPD problems are exactly the same in both the transmit sequence and the making operator's phonemes sequence, which negate each other.

Making the library sequence at the distribution

The main, zero, third and two-third arrays used in the library of the receive section needs to be made. Twelve different people should record the 44 phonemes. This will be done in the lab with audio spectrum analyzers and high tech computers. Each of the numbers in an array must start and end at zero crossing and have a positive slope at each start and a negative slope at each end. This is to prevent discontinuities when any two sets of numbers are connected then played into the computer speaker. After the main phoneme arrays are made, the zero arrays are made. This could be done in the lab by changing individual numbers in the zero array for best sound when connected and played on the computer's speaker. The third array and the two-thirds array are easy to do.

Conclusion

At this time I have not succeeded in learning any version of C++. Without help modifying and writing code, this project ends at this paper. If you have not made up your mind that this not work, please contact me at mike-lebo@ieee.org 858-278-5851

Supplement to digital speech within 100 Hz bandwidth

Mike Lebo, N6IEF
3172 Pasternack Place
San Diego, CA 92123-3058
mike-lebo@ieee.org
858-278-5851

Why not use voice to text software and
text to speech software that is available now?

1. Timing Lets assume the above is working perfectly. Lets also assume one speaker of a stereo is the original voice and the other is the sound from the speaker of the receivers computer. Lets also assume the original voice is delayed the same amount as the all the processes. In a three minute speech the sound from the stereo speakers will not be the same. The best example of this would be the varying sound of a record played with a hole drilled off center. The timing of the digital speech within 100 Hz bandwidth software is accurate to within plus-or-minus 5 mS at any time no mater how long the original live speech last.

2. The alphabet verses the phoneme This is just as true today as it was over 100 years ago.

A Plan for the Improvement of English Spelling

For example, in Year 1 that useless letter c would be dropped to be replased either by k or s, and likewise x would no longer be part of the alphabet. The only kase in which c would be retained would be the ch formation, which will be dealt with later.

Year 2 might reform w spelling, so that which and one would take the same konsonant, wile Year 3 might well abolish y replasing it with i and Iear 4 might fiks the g/j anomali wonse and for all.

Jenerally, then, the improvement would kontinue iear bai iear with Iear 5 doing awai with useless double konsonants, and Iears 6-12 or so modifaiing vowlz and the rimeining voist and unvoist konsonants.

Bai Iear 15 or sou, it wud fainali bi posibl tu meik ius ov thi ridandant letez c, y and x -- bai now jast a memori in the maindz ov ould doderez -- tu riplais ch, sh, and th rispektivli.

Fainali, xen, aafte sam 20 iers ov orxogrefkl riform, wi wud hev a lojikl, kohirnt speling in ius xrewawt xe Ingliy-spiking werld.

Mark Twain

3. Non-text word that are commonly spoken. Al Capp made an art-form of spoken words that are not easily written or read in his comic strip "Li'l Abner". How many times have you used the word waja? When you can't understand someone, you say "Waja say?" (What did you say?) Voice to text software and text to speech software can't handle this, but the digital speech within 100 Hz bandwidth software would not recognize anything unusual.

Shannon - Hartley Capacity theorem

Shannon's Law (actually Shannon - Hartley Capacity theorem) relating noise, bandwidth, signal power, and capacity can be found in any good communication text. In common terms the bandwidth of voice can not be compressed. I believe that voice is made of speech and pitch and volume and timing. Digital voice is high fidelity. Digital speech is understanding the meaning of the words without recognizing that persons voice. When you read something, does your mind recognize the voice of the writer? Speech is the part of voice that contains meaning. It is the part that gets ham radio operators DX contacts. Speech can be compressed in bandwidth because it is made of phonemes that are quantize into a fixed number of parts.

Since the phoneme comparator at the transmit sequence used the voice of the sender, it will also use the pitch of the sender. Since the phoneme length is variable in 10 mS step size, this software will take care of timing without exceeding the 100 Hz bandwidth. Our radios have automatic volume circuits (speech processors) to make them have a constant volume, which means, volume is not as important as speech. Again in common terms we can understand the words of an old man or a little girl even though the voices are very different.

The software actually uses the pitch of the senders voice when the transmit section of the software learns the senders phonemes. When the receive section plays one of the twelve sets of phonemes, that will include pitch of the twelve people who made the audio clips of the phoneme library.

Channel spacing

As ham radio operators, we have clear channels now that the sun spot cycle is at its minimum. When the sun spot cycle is at its maximum, there won't be many clear channels. With a 24 times reduction of bandwidth over SSB voice, there will be many more clear channels.

Why has this not been done before?

Unfortunately this project was not invented by one of the "big guns" in the digital voice and speech recognition software group. "Not invented here" applies.

Timing is critical for this to work and that part of software is not what is usually taught in schools.

The old computers are no match for speed and amount of memory of today's computers. But people only remember the sound of the "Speak-n-Spell".

In order for this project to work throughout the world, the software must be free. There is no hardware required beyond that used for PSK-31. The software cannot be patented because my paper has been published on the internet. There is no way to make money from doing this. People who work in the digital voice and speech recognition business could lose their livelihoods, if this project was to succeed.

What would it sound like if the voice of the person doing the sending was also used to make up the phoneme library for receiving?

I was hoping to be one of those twelve people. It makes me sad, disappointed and angry to know there was nothing else I could do to answer this question.

Conclusion

Without someone to do it, this project will end with my paper
http://docs.google.com/Doc?docid=dggwnj3m_28hcx4xkhg&hl=en and this supplement.