Doubletalk Detection

Adam Dolin

David Klaver
Abstract:

When processing a particular voice signal it is often assumed that the signal contains only one speaker, and all other signal energy is treated as noise. This is not always the case. The objective of this project was come up with a consistent and accurate method for detecting doubletalk – overlapping speakers in single signal. The method that we designed hoped to take advantage of the fact that a signal containing only one voice will appears periodic in the frequency domain, while one containing doubletalk will not display such periodic behavior. The method was implemented as follows: We created an array of comb filters with varying delays and therefore various numbers of equally spaced teeth in the frequency domain. After dividing the sample signal into smaller time windows, we passed each time window through this filter array in parallel and looked for one particular filter that would best match the period of the signal and filter out a large percentage of the energy of the original signal. If the sample contains only one voice then there should be one such filter that matches the voice’s period, but if the signal contains doubletalk, then no filter will be able to filter out a large percentage of the non-periodic signal energy. We concluded the algorithm by setting a threshold level to 25% and claiming that if one of the comb filters was successful in filtering out above this threshold of the original signal energy for any of the signals different time windows, then we could assume the signal is comprised of only a single speaker. If not, our method would determine the signal to contain doubletalk.
**Background:**

We began our project by looking at various signals, containing both single and multiple voices, in the time domain. This was done in an effort to determine the characteristics that may make it possible to easily differentiate between these two types of signal. We quickly noticed that the signal of one voice was very periodic, especially when we zoomed in the vowel sounds. In figure 1, for example, one can clearly see a period of about 47 time samples in the “i” sound of a female speaker pronouncing the word “nine.”

![Fig 1. Periodic Behavior of a Single Voice Signal in the Time Domain](image-url)
These vowel sounds seemed to contain most of the signal’s energy and were almost perfectly periodic. After verifying this with a few different samples, we took two of these periodic voices and combined them to create our sample of doubletalk. In this new doubletalk sample, the signal lost its periodic behavior as the periods seemed to overlap one another and interfere with each other’s period.

![Doubletalk in the Time Domain](image)

**Fig 2. Loss of Periodicity for Time Domain of Doubletalk Sample**

When we examined both the single speaker sample and the doubletalk sample in the frequency domain, we noticed that the single voice sample contained very periodic harmonics while the doubletalk sample did not. This is especially true at low frequencies, where most of the energy is for voice signals.
Fig 3. Frequency Domain for both Single Speaker and Doubletalk Sample

The reason for this loss of periodicity seemed to be that the periodic harmonics from each individual voice were overlapping each other in such a way that the overall signal did not display one consistent period throughout. With the help of Professor Ellis, we were able to come up the basic algorithm described in the abstract.
Implementation:

The first step in implementing our method of doubletalk detection was to create an array of comb filters that could potentially be used as best-fit filters to match the period of the voice signal. We realized that the algorithm would probably never choose a comb filter whose delay value was very low, but because we saw no harm in including such low frequency comb filters in our array, we chose a generous lower bound in our design. As we increased the value of the delay element in our filters, we realized that if the filter contained too many teeth then it would filter out a large percentage of the signal energy, regardless of whether or not it was matching the frequency of the signal. An upper bound was therefore set on the number of teeth that our filters were allowed to have, and we ended up with an array of 61 different comb filters whose delay \( L \) ranged from around 10 to 70. An example for one such comb filter with delay 23 was created with the following line of code: \( \text{freqz([2,zeros(1,23),-2],[2,zeros(1,23), -2*.5],512).} \)

Our next step was to multiply the array of filters, in parallel, with the frequency domain of a small window of a voice signal. The exact size of these windows was 1024 samples in the time domain, which we mapped to 512 samples in the frequency domain. We then compared the energy of the signal after it was passed through each one of our filters with the energy of the original signal. To make it easier to visualize what our data represented, we created a GUI that would allow us to view each one of the comb filters superimposed on the frequency domain of a given time window of the signal. This outputted, in addition to the graphical representation, the exact ratio of energy before and after the filter. Figure 4 shows the GUI before applying any filters. The list in the upper left hand corner gives the user the ability to choose the number of delays in the comb
filter that will be applied. After choosing a filter index and pressing the “apply” button, the plot will appear to the right and the energy ratio will be outputted beneath the x-axis. The reset button can be pressed at any time to clear the current comb filter from the interface.

![GUI Before Applying Any Filters](image)

**Fig 4. GUI Before Applying Any Filters**

The ideal button on the interface is the most important feature of our GUI. When it is pressed, the best-fit filter will automatically appear superimposed on the frequency domain of the sample. The user can easily look at the other filters to confirm that this is indeed the best-fit filter.
Figures 5 shows the best-fit comb filter of a time delay of 29 superimposed on the frequency domain of the sample. It is interesting to note, as figure 6 illustrates, that when changing the delay to 28 samples, energy ratio changes by 26%.

Fig 5. GUI After Applying The Best-fit Filter
We repeated this procedure to create a GUI that would do the same thing for the doubletalk sample. By comparing these two interfaces, one can see how there is one particular comb filter that nicely matches the period of the single speaker sample, while there is no such filter for the case of doubletalk.
Results:

For individual time windows, the results for this method of doubletalk detection were slightly inconsistent. There were certain time windows that demonstrated the method extremely well, such as in the following figures:
Fig 8. Frequency Domain With best fit Comb Filters Superimposed
In this particular example, the best-fit comb filter was able to filter out 35% of the energy of the original signal when dealing with only one voice, but managed to filter out only 8% for the sample that included doubletalk. One can see how the best-fit comb filter was able to match up with the original signal much better for the case of a single speaker than for the one that contained doubletalk. There were also certain time windows containing doubletalk where the best-fit comb filter was able to filter out a relatively high percentage of energy, and conversely, certain time windows containing a single speaker where the best-fit comb filter was not able to filter out enough energy to conclusively rule out the possibility of doubletalk. If we look at the results for all of the time windows for a given sample signal together, however, the results become both consistent and reliable. To demonstrate this fact, and in fact to provide us with a conclusive method of detecting doubletalk, we plotted all of the results for each filter and each time window on the following three-dimensional figures.
Fig 9. Three Dimensional Plot of Results for Single Speaker
In these plots, the x-axis, easily discernable as the shortest axis on the graph, represents the different time windows of the sample signal. Each of the different comb filters is represented along y-axis while the depth of the plot indicates the percentage of energy retained in the corresponding time window of the signal after being passed through the corresponding comb filter. What is important to notice about these two plots is that the lowest point anywhere on the graph representing a single voice signal is well below 65%, whereas the plot for the double talk signal never drops below 75%.
Conclusion:

If we choose a threshold level of 75%, we can suggest that if the signal energy ever dips below this threshold percent of its original energy, we can assume that the signal contains only voice. If, however, the signal remains above this threshold value for all time windows and for all filters, then this indicates that the signal lacks a strong periodic nature, and we can conclusively say that it contains doubletalk. This seems like a reliable and accurate method for detecting doubletalk in a sample voice signal. The only serious problem that we encountered in implementing this method was, as mentioned earlier, if we are trying to detect doubletalk “on the fly,” that is by only looking at one particular time window at a time, then our method will occasionally produce errors in both falsely claiming to detect doubletalk and also failing to detect doubletalk in a sample signal.

Please Note:

All of the Matlab code that was written for this project, including that written to implement the GUI is included on this CD. Included is also a readme file that includes instructions for how to use some of the more complex programs, particularly how to make use of the GUIs.