Lunds universitet

Algorithms in Signal Processors

ETIN80

Speech Recognition using a DSP

Authors
Johannes Koch, elt13jko
Olle Ferling, tna12ofe
Johan Persson, elt13jpe
Johan Östholm, elt13jos

Date
2017-03-05

Contents

1 Introduction

The purpose of this project was to research, design and implement a speech recognition algorithm on a ADSP-21262 SHARC digital signal processor (DSP). The original goal was to be able to distinguish two spoken words from each other and to match the recordings with a member of a previously built database. Later on, the database was extended to contain four words instead of two. Another objective was to validate if a recorded word is a good enough match with the library. Since the DSP had hardware limitations and an unfamiliar programming platform, the project was split into two main parts.

The first part consisted of implementing the speech recognition algorithm, hereby denoted as SRA, in Matlab. Compared to the DSP, Matlab allows the use of seemingly infinite memory and processing power. Matlab also gives the benefit of having access to the entire signal that needs processing at all times. The DSP, on the other hand, runs in real-time and has limited memory, meaning no data can be stored. Part one of the project consisted of making sure the SRA design worked when being simulated in Matlab.

Part two consisted of translating the Matlab code into C code and making sure the algorithms still worked despite the limitations of the DSP.

2 Theory

2.1 Basic Speech Recognition

A basic speech recognition algorithm can be split into three states: listening, processing, and matching. In the listening phase, the DSP analyses the present audio signal to determine if speech is present. When speech is detected, the DSP starts to process the information to describe the speech in a compact way. A common way to describe speech is to divide into subsets of 20 ms and build mathematical models of each subset. When the processing is done, the DSP continues to the last phase, which compares the speech model to a pre-defined database. Each of these phases will be more thoroughly described later on in this report.

2.2 Unvoiced and voiced speech

Speech can be categorized into unvoiced and voiced speech. Unvoiced speech is the noise-like parts of a word containing consonants, and the tonal voiced speech containing the vowels. As an example, take the word "ask", where "a" is tonal, and "sk" is noisy. In this project we assume that the reflection coefficients for the voiced speech will result in dominating reflection coefficients.

2.3 Reflection coefficients and the Schur Algorithm

As mentioned above, a common way to describe speech is to split the speech in subsets of 20 ms each and mathematically describe each subset. For a short time as 20 ms, speech can be viewed as a wide-sense stationary stochastic process and can thus be fairly well described with an direct-form FIR-model (AR-model). This is a process that looks like

$$y_t = -a_1 y_{t-1} - a_2 y_{t-2} - ... - a_p y_{t-p} + e_t \tag{1}$$

where e_t is white Gaussian noise, the filter koefficients are represented by a_i, $i = 1, 2, \ldots, p$ and y_t is the output at time t. Of course, more coefficients means better abilities to describe the speech-subset. However, a more crude model with fewer coefficients will describe only the vital parts of the word. This makes matching speech from another person easier, since the vital parts describing a spoken word is similar from person to person.

Another way of describing the direct-form FIR-model is with a set of reflection coefficients $\{K\}$ of the lattice filter. These reflection coefficients are closely related to the AR-coefficients, and are calculated recursively. Linear prediction is used to calculate the optimal model describing the speech. The reflection coefficients can be calculated directly from the autocorrelation of the subset-signal using the Schur algorithm. The Schur algorithm is fast and will produce exactly the same result as the Levinson-Durbin algorithm, but without having to compute the AR-coefficients[1].

The resulting N reflection coefficients describe the speech-subset. Doing this for each subset will produce a matrix that describe the most vital parts of the spoken word, and can be used to compare to other matrices in order to match and validate what word was spoken.

2.4 DSP

A Digital Signal Processor (DSP) is a purpose-built device specialized in processing data which is continuously supplied. The DSP used in this project was a Sharc ADSP-21262 fitted on a custom made PCB. This DSP has two parallel buses on which messages can be sent, allowing for high-speed processing. The DSP has hardware support for addition, subtraction and multiplication. Division is emulated and therefore substantially slower than the other mathematical operations. When possible, multiplication by a constant, e.g. $scale = 1/q$ should be used instead of dividing repeatedly. The total memory of the DSP is limited to 4×128 kB all in all. This puts restrictions on the algorithms that should run on the device, as these cannot use too much memory. An average of all reflection coefficients describing a word is stored in a 10×10 array on the DSP in order to avoid noise having a too large effect on the database.

2.5 Filtering the sampled signal

Digital filters are applied to boost and cut certain characteristics of the sampled signal to make for better mathematical models. To deal with this, two different filters are needed. Environmental noise causing mic bias, i.e. a door being shut, are low in frequency but high in energy and are dealt with by a high pass filter. Much of the information in human speech lies in the higher frequencies. In order to boost the lower energy levels of high frequencies present in the human speech, a pre-emphasis filter is used. A pre-emphasis filter boosts the high frequencies while attenuating the lower ones, which flattens the spectrum. It just so happens that the filter characteristics of the high pass and the pre-emphasis filter combined can be reproduced using only one filter in order to conserve resources. As displayed in figures 1 to 3 below, combining the characteristics of the magnitude response of the high pass filter and the pre-emphasis filter will result in the combined filter shown in figure 3 . The combined filter have a damping effect on low frequencies of the high pass filter, and the boosting effect of high frequencies from the pre-emphasis filter. Figure 4 below display the magnitude response of the used filter, and it's easy to see the similarities to the combined filter.

[1] J. Proakis, *Digital Signal Processing. Principles, Algorithms and Applications*, Pearson Education Limited, United States of America, 2014, p. 859-862

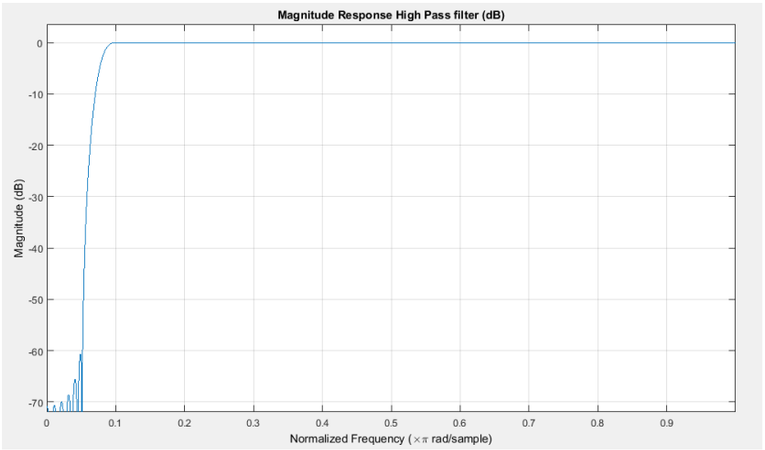

Figure 1: Magnitude response of the high pass filter

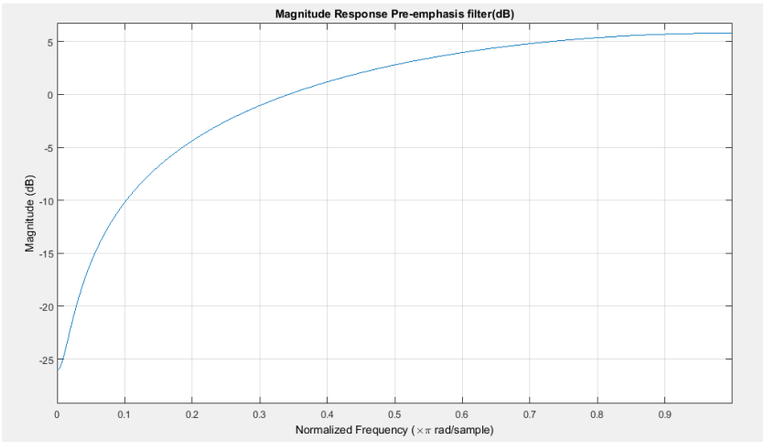

Figure 2: Magnitude response of the pre-emphasis filter

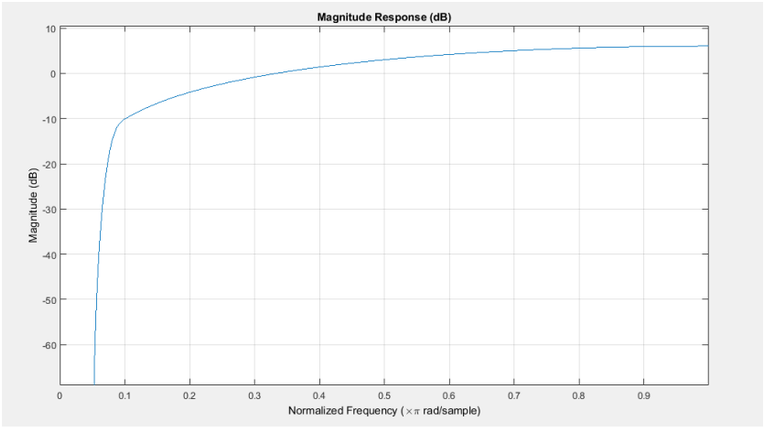

Figure 3: Magnitude response of the two filters above combined

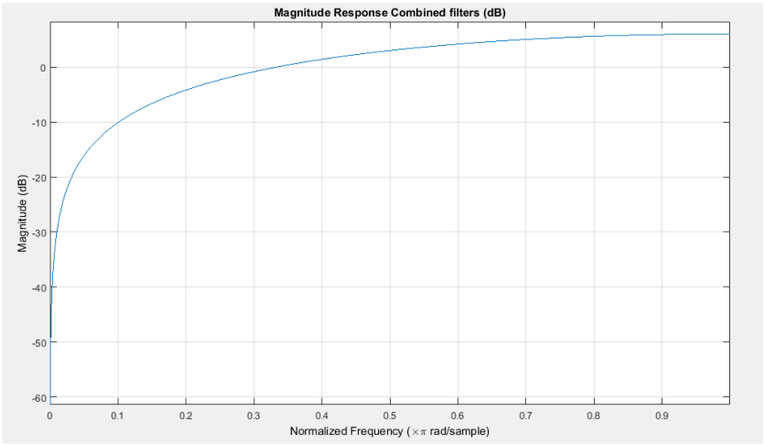

Figure 4: Magnitude response of the used filter

3 Method

3.1 Design

In figure 5, the general design structure of the speech recognition algorithm is shown. First of all, the recorded audio needs to be analysed to determine if speech is present. The algorithm that determines if speech is present listens on the background noise on the channel and sets a threshold for when the level is breached. To avoid spikes in the audio channel originating from e.g. a cup of coffee that is placed on the same table as the microphone, the algorithm requires the signal to maintain its energy for a short period of time. To avoid microphone bias from e.g. a pressure wave originating from a door being closed, a high pass filter was implemented before the speech-detection algorithm.

Reflection coefficients are always calculated, independent of whether speech has been detected or not. The difference is that if no speech is detected, the reflection coefficients of the three previous audio blocks are stored in a circular buffer. When finally speech is detected, the buffers are stored as the first columns in a coefficient matrix. This is done to avoid relevant data being lost due to a recording not starting in the beginning of a word. When speech has been detected, there is an algorithm that checks whether or not the speech has ended. This is done in a similar fashion as when detecting speech, with some additional features that prevents the algorithm to cut in a "natural pause" in a word. The DSP then continues to calculate reflection coefficients and storing them in the coefficient matrix. Ten coefficients are calculated, which is a common number that will describe the vital parts of the speech well enough without the risk of over-fitting. Finally, when speech has ended, the following three coefficient vectors are stored in the matrix. Three coefficient vectors in the beginning and end is intended both as a safeguard to catch all the speech, but also as simple windowing to apply weight on the middle parts. Now the DSP will compress the matrix from a $10 \times N$ matrix to a 10×10 matrix. This is done to allow for comparison between the recorded speech and the entries of the database. The compression-function was implemented in such way that it for the first $10 - u,\ u = mod(N, 10)$ columns in the compressed matrix calculates the row-wise average of K columns, where $K = floor(\frac{N}{10})$, in the original matrix. These average values are then saved in the new, compressed, matrix. The remaining u columns in the compressed matrix are calculated as the row-wise average of $K + 1$ columns in the original $10 \times N$ matrix.

The compressed matrix is compared to the 10×10 matrices in the database by evaluating

$$e_k = \sum_{i,j} (R_{ij} - D^k_{ij})^2 \tag{2}$$

where $D^k,\ k = 1, 2, 3, 4$ represent the different database matrices, and where R represents the recorded and compressed matrix at hand. The matrix D^k that results in the smallest e_k will be regarded as the best match.

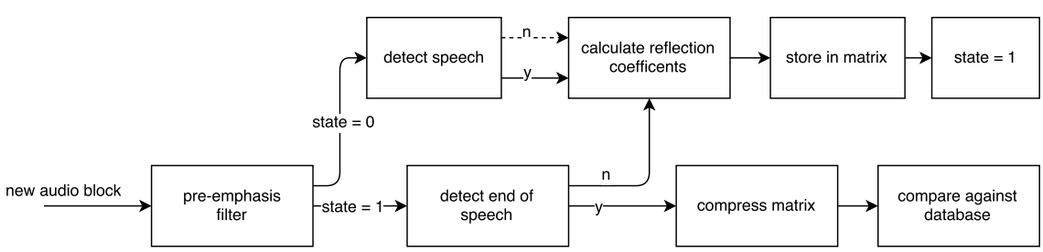

Figure 5: Flow chart of the general design. y = yes, n = no

3.2 Simulation

To verify that the previous mentioned design worked as intended, simulations in Matlab were made. As a first step the reflection coefficients for a pre-recorded speech sample was calculated using Matlab's built-in xcorr- and Schurrc-functions. It should be mentioned that the xcorr-function returns the auto-correlation sequence of the supplied signal and that this sequence is used in the Schur algorithm. It was suitable to use 10 reflection coefficients for each block of data, each block being 20 ms of speech. With a sample frequency of 16 kHz each block became 320 samples. Next step was to record an audio sample. Since Matlab supports audio recording, a microphone was used to record speech. The recording was then divided into N blocks, each block of 320 values, using Matlab's buffer-function.

For each block of data, a pre-emphasis filter was applied. This filter was a FIR-filter with coefficients 1 and -1, corresponding to a high-pass filter. Matlab's filter-function was used to implement this. A vector of 10 reflections coefficients for each block was then calculated. This was done for all N blocks of data. Each vector of coefficients was then saved in a $10 \times N$ matrix, where N is determined by the length of the recorded audio signal. To make the implementation in C easier, Matlab's built in Schurrc-function was replaced by a self-written function which had the same functionality as the built-in one.

Since recordings of speech will have some silence before and after the speech, a function that detects the start and the end of the spoken word and removes the silent parts, was written. This function was then split into two separate functions, one that removes the "silent" samples before the speech and one that removes the "silent" samples after the speech. The reason that this is important is that the "silent" samples contains no information that is of interest when the reflections coefficients are calculated .
When the samples containing no information have been removed, the data is divided and reflection coefficients calculated as described in the first paragraph. To make comparison between two matrices possible, all the matrices have to be compressed to the same size. This was achieved by implementing a function that compresses a $10 \times N$-matrix to a 10×10-matrix, more thoroughly described in the design section of this report.

After this was done, a database of three words was created; this was changed to four words during the implementation on the DSP. The words of the database, during the simulation, were: "Mikael", "Swartling" and "hundratjugosju". At first, only one recording of each word was used to build the database. This was to test that the recording worked as intended. Later this was replaced with the average of 12 recordings per word, 3 recordings per group member. The averaging was done on the compressed 10×10 matrices generated at every recording. The average was then saved in the database. If only one database was to be used, averaging gave better result for multiple persons than just a database built from one person.

3.3 Implementation

3.3.1 C-Code

The DSP was programmed using the C language. A DSP library was supplied, in which functions to facilitate the set-up and use of the DSP were included. Some macros that can be considered part of the design were also part of the library. Notably, the sampling frequency was set to $F_s = 16000$ Hz and the number of samples to be included in each data block was picked as 320. In this way, each block of data will correspond to 20 ms of speech, for reasons explained previously in this report.

To avoid overflowing the limited memory of the DSP (which would cause the speech recognition algorithm to crash), only static memory allocation was used in the imple-

mentation. This is a feasible approach because the nature of the program design. All matrices (multi-dimensional arrays) that the SRA uses are of pre-defined sizes. The only thing one cannot be certain of at compilation time, is how long the spoken word that should be recognized will be. Thus, the matrix containing the reflection coefficients of the recorded word is of unknown size. One solution to this problem is to allocate a two-dimensional array which is *sufficiently large* for any possible word. The implementation described in this report makes the assumption that no word will be longer than 6 seconds, implying that the array containing the speech recording should be a 10×300 array of float type. Thus, all memory in use will be known at compilation time and static memory allocation can be used. The advantage of this approach is that no program crash due to memory overflow is possible when the SRA runs. If the program tries to use more memory than available, this will be recognized during compilation and produce errors. This is preferable to the case of dynamic memory allocation, where overflowing the memory will not be caught at compilation time.

The database was built using the averaging technique described previously in this report. Every member of the project group was allowed to make three recording of the same word (e.g. "Mikael"). The average of the twelve resulting reflection coefficient matrices was then saved in a database.c-file as a 10×10 matrix. To be able to use the database in the main program, the corresponding header file was simply included in the main program.

The DSP has four different buttons, which can be used to send interrupts to the DSP. In this project, the buttons were used to

1. Set the DSP in different speech recognition modes
2. Record a database

In this project, the four buttons were used to set the DSP in one of four pre-defined recognition modes. These modes were

BUTTON1 Validation ON/OFF.

BUTTON2 Use Johannes' personal database.

BUTTON3 Use Johan Persson's personal database.

BUTTON4 Use the common (averaged) database. Standard case.

As shown above, the DSP can either run with validation mode on or off. If, for instance, button 2 has been pressed (and the DSP is thus running with the database of Johannes) and button 1 is pressed, the DSP will continue to run with Johannes' database but with a new validation state. When the DSP boots up, it will use the averaged database and no validation as default.

When building the database, the DSP was set to start listening for speech only when any of the keys on the DSP was pressed. This was done in order to re-initialize the DSP between the recordings described in the design section above. The need for a key-press introduced a *debouncing phenomenon*. When debouncing occurs, one key-press is interpreted as numerous by the processor. This in turn gives numerous interrupts *if* all keyboard interrupts are handled directly in the keyboard interrupt function. A feasible solution to this problem is to set a timer in the keyboard interrupt handler and handle the key-press in the timer interrupt handler once the timer expires. By using this approach, any debouncing in mechanical buttons will be allowed to die out before the timer expires and only one keyboard interrupt will be handled. The length of the timer, t_{delay}, has to be chosen depending on what hardware is being used. Admittedly, this approach introduces a delay in the total processing time equalling the length of the timer interrupt. However, the delay is introduced before any recording takes place, implying that the speaker has to wait for t_{delay} seconds. The actual processing time is not affected. If the timer is set to a reasonably low value (e.g. $t_{delay} = 100$ ms), the damage is quite small. The person using the application will most likely not be able to notice the small delay.

The DSP used in this project has six LEDs, which are used to indicate different functions and results to the user. Basic functionalities that the DSP indicates include

LED1 DSP status. LED on indicates DSP on and listening for speech.

LED2 DSP status. Speech detected. Recording.

LED3 Word match. Detected word is "Mikael".

LED4 Word match. Detected word is "Swartling".

LED5 Word match. Detected word is "Screen".

LED6 Word match. Detected word is "Black-board".

To compare two matrices, equation 2 was used to match each word in the database with the recording. In the case where no validation was used, the word with the smallest error was considered as the correct word and therefore considered to be a match. When validation was used, the best match was also required to have a sum of squared errors less than a pre-defined value for the word to be considered a match. If the threshold was violated, the DSP output "no match" instead.

4 Result

In Matlab the algorithms worked just as intended, and the program was able to distinguish the right word most of the times. It seemed to match roughly the same amount of times regardless of which of the four recorded words in the database that was spoken. The amount of matches seemed to be the same no matter whether or not the person speaking was part of creating the database. However, the smallest value of the error described by equation 2, which is used to determine what word was being said, tended to vary. The smallest recorded error was around 0.9, and the largest error still yielding a correct match was somewhere around 7. The big difference made validation difficult, even though the program still managed to find a correct match.

In a conducted test, three of the group members said each member-word of the database ten times and the match frequency was noted. They can be seen in the table 1 below.

	Mikael	Swartling	Screen	Blackboard	Total
Correct matches	22/30	24/30	29/30	30/30	105/120
Incorrect match	8/30	6/30	1/30	0/30	15/120

Table 1: The amount of correct matches/tries without validation

Further on, a test where a group member using his own database and validation mode ON was performed. In this test, the four words of the database was repeated ten times each. Also, the words "Jacket" and "DSP" was repeated ten times to test the validation threshold. The results are shown in table 2 below.

	Mikael	Swartling	Screen	Blackboard	Jacket	DSP
Correct matches	10/10	4/10	4/10	10/10	0/10	0/10
Incorrect match	0/10	0/10	0/10	0/10	0/10	0/10
No match	0/10	6/10	6/10	0/10	10/10	10/10

Table 2: The amount of correct matches/tries using validation on personal database.

5 Discussion and conclusion

The result of the project was very satisfying; we managed to accomplish more than the basic requirements. These were to make a program that could distinguish between two words for multiple speakers, our speech recognition program can distinguish between four words for multiple speakers with satisfying results.

As can be seen in table 2, we never get an incorrect match, which is the most important thing in speech recognition. However, by choosing a threshold giving raise to this result will cause some of the words in the database to not be recognized as a word in the database. Different words give different magnitudes of errors, making it hard to implement a common threshold level. Setting a separate threshold for each word can be done with some more fine tuning. Of course, there might be a better way to set a threshold than just a error magnitude threshold.

It is also possible that different matrix compression techniques would generate a better result, or that different sizes of the compressed matrix would work better for our purposes. The reason that we chose the previous mentioned method to compress a $10 \times N$ matrix was because of its simplicity. The size of the compressed matrix was chosen to 10×10 since it was desirable to use as little memory as possible, and the size 10×10 was considered a reasonable size. This might however not be the optimal size, nor the method to compress the $10 \times N$ matrix.

Also worth mentioning is that at first, we only had three words in the database. This amount was later increased to four words, the reason being that one of the three words was quite hard to identify and often resulted in incorrect matches being displayed. The reason behind the problems might the word length, or that the structure of the words are too similar. In any case, the word was changed and two other words were entered into the database instead. This change resulted in better matching even though the size of the database was increased.

When discussing the words that were used in the database, it should be mentioned that the SRA relies on two parameters for correct identification of a recording. These are the vowels of the detected word as well as on the position of these vowels within the word. The reflection coefficients are very much depending on what vowels are being spoken. Consonants contains less information, and reflection coefficients describing consonants are less prominent than reflection coefficients of vowels. Perhaps using other words as our database would have increased the hit rate using validation methods, but this was nothing we investigated further due to lack of time. We've also come to the conclusion that accentuation matters. Since the vowels are the information containing part of a word, and the vowel is spoken differently depending on your accent, it is quite hard to build a database that works for a common speaker. This was especially apparent when a group member tried to use the database of another person, as this resulted in more errors by the SRA.