Bastien Gouila, Marius Giedraitis, Katrin Krüger

Acoustic tachometer to detect car engine rpm readings

Helsinki Metropolia University of Applied Sciences

Information Technology

Digital Signal Processing

Final Project

30.10.2016

Helsinki Metropolia University of Applied Sciences

Authors Title Number of Pages Date	Bastien Gouila, Marius Giedraitis, Katrin Krüger Acoustic tachometer to detect car engine rpm readings 8 pages 30 October 2016
Degree	Bachelor
Degree Programme	Information Technology
Specialisation option	
Instructor	Anttii Piironen, Dr. (Principal Lecturer)

Keywords	

Contents

1 Introduction

The main objective of our project is to create an acoustic tachometer of a car's engine. For achieving this, the revolutions per minute (rpm) need to be calculated based on the soundtrack. By using a special tool, we first created some simulated sounds of an engine to know the parameters we have to calculate later. Based on this sound example, we started to make specified analytics using matlab. Based on the Fast Fourier Transform we figured out the difference between the peaks of the frequency. Using a formula, the rpm can be calculated.

2 Implementation

2.1 Create a sound example

To start with any analytics for our project, we first needed a good sound example of an engine where we knew all the parameters to check the results of our signal analysis with the known input.

Luckily we found the program called "REV", which is a well-known software to simulate the sound of any vehicle engine model. The program is shown in figure 1.

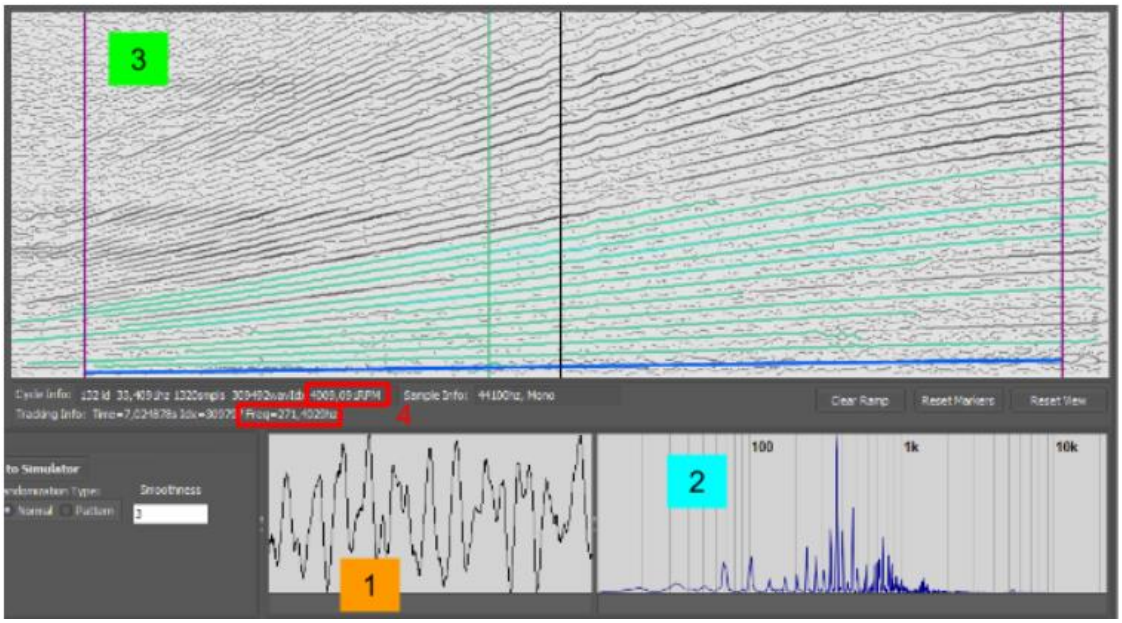

Figure 1. Program REV to create engine sounds.

In figure 1 the four following points are indicted:

1. Analog signal for 4009 rpm
2. FFT for the 4009 rpm signal
3. Harmonics frequency in function of the speed
4. Motor speed (4009 rpm) and frequency if the 8th harmonic (271 Hz)

Using this tool, we created a sound of a Chevrolet Camaro SS in the 5th generation with an V8 engine. The first sound examples we used were different rpm from 2000 to 6000 but all with a constant speed and no acceleration or breaking. This should make it easier to check if the first tries we made were correct or not. By using “audacity” we recorded the sound and finally we could use this wav-file to make our analyses in matlab.

2.2 Sound analyses in MATLAB

To calculate the revolution per minute we need to solve several problems that are explained in the following chapters with the solution that we found. We started by doing basic analytics of the signal, figured out different modifications that were necessary to finally calculate the revolution per minute.

2.2.1 Analyzing the original signal

As we had simple parameters and information given in the program “REV” where we created the sound, we started with calculating the Fast Fourier Transform (FFT) of the signal. The code below shows how we could get the result that is visible in the graphs.

```
[signal,fs] = wavread('filename');
signal = signal(1:fs*filelength);
nf = 4096*3;
y1 = fft(signal,nf);
f1 = fs/2*linspace(0,1,nf/2+1);
plot(f1,abs(y1(1:nf/2+1)));
```

Figure 2. Code to plot the FFT of a signal.

The two variables that are cursive in the code above must be replaced. “filename” is simply the name of the wav-file we want to read and “filelength” is the length of the file in seconds.

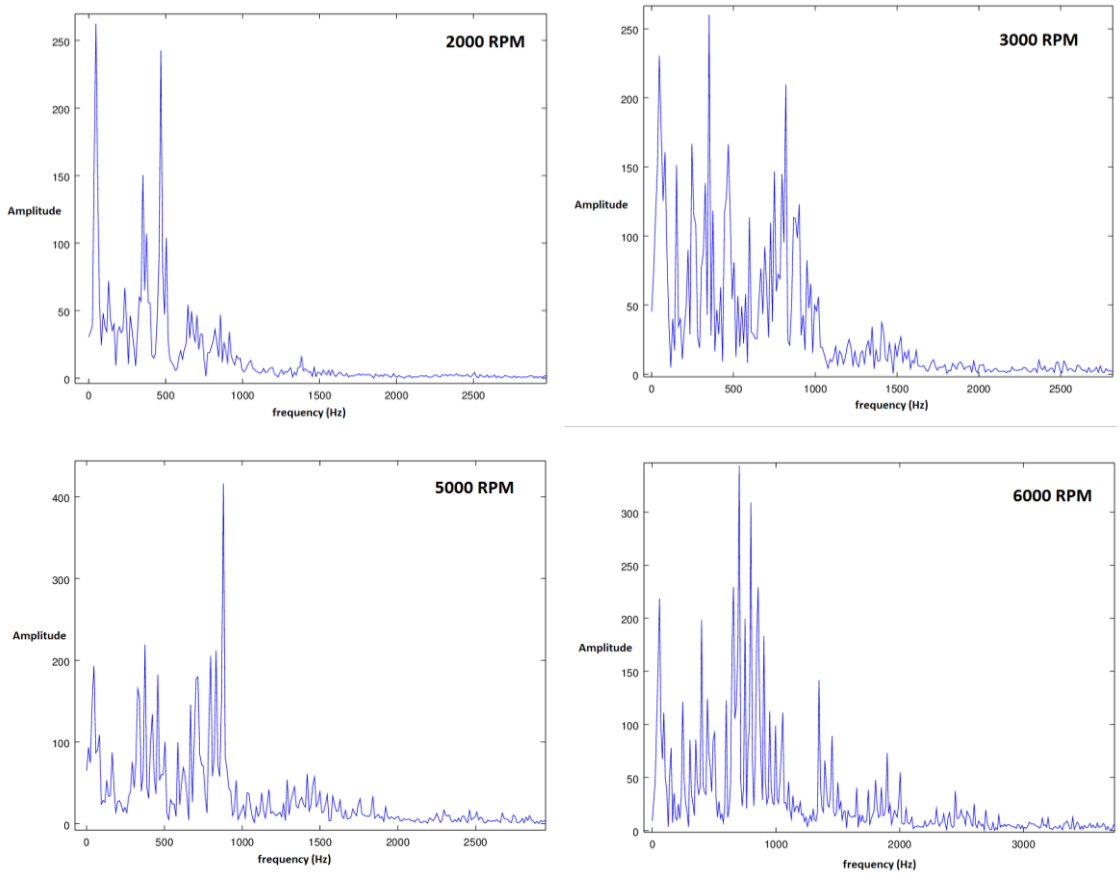

Figure 3. FFT of different RPMs.

Comparing these results and the FFT we got in the program, we came to the conclusion, that we can create and add some filters on our signal to depress the mechanical noise.

2.2.2 Modifying the original signal

As the graphs above already show, every sound example has a very high response for frequencies under approximately 100 Hz. The noise above 1000 Hz can be depressed as well as this does not have any relevance to the sound.
For achieving the first requirement, a high-pass filter with following parameters was created:

- cut-off frequency = 200 Hz
- stopband frequency = 100 Hz

```
fc = 200;
tb = 100;
fsb = fc - tb;
Fn = fs/2;
[NN, Wn] = buttord(fc/Fn, fsb/Fn, 0.1, 3)

[bbh,aah] = butter(NN, Wn, 'high');
freqz(bbh,aah,1024,fs)
```

Figure 4. Code to create high-pass filter.

As you can see in figure 5, the result of the filter fits to our requirements.

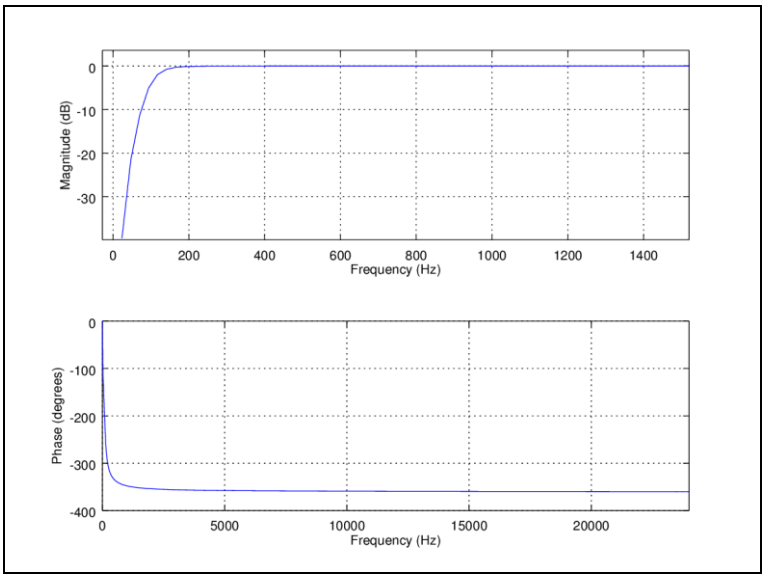

Figure 5. High-pass filter.

For depressing the noise above 1000 Hz, a low-pass filter was needed, where the cut-off frequency is set to 1000 Hz and the transition bandwidth is around 100 Hz.

```
fcl = 1000;
fsbl = fcl + tb;
[NN, Wn] = cheb1ord(fcl/(fs/2), fsbl/(fs/2), 2, 51)
[bb2,aa2] = cheby1(6,2,fcl/(fs/2));
freqz(bb2,aa2,1024,fs)
```

Figure 6. Code to create low-pass filter.

Figure 6 shows the graph of this low-pass filter.

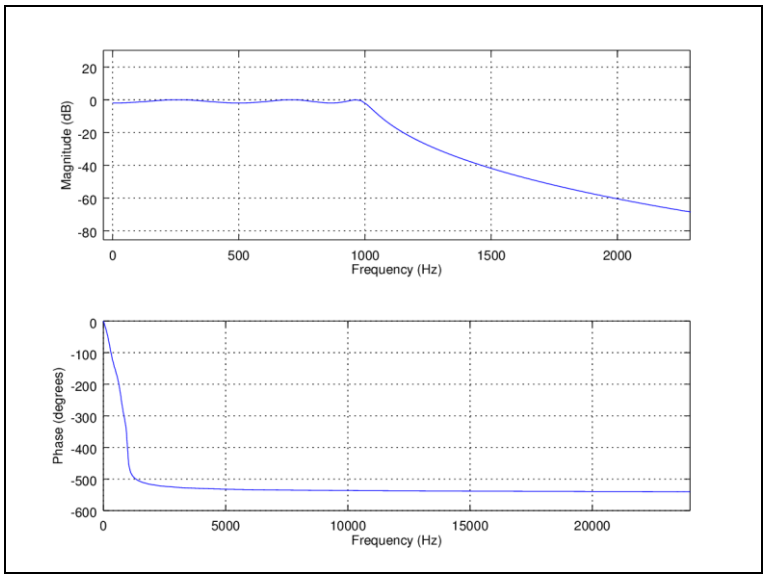

Figure 7. Low-pass filter.

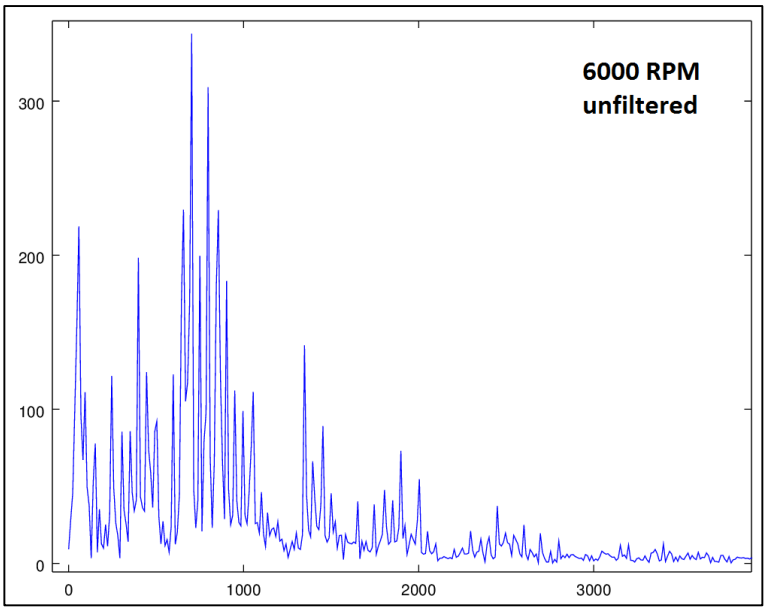

Figure 8. Unfiltered FFT result of 6000 rpm.

As mentioned above we just add the two filters on our signal. For the example we used 3000 rpm. So the first graph shows the FFT without any filters and the second shows the filtered signal.

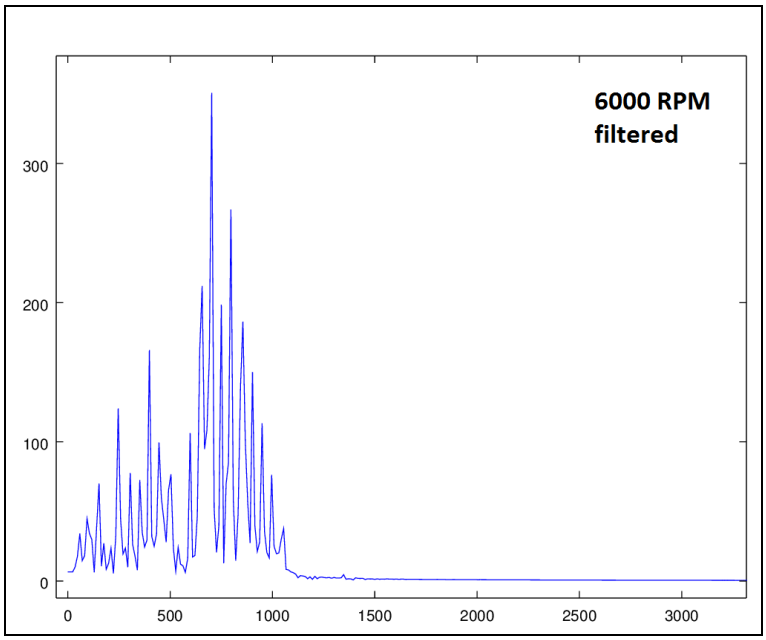

Figure 9. Filtered FFT result of 6000 rpm.

All the different steps (reading the signal, creating the filters and adding them to the signal) where joined together in one function for having a faster result.

Thanks to the filters that depress unnecessary sound, we have a good signal to continue with the calculation of the rpm.

2.2.3 Calculating the rpm

As we finally get the graph that shows the FFT of one constant speed, we are able to calculate the rpm.

As we know, that the harmonics frequency rises with the speed we can conclude, that the difference between the peaks rises as well. By calculating the difference, we can be more accurate than by counting the harmonics because we can avoid the fact that a harmonic could not be correctly visible. The basic idea was to get the difference between the peaks in the graph and take the median out of the value.
We first had to get the values out of the plotted graph of the rpm what was done by using the commands in figure 10.

```
h = gcf; %current figure handle

axesObjs = get(h, 'Children');  %axes handles

dataObjs = get(axesObjs, 'Children'); %handles to low-level
graphics objects in axes

objTypes = get(dataObjs, 'Type');  %type of low-level graphics ob-
ject

xdata = get(dataObjs, 'XData');  %data from low-level grahics ob-
jects

ydata = get(dataObjs, 'YData');
```

Figure 10. Commands to get values of plotted FFT.

Once we had the values we needed to get the peaks in the graph what we did by using the predefined function “findpeaks”. By setting a value for Threshold, the minimum value on the y-axis that the peak trigger would take is defined.

```
[pks,locations,widths,minAmplitude] =
findpeaks(ydata,xdata,'Threshold',50);
```

Figure 11. Code to find peaks in graph.

The last step is to calculate the rpm. For doing this, we need to take the median of the differences between the locations of the peaks, so the frequency. This value needs to be transformed from frequency into the revolution per minute. First the frequency in seconds needs to be changed to minutes, so multiplied by 60. As we know that our engine is an V8, we can conclude, that we need two revolutions to get one explosion. So we have to multiply the result by two. Figure 12 shows the final equation.

```
RPM = median(diff(locations)) * 15 * 8
```

Figure 12. Code to calculate the rpm.

3 Results

Finally, we get very close results that the next table shows.

Original RPM	Calculated RPM	Difference in rpm	Difference in %
2000	2.1094e+03	+109	5,45
2500	2.4609e+03	-40	1,6
3000	3.1641e+03	+164	5,4666
4000	3.8672e+03	-133	3,325
4500	4.5703e+03	+70	1,555
5000	4.9219e+03	-79	1,58
6000	5.9766e+03	-24	0,4

Figure 13. Compared results.

As figure 13 shows, the difference of the calculated result and the original rpm is always under 5.5 percent. So we finally found a proved way to calculate the revolution per minute.