Danmarks Tekniske Universitet

Color flow mapping using cross-correlation

22485 Medical imaging systems

October 29, 2020

Alvaro Carrera Cardeli
s172254

Federico Medea
s193612

Instructor: Jørgen Arendt Jensen

Contents

1 Introduction **2**

2 Methods **2**

2.1 Matched filtration 2

2.2 Stationary echo removal 3

2.3 Velocity estimation 4

3 Simulated data **5**

3.1 Matched filtration 6

3.2 Stationary echo removal 7

3.3 Velocity estimation 9

3.4 System integrating all the blocks 12

4 In-vivo data **13**

5 Conclusion **14**

1 Introduction

Color flow mapping (CFM) is considered one of the most important technique in medical imaging systems to estimate blood velocity in real time. By using ultrasound the velocity is coded as color intensity and direction is coded as different color in order to show a full map of the blood flow [1]. There are different types of CFM systems such as CFM using phase shift estimation with the auto-correlation approach [2] and CFM using time shift estimation with the cross-correlation approach [3]. For this study the cross-correlation approach will be taken into account. An advantage of this CFM system is its ability to detect higher velocities without aliasing in comparison to the phase shift system. Nevertheless, the velocity detection is uncertain because a false cross-correlation peak might be detected estimating a wrong velocity [1].
The aim of this report is to display a color flow map image using cross-correlation. Before estimating velocity, two previous steps are required: the matched filtration of the data (Section 2.1) and the stationary echo removal (Section 2.2). After that, the velocity estimation can be carried out (Section 2.3). In Section 3 each of the blocks in the system will be validated by using simulated ultrasound RF data from flowing blood at a constant velocity. Lastly, in Section 4 a CFM image will be generated using the in-vivo color flow data from the carotid artery. The model of the system and the simulation of the data will be developed in *Matlab*.

2 Methods

In this section an overview of the methods implemented is given. A block diagram of the system is shown in Fig 1, where the input data is the RF signal recovered from the interaction between pulses and the blood, and the output of the system is the blood velocity estimation. Through the system, the input data is filtered in order to remove white noise and stationary echo, and, then, it is cross-correlated to estimate the velocity.

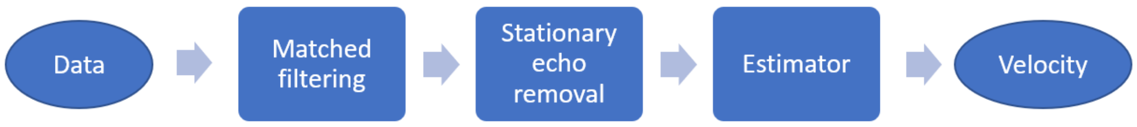


Figure 1: Block diagram of the velocity estimator.

2.1 Matched filtration

As stated above, the purpose of the matched filter is to remove noise, generated by electronics and other external factors, from the received signal. The idea of the filter is to try to find the match between the received noisy signal and the pulse emitted by the transducer (information known by the system, i.e.: number of cycles and center frequency f_0 of the pulse). Thus, the filter behaves as a band-pass filter centered in f_0. As a result of this, the frequency components of interest are maintained, while the noise frequency components are reduced or removed so that the signal-to-noise ratio (SNR) is maximized. The transfer function of the filter is:

$$H_m(f) = G_a R_s^*(f) exp(-j2\pi f t_1) \tag{1}$$

where t_1 is a time delay, G_a is a constant (in the case of study set to 1), and $R_s^*(f)$ is the conjugated spectrum of the expected received signal (i.e.: the pulse sent by the transducer) [1]. The time delay (t_1), in spectral domain expressed as an exponential factor, is to compensate the time shift introduced by the filter. By using the Inverse Fourier Transform (IFT), it is possible to obtain the filter transfer function in the time domain that is equals to $r_s(t_1 - t)$. Therefore, the filter in time domain is a time reversed and delayed version of the pulse emitted that convolves with the received signal.

2.2 Stationary echo removal

The RF signals received are composed of a flow signal r_s and a stationary signal r_t from the tissue:

$$r_i(t) = r_{s_i}(t) + r_{t_i}(t) \tag{2}$$

where i represents the RF-line number and t is time relative to the ith pulse emission [1]. The stationary echo is a signal that is constant along the time and that has an amplitude usually between 10 to 100 times higher than the flow signal. As a result of this, the stationary echo would dominate the received signal making the signal processing misleading. Thus, a high-pass filter is applied in order to remove this stationary echo signal (since it is constant noise, low frequencies should be removed in order to filter the stationary echo).
A simple way to obtain this removal is to subtract two consecutive RF lines knowing the time pulse repetition frequency T_{prf}. However, this has a critical point when there is not velocity or the velocity is really low, because it would lead to remove both, the echo stationary and the flow signal, thus, no signal would be received (SNR would dramatically worsen after the filter). Therefore, in the case of study it was decided to implement a more robust method that takes into account the mean of all the RF-lines obtained along the time (thus, get the constant component). Then, this mean is subtracted from each of the RF-lines.

2.3 Velocity estimation

When a transducer emits a pulse directed towards a vessel, part of the energy is scattered by the blood cells. Then, it propagates through tissue and is received by the transducer. For one emission the received signal includes a time delay that increases when the distance between the transducer and the scattering particles increases.
In the case of several pulses emitted every T_{prf} seconds, the movement of the blood cells will produce a small displacement in position between emissions. This displacement between consecutive RF-lines is know as time shift, and it can be calculated as:

$$t_s = \frac{2v_z}{c} T_{prf} \tag{3}$$

where c is speed of sound, $v_z = |\vec{v}| cos\Theta$ is the velocity vector of the scatterer and Θ is the angle between the ultrasound beam and the velocity vector (blood flow). It is noticeable that the time shift increases in time of reception, so it can be used for velocity estimation in pulsed systems.
In signal processing, cross-correlation is computed in order to have a measure of similarity between two signals. Therefore, by calculating the cross-correlation of two consecutive RF-lines, it is possible to see when these two signals match. Since, two consecutive lines are the same signal but time shifted, the cross-correlation between them will be like the auto-correlation of the signal but shifted from the origin. This shift from the origin will correspond to the time shift between them. Eventually, a velocity estimation can be obtained by using Eq. 3, so the final expression will be:

$$v_z = \frac{t_s\, c}{2\, T_{prf}} \tag{4}$$

Overall, a map of velocity variation can be obtained by calculating the cross-correlation between segments of consecutive RF-lines and, then, finding the shift of their peaks from the origin.
For this study, in order to have a more robust estimation it was considered to obtain one velocity estimation out of the average of a number of consecutive cross-correlations. Indeed, by doing this, it is possible to reduce the number of wrong cross-correlation peak detections that would lead to a bad velocity estimation. In contrast, the resolution in time for the velocity map will be reduced. Thus, it is important to find the right balance between these two aspects for a proper velocity estimation.

3 Simulated data

Simulations are the way to control that a program is correctly developing a specific function since the results expected are known beforehand. Thus, a simulation was generated in order to validate the model of the velocity estimator. In Fig. 2 it is shown the pulse that was used to run the simulation. The ultrasound pulse has a sampling frequency of $100\,MHz$. Calculating the FFT of the signal it is possible to obtain the spectrum (see Fig. 2, right) of the pulse. The center frequency of this spectrum is $2.28\,MHz$.

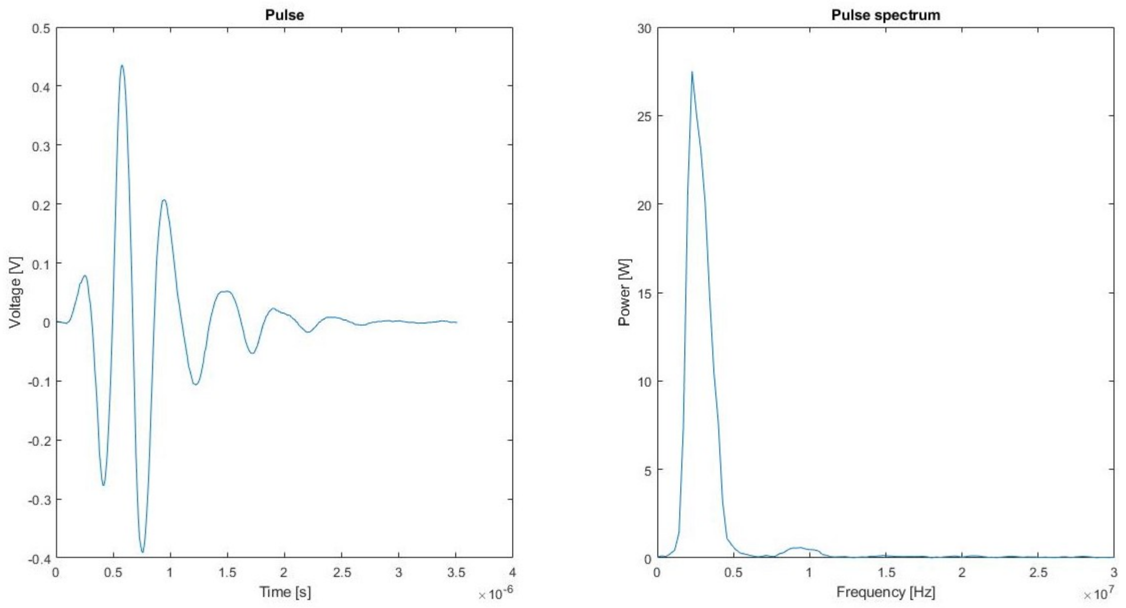
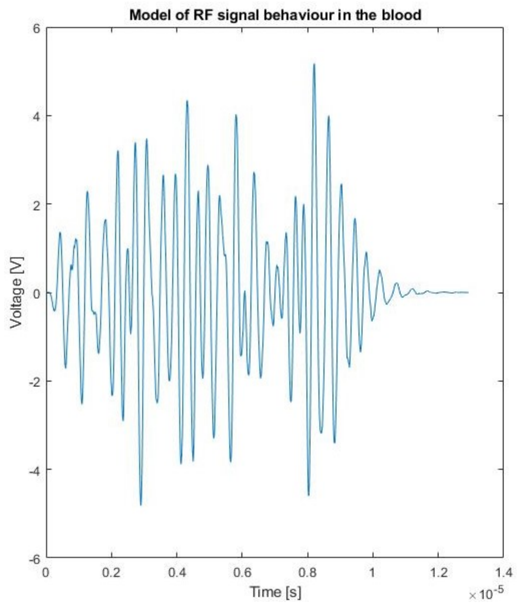


Figure 2: (left) Pulse in time domain, (right) spectrum of the pulse.

In Fig. 3 (left) it is shown a single received signal which represents how the pulse interacts with the blood throughout the vessel, in the simulation the interaction was modeled as a certain number of random scattering points. Thus, the received signal is obtained by doing the convolution of the emitted pulse with the scatters in time domain. Out of this signal, the simulation data is generated by applying a constant shift (constant velocity, $0.15\,m/s$ in this case) to 100 received signals (see Fig. 3, right). It is easy to notice from the graph how the signals are shifted by a constant value. Therefore, it can also be seen the constant velocity profile of the flow in the vessel, as stated in Eq. 3.



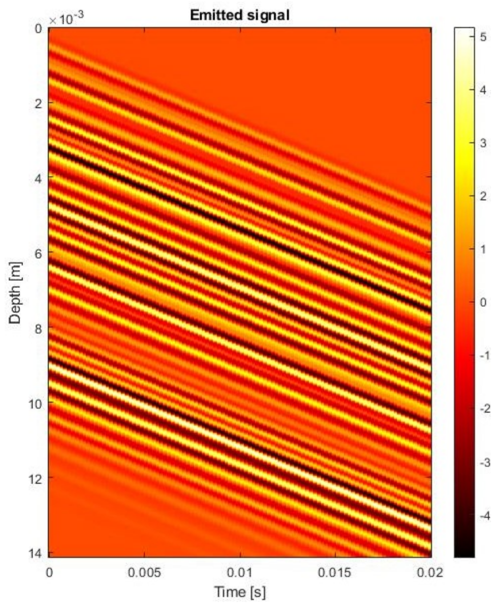
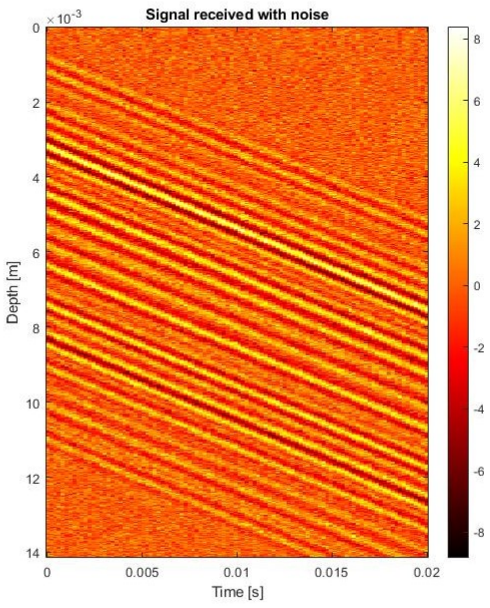


Figure 3: (left) Single received signal, (right) 100 received signals for velocity $= 0.15\,m/s$.

3.1 Matched filtration

To prepare the signal for the matched filtration, it was needed to add white noise to the signal received (see Fig. 4, left). Making a comparison with Fig. 3 (right), it is seen how the information received worsens. After applying the matched filter to the signal, it is possible to see how the original signal is almost recovered (see Fig. 4, right), although the power of the signal is increased due to the gain introduced by the filter.



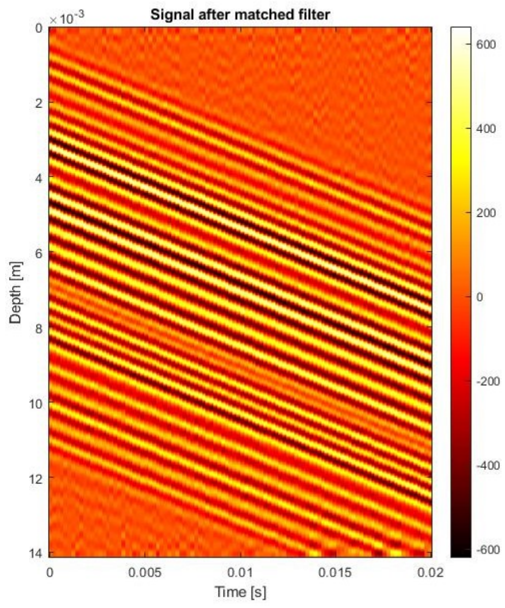


Figure 4: (left) Signal received with white noise, (right) signal recovered after matched filter.

In Fig. 5 (left) it can be seen one original RF-line plotted in comparison with the same received signal and the signal after filtration in time. It is easy to notice how the signal looks smoother after filtering it. Making a comparison of these three signals in frequency domain (see Fig. 5, right), it is possible to see the band-pass behaviour of the filter which is able to maintain the frequency components of interest from the original signal and removes the other frequencies added by the noise (the removal of high frequency components smooths the signal in time domain). In contrast, the frequency components that go from 8 MHz to 11 MHz from the original signal are removed too which will make impossible a perfect reconstruction of the original signal. Nevertheless, it is possible to conclude that the filter works properly recovering only the main frequency components. Lastly, it is important to mention that the signals were normalized in both figures in order to make an appropriate comparison between them.

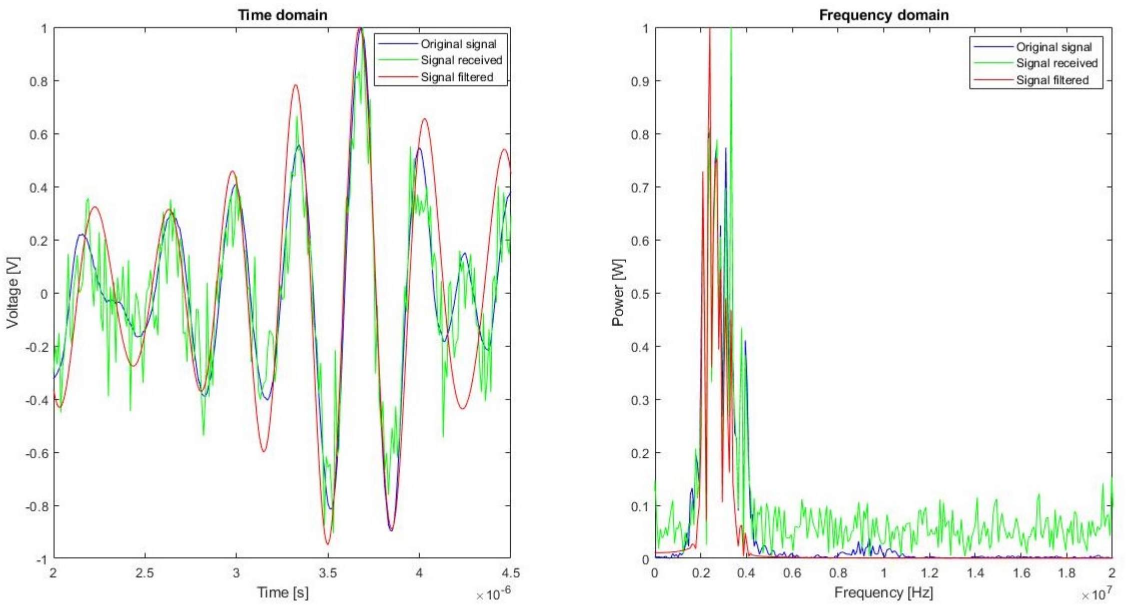
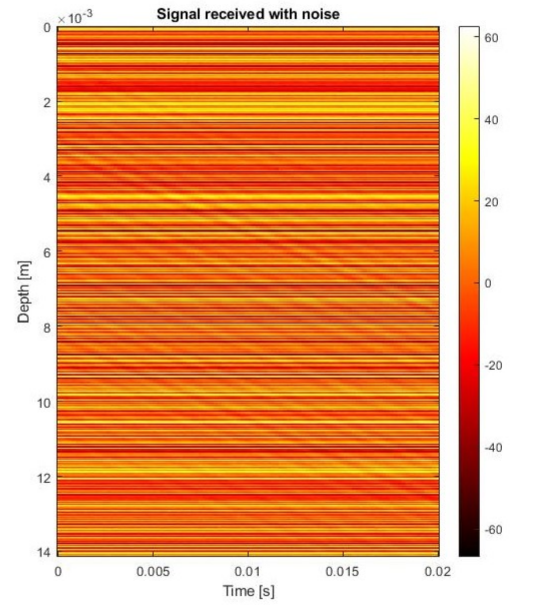


Figure 5: Original signal, received signal and match filtered signal normalized and colored in blue, green and red, respectively. (left) Time domain, (right) frequency domain.

3.2 Stationary echo removal

To prepare the signal for the stationary echo removal, it was decided to add stationary echo and white noise to the original signal. In Fig. 6 (left) it is shown the signal received with noise. It is easy to notice how most of the signal information is vanished due to the stationary echo. Indeed, the echo has an amplitude that is between 10 and 100 times higher than the original signal, so the signal received is completely hidden by the echo. Then, after applying the stationary echo removal, it is possible to recover the original signal with white noise (see Fig. 6, right).



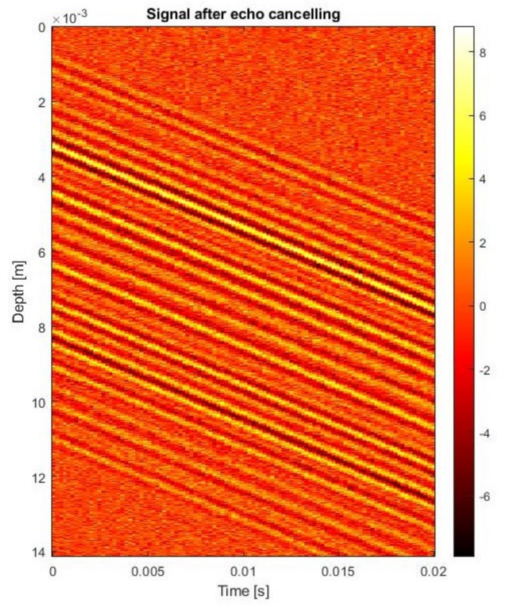
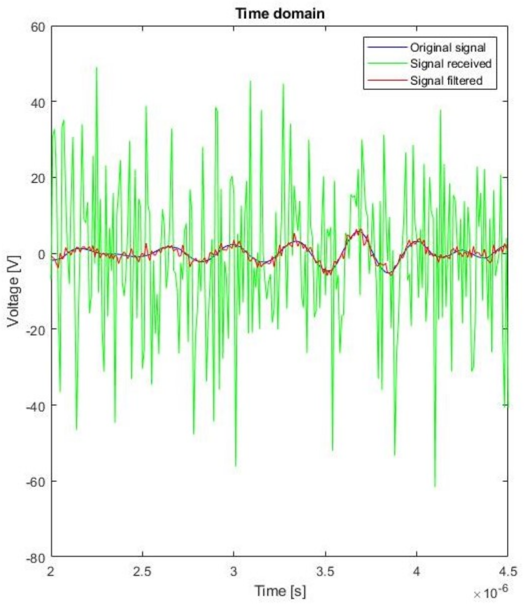


Figure 6: (left) Signal received with stationary echo and white noise, (right) signal recovered with only white noise after echo cancelling.

In Fig. 7 (left), like in the previous section, it is shown one original RF-line plotted along with the same received signal and the signal after filtration in time. It is noticeable that echo removal works properly since the filtered signal has the shape of the original signal but with white noise. Moreover, comparing the three signals in frequency domain (see Fig. 7, right) it is possible to see how the high amplitudes related to the frequency components of echo are removed by the filter.



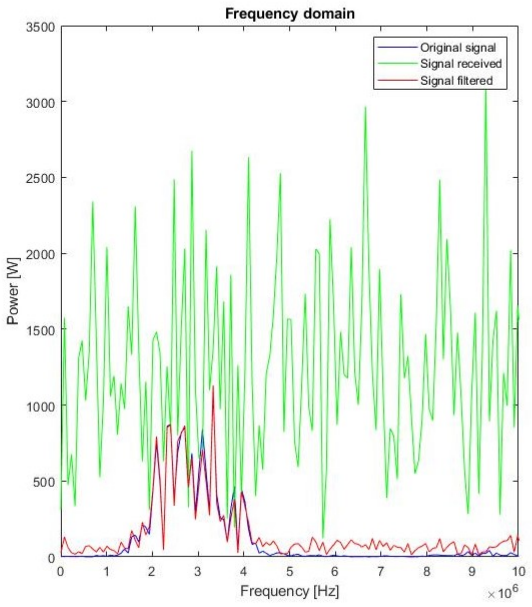


Figure 7: Original signal, received signal and echo filtered signal colored in blue, green and red, respectively. (left) Time domain, (right) frequency domain.

To conclude the two filtering sections, in Fig. 8 it is shown the Signal-to-Noise Ratio (SNR) obtained in each of the filters. In the graph on the left, it is shown the boxplot of the SNR before (blue) and after (red) echo removal for the 100 RF signals, as well as the SNR improvement generated by the filter (green). It can be noticed that the SNR before filtering is $-21\,dB$ (there is almost no variation in SNR between RF-lines due to constant feature of the echo and the big ratio echo-signal), while after applying the echo filter it is around $5\,dB$. So the SNR improves with a mean value of $25.92\,dB$. On the other hand, making the same comparison but for the matched filter (see Fig. 8, right), it is possible to see how the SNR before filtering is around $5\,dB$, whereas after applying the matched filter has a mean value of $20\,dB$ (the wider range of values shown in this boxplot may be caused by the fact that the white noise is not always filtered in the same proportion, in contrast with the echo where is always filtered the same amount of noise). Therefore, the SNR improves with a mean value of $14.46\,dB$.

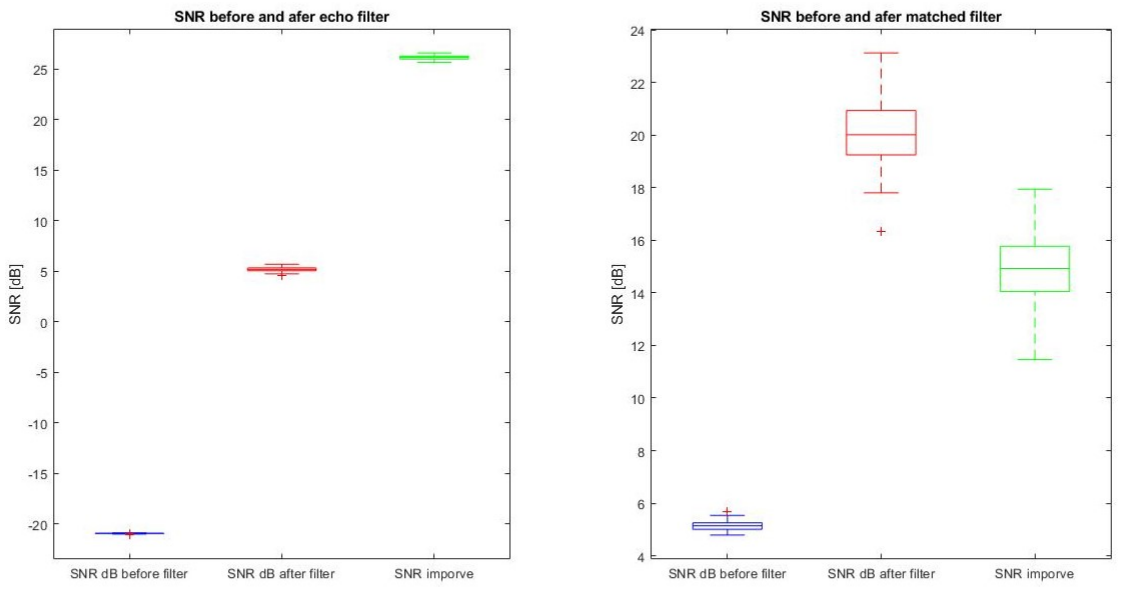


Figure 8: Boxplots of the SNR for the 100 RF-lines before filter, after filter and improvement introduced by the filter, colored in blue, red and green, respectively. (left) Results for echo removal, (right) results for matched filtration.

3.3 Velocity estimation

The velocity estimator returns the velocity out of the time shift obtained from the cross-correlation of two consecutive RF-lines. This block has three variable parameters that will determine the resolution and the precision of the system:

- **Segment Size:** defines how the RF-lines will be segmented, therefore it sets the resolution in depth (y-axis).

- **Span Size:** defines the window size used in order to obtain the cross-correlation for a segment, therefore it sets the velocity range considered and plays an important role for the accuracy of the estimation and the computing time.

- **RF-lines Overlap:** defines how many RF-lines will be used to obtain one cross-correlation average, therefore it sets the resolution in time (x-axis).

The velocity estimator was tested setting as an input data the original simulated signal, which has a velocity of $0.15\,m/s$, without any type of noise (see Fig. 3, right). Since there are three variable parameters, it was tested the performance of them considering different scenarios.

Fig. 9 shows the first scenario, where *segment size* is fixed to 10 points[1], *RF-lines overlap* is fixed to 1 line and *span size* is tested for three values: 10, 50 and 100 points, from left to right, respectively. It is plotted the velocity estimation (top) and the histogram of the velocities estimated (bottom). The *span size* of 100 points generates the best result (with 64.03% of probability detection), which is a reasonable result since a minimum number of cycles is needed in order to obtain a proper estimation. In contrast, if the value of this parameter is too big, the resolution of the system in depth will deteriorate and the computing time will increase drastically.

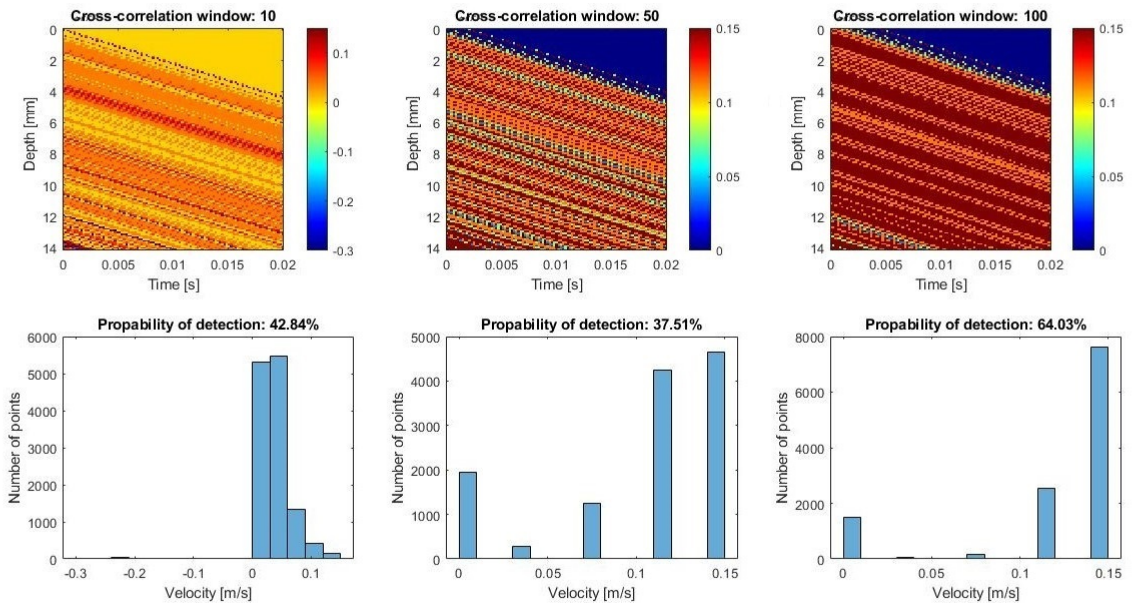


Figure 9: Velocity estimation for *segment size* and *RF-lines overlap* fixed, and *span size* variable. (top) Velocity map, (bottom) histogram of velocities.

[1]**points:** since it is a discrete system it was considered to use the notation of *points*. Since all the windows considered in this section lay along depth (y-axis), the conversion factor for all the windows in this section will be: $1.0937 * 10^{-5}\,m/point$ (i.e.: in this case 10 points window will correspond to $1.0937 * 10^{-4}\,m$)

On the other hand, Fig. 10 shows a new scenario where *RF-lines overlap* is the variable with values of 1, 3 and 9 RF-lines, from left to right, and *segment size* and *span size* are fixed to 10 and 100 points, respectively. In this case, the probability of detection has similar values for the three simulations (from 64.03%, in the worst case, to 66.97%, in the best case), due to the free-noise signal and the constant flow velocity that are used for the simulation. However, the velocity maps show the importance of this parameter for the time resolution. The more RF-lines used to estimate a velocity, the worse resolution. In contrast, if few RF-lines are used, then the computing time will increase and it is also more likely that there is a wrong estimation.

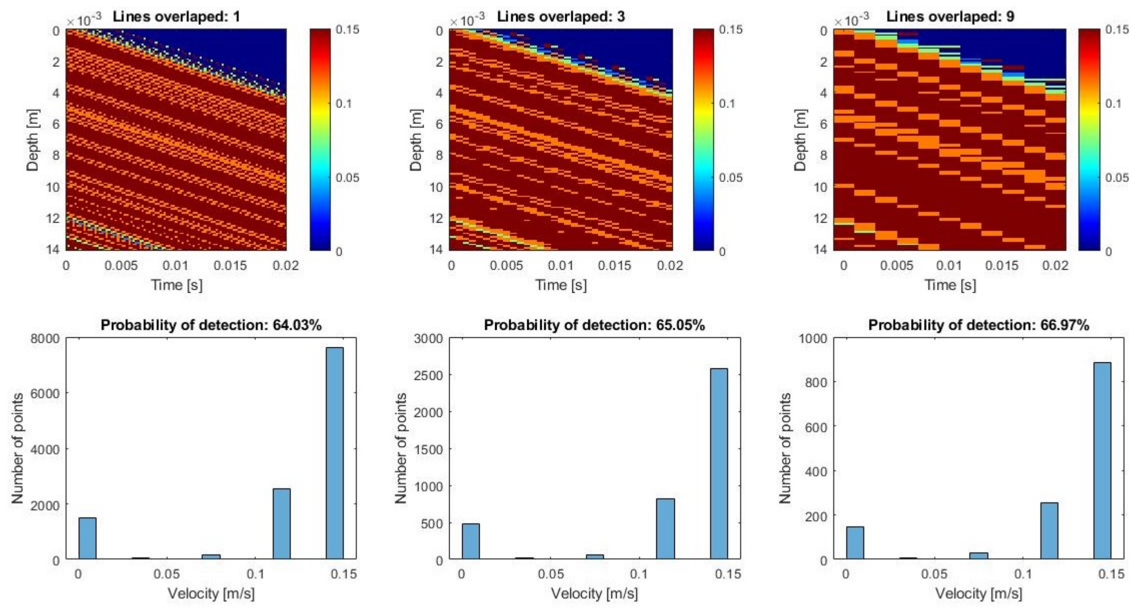


Figure 10: Velocity estimation for *segment size* and *span size* fixed, and *RF-lines overlap* variable. (top) Velocity map, (bottom) histogram of velocities.

Fig. 11 plots the velocity profile of a simulation at a certain time. The result expected from the data generated should be a constant velocity at $0.15\,m/s$, however the profile obtained shows how the velocity at some depths is bad estimated (fulfilling the probability detection of 64.03%, see Fig. 10, bottom-left).
Moreover, in Fig. 11 it can be appreciated the precision that has the system for the velocity estimation. Eq. 5 shows how the velocity calculated is proportional, by a constant value K, to the number of points shifted from the origin of the cross-correlation.

$$v_z = \frac{Num\_points\_shifted * c}{2 * f_s * T_{pfr}} = Num\_points\_shifted * K \tag{5}$$

Since the number of points shifted is an integer, the velocity will have a step value of K between two consecutive velocities estimations. Thus, the parameters in the constant

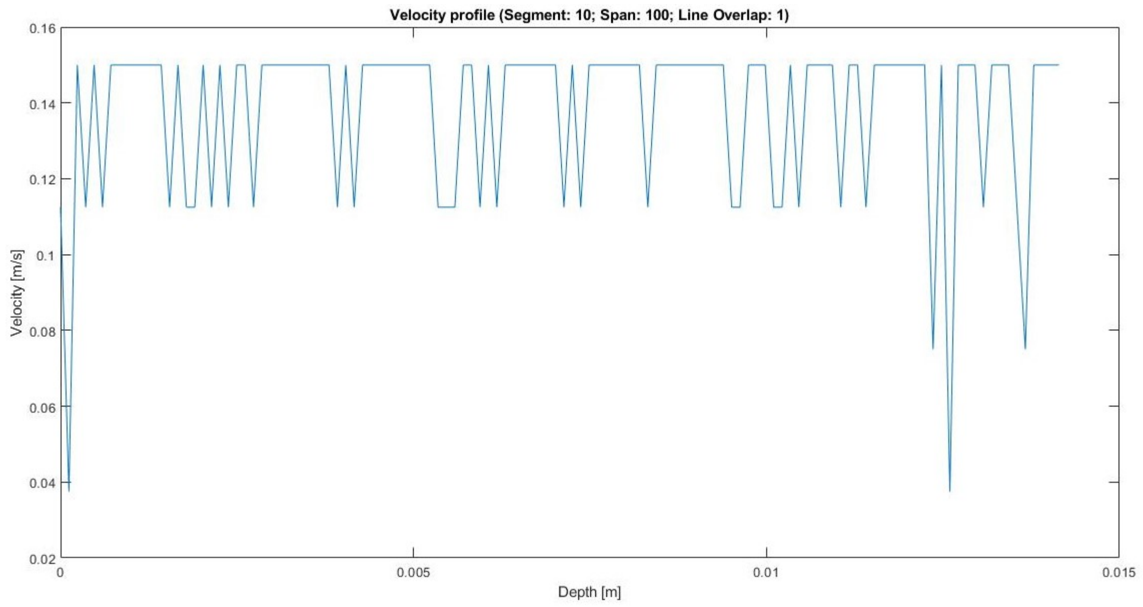


Figure 11: Velocity profile at a certain time simulated with a *segment size* of 10 points, *span size* of 100 points and *RF-lines overlap* of 1 RF-line.

value determine the grade of velocity resolution of the estimator (i.e. for a higher precision in velocity estimation, either f_s should be increased or f_{pfr} should be decreased). For the simulation parameters Eq. 6 demonstrates the different velocity values seen in Fig. 11.

$$K = \frac{c * f_{prf}}{2 * f_s} = \frac{1500[m/s] * 5 * 10^3[Hz]}{2 * 100 * 10^6[Hz]} = 0.0375 \tag{6}$$

Lastly, this is directly related with the *span size* since it will define the maximum velocity that can be detected (i.e. if *span size* equals to 4 points in the simulation, the cross-correlation vector will range from -3 to 3, therefore the velocity range will be from $-0.1125\,m/s$ to $0.1125\,m/s$, so the system would never get to estimate the correct velocity).

3.4 System integrating all the blocks

To conclude the simulations, it was tested the system (matched filtration, stationary echo removal and velocity estimation) with an input signal with white and echo noise and a constant blood velocity of $0.15\,m/s$ (see Fig. 6, left). The estimation for a *segment size* of 5 points, *span size* of 126 points and *RF-lines overlap* of 3 RF-lines is shown in Fig. 12, with a probability of detection of 64.03%.

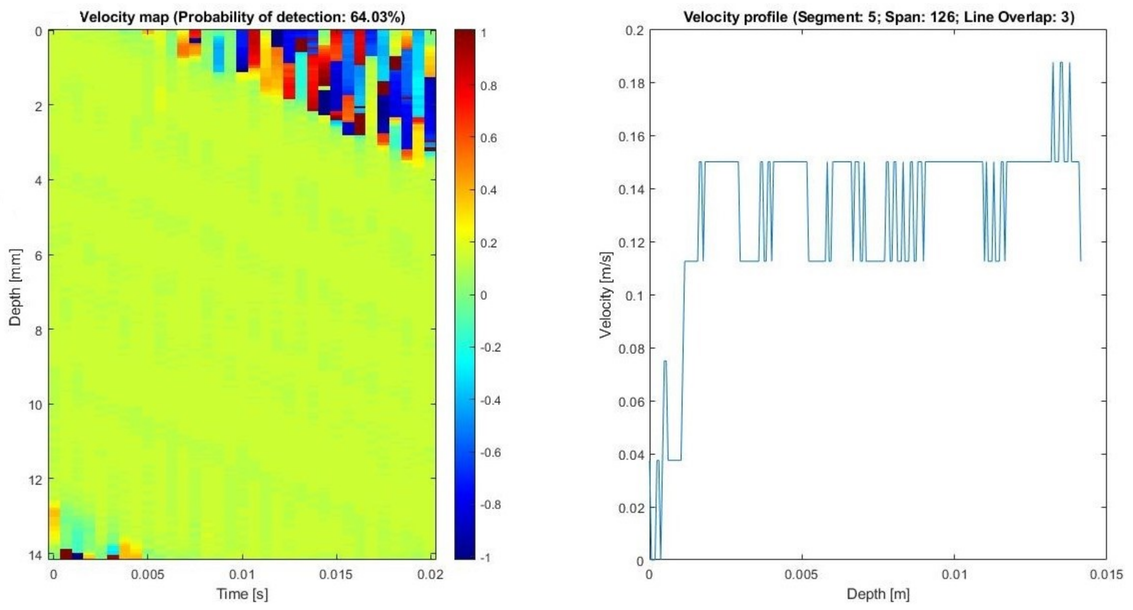


Figure 12: Velocity estimation by the system. (left) Velocity map, (right) velocity profile at a certain time.

4 In-vivo data

In this section the implemented system is tested with in-vivo data taken from a healthy male in order to display a color flow mapping from carotid artery. The data was acquired by using a linear array transducer. The ultrasound pulse has a sampling frequency of $40\,MHz$ with a center frequency of $5\,MHz$ and the number of cycles in the pulse equals to 8.

In order to generate the B-mode image that represents the tissue structure it was necessary to calculate the envelope of the data. The envelope is obtained by making a Hilbert transform on each of the RF lines. In Fig. 13 (left) it is shown the B-mode image calculated, where it is easy to distinguish the different stripes of the tissue.

As regards the CFM image, the implemented system in the previous sections was applied to the data with the following parameters: *segment size* equals to 10 points, *span size* of 38 points and the number of *RF-lines overlap* is 9. In Fig. 13 (right) it is displayed the CFM image of the carotid artery. It can be noticed how the blood flow goes in the same direction into the vessel since only negative velocity values are estimated.

In Fig. 14 (left) it is shown the image obtained by merging the two figures from Fig. 13. To achieve this final image it was necessary to match the sizes of the two images. Therefore, an interpolation factor of 10 was applied to the y-axis of the velocity matrix (axial distance). Regarding the x-axis (lateral distance), an interpolation factor of 4 was used and then a decimation factor of 7 was applied, in order to obtain a factor of 4/7.

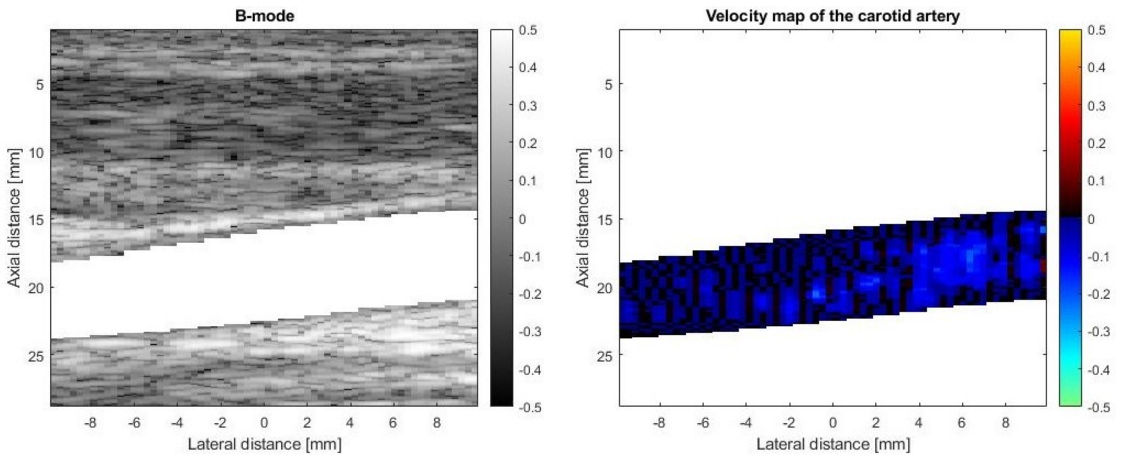


Figure 13: (left) B-mode image from the surrounding tissue of carotid artery, (right) color flow map from carotid artery.

The interpolation was carried out before the decimation to avoid losing any information from the previous calculation.
Lastly in Fig. 14 (right) it can be seen the velocity profile at a lateral distance of 5.14 mm. It is easy to notice how the flow profile looks parabolic as expected, with a peak value of $-0.15\,m/s$ around the center of the vessel. Indeed, the blood velocity from carotid artery is higher in the center of the vessel rather than at the edges, which tends to $0\,m/s$, generating this parabolic profile.

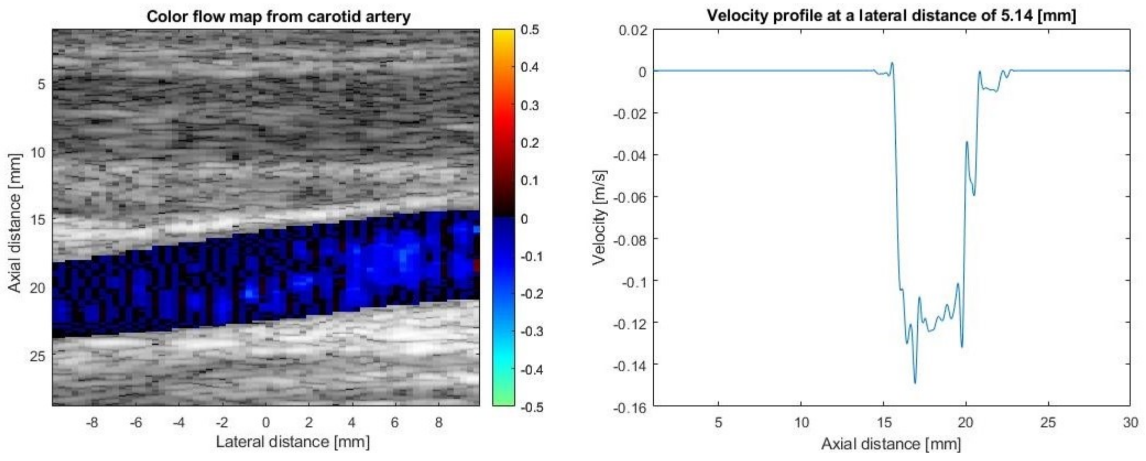


Figure 14: (left) B-mode image merged with CFM image, (right) velocity profile at a lateral distance of 5.14 mm.

5 Conclusion

After the test run in the simulations it can be concluded that all the blocks are working independently as expected.

In the case of filters, it has been possible to improve the SNR by 14.46 dB and 25.92 dB for the matched filter and the echo removal respectively. The difference between these improvements of around 10 dB as expected, since the echo provides much more energy to the signal than the white noise and, therefore, when removing it, bigger differences in SNR are achieved.
Moreover, as regards the velocity estimator block, a maximum detection probability of 66.97% was reached, which is positive and could even be increased by testing other values for the variable parameters of the estimator.
Finally, all the blocks as a whole (the system) have also behaved positively, giving a good distribution of velocities and maintaining a detection probability of 64.03%.
To conclude, the results obtained in the in-vivo data are also satisfying since the carotid artery is expected to have only one flow direction, and the distribution of the velocity in the vessel is parabolic-shaped.

References

[1] J. A. Jensen, *Estimation of blood velocities using ultrasound: a signal processing approach.* Cambridge University Press, 1996.

[2] C. Kasai, K. Namekawa, A. Koyano, and R. Omoto, "Real-time two-dimensional blood flow imaging using an autocorrelation technique," *IEEE Transactions on sonics and ultrasonics*, vol. 32, no. 3, pp. 458–464, 1985.

[3] O. Bonnefous, P. Pesqué, and X. Bernard, "A new velocity estimator for color flow mapping," in *Proc. IEEE Ultrason. Symp*, 1986, pp. 855–860.

Appendices

1 Matlab functions

```matlab
function [velocity_matrix] = mainFunction(data,fs,f0,cycles,c,T_prf,segmentSize,numPointsCorr,
    velocityRange,columnsOverlaped)
%// Method that runs all the blocks in the system and returns to the
%   original script the velocity map obtained after processing.

    data_matched = matchedFilter(data,fs,f0,cycles); %time domain
    data_matched_echo = echoCancelling(data_matched); %time domain
    velocity_matrix = crossCorrelationVariableSpan(data_matched_echo,segmentSize,numPointsCorr,fs,c,T_prf,
    velocityRange,columnsOverlaped);
end
```

Listing 1: mainFunction.m

```matlab
function [data_matched] = matchedFilter(data,fs,f0,cycles)
%// Method that filters the white noise from the signal.
% ------------ Input parameters:
% - data -> signal with noise
% - fs -> sampling frequency
% - f0 -> center frequency
% - cycles -> number of cycles in a pusle
% ------------ Output paramenters:
% - data_matched -> signal after filtration

    if(length(cycles) == 1)
        % Prepare the original pulse:
        t_pulse = linspace(0,cycles/f0,cycles*fs/f0);
        signal = sin(2*pi*f0*t_pulse);
        hanning_window = hann(length(signal));
        signal_sent = signal.*hanning_window';
    else
        % In the case of the simulation, cycles variable will be the pulse
        % vector:
        signal_sent = cycles;
    end

    % Generate tranfer funtion of the filter:
    h_filter = flip(signal_sent);

    % Filter the data:
    data_matched = filtfilt(h_filter,1,data);
end
```

Listing 2: matchedFilter.m

```matlab
function [data_filtered] = echoCancelling(data)
%// Method that filters the stationary echo from the signal.
% ------------ Input parameters:
% - data -> signal with statoinary echo
% ------------ Output paramenters:
% - data_filtered -> signal after removing the echo

    stationary_signal = mean(data,2);
    data_filtered = data - stationary_signal;
end
```

Listing 3: echoCancelling.m

```matlab
function [velocity_matriz] = crossCorrelationVariableSpan(data_matched_echo,segmentSize,numPointsCorr,fs,c
    ,T_prf,velocityRange,columnsOverlaped)
%// Method that defines a velocity map depending the resolution of the system, calculates
%   the cross-correlation for each 'pixel' of the velocity map and call the method
%   'velocityEstimator.m' to obtain the velocity from one cross-correlation.
% ------------ Input parameters:
% - data_matched_echo -> signal after filtering out the noise
% - segmentSize -> resolution of the system in depth
% - numPointsCorr -> number of points considered to calculate the
%                    cross-correlation for one segment
% - fs -> sampling frequency
% - c -> velocity of propagation in the mean
% - T_prf -> period of pulse repetition frequency
% - velocityRange -> maximum velocity in absolut value that the system can
%                    detect
% - columnsOverlaped -> number of lines used in order to calculate one
%                       average cross-correlation, resolution in time
% ------------ Output paramenters:
% - velocity_matriz -> velocity map obtained

    % Calculate all the correlations:
    if (numPointsCorr <= segmentSize)
        diff = segmentSize - numPointsCorr;
        number_of_segments = floor(size(data_matched_echo,1)/segmentSize);
        corr_3D_matrix = zeros(2*numPointsCorr-1,size(data_matched_echo,2)-1,number_of_segments);
        for j = 1:size(data_matched_echo,2)-1
            signal2 = data_matched_echo(:,j+1);
            signal1 = data_matched_echo(:,j);
            for g = 0:number_of_segments-1
                corr = xcorr(signal2((1+g*segmentSize+ceil(diff/2)):((1+g)*segmentSize-floor(diff/2))),...
                    signal1((1+g*segmentSize+ceil(diff/2)):((1+g)*segmentSize-floor(diff/2))));
                corr_3D_matrix(:,j,g+1) = corr;
            end
        end
    else
        diff = numPointsCorr - segmentSize;
        number_of_segments = floor((size(data_matched_echo,1)-diff)/segmentSize);
        corr_3D_matrix = zeros(2*numPointsCorr-1,size(data_matched_echo,2)-1,number_of_segments);
        for j = 1:size(data_matched_echo,2)-1
            signal2 = data_matched_echo(:,j+1);
            signal1 = data_matched_echo(:,j);
            for g = 0:number_of_segments-1
                corr = xcorr(signal2((1+g*segmentSize):(numPointsCorr+g*segmentSize)),...
                    signal1((1+g*segmentSize):(numPointsCorr+g*segmentSize)));
                corr_3D_matrix(:,j,g+1) = corr;
            end
        end
    end
    % Calculate all the velocities:
    timeResolution = (size(data_matched_echo,2)-1)/columnsOverlaped;
    velocity_matriz = zeros(number_of_segments,timeResolution);
    for j = 1 : number_of_segments
        for i = 1 : timeResolution
            mean_corr = sum(corr_3D_matrix(:,1+(i-1)*columnsOverlaped:i*columnsOverlaped,j),2)/
    columnsOverlaped;
            velocity_matriz(j,i) = velocityEstimator(mean_corr,fs,c,T_prf,velocityRange);
        end
    end
end
```

Listing 4: crossCorrelationVariableSpan.m

```
function [velocity] = velocityEstimator(crossCorrelation_vector,fs,c,T_prf,velocityRange)
%// Method that calculates the velocity from one cross-correlation vector
%   by looking for the maximum value in the vector.
% ------------ Input parameters:
% - crossCorrelation_vector -> vector of cross-correlation
% - fs -> sampling frequency
% - c -> velocity of propagation in the mean
% - T_prf -> period of pulse repetition frequency
% - velocityRange -> maximum velocity in absolut value that the system can
%                    detect
% ------------ Output paramenters:
% - velocity -> estimation of the velocity from the cross-correlation

    if(max(crossCorrelation_vector) ~= 0 || min(crossCorrelation_vector) ~= 0)
        % Generate a mask for the velocity range that we are looking for:
        mask = zeros(1,length(crossCorrelation_vector));
        centre_Value_Mask = round(length(mask)/2);
        range_points_shifted = round((velocityRange*2*fs*T_prf)/c);
        if (centre_Value_Mask - 1 < range_points_shifted)
            % If the range of velocities is bigger than the mask generated
            mask = ones(1,length(crossCorrelation_vector));
        else
            mask_in_range = ones(1,range_points_shifted);
            mask(centre_Value_Mask) = 1;
            mask(centre_Value_Mask + 1:centre_Value_Mask + length(mask_in_range)) = mask_in_range;
            mask(centre_Value_Mask - length(mask_in_range):centre_Value_Mask - 1) = mask_in_range;
        end
        % Apply mask:
        crossCorrelation_vector_cut = crossCorrelation_vector.*mask';
        % Vector that centers the cross-correlation in the origin:
        x_n = -(length(crossCorrelation_vector)-1)/2:(length(crossCorrelation_vector)-1)/2;
        [~,ind] = max(crossCorrelation_vector_cut); % index of the maximum value in the cross-correlation
        points_shifted = x_n(ind);
        % Conversion from points to time:
        t_shift = points_shifted/fs;
        if (ind == 1)
            % In case after applying the maks there is a vertor of 0
            velocity = 0;
        else
            velocity = (t_shift*c)/(2*T_prf);
        end
    else
        % In case the cross-correlation is a vector of 0
        velocity = 0;
    end
end
```

Listing 5: velocityEstimator.m

```
function [snr,snr_dB] = calculateSNR(originalSignal,noiseSignal,isFilter)
%// Method that calculates the signal to noise ratio.
% ------------ Input parameters:
% - originalSignal -> signal without noise
% - noiseSignal -> signal with noise or only noise
% - isFilter -> boolean that defines if the 'noiseSignal' is signal with noise or only noise
% ------------ Output paramenters:
% - snr -> signal-to-noise ratio
% - snr_dB -> signal-to-noise ratio in dB

    if (isFilter)
        expected_signal = mean(originalSignal.^2);
        expected_noise = mean(noiseSignal.^2);
    else
        noise = noiseSignal - originalSignal;
        expected_signal = mean(originalSignal.^2);
        expected_noise = mean(noise.^2);
    end
    snr = expected_signal./expected_noise;
    snr_dB = 10*log10(snr);
```

```
end
```

Listing 6: calculateSNR.m

2 Matlab running scripts

```
%% SETUP of script for in vivo data:
close all; clear all; clc;

load('cfm_carotis.mat');

%% VARIABLES:
% FIXED PARAMETERS:
fs = 40*10^6; % 40 MHz (sampling frequency)
Resolution = 16; % 16 bits samples
c = 1540; % 1540 m/s
f_prf = 6*10^3; % 6 kHz
T_prf = 1/f_prf;
f0 = 5*10^6; % 5 MHz (center frequency)
focus_depth = 18*10^(-3); % 18 mm
cycles = 8; % 8 cycles in one pulse
start_depth = 1; % 1 mm
end_depth = 30; % 30 mm
% Dimensions left to right:
x_min = -9.75; % -9.75 mm
x_max = 9.75; % 9.75 mm

% VARIABLE PARAMETERS:
segmentSize = 10; % 10
numPointsCorr = 38; % 10
columnsOverlaped = 9; % It can be: 1, 3, 7, 9 ,21 and 63
velocityRange = 1; % +-1 [m/s] of velocity range in the Carotid

%% CLACULATIONS:

% CALCULATE VELOCITIES:
velocity_matrix = [];
for j = 1:size(vessel,2)
    data = double(rf_cfm_data(:,:,j)).*vessel(:,j);
    velocity_matrix_j = mainFunction(data,fs,f0,cycles,c,T_prf,segmentSize,numPointsCorr,velocityRange,
    columnsOverlaped);
    velocity_matrix = [velocity_matrix velocity_matrix_j];
end

% CALCULATE B-MODE:
bmode_data(:,end) = [];
Hilb_bmode=hilbert(bmode_data); % hilbert matrix
env_bmode = abs(Hilb_bmode); % signal envelope through hilbert transformation
env_db_bmode=20*log10(env_bmode/max(max(env_bmode)));
E_dB_60_bmode=env_db_bmode+60; %previous values added 60
% We delete the values under the bottom 60 dB range.
E_dB_60_bmode(E_dB_60_bmode<0) = 0; %all the negative values become 0

%% PREPARE RESULTS FOR PLOTING:

% Interpolate Y-axis from velocity matrix:
interpFactor = floor(1520/size(velocity_matrix,1));
newVelocityMatrix = zeros(size(velocity_matrix,1)*interpFactor,size(velocity_matrix,2));
for i = 1:size(velocity_matrix,2)
    newVelocityMatrix(:,i) = interp(velocity_matrix(:,i),interpFactor);
end

% Interpolate X-axis from velocity matrix (for 9 columns overlaped the factor is: 4/7 -> 112(columns)
    *(4/7) = 64:
interpFactorX = 4;
dicimateX = 7; %7
velocity_matrix_interp = zeros(size(newVelocityMatrix,1),size(newVelocityMatrix,2)*interpFactorX/dicimateX
    );
for i = 1:size(newVelocityMatrix,1)
    aux = interp(newVelocityMatrix(i,:),interpFactorX);
    velocity_matrix_interp(i,:) = decimate(aux,dicimateX);
end

if(size(velocity_matrix_interp,1) ~= size(E_dB_60_bmode,1))
    diff_lines = size(E_dB_60_bmode,1) - size(velocity_matrix_interp,1);
    array = zeros(diff_lines/2,size(velocity_matrix_interp,2));
    velocity_matrix_interp = [array; velocity_matrix_interp; array];
end

%% PREPARE MASK FOR OVERLATING B-MODE and VELOCITIES IN ONE PLOT:

% Interpolate original Mask (vessel mask has 16 columns and we need 64 -> factor of 4):
interpFactorMask = 4;
mask_interpolated = zeros(size(vessel,1),size(vessel,2)*interpFactorMask);
for i = 1:size(vessel,1)
    mask_interpolated(i,:) = round(interp(vessel(i,:),interpFactorMask));
end
mask_interpolated(mask_interpolated > 1) = 1;
mask_interpolated(mask_interpolated < 0) = 0;

%% MERGE B-MODE AND VELOCITIES:
% NORMALIZE VALUES FOR B-MODE:
% NewValue = (((OldValue - OldMin) * (NewMax - NewMin)) / (OldMax - OldMin)) + NewMin
E_dB_60_bmode_color=(((E_dB_60_bmode - 0)*(0.5-(-0.5)))/(60-0))+(-0.5);

% MERGE IMAGES:
```

```matlab
merged_matrix = zeros(size(mask_interpolated,1),size(mask_interpolated,2));
for i = 1:size(merged_matrix,1)
    for j = 1: size(merged_matrix,2)
        if (mask_interpolated(i,j) == 0)
            merged_matrix(i,j) = E_dB_60_bmode_color(i,j);
        else
            merged_matrix(i,j) = velocity_matrix_interp(i,j);
        end
    end
end

%% PLOTS:

% CREATE COLORMAP:
jetLine = jet(256);
hotLine = hot(180);
colorMap = zeros(256,3);
colorMap(1:128,:) = flip(jetLine(1:128,:));
colorMap(129:end,:) = hotLine(1:128,:);

figure;
imagesc([x_min x_max],[start_depth end_depth],E_dB_60_bmode_color,'AlphaData',(-1)*(mask_interpolated-1)
    ,[-0.5 0.5]);
colorbar;
ylim([start_depth 28.84])
colormap('gray');
xlabel('Lateral distance [mm]');
ylabel('Axial distance [mm]');
title('B-mode');

figure;
imagesc([x_min x_max],[start_depth end_depth],velocity_matrix_interp,'AlphaData',mask_interpolated,[-0.5
    0.5]);
colorbar;
ylim([start_depth 28.84])
colormap(colorMap);
xlabel('Lateral distance [mm]');
ylabel('Axial distance [mm]');
title('Velocity map of the carotid artery')



figure;
ax1 = axes;
A = imagesc([x_min x_max],[start_depth end_depth],E_dB_60_bmode_color,[-0.5 0.5]);
ylim([start_depth 28.84])
colorbar
ax2 = axes;
B = imagesc([x_min x_max],[start_depth end_depth],velocity_matrix_interp,'AlphaData',mask_interpolated
    ,[-0.5 0.5]);
ylim([start_depth 28.84])
linkaxes([ax1,ax2]);
ax2.Visible = 'off';
ax2.XTick = [];
ax2.YTick = [];
colormap(ax1,'gray');
colormap(ax2,colorMap);
set(ax2,'color','none','visible','off');
colorbar
title('Color flow map from carotid artery')
xlabel('Lateral distance [mm]');
ylabel('Axial distance [mm]');

% Velocity profile:
figure;
plot(linspace(start_depth,end_depth,length(velocity_matrix_interp(:,50))),velocity_matrix_interp(:,50))
xlim([start_depth end_depth])
title('Velocity profile at a lateral distance of 5.14 [mm]')
xlabel('Axial distance [mm]');
ylabel('Velocity [m/s]');
```

Listing 7: mainRealData.m

```matlab
% Script to run all the simulations:

close all; clear all; clc;

%% PREPARE SIMULATION:

% PULSE:
load pulse.mat
t = 1/fs:1/fs:(1/fs*length(pulse)); % time axis of the pulse
pulse_freq = fft(hilbert(pulse));
pulse_freq_abs = abs(pulse_freq);
freq_axis = linspace(0,fs,length(pulse_freq_abs));
% Centre frequency of the transducer:
[~, ind] = max(pulse_freq_abs);
f0 = freq_axis(ind);

% SCATTERING IN THE VESSEL:
d = 10*10^-3; % vessel diameter
ang = pi/4; % angle of the transducer
c = 1500; % m/s
lambda = c/fs; % wave length
depth = sqrt(2*d^2); % depth in the vessel
nsamples = round(depth/lambda); % 943 wavelengths -> penetration in the vessel

% SIMULATION OF SCATTERING:
```

```
scatter = randn(nsamples,1); % scatter (column vector 943x1)
rf_signal = conv(pulse,scatter); % RF signal
time_rf_signal = 0:1/fs:1/fs*(length(rf_signal)-1); % time axis of RF signal
rf_signal_spectrum = fft(hilbert(rf_signal)); % Spectrum of RF signal
freq_rf_signal = linspace(0,fs,length(rf_signal_spectrum)); % frequency axis of the spectrum

% SIMULATION OF PLUG FLOW WITH 100 SIGNALS RECEIVED (contant velocity):
num_signals = 100; % number of signals received
vz = 0.15; % velocity of the flow in the vessel
fprf = 5*10^3; % pulse repetition frequnecy
Tprf = 1/fprf; % period of pulse repition frequency
timeShift = 2*vz*Tprf/c; % time shift generated in the pulse by the velocity of the blood

% PLOTS:
%Plot pulse:
figure();
subplot(1,2,1);
plot(t,pulse)
xlabel("Time [s]")
ylabel("Voltage [V]")
title("Pulse")
% Plot spectrum of the pulse:
subplot(1,2,2);
plot(freq_axis,pulse_freq_abs)
xlabel("Frequency [Hz]")
title("Pulse spectrum")
ylabel("Power [W]")
xlim([0 3e7])

signal = delayseq(rf_signal,(1:num_signals)*timeShift,fs); % signals received
figure();
imagesc([0 Tprf*num_signals],[0 depth],signal);
colormap (hot)
colorbar
title("Emitted signal")
xlabel("Time [s]")
ylabel("Depth [m]")

figure();
plot(time_rf_signal,rf_signal)
xlabel("Time [s]")
title("Model of RF signal behaviour in the blood")
ylabel("Voltage [V]")

%% ECHO CANCELLING TEST:
% Prepare signals with white noise and stationary signals:
stationary_echo = 20*randn(length(rf_signal),1); % between 10 and 100 times
noise_signal = signal + stationary_echo;
for i = 1:num_signals
    white_noise = randn(length(rf_signal),1);
    noise_signal(:,i) = noise_signal(:,i) + white_noise;
end
% Cancel echo:
signalRecovered = echoCancelling (noise_signal);

% Calculate results:
signal_index = 20;
[snr_before_echo_filter,snr_dB_before_echo_filter] = calculateSNR(signal,noise_signal,0);
[snr_after_echo_filter,snr_dB_after_echo_filter] = calculateSNR(signal,signalRecovered,0);
snr_echo_improve = snr_dB_after_echo_filter - snr_dB_before_echo_filter;

g = [repmat({'SNR dB before filter'},100,1); repmat({'SNR dB after filter'},100,1); repmat({'SNR imporve'
    },100,1)];
figure;
boxplot([snr_dB_before_echo_filter'; snr_dB_after_echo_filter'; snr_echo_improve'],g,'color','brg')
ylabel("SNR [dB]")
title("SNR before and afer echo filter")

% Plot results:
figure;
subplot(1,2,1);
imagesc([0 Tprf*num_signals],[0 depth],noise_signal);
colormap (hot)
colorbar
title("Signal received with noise")
xlabel("Time [s]")
ylabel("Depth [m]")

subplot(1,2,2);
imagesc([0 Tprf*num_signals],[0 depth],signalRecovered);
colormap (hot)
colorbar
title("Signal after echo cancelling")
xlabel("Time [s]")
ylabel("Depth [m]")

% --------- Represent one line (pulse)----------
% Time domain:
num = 20;
figure;
subplot(1,2,1);
plot(time_rf_signal, signal(:,num), 'b')
hold on
plot(time_rf_signal, noise_signal(:,num), 'g')
plot(time_rf_signal, signalRecovered(:,num), 'r')
title("Time domain")
xlabel("Time [s]")
ylabel("Voltage [V]")
```

```
xlim([0.2e-5 0.45e-5])
legend('Original signal','Signal received','Signal filtered')
% Frequncy domain:
freq = linspace(0,fs,length(signal(:,num)));
subplot(1,2,2);
plot(freq, abs(fft(hilbert(signal(:,num)))), 'b')
hold on
plot(freq, abs(fft(hilbert(noise_signal(:,num)))), 'g')
plot(freq, abs(fft(hilbert(signalRecovered(:,num)))), 'r')
title("Frequency domain")
xlabel("Frequency [Hz]")
ylabel("Power [W]")
xlim([0 1e7])
legend('Original signal','Signal received','Signal filtered')


%% MATCHED FILTER TEST:
% Prepare signals with white noise and stationary signals:
whiteNoise_signal = signal;
whiteNoise_matrix = zeros(size(signal,1),size(signal,2));
for i = 1:num_signals
    white_noise = randn(length(rf_signal),1);
    whiteNoise_signal(:,i) = whiteNoise_signal(:,i) + white_noise;
    whiteNoise_matrix (:,i) = white_noise;
end

% Generate tranfer funtion of the filter:
h_filter = flip(pulse);
% Filter the data:
data_matched = filtfilt(h_filter,1,whiteNoise_signal);
noise_matched = filtfilt(h_filter,1,whiteNoise_matrix); % filt the noise
signal_matched = filtfilt(h_filter,1,signal); % filt the original signal

% Calculate results:
[snr_before_matched_filter,snr_dB_before_matched_filter] = calculateSNR(signal,whiteNoise_matrix,1);
[snr_after_matched_filter,snr_dB_after_matched_filter] = calculateSNR(signal_matched,noise_matched,1);
snr_matched_improve = snr_dB_after_matched_filter - snr_dB_before_matched_filter;

gMatch = [repmat({'SNR dB before filter'},100,1); repmat({'SNR dB after filter'},100,1); repmat({'SNR
    imporve'},100,1)];
figure;
boxplot([snr_dB_before_matched_filter'; snr_dB_after_matched_filter'; snr_matched_improve'],gMatch,'color'
    ,'brg')
ylabel("SNR [dB]")
title("SNR before and afer matched filter")

% Plot results:
figure;
subplot(1,2,1);
imagesc([0 Tprf*num_signals],[0 depth],whiteNoise_signal);
colormap (hot)
colorbar
title("Signal received with noise")
xlabel("Time [s]")
ylabel("Depth [m]")

subplot(1,2,2);
imagesc([0 Tprf*num_signals],[0 depth],data_matched);
colormap (hot)
colorbar
title("Signal after matched filter")
xlabel("Time [s]")
ylabel("Depth [m]")

% --------- Represent one line (pulse)----------
% Time domain:
num = 20;
figure;
subplot(1,2,1);
plot(time_rf_signal, signal(:,num)/max(signal(:,num)), 'b')
hold on
plot(time_rf_signal, whiteNoise_signal(:,num)/max(whiteNoise_signal(:,num)), 'g')
plot(time_rf_signal, data_matched(:,num)/max(data_matched(:,num)), 'r')
title("Time domain")
xlabel("Time [s]")
ylabel("Voltage [V]")
xlim([0.2e-5 0.45e-5])
legend('Original signal','Signal received','Signal filtered')
% Frequncy domain:
freq = linspace(-fs/2,fs/2,length(signal(:,num)));
subplot(1,2,2);
plot(freq, abs(fftshift(fft(signal(:,num))))/max(abs(fftshift(fft(signal(:,num))))), 'b')
hold on
plot(freq, abs(fftshift(fft(whiteNoise_signal(:,num))))/max(abs(fftshift(fft(whiteNoise_signal(:,num))))),
    'g')
plot(freq, abs(fftshift(fft(data_matched(:,num))))/max(abs(fftshift(fft(data_matched(:,num))))), 'r')
title("Frequency domain")
xlabel("Frequency [Hz]")
ylabel("Power [W]")
xlim([0 2e7])
legend('Original signal','Signal received','Signal filtered')


%% VELOCITY ESTIMATOR TEST:
segmentSize = 10;
numPointsCorr = 100;
velocityRange = 1;
columnsOverlaped = 1;
```

```

velocity_matrix = crossCorrelationVariableSpan(signal,segmentSize,numPointsCorr,fs,c,Tprf,velocityRange,
    columnsOverlaped);

figure;
imagesc([0 Tprf*num_signals],[0 depth],velocity_matrix);
colormap (jet)
colorbar
title("")
xlabel("Time [s]")
ylabel("Depth [m]")


figure;
plot(linspace(0,depth,length(velocity_matrix(:,10))),velocity_matrix(:,10))
title("Velocity profile (Segment: 10; Span: 100; Line Overlap: 1)")
xlabel("Depth [m]")
ylabel("Velocity [m/s]")

% probability of detection:
figure;
hist_velocity = histogram(velocity_matrix,15);
ylabel("Number of points")
xlabel("Velocity [m/s]")

detectionProbability = max(hist_velocity.Values)/sum(hist_velocity.Values)

%% VELOCITY DETECTION SYSTEM TEST:
segmentSize = 5;
numPointsCorr = 126;
velocityRange = 1;
columnsOverlaped = 3;

velocity_detected = mainFunction(noise_signal,fs,f0,pulse,c,Tprf,segmentSize,numPointsCorr,velocityRange,
    columnsOverlaped);

figure;
imagesc([0 Tprf*num_signals],[0 depth],velocity_detected);
colormap (jet)
colorbar
title("Velocity map (Probability of detection: 64.03%)")
xlabel("Time [s]")
ylabel("Depth [m]")


figure;
plot(linspace(0,depth,length(velocity_detected(:,14))),velocity_detected(:,14))
title("Velocity profile (Segment: 5; Span: 126; Line Overlap: 3)")
xlabel("Depth [m]")
ylabel("Velocity [m/s]")

% probability of detection:
figure;
hist_velocity_system = histogram(velocity_matrix,15);
ylabel("Number of points")
xlabel("Velocity [m/s]")

detectionProbabilitySystem = max(hist_velocity_system.Values)/sum(hist_velocity_system.Values)
```

Listing 8: main.m