Variable Delay Line Algorithm for Simulating the Doppler Effect Created by a Moving Object and Estimating the TDOA

2.1 Introduction

The aim of this code is to develop an algorithm that separates the sound generated by the individual sources with proper processing at the array. Three following scenario have been considered:

- Signal from rotating source and time of arrival (TOA), or difference time of arrival (DTOA)

2.1 Signal from rotating source and TOA/DTOA

In this scenario $N = 2$ that placed in the center of the x, y axis in position (-d/2,0) and (d/2,0), for d=18cm. Source of signal is rotating in a circle with radius r_0 =3m centered in (0,0) with an angular speed $\omega = \pi$ rad/s , and the signals $s1(t)$ at microphone 1 in (-d/2,0) and $s2(t)$ at microphone 2 in (d/2,0) are received. Fig. 1 shows the scenario in detail.

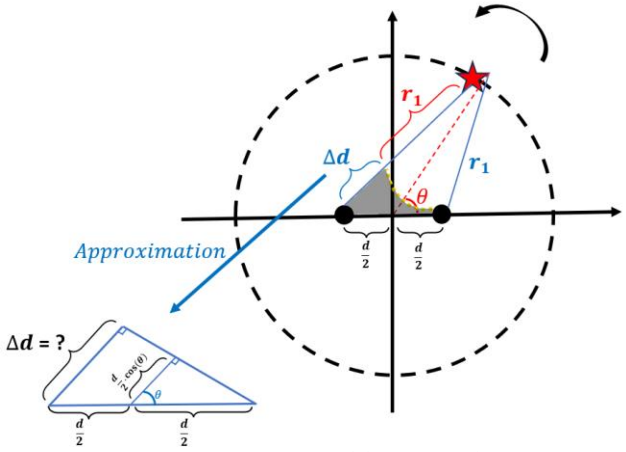

Fig. 1 scenario of the Homework

2.1.1 Simulation of the Received Signals in Microphones

the signals $s_1(t)$ at microphone 1 in (-d/2,0) and $s_2(t)$ at microphone 2 in (d/2,0) recalling that propagation is affected by a delay that depends on the distance source-microphone r and the attenuation $\alpha(r)$ is $\alpha(r) = (r/r_0)^2$. Signals are sampled, with frequency fs (or period $T = 1/fs$). To generate the signals at each of the microphone, first the delays from the source to each of the microphones should be calculated according to distance between the source and the microphones (τ_1, τ_2). The delay is based on the following formula:

$$\tau_{1,2} = \frac{1}{v}\Delta d = \frac{1}{v}\sqrt{\left(r\cos\theta \pm \frac{d}{2}\right)^2 + (r\sin\theta)^2} \qquad (2.1)$$

where r, and θ are time varying due to the source rotation. The following code is used to calculate the TOA in each microphone.

```
par.d = 0.18 ;                              % location of the Microphones
par.omega = pi/10;                             % w = pi [rad/s]
par.r0 = 3 ;                               % 3[m]
par.N  = 2;
par.v = 335;                                        % speed of wavefront
t = (0:Ns-1)./fs;                                   % time axis
par.x = par.r0 .* cos(par.omega * t);               % x speaker
par.y = par.r0 .* sin(par.omega * t);               % y speaker
theta_true = linspace (0 ,2*pi , Ns);               % tetha speaker
```

```
r1  =  sqrt( par.y.^2 + (par.x + par.d/2).^2 ) ;   % r1
r2  =  sqrt( par.y.^2 + (par.x - par.d/2).^2 ) ;   % r2
DTOA_true = (cos( theta_true ).* par.d ./ par.v)';
delayl = (r1./ par.v ).*fs;
delayr = (r2./ par.v ).*fs;
plot(d_sample_l)
```

To delay music, the variable delay line is used to introduce a time delay between its input and output [1]. As shown in Figure.2, a discrete x[n] is applied to a delayed system with a delay of M.

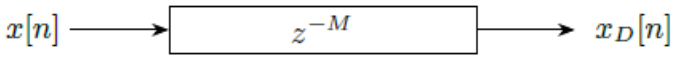

Fig 2. variable time delay

Therefore, the output of the system can be written as follows [1]:

$$x_D[n] = x[n - M] \qquad (2.2)$$

which is caused by the rotating acoustic source on a circle by using different values of M. Our target is to write the input signal to a buffer sample by sample. We consider the length of buffer as N, and it should be larger than any delay $M \leq N$. In this buffer, two types of pointers can be used. one for reading the samples and the other one is to write in the buffer. As it is shown in Fig. 3, in each window, first, we read the sample by read pointer. Then, shift right it by M sample. Finally, write to the buffer with a write pointer. Using this method enables us to simulate the doppler shift effect cause by the moving source.

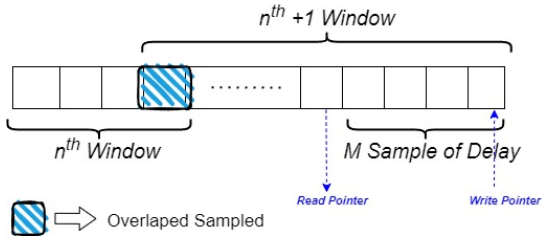

Fig 3. variable delay line algorithm explained.

Delay lines generate precise time delays, enabling the modeling of acoustic waves propagating in a specific direction with a consistent wave shape. The first step is to find the M as the number of samples to be shifted. Interpolation was also exploited to implement the discrete value of delay in each sample. The Following function implemented this method:

```
function [s_rot] = Variable_Delay(s,d_sample)
rpt = 1;
Ns = length(s);
clear buff nanIndices nonNaNIndices
for i = 1:length(1:Ns)
    buff(i) = s(i);
    if i == 1
        if d_sample(i) == 0
            s_rot(i) = buff(i);
            rpt = rpt+1;
            continue;
        else
            s_rot(i) = NaN;
            continue;
        end
    end
    if d_sample(i) == 0 && d_sample(i-1)== 0
```

```
        s_rot(i) = buff(i);
        rpt=rpt+1;
        continue;
    end
    if (i - rpt) ~= d_sample(i)
        if d_sample(i) > d_sample(i-1) || d_sample(i) == d_sample(i-1)
            s_rot(i)= NaN;
            continue;
        elseif (d_sample(i) < d_sample(i-1))
            rpt = rpt + 1;
        end
    end
    if (i - rpt) == d_sample(i) && i>1
        if d_sample(i)== 0 && d_sample(i-1) == 1
            rpt = rpt-1;
            s_rot(i) = buff(rpt);
            rpt = rpt+1;
        else
            rpt = rpt +1;
            s_rot(i) = buff(rpt);
        end
    end
end
s_rot = fillmissing(s_rot , 'linear');
nanIndices = isnan(s_rot);
nonNaNIndices = find(~isnan(s_rot));
s_rot(nanIndices) = interp1(nonNaNIndices, s_rot(nonNaNIndices), find(nanIndices), 'linear');
end
```

The output of this method after the delay line is as Fig. 2, where the received signal after attenuating in each microphone is illustrated. The obtained signal also makes the listener feel that the speaker is rotating around the head.

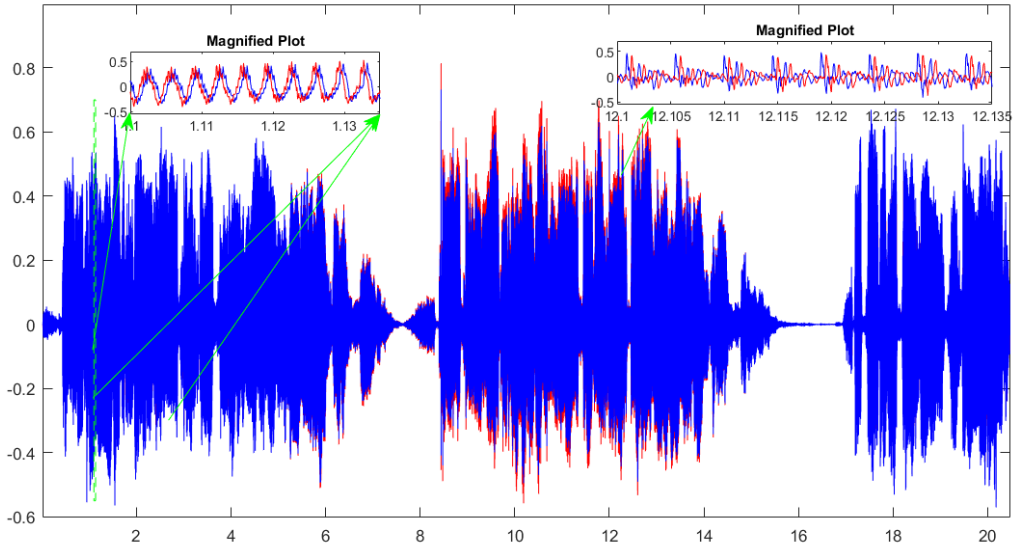

Fig 4. simulated signal at each microphone vs time (s).

a. TDOA Estimation

In order to estimate the DTOA (Δτ) between the received signals in two microphones, the correlation method, which is based on the maximum likelihood estimator (MLE) is implemented. The estimator is formulated as follows [2].[3]:

$$\Delta\hat{\tau} = arg \underbrace{max}_{|\xi|<T/2} \left\{ \int_{-\frac{T}{2}}^{+T/2} s_1(t) s_2(t+\xi)\, dt \right\} \qquad (2.3)$$

where both signals are wideband in this scenario. The parameter T is an indication of the windows length within which the cross correlation is computed. In this scenario, different window lengths are considered and mean square error (MSE) is computed for each choice then the best window length is selected. The Cramer-Rao bound (CRB) in TDOA is [2]:

$$Var[\hat{\Delta\tau}] = \frac{3}{\pi^2} \cdot \frac{1+2\rho}{\rho^2} \cdot \frac{1}{N} \qquad (2.4)$$

where the ρ is the power of the signal to noise ratio within the signal bandwidth. Quadratic interpolation is exploited to find the maximum of the cross-correlation function. The code for this section is as follows.

```
Windows =[50, 100, 200 ,500 ,1000 ,1500, 5000];           % different window length
DTOA_est = zeros( Ns , length(Windows) );
MSE = zeros(1, length(Windows));
for w = 1:length(Windows)               % loop over different windows length
    w_len = Windows(w);
    N = round( Ns / w_len);
    tau_est = zeros(w_len, 1);
    for k = 1:w_len                          % loop over w_len
        sliced1 = y_noisy((k - 1) * N + 1 : k * N,end-1);
        sliced2 = y_noisy((k - 1) * N + 1 : k * N,end);
        convolution = conv(sliced1, sliced2(end : -1:1));
        [~, mind] = max(convolution);
        if mind == 1 || mind == 2*N - 1
            tau_est(k) = NaN;
        else                                                                % quadratic interpolation
            delta = 0.5 * (convolution(mind - 1) - convolution(mind + 1)) / (convolution(mind - 1) +
convolution(mind + 1) - 2 * convolution(mind));
            tau_est(k) = (delta + mind - N) / fs;
        end
    end
    DTOA_est(:, w) = interp1(1:w_len, tau_est, linspace(1, w_len, Ns));
    err = DTOA_est(:, w) - DTOA_true;
    err(isnan(err)) = [];
    MSE(w) = err' * err / w_len;
end
[~, best] = min(MSE);                                                       % choose the best block size
```

iteration for different SNRs with proper choice of the Window

```
DTOA_est_SNR = zeros( Ns , Noise.numSNRLevels );
MSE_SNR = zeros(1, Noise.numSNRLevels);
w_len = Windows(best);
N = round( Ns / w_len);
for  i =1:Noise.numSNRLevels
    tau_est = zeros(w_len, 1);

    for k = 1:w_len                          % loop over w_len
        sliced1 = y_noisy((k - 1) * N + 1 : k * N, (2*i) - 1);
        sliced2 = y_noisy((k - 1) * N + 1 : k * N, 2*i );
        convolution = conv(sliced1, sliced2(end : -1:1));
        [~, mind] = max(convolution);
        % quadratic interpolation
        if mind == 1 || mind == 2*N - 1
            tau_est(k) = NaN;
        else
            delta = 0.5 * (convolution(mind - 1) - convolution(mind + 1)) / (convolution(mind - 1) +
```

```
convolution(mind + 1) - 2 * convolution(mind));
            tau_est(k) = (delta + mind - N) / fs;
        end
    end
    DTOA_est_SNR(:, i) = interp1(1:w_len, tau_est, linspace(1, w_len, Ns));
    err = DTOA_est_SNR(:, i) - DTOA_true;
    err(isnan(err)) = [];
    MSE_SNR(i) = err' * err / w_len;
end
```

The results for the choosing time- window and corresponding DTOA estimation are shown in Fig. 5. Meanwhile, the results for TDOA estimation for different SNR levels in illustrated in Fig. 6.

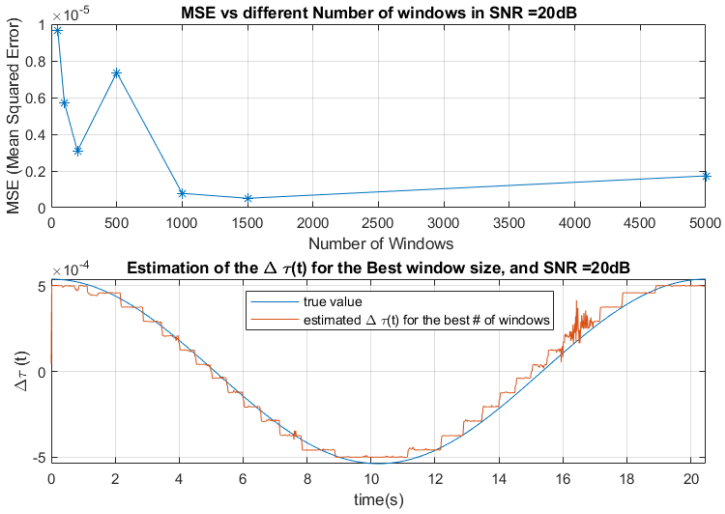

Fig 5. MSE vs number of windows (up), and estimation of the DTOA for the best-chosen window length in SNR = 20dB.

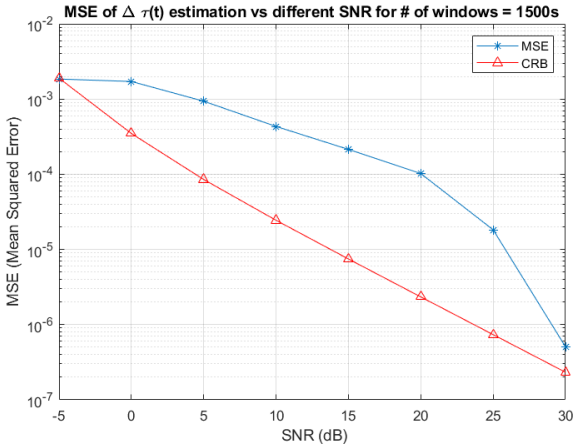

Fig 6. MSE for DTOA vs SNR for the best-chosen number of windows (blue), and CRB (red)

b. Signal Realignment

In this section using the TDOA estimated in the previous section is used for realignment of the received signal in order to mitigate the effect of rotation. The problem can be formulated as follows:

$$\begin{cases}\hat{R}_1 = r_1\left(t+\alpha_1(t)\right)\\ \hat{R}_2 = r_2\left(t+\alpha_2(t)\right)\end{cases} \xrightarrow{r_1\ as\ time\ reference} \begin{cases}\widehat{R_1} = r_1(t)\\ \hat{R}_2 = r_2\left(t+\hat{\Delta}\tau(t)\right)\end{cases} \quad (2.5)$$

Then the realignment is implemented using the same function for delay-line implementation as in the previous section. This would cause the listener to perceive the source of music in front as both signals have the same relative delay (ideally zero) with respect to each other. The output of this part is shown in Fig. 7.

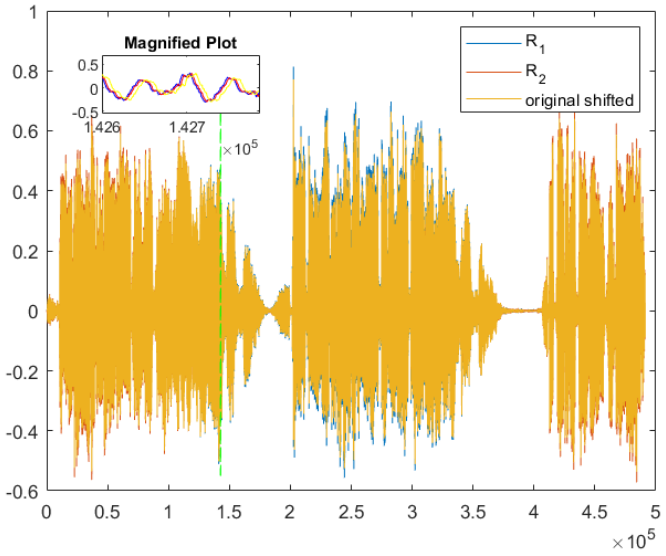

Fig 7. Result fir realignment of the signals compared with delayed original signal properly.

c. Signal Reconstruction

To reconstruct the original signal the following formula is used:

$$\hat{R}(t) = \frac{1}{2}\left(\hat{R}_1(t) + \hat{R}_2(t)\right) \quad (2.6)$$

Then MSE between the original signal, which shifted with the minimum delay value ($\tau_{min} = r - d/v$) and $\widehat{R}(t)$ is calculated using the following code. As shown in fig.7 the two signals are approximately aligned and after summing, they would resemble the original signal. The MSE in this case reported as 0.0074822. Fig. 8 shows the realignment MSE for different values of the SNR. Finally, the MSE of the TOA estimated vs different SNR is illustrated in Fig. 9. As was expected, higher SNR values make the MSE of the estimation lower.

```
R_1 = Variable_Delay(y_noisy(:,end-1),a_1);
R_2 = Variable_Delay(y_noisy(:,end),a_2);
```

```
plot(R_1);hold on;
plot(R_2);
hold on;
pp =[zeros(max(a_1),1);s1];
plot(pp)
R_hat =  0.5.*R_1 + 0.5.*R_2;
mse_result = calculate_mse(R_hat, pp');
disp(['Mean Squared Error: ', num2str(mse_result)]);
```

```
Mean Squared Error: 0.0074822
```

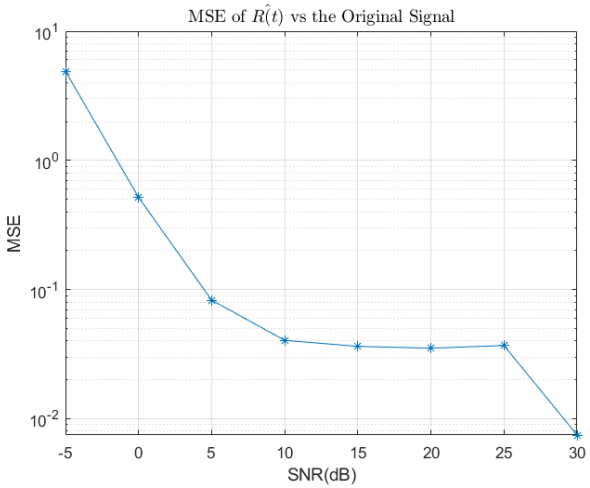

Fig 8. MSE vs SNR for MSE between R(t) and original signal

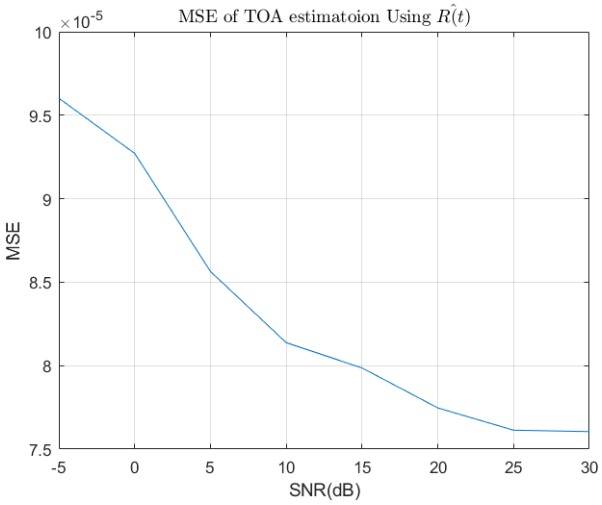

Fig 9. MSE vs SNR for TOA estimation.

References

[1] S. Damiano and T. van Waterschoot, *"Pyroadacoustics: A Road Acoustics Simulator Based on Variable Length Delay Lines,"* DAFx-2022, Vienna.

[2] Spagnolini, Umberto. *Statistical Signal Processing in Engineering*. John Wiley & Sons, 2018.

[3] Kay, Steven M. *Fundamentals of statistical signal processing: estimation theory*. Prentice-Hall, Inc., 1993.