EE4C03 STATISTICAL DIGITAL SIGNAL PROCESSING ASSIGNMENT

Erik Hagenaars
4272404

Sander van Katwijk
4334264

***Abstract*—An indoor, line of sight and robust localization algorithm using a microphone array in square arrangement was developed to localize and track a toy car in an indoor environment. Using an audio beacon on the car and channel estimation and peak finding algorithms, the TDOA can be obtained from the input of the microphones. With this TDOA, a location estimate is made. Incorrect measurements or outliers are discarded using a static algorithm, and a path prediction algorithm maps the actual route of the target. The results show that due to modeling issues and very noisy data, the channel estimation algorithm is unable to correctly estimate the channel, resulting in location estimates which are completely wrong. However, using simulated TDOA values, location estimation and path prediction are shown to be effective. It is concluded that the channel estimation and peak finding algorithm work according to theoretical research and as feedback, an improved measurement environment is recommended.**

***Keywords*—Indoor localization, TDOA, optimal filtering, adaptive filters**

I. Introduction

Localization is enormously important for many applications. Outdoor localization can easily be performed using commercially available GPS devices. However, indoor localization is still a difficult task due to the No Line Of Sight (NLOS), noisy and multipath situations often found in the dense areas often found indoors.

Localization can be done using acoustic signals in an indoor environment. For this, at least three microphones are required to be able to locate an track a target. To reduce the impact of multipath, noise and interference, a microphone array setup consisting of five microphones is used. The microphones are set up at known locations. Assuming Line Of Sight (LOS) properties of the signal, it is possible to determine the location of the signal source. Which in this case is a toy car with a mounted audio beacon.

The purpose of this research is to design and develop a robust localization algorithm. First, the problem is described in detail in section II. Next, the algorithm used to convert the microphone measurements into a path is detailed. First, the arrivals of the signal must be determined using a channel estimate, which is then turned into a set of Time-Difference of Arrival (TDOA) values as described in section III. This set of TDOAs is then converted into a set of locations and then into a path using the methods outlined in IV. Finally the results are discussed in section V. Finally, the MATLAB is available in the appendix.

II. Problem Description

The supplied problem consists of a toy car travelling inside an approximately 6 by 6 room. The room has a microphone in each corner and an additional fifth microphone in the center of one wall. The car is equipped with an audio beacon which transmits an acoustic code. This code is received by the microphones and is to be converted into a location.

To convert the data into locations a few steps are required. Firstly, a channel estimation must be done, which will generate peaks corresponding to reception of the transmitted signal. Secondly, the peaks from the channel estimate must be detected using a peak finding algorithm. Thirdly, the sample indices of the peaks must be converted to a location. Finally, the found locations must be filtered, rejecting outliers and reducing the impact of noise.

These steps are covered in the separate sections of this report. Channel estimation is discussed in section III-A. The peak finder is discussed in Section III-B. Translation of peak indices into locations is covered in Chapter III and IV. Finally, location filtering is discussed in Section IV-B.

III. TDOA estimation

For the TDOA estimation, first a filter needs to be designed in order to created clear peaks at the position of the received signal. The Wiener filter and Matched filter will be considered as filter to create these delta spikes. These methods are described in Section III-A. After filtering, due to channel noise and modeling errors, a perfect delta spike is not present. Therefore, the actual peak must be found using a peak finding algorithm as discussed in Section III-B.

A. Channel Estimation

To reduce the effect of noise and extract the arrival instances of the beacon signal, several different filtering methods are applied. These methods will be discussed here. First, the received signal must be defined mathematically.

To be able to model the signal, the data is analyzed. Fig. 1 shows the signal transmitted by the audio beacon. Fig. 2 shows the signal received by one of the microphones. Comparing these signals, it can be seen that the beacon signal is received twice in the sample signal with a certain amount of noise.

The received signal is now modeled as seen in Eq. 1. Here $d(n)$ is the channel response, $g(n)$ is the signal transmitted by the beacon and $v(n)$ is modelled as additive white Gaussian noise with variance σ_v^2. Eq. 2 shows the structure of the desired signal. It consists of two delta pulses for the two pulses arriving

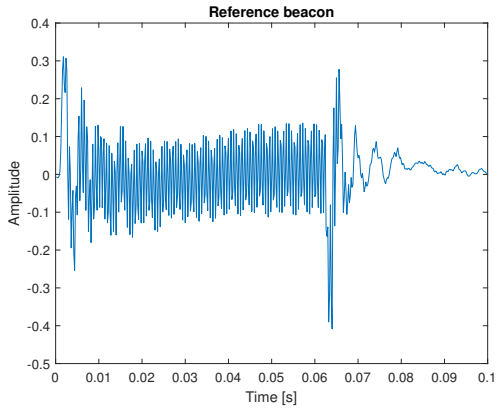

Fig. 1. Time-domain plot of the transmitted beacon signal.

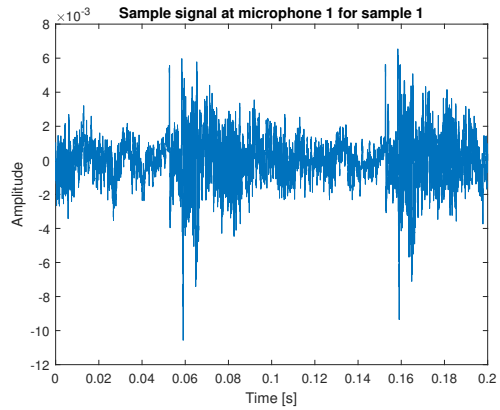

Fig. 2. Time-domain plot of the received signal.

within each measurement interval. The channel also contains multipath effects and other disturbances, which are modelled as additive white Gaussian noise with variance σ_w^2. The two delta pulses are spaced at a distance k, which is determined by the sampling rate and the beacon pulse frequency.

$$s(n) = g(n) * d(n) + v(n) \tag{1}$$

$$d(n) = \delta(n-\alpha) + \delta(n-\alpha-k) + w(n) \tag{2}$$

1) Wiener deconvolution: The first method used is Wiener deconvolution as described in [1, lecture 10]. This model conforms to Eq. 1 but it assumes a noise-free channel $d(n)$.

The signal is modelled as $\vec{x} = G\vec{d} + \vec{v}$, where G is the convolution matrix corresponding to the beacon signal $g(n)$ as shown in Eq. 3. Where L is the desired length of the filter.

$$M = \begin{bmatrix} \text{g(0)} & \dots & \text{g(L)} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \text{g(0)} & \dots & \text{g(L)} \end{bmatrix} \tag{3}$$

This results in the Wiener-Hopf equations as seen in Eq. 4.

$$(G^* \cdot R_d \cdot G^T + R_v)\vec{w} = \vec{r}_{dx} \tag{4}$$

The solution to this set of equations is given in Eq. 5.

$$\vec{w} = (G^* \cdot R_d \cdot G^T + R_v)^\dagger \cdot \vec{r}_{dx} \tag{5}$$

Filter orders of 2 up to 30 were attempted, and a 4th order filter provides the best results. However, even those results are worse than the raw data that was supplied. Because of this, other methods were attempted.

2) Matched filter: The Matched filter is an easy filter method modeled by Eq. 6. Where $h[n]$ will be the reference beacon, $x[n]$ the received signal and $y[n]$ the signal with the delta spikes at the receiving time.

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k] \tag{6}$$

For white noise, the equation for the optimal filter is shown in Eq. 7.

$$h = \alpha R_v^{-1} s \tag{7}$$

$$\alpha = \frac{1}{\sqrt{s^H R_v^{-1} s}} \tag{8}$$

In this equation, R_v^{-1} is the diagonal matrix of σ_v^2 and s the beacon signal. s could also be modeled as two beacon signals since it is present twice in receiving signal of the microphone.The code of the matched filter design is shown in Appendix B.

B. Peak Finder

To detect the peaks in the recovered channel, a peakfinder is required. It must be able to find the two peaks present in each measurement. This is done using the Cell Averaging Constant False Alarm Rate (CA-CFAR) algorithm[2]. A schematic representation of the process is shown in Fig. 3.

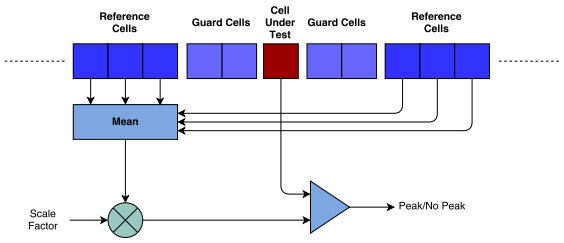

Fig. 3. Schematic representation of the CA-CFAR algorithm. Image taken with permission from [3].

The algorithm splits the signal into several cells, with a certain number of samples in each cell. The average of each cell is taken and is compared to the average of a set of reference cells in the local area. Between the cell under test and the reference cells, a few guard cells are placed. These guard cells are not taken into account when calculating the local average and therefore reduce the impact of multipath clutter on the peak detection.

After the peaks have been detected, they must be converted into a TDOA estimate. This estimate is determined by taking

the difference between the peaks found by the peak finding algorithm. However, there is room for ambiguity in these peaks. Fig. 4 shows two signals and their detected peaks (denoted by o). It shows the first peak of one signal might not always correspond to the same beacon transmission as the first peak of another signal.

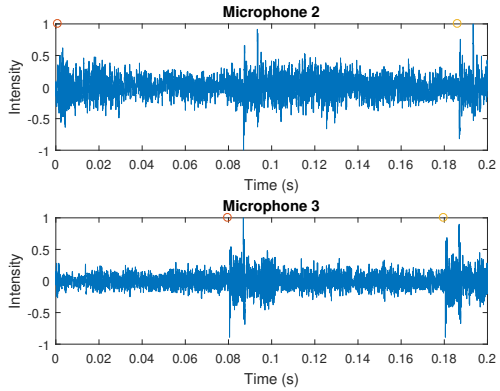

Fig. 4. Two microphone signals and their detected peaks.

To avoid this ambiguity, the minimum distance between peaks is used in determining the sets of peaks. The calculation for microphone m is given in Eq. 9. Where peaks_m is the set of peaks found in the signal for microphone m.

$$\text{TDOA}_m = \min_{i,j \in \{1,2\}} \text{peaks}_0[i] - \text{peaks}_m[j] \tag{9}$$

Whilst this can give falsely low distances when there are very large TDOAs, this will only become an issue when the time difference is over half the beacon transmission frequency, which is 0.05s, or 1700cm. This is larger than the maximum distance difference that can occur in the specified area, and will therefore not be an issue.

IV. Localization

This section is divided into two subsections. Firstly, the TDOA localization algorithm is explained in Section IV-A which retrieves the x and y coordinate from the time difference of arrival between the microphones. Secondly, an outlier rejection scheme and a prediction algorithm are discussed to filter the determined locations and create a smooth path in Section IV-B.

A. Location estimation

There are a lot of localization estimation algorithms. The one of interest is the unconstrained least squares (LS) estimate described in the article of Stroica as the first method [4]. The following matrices are given show in Eq. 10. Where $\mathbf{x}$ is the position of the sound, $||\mathbf{x}||$ the distance of the sound to the origin microphone, d_n the distance difference of the sound to the nth microphone and the sound tot he origin microphone, $\mathbf{a}_n$ the position of the nth microphone and b_n the equation shown in in Eq. 11.

$$y(\mathbf{x}) = \begin{bmatrix} ||\mathbf{x}|| \\ \mathbf{x} \end{bmatrix} \qquad \mathbf{d} = \begin{bmatrix} d_1 \\ \vdots \\ d_N \end{bmatrix}$$
$$\Phi = \begin{bmatrix} d_1 & \mathbf{a}_1^T \\ \vdots & \vdots \\ d_N & \mathbf{a}_N^T \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} \tag{10}$$

$$b_n = \frac{||\mathbf{a}_n||^2 - d_n^2}{2} \tag{11}$$

With these notations, the LS criterion notation can then be written from the standard form to the new form shown in Eq. 12.

$$c_1 = \sum_{n=1}^{N} (d_n||\mathbf{x}|| + \mathbf{a}_n^T\mathbf{x} - b_n)^2$$
$$c_1 = ||\Phi y(\mathbf{y}) - \mathbf{b}^2||^2 \tag{12}$$

This notation can be seen as the unconstrained minimization which has a solution that can be solved quite easily using Eq. 13

$$\bar{y} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{b} \tag{13}$$

B. Path prediction

Due to noise, filter mismatch and erroneous measurements some measurements will be incorrect. Bad measurements can be corrected in two ways, the first way is to simply reject measurements which do not obey some restrictions, which can be adaptive or static. The second way is to smooth the measurements using previous measurements by predicting the next location using previous locations.

1) Error rejection: Since the microphones are located in the corners of the room and the car is unable to leave the area between them, it is impossible for the car to be located outside of the area defined by the microphones' location. The localization algorithm can therefore assume that the car will be inside this area. This allows for easy outlier rejection where any location outside of this area can be discarded immediately.

2) Path smoothing: Path smoothing is done using a Kalman filter. This filter uses previous locations and a model to predict the next location that will be found. In this system, the car's behaviour can be modeled by a velocity vector. Since the sample rate of the location estimator is approximately 5 Hz, it can be assumed that the acceleration of the car will not be significant during this period. The velocity of the car can therefore be assumed to be constant. This allows for a simple model, as seen in Eq. 14. Here, α is used to tune the aggressiveness of the prediction, where lower values are more conservative in their estimates of the next location.

$$\begin{aligned} x(n) &= x(n-1) + \alpha(x(n-1) - x(n-2)) \\ &= x(n-1) + \alpha\Delta x(n-1) \end{aligned} \tag{14}$$

Using this model, the kalman filter equations as described in [5, p. 371-379], results in the set of equations seen in Eq. 19 in appendix A. The equations have been adapted to allow for tuning. Here, β is added to $K(n)$ to allow tuning of the responsiveness, where lower values of β result in a smoother path.

3) Wiener linear prediction: Instead of simply rejecting the outliers, the position could also be replaced by an estimation from a wiener prediction filter. If for every point that should be rejected is replaced by the wiener prediction, a more smooth path is obtained. The Wiener-Hopf equations can be translated into Eq. 15 where p is the filter order. The solution to the equation is shown in Eq. 16.

$$\hat{x}(n+1) = \sum_{k=0}^{p-1} w(k)x(n-k) \tag{15}$$

$$R_x * \mathbf{w} = \mathbf{r}_{x+1} \tag{16}$$

$$\mathbf{w} = R_x^{-1} * \mathbf{r}_{x+1} \tag{17}$$

V. Evaluation

A. Results

1) Wiener deconvolution: Several different wiener convolution filters were tested, with filter orders ranging from 1 to 100. Unfortunately, none of these filters was able to significantly improve the signal received from the microphones. The input and output of the wiener filter is shown in Fig. 5. As can be seen, there is a slight suppression of noise in the signal around 0.02s and 0.12s. The results of filtering vary significantly per measurement, with some filtered signals being worse than their respective unfiltered signals.

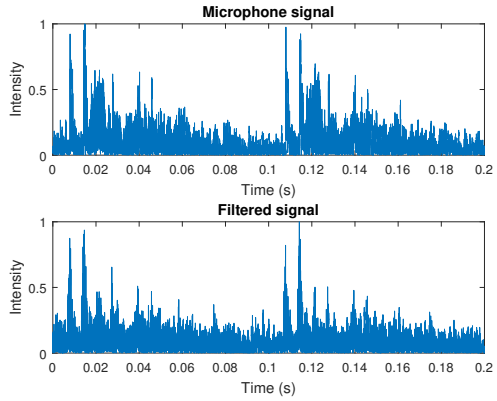

Fig. 5. The input signal (top) and the corresponding output signal (bottom). Absolute value used to emphasize differences.

2) Matched filter: Two matched filters were tested. One matched filter constructed from one reference signal and the other filter was composed from a repetition of this reference signal as would be expected. Almost the same results were found from the Wiener deconvolution. Fig. 6 shows the results of the matched filter created from one reference signal. It can be seen that distinct peaks can be seen at the position (shifted by a small number of samples due to the filter). However, these peaks are not consistent and accurate enough and have numerous peaks close by.

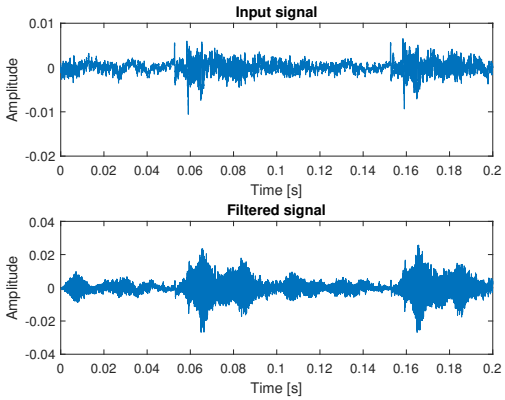

Fig. 6. The result of applying a matched filter to a signal.

3) Peak detection: The peak detection algorithm is capable of reliably detecting the peaks in both the filtered and the unfiltered signals. Fig. 7 shows that the algorithm correctly detects the first peak of each transmitted signal, even though the first multipath peak is higher than the initial peak.

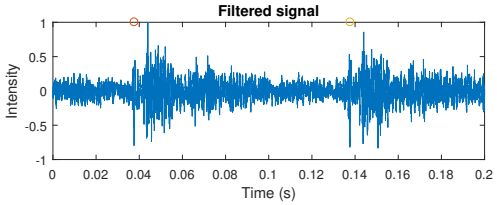

Fig. 7. The result of applying peak detection to a filtered signal.

4) Localization: Since the channel estimation was not accurate enough to generate reliable test samples for the localization algorithm, the input samples were simulated with noise. Fig. 8 shows the simulated results of the algorithm. As noise, a spectral density between 1 and 20 is used. As can be seen, some solutions have large outliers, but most of the results are close by. The path smoothing or path estimation will have to filter some of the outliers.

5) Path smoothing using Kalman filter: Since the localization did not generate usable results, a simulated dataset was used to test the Kalman algorithm. This was done by adding noise to the x and y coordinates of each sample of the supplied path. The noise addition process is described in Eq. 18, where $v(i)$ is a uniformly distributed random variable ranging from -15 to 15.

$$x(i) = x(i) + v(i) \text{ where } v(i) \sim U(-15, 15) \tag{18}$$

B. Conclusion

For every assignment, a solution was presented. Aside from the channel estimation, all the assignment solutions were

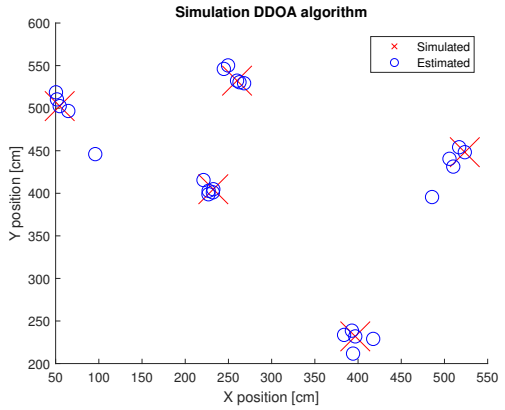

Fig. 8. Results of the simulation of the TDOA algorithm

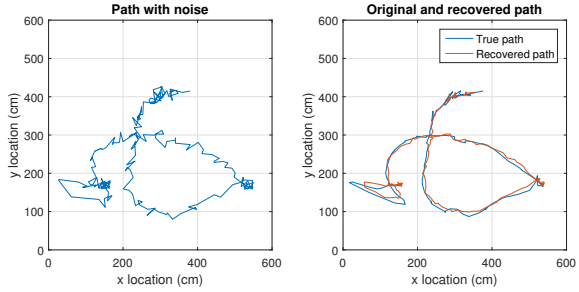

Fig. 9. The result of applying Kalman filtering to a noisy path. The left image shows the path with noise added to it. The right image shows the true path and the path found after filtering. Parameters used are $\alpha = 1/8$ and $\beta = 0.5$

successfully designed. The CA-CFAR algorithm was able to positively detect all important peaks which were needed to be found in every graph. The location estimation from DDOA information used a simple least squares estimate solution given in the literature that was provided. As expected from simulated data, the algorithm was able to find every location with a minor offset if the noise would not exceed 10 meter. The path prediction, or path smoothing with the Kalmann filtered showed that this easy model of the path was able to ignore the outliers for the most path and minimize the error from the true path and the estimate.

Although the channel estimation would not work in practise, the theoretical design of the wiener deconvolution was correctly implemented and should have worked. This failure of the channel estimation will be discussed in the next section.

C. Discussion

A possible explanation of the results of the wiener filter is a modeling error. The model assumes a noise-free channel response $g(n)$, whereas there are many factors contributing to noise in the channel. The main factor here being multipath reflections which distort the channel response, causing an incorrect assumption of the shape of $d(n)$. This causes the solution to the wiener equations to be incorrect, causing the poor performance. A better approach to this problem would be to use an adaptive filter which is able to adapt to the changing multipath environment, enabling it to give a better channel estimate.

Other methods not described in the literature to be used for this project were also applied, but these algorithms were also unsuccessful in determining an accurate channel estimation. This could be caused by numerous factors. As discussed earlier, a large source of interference or noise in the channel could explain the bad estimation. If there was another sound source, reflective walls, or noise in the cabling of the microphones, the signal could be heavily distorted. Another plausible explanation is an invalid reference signal. If the reference signal was incorrectly measured, the signals would not correlate. Additionally, if the microphones' audio responses differ between themselves, the reference signal would only be optimized for one of the microphones. The third valid possibility for bad channel estimation would be the fact that the signals are finite. Determining an autocorrelation sequence out of finite samples creates an error. This error could significantly influence the results.

REFERENCES

[1] A. J. van der Veen *et al.*, "Ee4c03 statistical digital signal processing and modeling," 9 2017. [Online]. Available: http://ens.ewi.tudelft.nl/Education/courses/ee4c03/index.php

[2] C. Algorithm, "Constant false-alarm rate cfar detection." [Online]. Available: https://nl.mathworks.com/help/phased/examples/constant-false-alarm-rate-cfar-detection.html

[3] A. J. v. Katwijk *et al.*, *Person Detection Using Ultra-Wideband Radar*, Delft University of Technology Std., June 2017.

[4] P. Stoica and J. Li, "Lecture notes - source localization from range-difference measurements," *IEEE Signal Processing Magazine*, vol. 23, no. 6, pp. 63–66, 11 2006.

[5] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[6] S. Bancroft, "An algebraic solution of the gps equations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-21, no. 1, pp. 56–59, 1 1985.

APPENDIX A
KALMAN EQUATIONS

$$
\begin{aligned}
C(n) &= 1 \\
A(n-1) &= \frac{(\hat{x}(n-1|n-1) + \Delta\hat{x}(n-1|n-1))}{\hat{x}(n-1|n-1)} \\
\hat{x}(n|n-1) &= A(n-1)\hat{x}(n-1|n-1) \\
&= \hat{x}(n-1|n-1) + \Delta\hat{x}(n-1|n-1) \\
P(n|n-1) &= A(n-1)P(n-1|n-1)A^H(n-1) + Q_w \\
&= \frac{P(n-1|n-1)(\hat{x}(n-1|n-1) + \Delta\hat{x}(n-1|n-1))^2}{\hat{x}(n-1|n-1)^2} + Q_w \\
K(n) &= \beta P(n|n-1)C^H(n)[C(n)P(n|n-1)C^H(n) + Q_v]^{-1} \\
&= \beta P(n|n-1)[P(n|n-1) + Q_v]^{-1} \\
\hat{x}(n|n) &= \hat{x}(n|n-1) + K(n)[y(n) - C(n)\hat{x}(n|n-1)] \\
&= \hat{x}(n|n-1) + K(n)[y(n) - \hat{x}(n|n-1)] \\
P(n|n) &= [I - K(n)C(n)]P(n|n-1) \\
&= [I - K(n)]P(n|n-1)
\end{aligned}
\tag{19}
$$

APPENDIX B
MATLAB CODE

main.m

```
close all;
%clear all;
clc;

load('EE4C03_indoor_loc.mat');


L = 9600;
FilL = 4;
sigmav2 = 0.01;

nummeasurements = size(recording_time_samples_mic, 1);
% Swap mic 3 and 1.
recording_time_samples_mic([3 1],:,:) = recording_time_samples_mic([1 3],:,:);
mic_loc([3 1], :) = mic_loc([1 3], :);


w = zeros(5, FilL);
g = reference_beacon(:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generate wiener for all microphones, based on manually determined d(n).
% Microphone 1
x = recording_time_samples_mic(3,:,1);
d = zeros(L,1);
d(3031) = 1;
d(3031+4800) = 1;
w(1,:) = wiener(FilL, x, g, d, sigmav2);
% Microphone 2
x = recording_time_samples_mic(3,:,2);
d = zeros(L,1);
d(2752) = 1;
d(2752+4800) = 1;
w(2,:) = wiener(FilL, x, g, d, sigmav2);
% Microphone 3
x = recording_time_samples_mic(3,:,3);
d = zeros(L,1);
d(2427) = 1;
d(2427+4800) = 1;
w(3,:) = wiener(FilL, x, g, d, sigmav2);
% Microphone 4
x = recording_time_samples_mic(3,:,4);
d = zeros(L,1);
d(2853) = 1;
d(2853+4800) = 1;
w(4,:) = wiener(FilL, x, g, d, sigmav2);
% Microphone 5
x = recording_time_samples_mic(3,:,4);
d = zeros(L,1);
d(2824) = 1;
d(2824+4800) = 1;
w(5,:) = wiener(FilL, x, g, d, sigmav2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Process microphone data.
peaks = zeros(5, nummeasurements, 2);
for(mic = 1:5)
    disp(['Processing mic ' num2str(mic)]);
    for(i = 1:nummeasurements)
```

```
        if(mod(i, 50) == 0)
            disp(['Measurement ' num2str(i) ' / ' num2str(nummeasurements)]);
        end
        x = recording_time_samples_mic(i,:,mic);
        % Apply the wiener filter.
        d2 = conv(w(mic,:),x);
        % Resize d2 to be 9600 samples.
        d2 = abs(d2(FilL:end));
        % Find the peaks in the recovered channel.
        peaks(mic, i, :) = peakdetect(d2);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Convert peaks to TDOAs
TDOAs = zeros(nummeasurements, 5);
for(i = 1:nummeasurements)
    TDOAs(i, :) = tdoa(squeeze(peaks(:, i, :)));
end

DDOAs = TDOAs * soundspeed / Sampling_frequency;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Convert TDOAs to locations
locations = zeros(nummeasurements, 2);
for(i = 1:nummeasurements)
    [x, y] = ddoa(xy, squeeze(DDOAs(i, 2:5))');
    locations(i, :) = [x y];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Remove outliers
locations = removeoutliers(locations, mic_loc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Filter locations using a Kalman filter
for(i = 1:size(locations,1))
    [x, y] = kalman(locations(i, 1), locations(i, 2));
    locations(i, :) = [x y];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot
figure;
for(i = 1:nummeasurements)
    hold off;
    plot(locations(1:i, 1), locations(1:i, 2));
    hold on;
    plot(X_true(1:i), Y_true(1:i));
    axis([[0 600], [0 600]])
    pause(0.1);
end
```

---

wiener.m

```
% Calculate a wiener filter of length FilL using the supplied parameters.
function w = wiener(FilL, x, g, d, sigmav2)

    L = max(length(x), length(d));

    x = x(:);
    g = g(:);
```

```
    d = d(:);

    g = [g;zeros(L - length(g),1)];


    rd = xcorr(d,d);
    % Zero very low values.
    rd(rd<1e-5)=0;
    % Discard negative indices.
    rd = rd(ceil(length(rd)/2):end);
    % Zero-pad to correct length.
    rd = [rd;zeros(FilL-1,1)];

    rdx = xcorr(x,d);
    rdx = rdx(ceil(length(rdx)/2):end);
    rdx = [rdx;zeros(FilL-1,1)];

    % Build G-matrix.
    G = transpose(convm(g, FilL));
    % [ g(0) g(1) g(2) ...
    % [  0   g(0) g(1) ...
    % [  0    0   g(0) ...

    Rd = toeplitz(rd,rd);
    Rv = diag(sigmav2, FilL-1);

    % Build wiener equations.
    M = (G*Rd*G' + Rv);
    r = rdx(2:FilL+1);%G*rd;

    % Solve equations.
    w = pinv(M)*r;
end
```

---

convm.m, taken from Hayes[5, p. 573]

```
% Set up a convolution matrix of length p, based on data x.
function X = convm(x, p)

    N = length(x) + 2*p - 2;
    x = x(:);
    xpad = [zeros(p-1,1);x;zeros(p-1,1)];
    for(i = 1:p)
        X(:,i) = xpad(p-i+1:N-i+1);
    end
end
```

---

peakdetect.m

```
% Simple adaptive-scale CFAR peak finder
function peaks = peakdetect(h, scale)

    peaks = [];
    peaksfound = 0;
    nguard = 2;
    nref = 10;
    binsize = 16;
    if(nargin < 2)
        scale = 5;
    end
```

```
12
13      nbins = length(h) / binsize;
14      bins = reshape(h, binsize,[]);
15      binsum = sum(bins);
16
17      b = 0;
18      for(a = (1+nguard+nref):(nbins-nref-nguard))
19          if(b > a)
20              continue;
21          end
22          % Calculate sum of reference cells.
23          refsum = sum(binsum(a - nguard - nref:a - nguard)) ...
24                 + sum(binsum(a + nguard:a + nguard + nref));
25          % Average of reference cells.
26          refavg = refsum / (2 * nref);
27
28          % If the current peak is scale times higher than the average.
29          if(binsum(a) > refavg * scale)
30              peaksfound = peaksfound + 1;
31              peaks(peaksfound) = a * binsize;
32              % Skip bins to prevent duplicate peaks in the same peak
33              a = a + (4800 / binsize) / 2;
34              b = a;
35          end
36      end
37
38      % Automatically tune scale to ensure only 2 peaks are found.
39      if(peaksfound < 2)
40          peaks = peakdetect(h, scale * 0.8);
41      end
42      if(peaksfound > 2)              % randomize to prevent infinite loops.
43          peaks = peakdetect(h, scale + rand);
44      end
45  end
```

---

tdoa.m

```
1   % Convert peaks to TDOAs.
2   function TDOAs = tdoa(peaks)
3       TDOAs(1) = 0;
4       for(i = 2:5)
5           % Peak 1 - Peak 1
6           TDOA = (peaks(i, 1) - peaks(1, 1));
7           % Peak 2 - Peak 1
8           if(abs(peaks(i, 2) - peaks(1, 1)) < abs(TDOA))
9               TDOA = peaks(i, 2) - peaks(1, 1);
10          end
11          % Peak 1 - Peak 2
12          if(abs(peaks(i, 1) - peaks(1, 2)) < abs(TDOA))
13              TDOA = peaks(i, 1) - peaks(1, 2);
14          end
15          % Peak 2 - Peak 2
16          if(abs(peaks(i, 2) - peaks(1, 2)) < abs(TDOA))
17              TDOA = peaks(i, 2) - peaks(1, 2);
18          end
19          TDOAs(i) = TDOA;
20      end
21  end
```

---

ddoa.m

```
% Calculate location from ddoa
function [x, y] = ddoa(xy, d)

    % create phi and b
    phi = [d, xy];
    b = (sum(xy.^2, 2)-d.^2)/2;

    % get answer
    z = pinv(phi'*phi)*phi'*b;
    % pinv in case (phi'*phi) is singular

    %extract answer
    x = z(2);
    y = z(3);
end
```

---

removeoutliers.m

```
% Remove samples outside of the region of interest
function filteredlocations = removeoutliers(locations, mic_loc)

    % Calculate target region bounds.
    minx = min(mic_loc(:, 1));
    maxx = max(mic_loc(:, 1));
    miny = min(mic_loc(:, 2));
    maxy = max(mic_loc(:, 2));

    % Simply remove all locations that are outside of the target region.
    filteredlocations = locations(locations(:, 1) > minx & ...
                                  locations(:, 1) < maxx & ...
                                  locations(:, 2) > miny & ...
                                  locations(:, 2) < maxy, :);

    disp(['Filtered ' num2str(length(locations) - length(filteredlocations)) '
        locations']);
end
```

---

kalman.m

```
% Apply a Kalman filter on the provided samples.
function [xhn, yhn] = kalman(x, y)

    persistent vx lastxh lastPx;
    persistent vy lastyh lastPy;
    if(isempty(vx))
        disp('Kalman init');
        vx = 1;
        vy = 1;
        lastxh = x;
        lastyh = y;
        lastPx = 1;
        lastPy = 1;

        xhn = x;
        yhn = y;
        return;
    end

    Qw = 0.1;
    Qv = 0.1;
```

```
22
23      Cx = 1;
24      Cy = 1;
25
26      alpha = 1/8;
27      beta = 0.5;
28
29      Ax = (lastxh + vx*alpha) / lastxh;
30      Ay = (lastyh + vy*alpha) / lastyh;
31
32      % xh(n|n-1) = A(n-1) xh(n-1|n-1)
33      xhn1 = Ax * lastxh;
34      yhn1 = Ay * lastyh;
35
36      % P(n|n-1) = A(n-1) P(n-1|n-1) A(n-1) + Qw(n)
37      Pxn1 = Ax * lastPx * Ax + Qw;
38      Pyn1 = Ay * lastPy * Ay + Qw;
39
40      % K(n) = P(n|n-1)C(n)[C(n)P(n|n-1)C(n) + Qv(n)]^-1
41      Kxn = beta * Pxn1 * Cx * pinv(Cx * Pxn1 * Cx + Qv)*0.5;
42      Kyn = beta * Pyn1 * Cy * pinv(Cy * Pyn1 * Cy + Qv)*0.5;
43
44      % xh(n|n) = xh(n|n-1) + K(n)[y(n) - C(n)xh(n|n-1)]
45      xhn = xhn1 + Kxn * (x - Cx * xhn1);
46      yhn = yhn1 + Kyn * (y - Cy * yhn1);
47
48      % P(n|n) = [1 - K(n)C(n)]P(n|n-1);
49      Pxn = (1 - Kxn * Cx) * Pxn1;
50      Pyn = (1 - Kyn * Cy) * Pyn1;
51
52      % Store P(n|n) for next step, where it will be P(n-1|n-1)
53      lastPx = Pxn;
54      lastPy = Pyn;
55
56      % Store the velocity for the calculation of A(n-1) next step.
57      vx = xhn - lastxh;
58      vy = xhn - lastyh;
59
60      % Store xh(n|n) for next step, where it will be xh(n-1|n-1)
61      lastxh = xhn;
62      lastyh = yhn;
63  end
```

---

matched_filter.m

```
1  % function that gains the matched filter and the output for a given signal
2  function [y, h] = matched_filter(r, x, sigmav2)
3  % using two beacon signals
4  % s = zeros(length(x),1);
5  % s(1:length(r)) = r;
6  % s(end/2:end/2+length(r)-1) = r;
7
8  %using one beacon signal
9  s = r;
10
11 %calculating h
12 Rv = diag(sigmav2*ones(1,length(s))); % Rv is Rv^-1 for white noise
13 alpha= 1/(sqrt(s'*Rv*s));
14 h = alpha*Rv*s;
15
16 % filtering
```

```
y = filter(h, 1, x);

end
```

---

getDDOA.m

```
% function to generate a DDOA vector
function d = getDDOA(xy, x, y, noise)
% for every microphone
for i = 1 : length(xy)
    d(i) = sqrt((xy(i,1)-x)^2 + (xy(i,2)- y)^2) - sqrt(x^2+y^2);
    d(i) = d(i) + noise*randn(1);
end
d = d';
end
```