Robust Time Difference Estimation for Unknown Microphone Positions with Reverberation

Simon Segerblom Rex

Master's thesis
2015:E6

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Robust Time Difference Estimation for Unknown Microphone Positions with Reverberation

Simon Segerblom Rex

Abstract

In this thesis a system is developed for robust time difference estimation for multiple microphones in reverberant environments, without any knowledge of the 3D positions of the microphones or the room. RANSAC is used to find line segments in sets of frames containing a number of the largest GCC-PHAT peaks. Focus is on real experiments with one moving sound source continuously playing music. The estimated time differences can be used to find the microphone positions, track the sound source, and estimate flat surfaces in the room. The system has been evaluated on real data with manually annotated ground truth in order to estimate the precision of the system in terms of true positives and false positives. The algorithms have also been incorporated into a system that takes a number of sound files as input and produces the microphone positions, sound path motion and reverberant structures. The complete system has been evaluated on real data with promising results.

LUND UNIVERSITY

Master of Science in Engineering Physics,
Centre for Mathematical Sciences
Degree Project in Mathematics for Engineers
FMA820 – March 2015

Supervisor:
Karl Åström

Acknowledgements

I want to thank my supervisor Karl Åström for suggesting the thesis subject and for all input during the progress of the work. Thanks are also due to Zhayida Simayijiang, Matilda Landgren, and Hanna Källén for assisting the video recording of the experiments. Maiko, thank you for supporting me at home!

Contents

Appendices

Chapter 1

Introduction

Using a set of synchronized recordings from a number of microphones it is possible to track a randomly moving sound source in a room. This is true even if the microphone positions are unknown.

The aim of this thesis is to develop a system capable of finding the microphone time differences for recordings of a single sound source continuously playing music, moving in 3D, with only the sound recordings as input, see Figure 1.1.

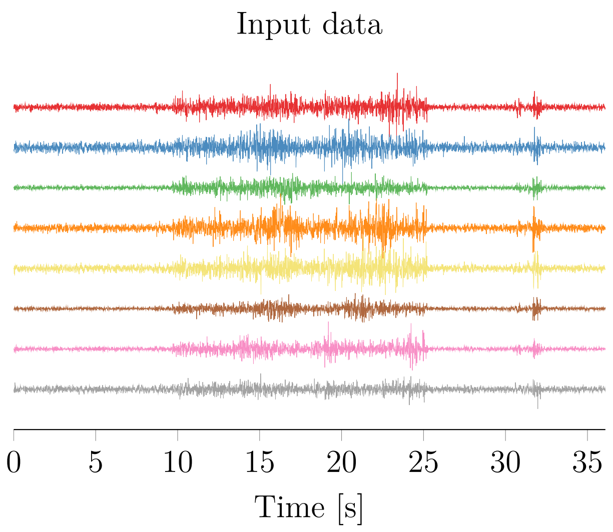

Figure 1.1: Eight synchronized sound recordings from the same number of stationary omnidirectional microphones serve as the input data.

The output consists of vectors containing estimated time differences (or range differences, if scaled with the speed of sound) between all microphones and one microphone for a number of positions in time, see Figure 1.2. This data can be used for estimating the 3D positions of the microphones, the path of the sound source, and reverberant flat surfaces in the room such as walls and floor.

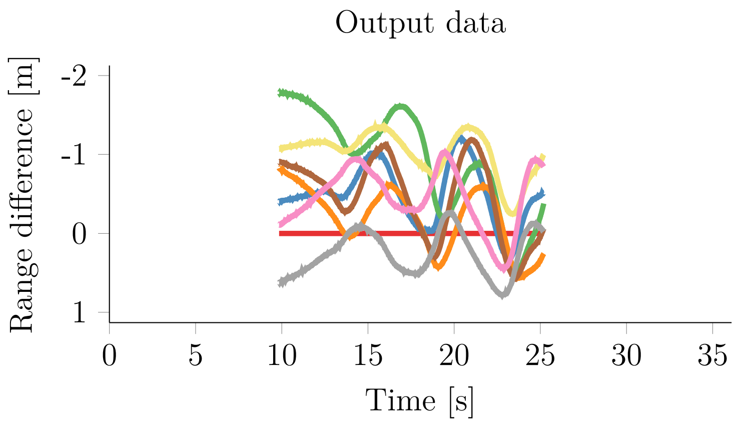

Figure 1.2: Estimated range differences between all eight channels and one of the channels (red). The colors in the plot correspond to the ones used in Figure 1.1.

The idea is to start off with a large data set containing both inliers and outliers by selecting multiple peaks from the generalized cross-correlation phase transform (GCC-PHAT, see Section 2.2) for every frame for all channel pairs, and then reduce the number of outliers with various techniques. Random sample consensus (RANSAC, see Section 2.3) is used to find line segments for tracks containing a number of subsequent frames.

1.1 Previous Research

The idea to use time differences for positioning is not new. Meldercreutz (1741) suggested a method for land surveying almost three centuries ago! Back then, the time measurement was limited to having observers flipping hourglasses.

Today, more advanced methods are available. One is to use cross-correlation. In particular GCC-PHAT, introduced by Knapp and Carter (1976), is popular. Parisi, Cirillo, Panella, and Uncini (2007) suggest using multiple peaks from the GCC-PHAT. They only use one stationary sound source in synthetic experiments.

Zhayida, Andersson, Kuang, and Åström (2014) provide a system for finding time differences using cross-correlation and solving the time difference of arrival (TDOA, see Section 2.1) problem. Their system is proven to work well for real experiments in anechoic chambers, but fails finding the time differences for experiments in reverberant rooms. This served as the starting point for this thesis project.

Chapter 2

Theory

This chapter introduces the concepts and algorithms used in Chapter 3. In Section 2.1 the equations needed to solve the time difference of arrival problem are formulated, without providing the full solution to the problem, which is out of the scope of this thesis. The notation and problem formulation follows the one used used by Zhayida et al. (2014). Readers familiar with cross-correlation and RANSAC can skip Sections 2.2 and 2.3 respectively.

2.1 Time Difference of Arrival

Time difference of arrival (TDOA) is the time difference it takes for a signal emitted from a transmitter, $\mathbf{s}_j$, at an unknown time instance $t_{0,j}$ to reach two receivers, $\mathbf{m}_1$ and $\mathbf{m}_2$, at time instances $t_{1,j}$ and $t_{2,j}$ respectively. This can be measured if the receivers are synchronized. If $t_{0,j}$ also is known one would instead talk of time of arrival (TOA) measurements.

To estimate the receiver and transmitter positions only from TDOA measurements one needs M receivers $\mathbf{m}_i$, $i = 1, \ldots, M$, and N matched transmitters $\mathbf{s}_j$, $j = 1, \ldots, N$. The requirements on M and N are discussed later in this section.

The distance between $\mathbf{s}_j$ and $\mathbf{m}_i$ can be expressed as

$$c(t_{i,j} - t_{0,j}) = \left\| \mathbf{s}_j - \mathbf{m}_i \right\|,$$

where c denotes the propagation speed of the signal.

Let $u^r_{i,j}$ denote the range difference between receivers $\mathbf{m}_i$ and $\mathbf{m}_r$ for transmitter $\mathbf{s}_j$, then

$$\begin{aligned} u^r_{i,j} &= c(t_{i,j} - t_{r,j}) = c(t_{i,j} - t_{0,j}) - c(t_{r,j} - t_{0,j}) \\ &= \|\mathbf{s}_j - \mathbf{m}_i\| - \|\mathbf{s}_j - \mathbf{m}_r\| = \|\mathbf{s}_j - \mathbf{m}_i\| - o_{r,j}, \end{aligned} \tag{2.1}$$

where $o_{r,j} = \|\mathbf{s}_j - \mathbf{m}_r\|$ is introduced as the offset between $\mathbf{s}_j$ and $\mathbf{m}_r$, $r = 1, \ldots, M$. In Figure 2.1 the range differences and offsets are illustrated for three receivers and one transmitter.

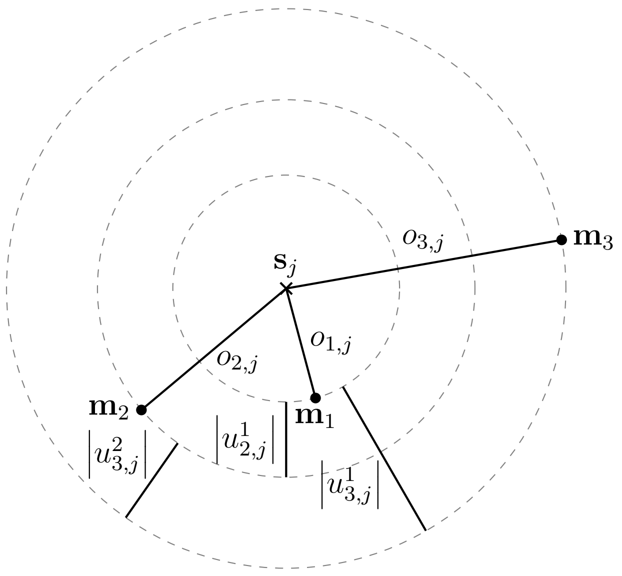

Figure 2.1: Illustration of the range differences for three receivers $\mathbf{m}_i$ (dots) and one transmitter $\mathbf{s}_j$ (cross). $u^r_{i,j} = \|\mathbf{s}_j - \mathbf{m}_i\| - o_{r,j}$, where $o_{r,j} = \|\mathbf{s}_j - \mathbf{m}_r\|$, $i, r \in 1, 2, 3$. (Note that the superscript is an index and not an exponent.)

To solve the problem in 3D with the method proposed by Kuang and Åström (2013) one needs a minimum of $M = 6$ receivers for which the required number of transmitters is $N = 8$. Zhayida et al. (2014) use $M = 7$, $N = 6$. Instead of having N stationary transmitters one can study a single moving sound source at N different positions.

Note that

$$u^r_{i,j} = u^r_{k,j} - u^i_{k,j},\ k \in \{1, \dots, M\} \setminus \{i, r\} \tag{2.2}$$

gives $M - 2$ additional expressions for $u^r_{i,j}$.

2.2 Cross-Correlation

The cross-correlation between two sequences, u and v, is defined by (Proakis & Manolakis, 1996)

$$(u \star v)(\tau) = \sum_{n=-\infty}^{\infty} u(n)v(n - \tau),\ \tau = 0, \pm 1, \pm 2, \dots$$

It can be used as a measure for how similar two sequences are for different shifts τ. The cross-correlation between the two sequences illustrated in Figure 2.2 can be found in Figure 2.3.

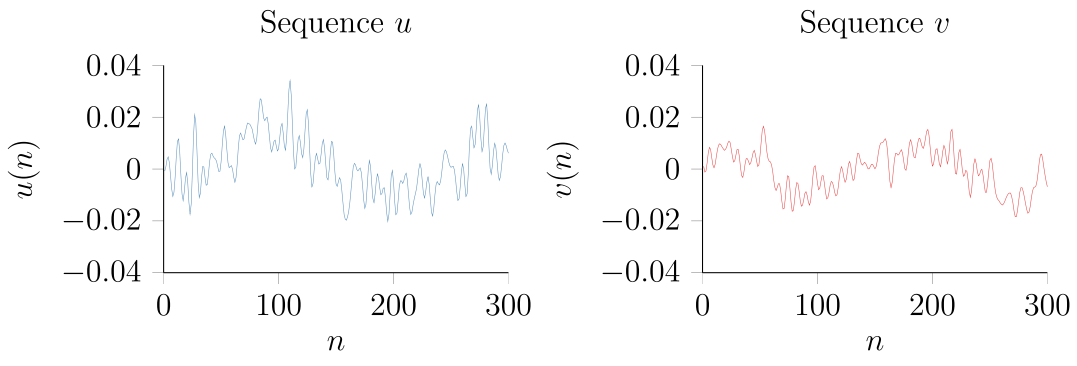

Figure 2.2: A short segment of a sound recording from two synchronized microphones at different positions.

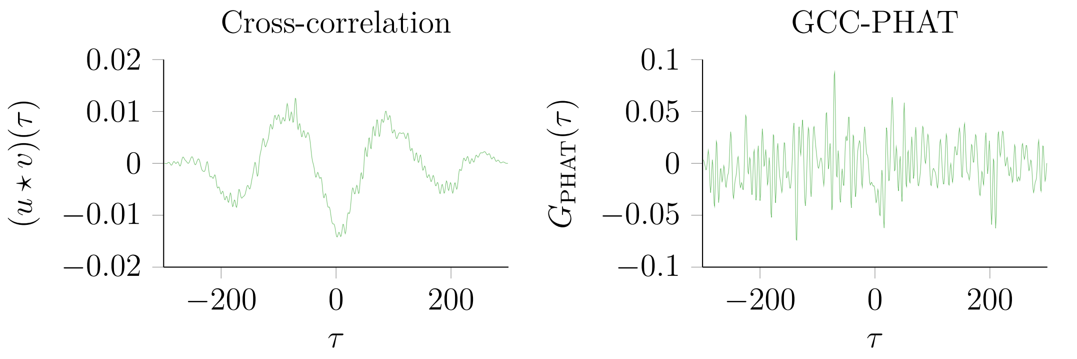

Figure 2.3: Cross-correlation (left) and GCC-PHAT (right) between the two signals in Figure 2.2. Both have $\arg\max_\tau \, (u \star v)(\tau) = -70$, which corresponds to the time difference between the two microphones and the sound source in sample points.

Using the discrete-time Fourier transform (DTFT)

$$U(f) = \sum_{n=-\infty}^{\infty} u(n) e^{-i2\pi f n}$$

with the inverse

$$u(n) = \int_{-\infty}^{\infty} U(f) e^{i2\pi f n} \, \mathrm{d}f = \int_{-1/2}^{1/2} U(f) e^{i2\pi f n} \, \mathrm{d}f,$$

one can also express the cross-correlation as

$$(u \star v)(\tau) = \int_{-1/2}^{1/2} U(f) \overline{V(f)} e^{i2\pi f \tau} \, \mathrm{d}f,$$

where $\overline{V(f)}$ is the complex conjugate of $V(f)$. This can be generalized by inserting a weighting function $W(f)$:

$$G(\tau) = \int_{-1/2}^{1/2} W(f)U(f)\overline{V(f)}e^{i2\pi f\tau}\,\mathrm{d}f$$

$G(\tau)$ is referred to as the generalized cross-correlation function (GCC). $W(f) = 1$ gives the normal cross-correlation. Another popular one is the so called GCC phase transform (GCC-PHAT) (Knapp & Carter, 1976), which is given by

$$W(f) = \frac{1}{\left|U(f)\overline{V(f)}\right|}. \tag{2.3}$$

In Figure 2.3 it is illustrated how the GCC-PHAT can result in more distinct peaks than the normal cross-correlation.

2.3 Random Sample Consensus

Random sample consensus (RANSAC), introduced by Fischler and Bolles (1981), is a method for fitting a model to a data set containing a potentially large amount of outliers.

The idea is to only use a minimal (or small) subset of points to fit the model, and then count the number of inliers, i.e., the points with residuals smaller than some threshold. This process is repeated for a predefined number of iterations, or until a set containing a satisfactory number of inliers has been found. The largest inlier set is then used to fit the model.

In Figure 2.4, RANSAC is compared to using all points with linear least squares for fitting a line.

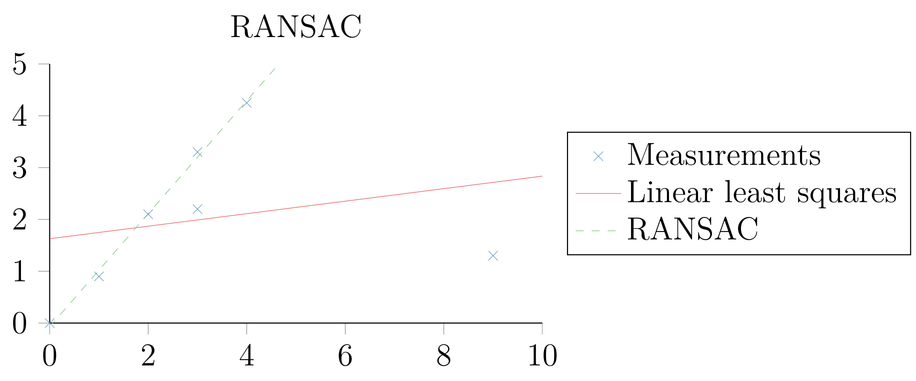

Figure 2.4: Line fitting with RANSAC compared to linear least squares.

Chapter 3

Method

The algorithm for finding the range differences consists of a number of steps that will be described in this chapter. The main idea is to start out with a set containing both a large number of outliers and inliers, and then trying to reduce the number of outliers while keeping, or increasing, the number of inliers step by step.

The Matlab implementation of the system can be found in Appendix A.

3.1 Input Data and Frame Creation

The input data consists of at least two synchronized sound files. It is assumed that recordings have been made with omnidirectional microphones with one moving sound source continuously making some sound.

To compare different channels it is necessary to be able to divide the data into frames short enough to consider the sound source stationary. To get a good resolution a frame should also contain a large number of sample points. This can be achieved by using a high sample rate for the recordings. With a sample rate $F_s = 96000$ Hz, and a frame size n_{fs}, one frame is

$$\frac{n_{\text{fs}}}{F_s} = \frac{2048}{96000} \text{ s} \approx 0.02 \text{ s}$$

long. That is, a sound source moving at a speed of 1 m/s is off by approximately ± 1 cm, while the range difference can be ± 3 m, when comparing the same frame for two different channels. The reason for choosing a $n_{\text{fs}} = 2048$ instead of 2000 is to save a few seconds when calculating the fast Fourier transform later (which is suited for vectors of a length that is a power of two).

The frames should be close enough in time to be able to consider the movement of the sound source linear when studying a number of subsequent frames. Frames 1000 sample points apart will suffice for the experiments in this project, resulting in an overlap between the frames of 1048 sample points.

3.2 Cross-Correlation

The cross-correlation (with GCC-PHAT, see Section 2.2), is computed between corresponding frames for all channel pairs. This can be done efficiently using the 2D fast Fourier transform.

To avoid problems when $\left|U(f)\overline{V(f)}\right|$ is close to zero, the weighting function defined in equation 2.3 is modified to

$$W(f) = \frac{1}{\left|U(f)\overline{V(f)}\right| + \delta},$$

where

$$\delta = \begin{cases} 1 & \text{if } \left|U(f)\overline{V(f)}\right| < \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

in which $\varepsilon = 5 \cdot 10^{-3}$. Another option could be to instead use

$$W(f) = \frac{1}{\max\left(\left|U(f)\overline{V(f)}\right|, \varepsilon\right)}.$$

In Figure 3.1 it is illustrated what the GCC-PHAT looks like for one pair when arranging the frames as columns in a matrix. Both range differences that are a result from the direct sound path and parts of indirect paths are distinguishable.

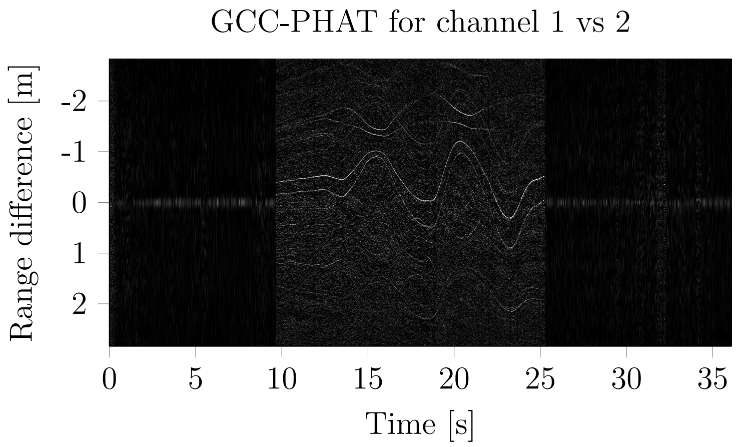

Figure 3.1: GCC-PHAT between all frames for two channels. Each column of pixels represent one (slightly cropped) frame. Only positive values are plotted. The most prominent curve structure is the result of the direct sound path. (The whites have been clipped to increase the contrast in the plot.)

3.3 Peak Selection

The $K = 4$ strongest peaks for each frame are considered. To avoid some outliers, peaks under a predefined threshold are ignored. Setting this threshold high will increase the ratio of inliers to outliers, but for the following steps it's more important to have a large number of inliers, even though the number of outliers is also high. See Figure 3.2.

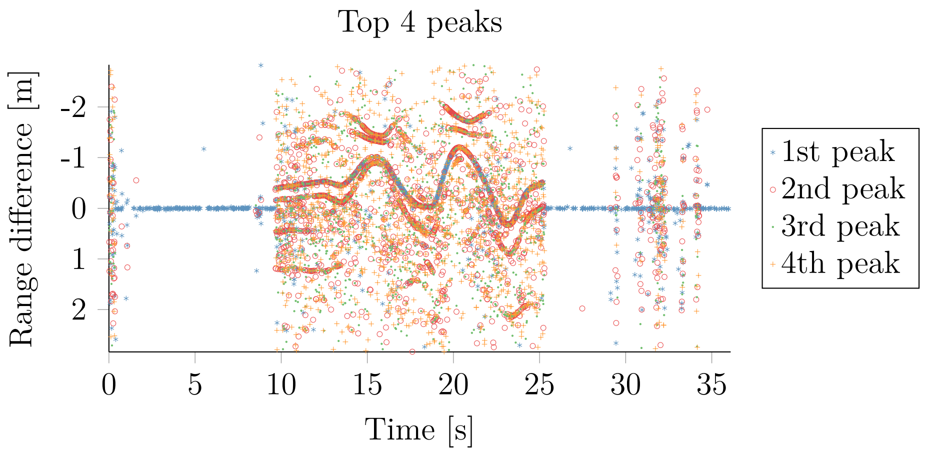

Figure 3.2: The four strongest peaks above some threshold for each frame for one pair of channels. Most are outliers, but range differences corresponding to the direct sound path and parts of indirect paths can be hinted.

3.4 Matching Peaks Over all Channels

Studying a frame for one channel pair the previous step gives K values for $u^r_{i,j}$ that are potential inliers. Equation 2.2 gives an additional $K^2(M-2)$ values, and a total of $K + K^2(M-2)$ values. They are put into bins 3 sample points wide. Bins with a count of at least 3 are considered, for which an inlier value is calculated as the median. This results in both new inliers and removal of outliers. See Figure 3.3.

3.5 Using RANSAC to Find Line Segments

To find tracklets, i.e., small parts of a curve, each pair of channels is studied for sets of 21 subsequent frames with 1 frame overlap. RANSAC (see Section 2.3) is used to find the line segments containing the most inliers. A line segment

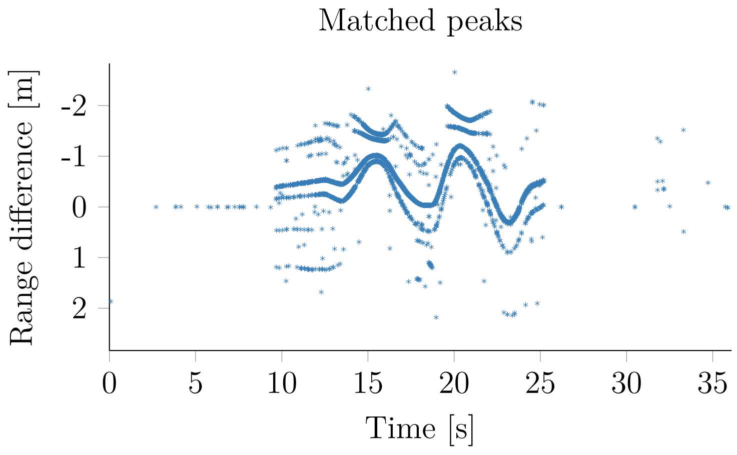

Figure 3.3: Potential inliers after matching over the channels using equation 2.2. Many outliers have been removed.

must contain a minimum of 5 inliers, fulfill a threshold for the slope, and share a maximum of 1 inlier with the other returned line segments. To decrease the required number of iterations the RANSAC algorithm is non-repeating, i.e., not using the same values more than once. The tracklets are presented in Figure 3.4. Both the tracklets and their corresponding line segments are stored.

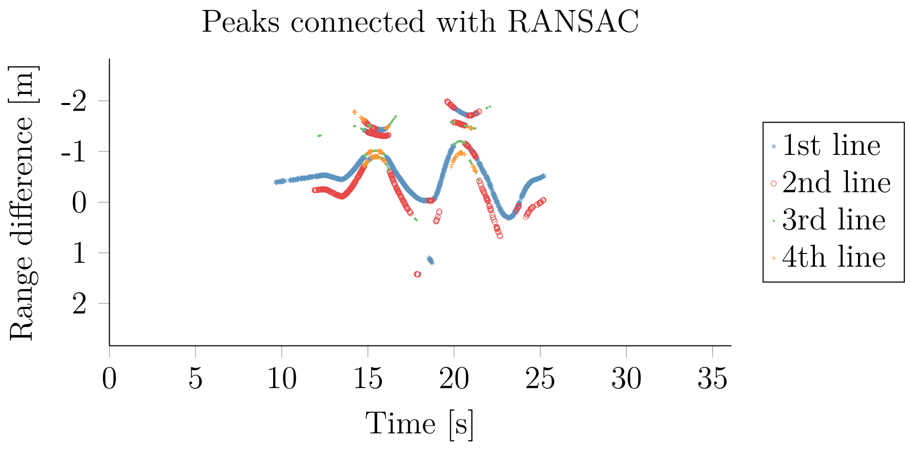

Figure 3.4: Tracklets of inliers corresponding to the line segments found with RANSAC. The tracklets are sorted by the number of inliers they contain.

3.6 Connecting Line Segments

Line segments closer than 10 sample points are connected to form longer tracklets. This is illustrated in Figure 3.5.

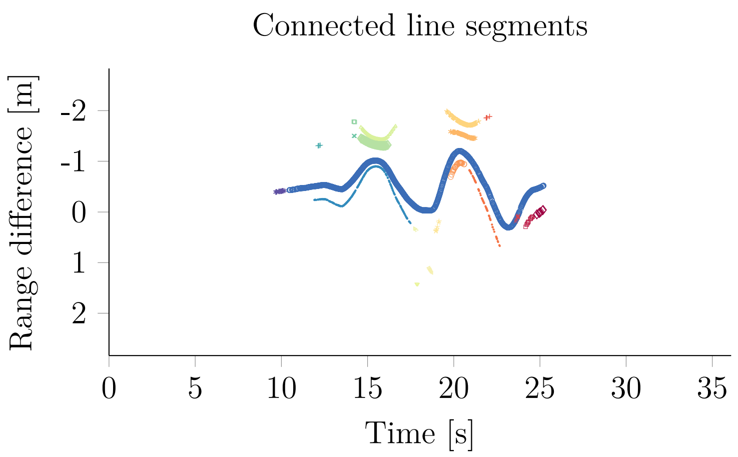

Figure 3.5: Connected tracklets for one channel pair. The direct path is almost complete, but there is still a small piece to the left that is loose.

Next tracklets separated by a short amount in time, sharing the same line equation are connected. See Figure 3.6.

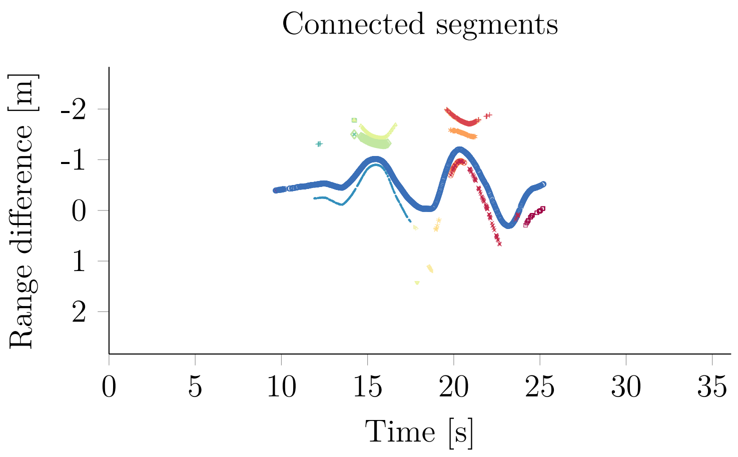

Figure 3.6: Connected tracklets from the previous step. One tracklet is now representing the whole direct path.

3.7 Smoothing

As a last step the longest segment is smoothed with a moving average filter before fitting a spline. Peaks that are closer than 2 sample points from the spline are considered inliers. For other frames the spline value becomes a new inlier.

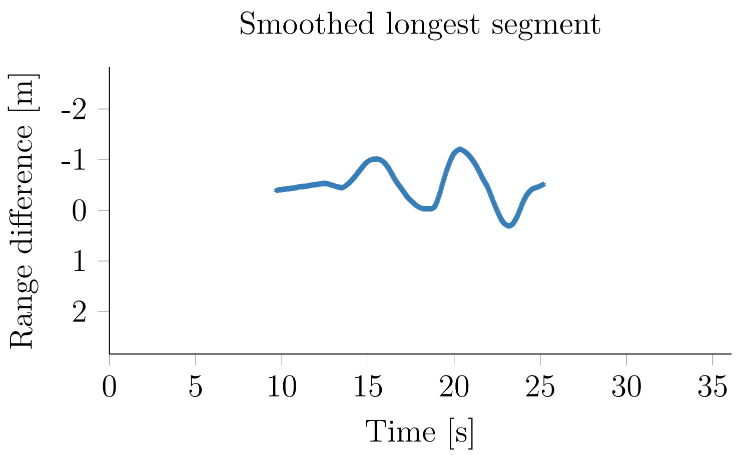

Figure 3.7: Smoothed longest segment. Gaps have been filled in by fitting a spline. Shorter tracklets are ignored.

Chapter 4

Experiments

Several experiments were carried out in different environments containing reverberant flat surfaces. The experimental setup is illustrated in Figure 4.1. Eight omnidirectional microphones (t.bone MM 1) connected to a USB sound card (M-Audio Fast Track Ultra 8R) were used to record a single moving sound source consisting of a portable loudspeaker (Roxcore Portasound) connected to a mobile phone through the headphone output. The sound was recorded with a sample rate of 96000 Hz and stored with lossless compression. For visualization some of the experiments were also recorded on video.

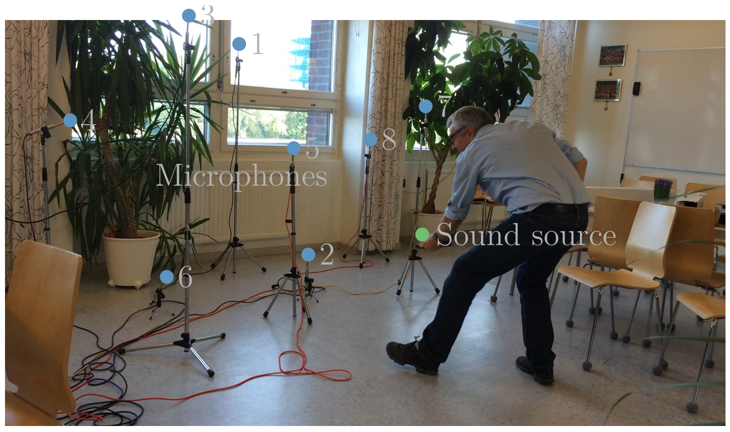

Figure 4.1: Experimental setup with one moving sound source and eight microphones.

Chapter 5

Results

The results presented in Section 5.1 are the direct result from the system implemented in this thesis project. In the subsequent sections of this chapter the TDOA solver provided by Zhayida et al. (2014) is used to evaluate the accuracy of the output.

5.1 Final Output

The final output of the system implemented in this project is illustrated for one of the experiments in Figure 5.1. This is the same experiment that has served for the illustrations throughout Chapter 3.

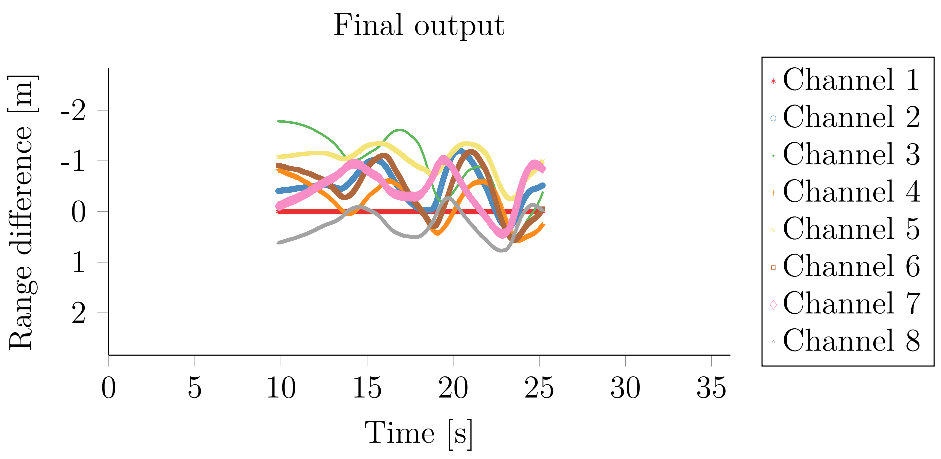

Figure 5.1: Range differences for all eight channels compared to Channel 1 for one experiment.

5.2 Projecting the Solution

Using the range differences in Figure 5.1 as input in the system presented by Zhayida et al. (2014) the microphone and sound source positions can be estimated, see Figure 5.2.

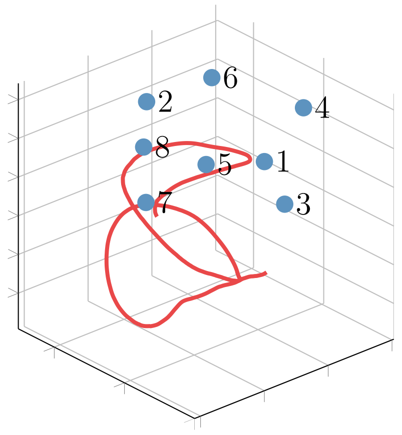

Figure 5.2: 3D path Zhayida et al. (2014). An interactive version of this figure is presented in Appendix B.

After calculating the camera matrix (by manually marking the camera positions in the video) the 3D path of the sound in Figure 5.2 can be projected, see Figure 5.3.

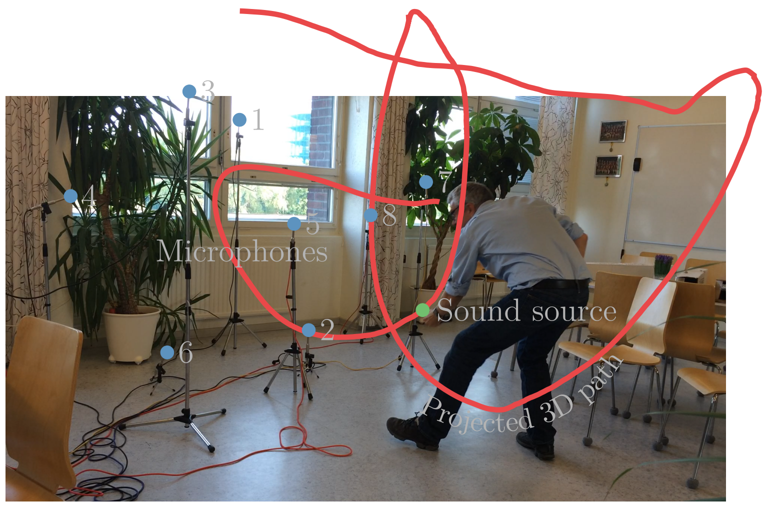

Figure 5.3: The 3D path from Figure 5.2 projected at video recording of the experiment. The video is presented in Appendix B.

Table 5.1: The number of outliers and inliers after each step for one channel pair for one of the experiments.

Step	Inliers	Outliers	Corresponding figure
Step 1	1260	4639	Figure 3.2
Step 2	1228	633	Figure 3.3
Step 3	1027	30	Figure 3.4
Step 4	1199	93	Figure 3.5
Step 5	1199	7	Figure 3.6
Step 6	1475	17	Figure 3.7

The number of inliers and outliers after each step of the algorithm compared to the projected solution is presented in Table 5.1 for one channel pair.

Figure 5.4 shows a histogram of the residuals when comparing the calculated direct path with the projected soulution.

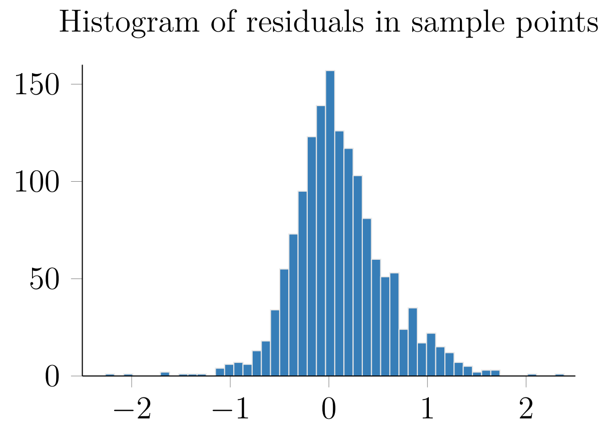

Figure 5.4: Residuals for the direct path compared to the projected solution.

5.3 Indirect Paths

In Figure 5.5 the solution for the direct path is projected on top of the matched peaks. Also indirect paths calculated from mirroring the microphones in a surface (found with trilateration for the second longest segment) are projected.

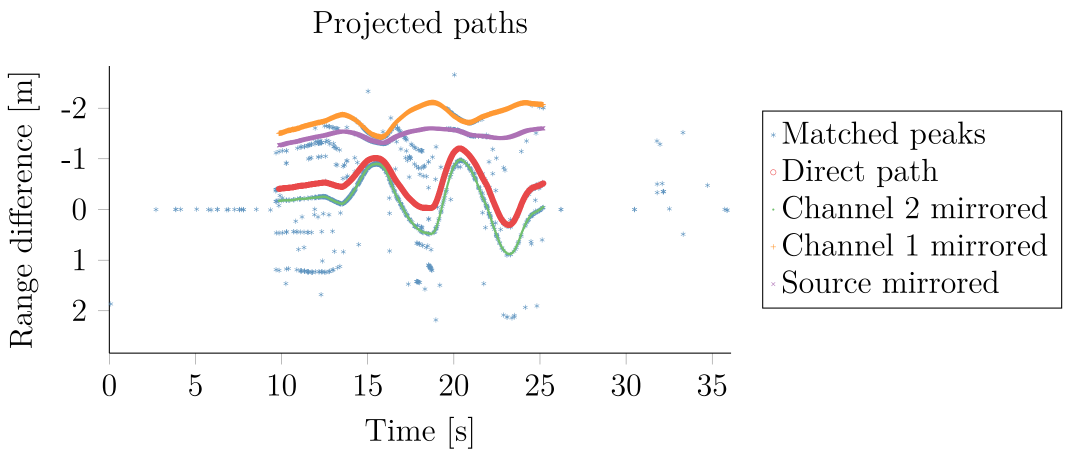

Figure 5.5: Projected paths for one of the channel pairs.

Chapter 6

Discussion

The developed system works well, at least for some experiments, as seen in Figure 5.1 and 5.3. Table 5.1 shows that the number of outliers is reduced while the number of inliers is kept, or increased. The residuals, illustrated in Figure 5.4, are small, most within ±1 sample point, which corresponds to ±3.5 mm. Errors in the projection of the solution from Figure 5.2 in Figure 5.3 are due to the rough estimate of the camera matrix only using eight points in the image.

There are some situations for which the system might fail. RANSAC is based on randomness and it is important that the number of iterations is kept high to reduce the risk of missing data. When connecting the tracklets there is a risk of loosing track of the direct path when it is crossed by indirect paths. This is a difficult situation that might require some additional step for increased robustness.

Using the second longest segment, which should represent some part of an indirect path, it is possible to find the plane equation for the reverberant surface using trilateration. This is illustrated in Figure 5.5 where the the equation for the floor has been estimated and used to reflect the microphone positions.

The implementation in Matlab can be improved in many ways. The current use of cell arrays is not optimal in some cases. It would also be interesting to try doing all calculations for small sets of frames instead of doing it step by step for the whole signal. Maybe the system would then work for real time applications.

Further research is required to test the system in different situations. Using other sound sources than speakers might not work. Sound source separation of two or more sound sources would be interesting to try. A robust way of automatic estimation of reverberant flat surfaces is also yet to be implemented.

Chapter 7

Conclusion

In this thesis a method is proposed for microphone time difference estimation in reverberant rooms with one moving sound source and unknown microphone positions. RANSAC is used to find line segments in sets of frames containing a number of the largest GCC-PHAT peaks. The method is proved to work well in real experiments with a speaker playing music moving in 3D. The output can be used as input in a system for TDOA solving to estimate microphone positions, sound source position, and flat reverberant surfaces. Further research is required to increase the robustness of the system.

References

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, *24*(6), 381–395. doi: 10.1145/358669.358692

Knapp, C., & Carter, G. C. (1976). The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *24*(4), 320–327. doi: 10.1109/TASSP.1976.1162830

Kuang, Y., & Åström, K. (2013). Stratified sensor network self-calibration from TDOA measurements. In *21st European Signal Processing Conference (EUSIPCO).* Retrieved from `http://www.eurasip.org/Proceedings/Eusipco/Eusipco2013/papers/1569743721.pdf`

Meldercreutz, J. (1741). Om längders mätning genom dåns tilhielp. *Vetenskapsakademiens Handlingar*, *2*, 73–77.

Parisi, R., Cirillo, A., Panella, M., & Uncini, A. (2007). Source localization in reverberant environments by consistent peak selection. In *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP* (Vol. 1, pp. 37–40). doi: 10.1109/ICASSP.2007.366610

Proakis, J. G., & Manolakis, D. G. (1996). *Digital signal processing: Principles, algorithms and applications* (3rd ed.). Upper Saddle River, N.J.: Prentice-Hall.

Zhayida, S., Andersson, F., Kuang, Y., & Åström, K. (2014). An automatic system for microphone self-localization using ambient sound. In *22nd European Signal Processing Conference (EUSIPCO)* (pp. 954–958). Retrieved from `http://www.eurasip.org/Proceedings/Eusipco/Eusipco2014/HTML/papers/1569923937.pdf`

Appendix A

Matlab Code

Here the full implementation of the range difference estimating system is presented. The scripts have only been tested in Matlab 2014a and have some dependencies on the Signal Processing Toolbox and the Statistics Toolbox. Internal dependencies are presented in Figure A.1.

The code has also been published under a free software license on GitHub. Please refer to this repository for the latest version: `https://github.com/SimonSegerblomRex/tdoa`.

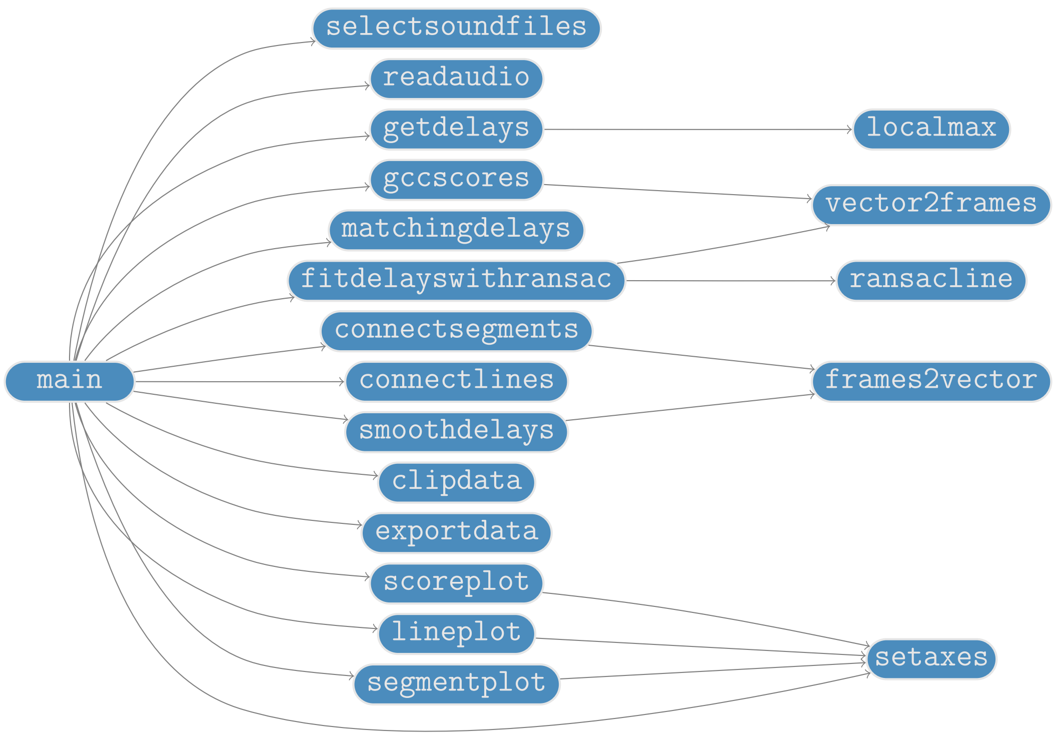

Figure A.1: Function dependencies for the Matlab-code.

A.1 main

This is the only script to run, preferably section by section.

```
%Script for time difference estimation
% Run section by section

%% Read audio files:
settings.v = 340;         %speed of sound
settings.mm = 8;          %number of microphones
settings.channels = 1:8; %channels to read
settings.refChannel = 1; %reference channel
%[340,8,1:8,1]

[fileNameBase,dataDir,fileExt] = selectsoundfiles();
[a,settings.sr] = readaudio([dataDir fileNameBase],fileExt,settings.mm,...
    settings.channels);
settings.nbrOfSamples = length(a);

%% Correlation:
settings.wf = @(x) 1./(abs(x)+(abs(x)<5e-3)); %weighting function
settings.firstSamplePoint = 1; %center sample point of first frame
settings.frameSize = 2048;      %width of frame in sample points
settings.dx = 1000;             %distance between frames in sample points
settings.frameOverlap = settings.frameSize-settings.dx; %overlap between frames
settings.sw = 800;              %clipping of search window
%Default: [@(x) 1./(abs(x)+(abs(x)<5e-3)),1,2048,1000,800]

scores = gccscores(a,settings);

settings.nbrOfFrames = size(scores{1,1},2);

%Plot:
ch = settings.channels(2); %channel to plot
figure(1),scoreplot(scores{settings.refChannel,ch},settings)

%% Delays:
settings.nbrOfPeaks = 4;       %max number of peaks
settings.minPeakHeight = 0.01; %threshold for min value of local maxima
%Default: [4,0.01]

u = getdelays(scores,settings);

%Plot:
figure(2),scoreplot(scores{settings.refChannel,ch},settings),hold on
plot(1:size(u{settings.refChannel,ch},2),...
    u{settings.refChannel,ch}+settings.sw+1,'*')
%% Clean up delays:
settings.binSize = 3;          %inlier threshold
settings.minNbrOfInliers = 3; %min number of matching equations
%Default values: [3,3]

uref = matchingdelays(u,settings);

%Plot:
figure(3),scoreplot(scores{settings.refChannel,ch},settings),hold on
plot(1:size(uref{ch},2),uref{ch}+settings.sw+1,'g*')

%% RANSAC:
settings.RANSACnbrOfIterations = 350;      %number of RANSAC iterations
settings.RANSACframeSize = 21;             %line width
settings.RANSACframeOverlap = 1;           %overlap of lines
settings.RANSACmaxNbrOfGroups = 5;         %max nbr of lines
settings.RANSACminNbrOfInliers = 6;        %min nbr of inliers
settings.RANSACinlierThreshold = 1;        %max distance to line
settings.RANSACsharedPointsThreshold = 1; %max nbr of shared points
settings.RANSACmaxSlope = 3.5;             %max derivative
%Default values: [350,21,1,5,6,1,1,3.5]

[delays,lines,ind] = fitdelayswithransac(uref,settings);

%Plot:
figure(4),lineplot(delays{ch},ind,settings)
figure(5),lineplot(lines{ch},ind,settings)

%% Connect lines:
settings.lineDistanceThreshold = 10; %max vertical distance between lines
%Default value: [10]

[delaysegments,linesegments] = connectlines(delays,lines,ind,settings);

%Plot:
figure(6)
segmentplot(scores(settings.refChannel,:),...
    delaysegments,ind,@(x)find(x>0),settings)

%% Connect line-segments:
settings.linesInlierThreshold = 10;
settings.linesOverlap = 5;
settings.linesInlierRatio = 0.15;
[newdelaysegments,newlinesegments] = connectsegments(delaysegments,...
    linesegments,ind,uref,settings);
%Default values: [10,5,0.15]

%Plot:
figure(7)
segmentplot(scores(settings.refChannel,:),...
    newdelaysegments,ind,@(x)find(x==max(x),1,'first'),settings)

%% Smoothing:
settings.smoothingDegree = 0.01; %degree of smoothing of uref
settings.smoothingDistance = 2;  %set to 0 to keep smoothed curve
%Default values: [0.01,2]

urefsmooth = smoothdelays(newdelaysegments,newlinesegments,uref,settings);

%Plot:
figure(8),scoreplot(scores{settings.refChannel,ch},settings),hold on
plot(urefsmooth{ch}+settings.sw+1,'g.')

%% Clip data:
uout = clipdata(urefsmooth,settings);

%Plot:
figure(9),pl = plot(1:length(uout),uout+settings.sw+1,'*');
marks = {'+','o','*','.','x','s','d','^','v','>','<','p','h'}.';
colors = num2cell(hsv(settings.mm),2);
set(pl,{'Marker'},marks(1:settings.mm),{'MarkerFaceColor'},...
    colors,{'MarkerEdgeColor'},colors);
leg = num2cell(1:settings.mm);
leg = cellfun(@(k) ['Channel ' num2str(k)],leg,'UniformOutput',false);
legend(leg)
setaxes(settings)

%% Save data:
if ~exist([fileNameBase 'data.mat'],'file')
    save([fileNameBase 'data'],'settings','u','uref','delays','lines','ind',...
        'delaysegments','linesegments','newdelaysegments','newlinesegments',...
        'urefsmooth','uout')
end

%% Export data:
%Only if you intend to use the 'tdoasystem'-package
exportdata
```

A.2 selectsoundfiles

Brings up a dialogue to choose input data.

```
function [fileNameBase,dataDir,fileExt] = selectsoundfiles(delimeter)
%SELECTSOUNDFILES - Select one of the input sound files
%
%    [fileNameBase,dataDir,fileExt] = SELECTSOUNDFILES()
%    [fileNameBase,dataDir,fileExt] = SELECTSOUNDFILES(delimeter)
%
%    Input (optional):
%    delimeter - string containg the delimeter separating the sound files
%                default value is '-'
%
%    Output:
%    fileNameBase - part of file name before the delimeter
%    dataDir      - path to directory, e.g., 'C:/'
%    fileExt      - file extension, e.g., '.aiff'

if nargin < 1
    delimeter = '-';
end

[file,dataDir] = uigetfile(...
    {'*.wav;*.ogg;*.flac;*.au;*.mp3;*.aac;*.aiff',...
    'Sound files (*.wav, *.ogg, *.flac, *.au, *.mp3, *.aac, *.aiff)';...
    '*.*',  'All Files (*.*)'},...
    'Select one of the sound files in the set');
[~,fileName,fileExt] = fileparts(file);
fileNameBase = strtok(fileName,delimeter);
fileNameBase = [fileNameBase delimeter];
end
```

A.3 readaudio

Note that the built-in Matlab-function `audioread` is currently not available in Octave.

```
function [a,sr] = readaudio(fileNameBase,fileExtension,mm,channels)
%READAUDIO - Read audio files
%
%    This function returns a matrix containing audio data from the files
%    specified with the input arguments. All files must have the same
%    sample rate and names ending with 1,..,mm
%
%    [a,sr] = READAUDIO(fileNameBase,fileExtension,mm)
%
%    Input:
%    fileNameBase  - filenamebase including path, e.g, 'C:/foo-' if the
%                     filenames are 'C:/foo-1.aiff','C:/foo-2.aiff',...
%    fileExtension - file extension, e.g., '.aiff'
%    channels      - vector with channel numbers, e.g., 1:8, or [2,3,5,8]
%
%    Output:
%    a  - matrix where each row contains the samples of corresponding file
%    sr - sample rate

ainfo = audioinfo([fileNameBase num2str(channels(1)) fileExtension]);
sr = ainfo.SampleRate;
a = zeros(mm,ainfo.TotalSamples);
for k = channels
    a(k,:) = audioread([fileNameBase num2str(k) fileExtension]);
end
end
```

A.4 gccscores

Edit out the call to `single` for double precision.

```
function scores = gccscores(a,settings)
%GCCSCORES - Genereal cross-correlation
%
%    This function calculates the GCC for all pairs of frames.
%
%    scores = GCCSCORES(a,settings)
%
%    Input:
%    a        - matrix where each row contains samples of a signal for
%               different receivers
%    settings - struct that must contain:
%      .mm               - number of columns in scores
%      .channels         - indeces of rows in a to compute
%      .frameSize        - number of columns in a frame
%      .frameOverlap     - overlap between frames
%      .firstSamplePoint - column index for the center of the first frame
%      .wf               - weighting function for the GCC
%      .sw               - half width of frame to consider
%
%    Output:
%    scores - cell array containing matrices where each column is the
%             calculated between corresponding frames of two channels

%Calculating matrices containing the DFT of all frames:
fftframes = cell(1,settings.mm);
for k = settings.channels
    frames = single(vector2frames(a(k,:),settings.frameSize,...
        settings.frameOverlap,settings.firstSamplePoint));
    %frames = frames-repmat(mean(frames),size(frames,1),1);
    fftframes{k} = fft(frames,2*size(frames,1)-1);
end
clear frames;

%Getting indeces indu only for the pairs that we have to calculate:
ii = zeros(1,settings.mm);
ii(settings.channels) = 1;
ind = logical(ii'*ii);
indu = triu(ind);

%Cross-correlation for all pairs with indeces indu:
scores = cell(settings.mm);
```

```
for k = find(indu)'
    [ii,jj] = ind2sub(size(ind),k);
    tmp = fftframes{jj}.*conj(fftframes{ii});
    tmp = fftshift(ifft(tmp.*settings.wf(tmp)),1);
    scores{ii,jj} = tmp(round(end/2)-settings.sw:round(end/2)+settings.sw,:);
end
clear fftframes;

%Filling in the pairs not in indu by mirroring the data:
tmp = cellfun(@flipud,scores','UniformOutput',false);
scores(tril(ind,-1)) = tmp(tril(ind,-1));
end
```

A.5 scoreplot

For plotting the GCC-PHAT data. Is set to only plot positive values and to clip colors for high values.

```
function scoreplot(score,settings)
%SCOREPLOT - Plots gray-scale version of GCC-PHAT data

c = 4; %set to 1 not to clip any colors
mm = max(max(score))/c;
imagesc(score,[0,mm]),colormap(gray)

setaxes(settings)
end
```

A.6 setaxes

Changes ticks. Not necessary if using a different call to `imagesc` in `scoreplot` and scaling the rest of the data.

```
function setaxes(settings)
%SETAXES - Scales axes ticks
%
%    This function scales the axes of a plot.
%
%    SETAXES(settings)
%
%    Input:
%    settings - struct that must contain:
%      .nbrOfFrames  - number of frames
%      .nbrOfSamples - number of samples in the signal
%      .sr           - sample rate of the signal
%      .sw           - half width of frame to consider
%      .v            - propagation speed of the signal

axis equal

%x-axis:
xlim([1,settings.nbrOfFrames])
x = 1:settings.nbrOfFrames;
t = x*settings.nbrOfSamples/settings.nbrOfFrames/settings.sr;
tticks = 5*unique(fix(t/5));
```

```
tind = arrayfun(@(x) find(t-x>0,1,'first'),tticks);
set(gca,'XTick',x(tind));
set(gca,'XTickLabel',tticks);
xlabel('Time')

%y-axis:
ylim([1,2*settings.sw+1])
y = 1:2*settings.sw+1;
u = (y-settings.sw)/settings.sr*settings.v;
uticks = 1*unique(fix(u/1));
uind = arrayfun(@(x) find(u-x>0,1,'first'),uticks);
set(gca,'YTick',y(uind));
set(gca,'YTickLabel',uticks);
set(gca,'YDir','reverse');
ylabel('Range-difference')
end
```

A.7 vector2frames

Could use `buffer` directly if not caring about the parameter `start`.

```
function F = vector2frames(v,frameSize,frameOverlap,start)
%VECTOR2FRAMES - Similar to buffer
%
%    This function divides the input vector v into frames organized in a
%    matrix.
%
%    F = VECTOR2FRAMES(v,frameSize,frameOverlap,start)
%
%    Input:
%    v            - vector
%    frameSize    - length of frame
%    frameOverlap - overlap between frames
%    start        - center of first frame
%
%    Output:
%    F - frames organized as columns in a matrix

if nargin < 4
    start = 1;
end

v = v(:); %ensure v is a column vector

hw = round((frameSize-1)/2);
if frameOverlap > 0
    n = hw-frameOverlap;
else
    n = hw;
end

if (start <= hw)
    z = zeros(hw-start+1,1);
    v = [z; v];
    start = hw+1;
end

F = buffer(v(start-n:end),frameSize,frameOverlap,...
    v(start-hw:start-hw+frameOverlap-1));
end
```

A.8 getdelays

Could be shortened if the optimization for the special case of one peak was handled in `localmax` instead.

```
function u = getdelays(scores,settings)
%GETDELAYS - Find local maxima
%
%    This function finds local maxima for the input.
%
%    u = GETDELAYS(scores,settings)
%
%    Input:
%    scores   - cell array conataining matrices
%    settings - struct that must conatin:
%      .mm            - number of columns in scores
%      .channels      - indeces of columns in scores to compute
%      .minPeakHeight - minimum value for local maxima
%      .sw            - should be round((size(scores{ch},1)-1)/2)
%      .nbrOfPeaks    - max number of local maxima to return
%
%    Output:
%    u - cell array of size mm x mm containing matrices

u = cell(settings.mm);

%Getting indeces indu only for the pairs that we have to calculate:
ii = zeros(1,settings.mm);
ii(settings.channels) = 1;
ind = logical(ii'*ii);
indu = triu(ind);

if settings.nbrOfPeaks == 1 %special case for 1 peak (for optimization)
    for k = find(indu)'
        [v,ii] = max(scores{k});
        ii(v < settings.minPeakHeight) = NaN;
        u{k} = ii-settings.sw-1;
    end
else
    for k = find(indu)'
        [~,ii] = localmax(scores{k},settings.minPeakHeight,settings.nbrOfPeaks);
        u{k} = ii-settings.sw-1;
    end
end

%Filling in the pairs not in indu by mirroring the data:
tmp =  cellfun(@(A) -A,u','UniformOutput',false);
u(tril(ind,-1)) = tmp(tril(ind,-1));
end
```

A.9 localmax

This could be implemented using `findpeaks`. Sorting all the values even when just returning a few is not optimal.

```
function [v,ind] = localmax(A,minPeakValue,nbrOfPeaks)
%FINDLOCALMAX - Find local maxima
%
```

```
%    This function finds local maxima for each column in the input matrix.
%
%    [v,ind] = LOCALMAX(A,minPeakValue,nbrOfPeaks)
%
%    Input:
%    A             - matrix
%    minPeakValue - minimum value for local maxima
%    nbrOfPeaks   - max number of local maxima to return (for each column)
%
%    Output:
%    v   - value vectors for local maxima
%    ind - index vectors for local maxima

%Finding local maxima:
n = size(A,2);
df = [diff(A); zeros(1,n)];
df = sign(df);
dfdf = [zeros(1,n); diff(df)];

ind = dfdf < 0; %indeces of local maxima
v = -Inf(size(A));
v(ind) = A(ind);

%Sorting local maxima:
[v,ind] = sort(v,'descend');
v = v(1:nbrOfPeaks,:);
ind = ind(1:nbrOfPeaks,:);

%Setting local maxima < minPeakValues to NaN:
badind = v < minPeakValue;
v(badind) = NaN;
ind(badind) = NaN;
end
```

A.10 matchingdelays

Uses equation 2.2 to find inliers.

```
function uref = matchingdelays(u,settings)
%MATCHINGDELAYS - Limits the delay candidates
%
%    This function...
%
%    uref = MATCHINGDELAYS(u,settings)
%
%    Input:
%    u  - delay data output from getdelays
%    settings - struct that must have...
%
%    Output:
%    uref - delays for

channels = settings.channels;
refChannel = settings.refChannel;
[m,n] = size(u{channels(1),channels(1)});

uref = cell(1,settings.mm);
for ch = channels(channels~=refChannel)
    candidates = NaN(1+m*m*(numel(channels)-2),n);
    candidates(1:m,:) = u{refChannel,ch};
    %Using u{refChannel,ch} = urefChannel{refChannel,k}-u{ch,k}:
```

```
    pp = 1; %loop counter
    for k = setdiff(channels,[refChannel,ch])
        candidates(2+(pp-1)*m*m:pp*m*m+1,:) = dfcombs(u{refChannel,k},u{ch,k});
        pp = pp+1;
    end
    %Finding the candidates that have at least minNbrOfInliers inliers:
    [bin,binNbr] = histc(candidates,-settings.sw:settings.binSize:settings.sw);
    ind = bin > settings.minNbrOfInliers;
    urch = NaN(max(sum(ind)),size(bin,2));
    for cc = find(sum(ind) > 0)
        pp = 1; %loop counter
        for k = find(ind(:,cc))'
            urch(pp,cc) = median(candidates(binNbr(:,cc) == k,cc));
            pp = pp+1;
        end
    end
    uref{ch} = urch;
end

end

%Help function:
function combs = dfcombs(A,B)
%...
%  A, B, same size

[m,n] = size(A);
combs = NaN(m*m,n);
for k = 1:m
    combs(1+(k-1)*m:k*m,:) = A-B;
    B = circshift(B,-1);
end
end
```

A.11 fitdelayswithransac

Doesn't use perform any calculations for the last frame.

```
function [delays,lines,ind] = fitdelayswithransac(u,settings)
%FITDELAYSWITHRANSAC - Non-repeating RANSAC
%
%    This function performs a non-repeating RANSAC to find line segments.
%
%    [delays,lines,ind] = FITDELAYSWITHRANSAC(u,settings)
%
%    Input:
%    u         - cell array of matrices where each column represents an
%                instance in time with a number of values (or NaN)
%    settings - struct that must contain:
%      .channels               - indeces of u to compute
%      .refChannel             - reference index of u
%      .RANSACframeSize        - number of columns in a frame
%      .RANSACframeOverlap     - overlap
%      .RANSACmaxNbrOfGroups   - max number of line segments to return
%      .RANSACnbrOfIterations - number of RANSAC iterations
%      .RANSACmaxSlope         - max allowed abs derivative
%      .RANSACminNbrOfInliers - min number inliers for a line segment
%      .RANSACinlierThreshold - distance to line threshold
%      .RANSACsharedPointsThreshold - max number of shared points between
%                                     returned line segments
%
```

```
%    Output:
%    delays - matrix containing the values in A correspoinding to the
%             found line segments row by row sorted by the number of inliers
%    lines  - matrix containing the values for the found line segments row
%             by row sorted by the number of inliers
%    ind    - matrix containing the column indeces of all frames as columns

channels = settings.channels;
refChannel = settings.refChannel;

frameSize = settings.RANSACframeSize;
frameOverlap = settings.RANSACframeOverlap;
maxNbrOfGroups = settings.RANSACmaxNbrOfGroups;

start = round((frameSize+1)/2);
ind = vector2frames(1:max(cellfun('length',u)),frameSize,frameOverlap,start);
ind = ind(:,1:end-1); %since there might be zeros in the last column
n = size(ind,2);

delays = cell(1,settings.mm);
lines = cell(1,settings.mm);

for ch = channels(channels~=refChannel)
    delays(ch) = {NaN(maxNbrOfGroups,frameSize,n)};
    lines(ch) = {NaN(maxNbrOfGroups,frameSize,n)};
    for k = 1:n
        [delays{ch}(:,:,k),lines{ch}(:,:,k)] = ...
            ransacline(u{ch}(:,ind(:,k)),settings);
    end
end
end
```

A.12 ransacline

Non-repeating RANSAC algorithm.

```
function [delays,lines] = ransacline(A,settings)
%RANSACLINE - Non-repeating RANSAC
%
%    This function performs a non-repeating RANSAC to find line segments.
%
%    [delays,lines] = RANSACLINE(A,settings)
%
%    Input:
%    A        - matrix where each column represents an instance in time with
%               a number of values (or NaN)
%    settings - struct that must conatin:
%      .RANSACmaxNbrOfGroups   - max number of line segments to return
%      .RANSACnbrOfIterations - number of RANSAC iterations
%      .RANSACmaxSlope         - max allowed abs derivative
%      .RANSACminNbrOfInliers - min number inliers for a line segment
%      .RANSACinlierThreshold - distance to line threshold
%      .RANSACsharedPointsThreshold - max number of shared points between
%                                     returned line segments
%    Output:
%    delays - matrix containing the values in A correspoinding to the
%             found line segments row by row sorted by the number of inliers
%    lines  - matrix containing the values for the found line segments row
%             by row sorted by the number of inliers

delays = NaN(settings.RANSACmaxNbrOfGroups,size(A,2));
```

```
lines = NaN(settings.RANSACmaxNbrOfGroups,size(A,2));

tmp = find(sum(~isnan(A)));
if numel(tmp) < settings.RANSACminNbrOfInliers
    return
end

%Non-NaN points in A:
[rows,cols] = find(~isnan(A));
points = [cols'; A(sub2ind(size(A),rows,cols))'];

%All possible combinations:
n = length(points);
[r,c] = meshgrid(1:n,1:n);
pairs = unique(sort([r(:),c(:)],2)','rows');
pairs = pairs(:,randperm(length(pairs))); %shuffling

%RANSAC:
nbrOfIterations = settings.RANSACnbrOfIterations;
tmpdelays = NaN(nbrOfIterations,size(A,2));
tmplines = NaN(nbrOfIterations,size(A,2));
for k = 1:min(settings.RANSACnbrOfIterations,length(pairs))
    %Line using two random points:
    p = points(:,pairs(:,k));
    ll = null([p' ones(2,1)]);
    okSlope = abs(ll(1)/ll(2)) < settings.RANSACmaxSlope;
    if ~okSlope
        continue
    end
    distances = abs(ll'*[points;ones(1,size(points,2))])/norm(ll(1:2));
    ind = distances < settings.RANSACinlierThreshold;
    enoughInliers = sum(ind) >= settings.RANSACminNbrOfInliers;
    if enoughInliers
        %Line fitting using all the inliers:
        pp = [points(1,ind)' ones(size(points(1,ind)'))]\points(2,ind)';
        ll = [pp(1)/pp(2); -1/pp(2); 1];
        x = 1:size(A,2);
        y = (-ll(3)-ll(1)*x)/ll(2);
        distances = abs(ll'*[points;ones(1,size(points,2))])/norm(ll(1:2));
        ind = distances < settings.RANSACinlierThreshold;
        tmpdelays(k,points(1,ind)) = points(2,ind);
        tmplines(k,:) = y;
    end
end

%Deleting NaN-rows:
ind = ~all(isnan(tmpdelays),2);
tmpdelays = tmpdelays(ind,:);
tmplines = tmplines(ind,:);

%Getting rid of duplicate lines:
for k = 1:settings.RANSACmaxNbrOfGroups
    if isempty(tmpdelays)
        return
    end
    nbrOfInliers = sum(~isnan(tmpdelays),2);
    [~,ind] = max(nbrOfInliers);
    delays(k,:) = tmpdelays(ind,:);
    lines(k,:) = tmplines(ind,:);
    ind = ~(sum(ismember(tmpdelays,delays(k,:)),2) > ....
        settings.RANSACsharedPointsThreshold);
    tmpdelays = tmpdelays(ind,:);
    tmplines = tmplines(ind,:);
end
end
```

A.13 connectlines

Only works for positive values of `settings.RANSACframeOverlap`.

```
function [delaysegments,linesegments] = connectlines(delays,lines,ind,settings)
%CONNECTLINES - Connects line segments
%
%    This function connects line segments to form longer tracklets.
%
%    [delaysegments,linesegments] = CONNECTLINES(delays,lines,ind,settings)
%
%    Input:
%    delays - cell array
%    lines  - cell array
%    ind    - matrix
%    settings         - struct that must contain:
%      .mm                     - number of columns in scores
%      .channels               - indeces of columns in scores to compute
%      .refChannel             - reference index
%      .RANSACframeOverlap     - overlap
%      .lineDistanceThreshold - threshold
%
%    Output:
%    delaysegments - cell array
%    linesegments  - cell array

channels = settings.channels;
refChannel = settings.refChannel;

frameOverlap = settings.RANSACframeOverlap;

delaysegments = cell(1,settings.mm);
linesegments = cell(1,settings.mm);

for ch = channels(channels~=refChannel)

    delaysegments(ch) = {{NaN(size(ind))}};
    linesegments(ch) = {{NaN(size(ind))}};

    pp = numel(delaysegments{ch});
    prev = cellfun(@(c) c(end-floor(frameOverlap/2),1),linesegments{ch})';
    for k = 1:size(ind,2)
        curr = lines{ch}(:,frameOverlap-floor(frameOverlap/2),k);
        lineInd = find(~isnan(curr));
        tmpMin = Inf(size(prev));
        for j = lineInd'
            [mv,tmp] = nanmin(abs(prev-curr(j)));
            if (mv < min(settings.lineDistanceThreshold,tmpMin(tmp)))
                %Connect to old segment:
                delaysegments{ch}{tmp}(:,k) = delays{ch}(j,:,k)';
                linesegments{ch}{tmp}(:,k) = lines{ch}(j,:,k)';
                tmpMin(tmp) = mv;
            else
                %New segment:
                delaysegments{ch}{pp}(:,k) = delays{ch}(j,:,k)';
                linesegments{ch}{pp}(:,k) = lines{ch}(j,:,k)';
                delaysegments{ch}(pp+1) = {NaN(size(ind))};
                linesegments{ch}(pp+1) = {NaN(size(ind))};
                pp = pp + 1;
            end
        end
        prev = cellfun(@(c) c(end-floor(frameOverlap/2),k),linesegments{ch})';
    end
    if pp > 1
        delaysegments{ch} = delaysegments{ch}(1:end-1);
        linesegments{ch} = linesegments{ch}(1:end-1);
    end
end
end
```

A.14 lineplot

Some hacks to remove NaN-values from the plot to reduce the size of exported vector graphics.

```
function lineplot(u,ind,settings)
%LINEPLOT - Plots line segments

marks = '+o*.xsd^v><ph';
colors = hsv(12);

rows = find(~all(all(isnan(u),3),2))';
n = rows(end);

p = zeros(n,1);

hold on
for k = 1:n
    mar = marks(mod(k-1,numel(marks))+1);
    col = colors(k,:);
    tmp = squeeze(u(k,:,:)+settings.sw+1);
    tmpind = ~isnan(tmp);
    tmp = plot(ind(tmpind),tmp(tmpind),mar,'Color',col);
    p(k) = tmp(1);
end
legend(p,'1st line','2nd line','3rd line','4th line','5th line')
setaxes(settings)
end
```

A.15 connectsegments

There are some ugly hacks here that should be fixed.

```
function [newdelaysegments,newlinesegments] = ...
    connectsegments(delaysegments,linesegments,ind,uref,settings)
%CONNECTSEGMENTS - Connects tracklets
%
%    This function connects tracklets to form longer tracklets.
%
%    [newdelaysegments,newlinesegments] = ...
%      CONNECTSEGMENTS(delaysegments,linesegments,ind,uref,settings)
%
%    Input:
%    delaysegments - cell array
%    linesegments  - cell array
%    ind           - matrix
%    uref          - cell array
%    settings         - struct that must contain:
```

```
%      .mm                     - number of columns in scores
%      .channels               - indeces of columns in scores to compute
%      .refChannel             - reference index
%      .RANSACframeOverlap     - overlap
%      .lineDistanceThreshold - threshold
%
%    Output:
%    newdelaysegments - cell array
%    newlinesegments  - cell array

channels = settings.channels;
refChannel = settings.refChannel;

newdelaysegments = cell(1,settings.mm);
newlinesegments = cell(1,settings.mm);

for ch = channels(channels~=refChannel)

    newdelaysegments{ch} = cell(size(delaysegments{ch}));
    newlinesegments{ch} = cell(size(newlinesegments{ch}));

    n = numel(linesegments{ch});

    u = uref{ch};

    for k = 1:n
        segment0 = linesegments{ch}{k};
        index = find(~isnan(segment0),settings.linesOverlap,'last');
        point0 = [ind(index(1)); segment0(index(1))];
        rMax = -Inf;
        for j = k+1:n
            segment1 = linesegments{ch}{j};
            index = find(~isnan(segment1),settings.linesOverlap,'first');
            point1 = [ind(index(end)); segment1(index(end))];
            if point0(1) > point1(1)
                tmp0 = frames2vector(segment0,settings.RANSACframeOverlap,...
                    round((settings.RANSACframeSize+1)/2));
                tmp1 = frames2vector(segment1,settings.RANSACframeOverlap,...
                    round((settings.RANSACframeSize+1)/2));
                index = point1(1):point0(1);
                r = sum(abs(tmp0(index)-tmp1(index)) < 10)/norm(point0-point1);
            else
                ll = null([point0' 1; point1' 1]);
                [rows,cols] = find(~isnan(u));
                index = logical((cols > point0(1)).*(cols < point1(1)));
                rows = rows(index);
                cols = cols(index);
                points = [cols'; u(sub2ind(size(u),rows,cols))'];
                distances = abs(ll'*[points;ones(1,size(points,2))])/norm(ll(1:2));
                r = sum(distances < settings.linesInlierThreshold)/norm(point0-point1);
                if norm(point0-point1) > 300 %...fult
                    r = 0;
                end
            end
            if r > max(settings.linesInlierRatio,rMax)
                rMax = r;
                %Add segment k...
                newdelaysegments{ch}{k} = delaysegments{ch}{k};
                index = ~isnan(delaysegments{ch}{j});
                newdelaysegments{ch}{k}(index) = delaysegments{ch}{j}(index);
                %...and segment j:
                newlinesegments{ch}{k} = linesegments{ch}{k};
                index = ~isnan(linesegments{ch}{j});
                newlinesegments{ch}{k}(index) = linesegments{ch}{j}(index);
                %Copy new connected segments to j:
```

```
                delaysegments{ch}(j) = newdelaysegments{ch}(k);
                linesegments{ch}(j) = newlinesegments{ch}(k);
                continue
            elseif j == n
                newdelaysegments{ch}(k) = delaysegments{ch}(k);
                newlinesegments{ch}(k) = linesegments{ch}(k);
            end
        end
    end
end
end
```

A.16 frames2vector

Should work for negative overlaps too.

```
function v = frames2vector(F,frameOverlap,start)
%FRAMES2VECTOR - Frames back to vector
%
%    This function reconstructs a vector from frames.
%
%    v = FRAMES2VECTOR(F,frameOverlap,start)
%
%    Input:
%    F            - matrix with frames as columns
%    frameOverlap - overlap between frames
%    start        - center of first frame
%
%    Output:
%    v - vector

frameSize = size(F,1);
hw = (frameSize-1)/2;

if frameOverlap >= 0
    v = F(1:end-frameOverlap,:);
    v = v(:);
    if start <= hw
        v = v(hw-start+2:end);
    end
    v = [v; F(end-frameOverlap+1:end,end)];
else
    F = [zeros(-frameOverlap,size(F,2)); F];
    v = F(1:end,:);
    v = v(:);
end
end
```

A.17 segmentplot

Some hacks to remove NaN-values from the plot to reduce the size of exported vector graphics.

```
function segmentplot(scores,u,ind,f,settings)
%SEGMENTPLOT - Plots tracklets
```

```
marks = '+o*.xsd^v><ph';
for ch = settings.channels(settings.channels~=settings.refChannel)
    subplot(2,round(settings.mm/2),ch)
    nel = cellfun(@(c) numel(find(~isnan(c))),u{ch});
    indd = f(nel);
    colors = hsv(numel(indd));
    %scoreplot(scores{ch},settings)
    hold on
    pp = 1;
    for kk = indd
        mar = marks(mod(pp-1,numel(marks))+1);
        col = colors(pp,:);
        tmp = u{ch}{kk};
        tmpind = ~isnan(tmp);
        plot(ind(tmpind),tmp(tmpind)+settings.sw+1,mar,'color',col);
        pp = pp + 1;
    end
    setaxes(settings)
end
end
```

A.18 smoothdelays

There are many other built-in smoothers in Matlab that might work better, needs more testing.

```
function urefsmooth = smoothdelays(newdelaysegments,newlinesegments,uref,settings)
%SMOOTHDELAYS - Smooths input data
%
%    This function smooths the input data.
%
%    urefsmooth = SMOOTHDELAYS(newdelaysegments,newlinesegments,uref,settings)
%
%    Input:
%    newdelaysegments - cell array
%    newlinesegments  - cell array
%    uref             - cell array
%    settings         - struct that must contain:
%      .mm                   - number of columns in scores
%      .channels             - indeces of columns in scores to compute
%      .refChannel           - reference index
%      .RANSACframeOverlap   - overlap
%      .RANSACframeSize      - number of columns in a frame
%      .smoothingDegree      - degree of smoothing
%      .smoothingDistance    - inliers threshold
%
%    Output:
%    urefsmooth - cell array

channels = settings.channels;
refChannel = settings.refChannel;

longestdelaysegments = cell(1,settings.mm);
longestlinesegments = cell(1,settings.mm);
for ch = channels(channels~=refChannel)
    nel = cellfun(@(c) numel(find(~isnan(c))),newlinesegments{ch});
    [~,indd] = max(nel);
    longestdelaysegments(ch) = newdelaysegments{ch}(indd);
    longestlinesegments(ch) = newlinesegments{ch}(indd);
end
```

```
urefsmooth = cell(1,settings.mm);

for ch = channels(channels~=refChannel)

    ur = uref{ch};

    u = frames2vector(longestdelaysegments{ch},settings.RANSACframeOverlap,...
        round((settings.RANSACframeSize+1)/2));

    %Fit spline:
    x = find(~isnan(u));
    y = smooth(x,u(x),settings.smoothingDegree);
    xx = x(1):x(end);
    yy = spline(x,y,xx);

    y = NaN(1,size(ur,2));
    y(xx) = yy;

    d = abs(ur-repmat(y,size(ur,1),1));
    cind = find(sum(d < settings.smoothingDistance) > 0);
    [~,rind] = min(d(:,cind));
    ind = sub2ind(size(ur),rind,cind);
    tmp = ur(ind);

    y(cind) = tmp;
    urefsmooth{ch} = y;
end
end
```

A.19 clipdata

This is not really optimal if the TDOA solver can handle missing data.

```
function uout = clipdata(u,settings)
%CLIPDATA - Clip input data
%
%    This function clips the input data.
%
%    uout = CLIPDATA(u,settings)
%
%    Input:
%    u        - cell array
%    settings - struct that must contain:
%      .channels   - indeces of columns in scores to compute
%      .refChannel - reference index
%
%    Output:
%    uout - matrix

leftLim = max(cell2mat(cellfun(@(c) find(~isnan(c),1,'first'),u,...
    'UniformOutput',false)));
rightLim = min(cell2mat(cellfun(@(c) find(~isnan(c),1,'last'),u,...
    'UniformOutput',false)));

uout = NaN(settings.mm,max(cellfun('length',u)));
for ch = settings.channels(settings.channels~=settings.refChannel)
    uout(ch,leftLim:rightLim) = u{ch}(leftLim:rightLim);
end
uout(settings.refChannel,leftLim:rightLim) = 0;
end
```

Appendix B

Videos and 3D Figures

Figure B.1 and B.2 are intended to be viewed in a PDF viewer with multimedia support.

Figure B.1: Interactive version of Figure 5.2.

Figure B.2: Click to play video version of Figure 5.3.

Master’s Theses in Mathematical Sciences 2015:E6
ISSN 1404-6342

LUTFMA-3271-2015

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/