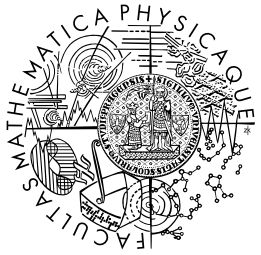


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jan Hrach

Passive emitter tracking

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	Mgr. David Klusáček, Ph.D.
Study programme:	Informatics
Study branch:	Software and data engineering

Prague 2019

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. This thesis was not used to achieve an academic grading elsewhere.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague, 2019-05-05 signature of the author

I would like to thank to my supervisor for great insights, tips and advice, and to SPOJE.NET and Wendulka.net ISPs for providing rooftop hosting to make this project possible.

Title: Passive emitter tracking

Author: Jan Hrach

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. David Klusáček, Ph.D., Institute of Formal and Applied Linguistics

Abstract: We have implemented a TDOA multilateration of transmitters on an unmodified rtl-sdr receiver using transmitters with known location as a timing reference. We present a brief theoretical background and describe the measurement process which includes several approaches that correct the timing and frequency errors between the receivers. Additionally, we have implemented an angle of arrival direction finder using coherent rtl-sdr.

Keywords: TDOA multilateration sdr rtl-sdr

Contents

Introduction

Passive emitter tracking

We summarize methods of location of uncooperative radio emitters and implement two of them on commonly available hardware (namely, a rtl-sdr radio).

"Uncooperative emitter" means a radio transmitter over which we have no control — e.g. a commercial broadcast radio station — and therefore we cannot instruct it to broadcast a special signal for easier location (such as precise timestamps or a specially crafted pseudorandom sequence as for example GPS does). We do not consider transmitters that are explicitly trying to avoid being located — a plethora of countermeasures and counter-countermeasures can be thought of.

"Passive" means that no transmitting from our side is involved. Therefore, our device does not require any regulatory license. This is a rather common approach when locating transmitters, though, and being "passive" is considered a special feature when tracking non-transmitting objects (based on reflections and disturbance of already existing ambient signals) such as airplanes with radios turned off[1] or even people[2].

Applications

Emitter location is used for example in the sport of radio fox hunting, where transmitters are hidden within a designated area and players are to find and recover them. Related activity is weather probe recovery — a probe from a bursted baloon falls to the ground while still transmitting, and if its location is known, it can be recovered.

A more serious task is a supervision of the radio spectrum by a regulatory body. Locating transmitters in bands with a "general license" can help to resolve radio interference problems and in licensed bands legal regulations can be enforced.

Tracking airplanes (or, more precisely, their radios) can serve various means, from hobbyist to military.

IoT networks such as Sigfox offer geolocation of the device based only on the received signal. This task traditionally required a GPS receiver on the device, increasing cost and energy consumption.

Emergency Position Indicating Radio Beacon (EPIRB) is a floating device automatically released from a sinking ship. It transmits signal that is picked up by rescue crews and the location is then determined using multilateration and direction finding.

Structure of this work

- In Chapter 1 we lay out the basics of digital signal processing. The chapter should make the reader familiar with the framework used later in this thesis.

- In Chapter 2 we describe multilateration by measuring the time difference of arrival (TDoA). We lay out the necessary theory and then proceed to implementation. Appendix A contains a user guide on how to use the resulting software, Appendix B technical details regarding the software and Appendix C the results and an example measurement protocol.
- In Chapter 3 we describe three approaches to measuring an angle of arrival (AoA). Their advantages and drawbacks are described and then the most promising one is implemented. Appendix D contains a user guide for the AoA software.

1. Introduction to digital signal processing

1.1 Signal model, from real to complex

We will model the signal received by the antenna as a sum of finitely many (M) cosine waves with various amplitudes b_k, frequencies f_k and phases ϕ_k, sampled at discrete time. That is, it is a function $\mathbb{Z} \to \mathbb{R}$. This approach is sufficient for our usage and we can side-step more complicated continuous-time signal theory and calculus. The discrete time sampling corresponds to the fact that we get the signal from our radio sampled by the ADC, and the finite sum of cosines can reasonably approximate the continuous frequency spectrum (c.f. approximation of smooth and periodic functions by Fourier series).

$$s_n = \sum_{k=0}^{M} b_k \cos(2\pi f_k n + \phi_k) \tag{1.1}$$

s_n is a a vector (of infinite length) indexed by n, one can imagine this as a n-th sample. We will assume the sampling frequency to be 1 [Hz] to simplify the equations. This can be later scaled if needed.

It is convenient to represent the signal in a complex form. We will substitute the Euler's trigonometric formula for $\cos(2\pi f_k n + \phi_k)$:

$$\begin{aligned}
\cos x &= \frac{e^{ix} + e^{-ix}}{2} \\
s_n &= \sum_{k=0}^{M} b_k \frac{e^{i(2\pi f_k n + \phi_k)} + e^{-i(2\pi f_k n + \phi_k)}}{2} \\
&= \sum_{k=0}^{M} \left(b_k \frac{e^{2\pi i f_k n + i\phi_k}}{2} + b_k \frac{e^{-2\pi i f_k n - i\phi_k}}{2} \right) \\
&= \sum_{k=0}^{M} \left(\overbrace{\frac{b_k}{2} e^{i\phi_k}}^{a_{2l}} e^{2\pi i \overbrace{f_k}^{f_{2l}} n} + \overbrace{\frac{b_k}{2} e^{-i\phi_k}}^{a_{2l+1}} e^{-2\pi i \overbrace{f_k}^{-f_{2l+1}} n} \right) \\
&= \sum_{l=0}^{M} \left(a_{2l} e^{2\pi i f_{2l} n} + a_{2l+1} e^{2\pi i f_{2l+1} n} \right) \\
&= \sum_{l=0}^{N} a_l e^{2\pi i f_l n} \text{ for } N = 2M+1, a_{2l} = \overline{a_{2l+1}} \text{ and } f_{2l} = -f_{2l+1}
\end{aligned} \tag{1.2}$$

These conditions ($a_{2l} = \overline{a_{2l+1}}$, $f_{2l} = -f_{2l+1}$) are satisfied by real signals, but we will use this model ($s_n = \sum_{l=0}^{N} a_l e^{2\pi i f_l n}$) to describe complex signals too (which we will generate in a moment) and our results will be valid for these too. Therefore, a general signal is from now on a function $s_n : \mathbb{Z} \to \mathbb{C}$.

Now, in the next few sections, we will show some basic operations with signals.

1.2 Frequency shift and time reversal

Consider signal $s_n = \sum_{k=0}^{N} a_k e^{2\pi i f_k n}$. Multiplying it pointwise by $r_n = e^{-2\pi i f n}$ results in signal $\widehat{s}_n$ where all frequencies are shifted to the left by f (i.e., the frequency f of the original signal becomes zero, and this is usually called down-conversion):

$$\widehat{s}_n = s_n \odot r_n = \sum_{k=0}^{N} a_k e^{2\pi i f_k n} \odot e^{-2\pi i f n} = \sum_{k=0}^{N} a_k e^{2\pi i (f_k - f) n} \tag{1.3}$$

The signal $\widehat{s}_n$ is (except for special cases) no longer real: the requirement $f_{2l} = -f_{2l+1}$ no longer holds in general.

For convenience, define a time-reversing operator ρ, $\rho\, s_n = s_{-n}$. Again, let us examine the result:

$$\rho\, s_n = s_{-n} = \sum_{k=0}^{N} a_k e^{2\pi i f_k (-n)} = \sum_{k=0}^{N} a_k e^{2\pi i (-f_k) n} \tag{1.4}$$

The amplitudes remained the same, however, the frequencies changed their sign. If the signal was defined by $A = \{a_k,\, k \in \{0 \dots N\}\}$ and $F = \{f_k,\, k \in \{0 \dots N\}\}$, it is now defined by A and $-F$.

1.3 Convolution and FIR filters

Convolution of two signals h and x is defined as

$$(h * x)_n = \sum_{m=-\infty}^{\infty} h_m x_{n-m}. \tag{1.5}$$

Finitely supported vector h is sometimes called *FIR filter*[1] and the numbers h_k are its *taps*.

We will show properties of convolution, but some of them only for finite signals. A more complicated reasoning would be needed to assure infinite sums convergence. So, altough, all the sums go from $-\infty$ to ∞, they should be thought of as being in fact finite because the signals are only finitely supported. Consequently, the summation order can be swapped freely.

- Additivity: $(f + h) * x = f * x + h * x$
- Commutativity: $h * x = \sum_{m=-\infty}^{\infty} h_m x_{n-m} \overset{l=n-m}{=} \sum_{l=-\infty}^{\infty} h_{n-l} x_l = x * h$

[1] Finite impulse response, as passing a signal through it affects only finitely many (the length of h) samples. This is in contrast with Infinite impulse response filters, which we will not use in this thesis.

- Associativity:

$$\begin{aligned}((h*g)*x)_n &= \sum_{m=-\infty}^{\infty}(h*g)_m x_{n-m} \\ &= \sum_{m=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} h_l g_{m-l} x_{n-m} \\ &= \sum_{l=-\infty}^{\infty}\sum_{m=-\infty}^{\infty} h_l g_{m-l} x_{n-m} \quad m = \widehat{m} + l \\ &= \sum_{l=-\infty}^{\infty}\sum_{\widehat{m}=-\infty}^{\infty} h_l g_{\widehat{m}} x_{n-\widehat{m}-l} \\ &= \sum_{l=-\infty}^{\infty}\sum_{m=-\infty}^{\infty} h_l g_m x_{n-l-m} \\ &= \sum_{l=-\infty}^{\infty} h_l (g*x)_{n-l} \\ &= (h*(g*x))_n\end{aligned} \tag{1.6}$$

1.3.1 Filtering

Let us examine convolution of h_n with signal containing only a single frequency: $s_n = ae^{2\pi i f n}$ (we can then generalize this to more complicated signals thanks to the additive property of convolution).

$$(h*s)_n = \sum_{k=-\infty}^{\infty} h_k a e^{2\pi i f(n-k)} = \underbrace{\sum_{k=-\infty}^{\infty} h_k e^{-2\pi i f k}}_{\text{constant factor } H_f} \underbrace{a e^{2\pi i f n}}_{s_n} \tag{1.7}$$

For $h_k = -h_{-k}\,\forall k$ the factors H_f are real. This is therefore a filter which attenuates or amplifies various frequencies. And as convolution is commutative and associative, one may also reason about the situation as if taps h_k were filtered by signal s_n.

A simple example of a filter is the moving average, defined by

$$h_k = 1/N \text{ for } k \in \{0 \ldots N-1\}, h_k = 0 \text{ otherwise.}$$

Intuitively, we would expect this filter will smooth the signal and therefore attenuate high frequencies. We can plot its H_f by computing (1.7) by definition:

```
N = 10
h = np.ones(N)/N
H = []
for f in np.linspace(0, 0.5, 100):
  acc = 0
  for k in range(len(h)):
    acc += h[k] * np.exp(-2*np.pi*1j*f*k)
  H.append(acc)
plt.plot(np.abs(H))
```

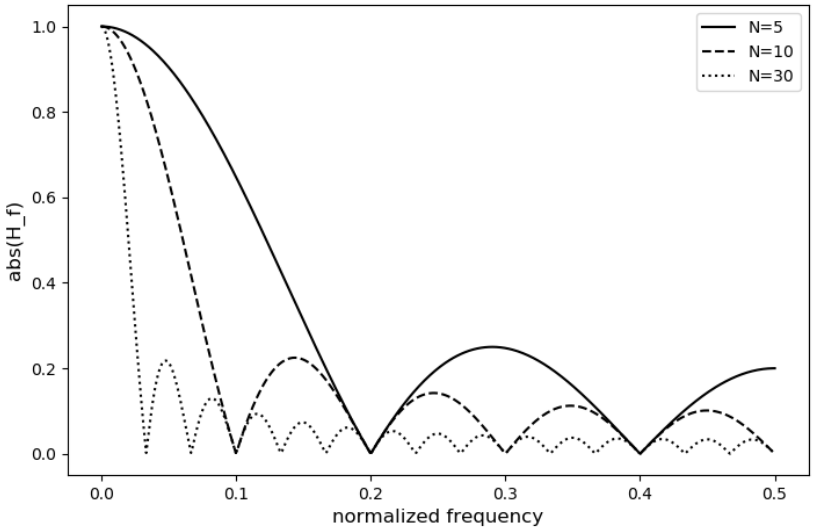


Figure 1.1: Frequency responses of moving average filters of various lengths

We use this to select radio transmitter at certain frequency and attenuate the others. Of course more advanced h_k than moving average must be used, as we can see that moving average exposes significant ripples.

1.4 Discrete Fourier transform and windowing

Discrete Fourier transform of signal s_n of length N is defined by

$$\mathrm{DFT}(s)_f = \sum_{n=0}^{N-1} s_n e^{-\frac{2\pi i f n}{N}}. \tag{1.8}$$

The inverse transform is defined by

$$\mathrm{IDFT}(X)_n = \frac{1}{N} \sum_{f=0}^{N-1} X_f \, e^{\frac{2\pi i f n}{N}}. \tag{1.9}$$

One can think of the DFT as a result of multiple convolutions of the signal with complex exponentials with frequencies from the range $[-0.5, 0.5)$. By taking the absolute value[2] of the result, we can infer "how much of a given frequency is in the signal" (usually called *spectrum of the signal*). Additionally, computing DFT by definition requires $\mathcal{O}(N^2)$ operations, but there exist algorithms[3] that are able to compute it in $\mathcal{O}(N \log N)$. We can exploit this for fast computation of arbitrary convolutions.

1.4.1 DFT and its inverse

Another formalism of DFT [3] is defining a matrix $F_N \in \mathbb{C}^{N \times N}$ and then executing the transform as a multiplication of the input vector with this matrix. Inverse

[2]More precisely, we take $10 \log_{10} \left(\frac{\mathrm{Re}(\mathrm{DFT}(s)_f)^2 + \mathrm{Im}(\mathrm{DFT}(s)_f)^2}{N} \right)$ to get the normalized power in decibels.

[3]Fast Fourier Transform (FFT)

DFT is then multiplying the vector with a Hermitian conjugate of the matrix.

$$(F_N)_{x,y} = \frac{1}{\sqrt{N}} e^{\frac{-2\pi i y x}{N}} \tag{1.10}$$

Using the matrix, we can show that the inverse transform is indeed an inverse, that is, F_N is unitary, $F_N F_N^H = I$. The innermost sum of the matrix multiplication is

$$\sum_{p=0}^{N-1} e^{\frac{-2\pi i p y}{N}} e^{\frac{2\pi i p x}{N}} = \sum_{p=0}^{N-1} e^{\frac{2\pi i p (x-y)}{N}} . \tag{1.11}$$

This is $\sum_{p=0}^{N-1} e^0 = N$ for $x = y$. For $x \neq y$, $e^{\frac{2\pi i p (x-y)}{N}} \neq 1$ (the exponent will be non-integer), and therefore we can apply the formula for the sum of a geometric series

$$\sum_{p=0}^{N-1} \left(e^{\frac{2\pi i (x-y)}{N}} \right)^p = \frac{1 - \left(e^{\frac{2\pi i (x-y)}{N}} \right)^N}{1 - e^{\frac{2\pi i (x-y)}{N}}} = \frac{1 - e^{2\pi i (x-y)}}{1 - e^{\frac{2\pi i (x-y)}{N}}}, \tag{1.12}$$

the numerator is the same as $1 - e^0$ and so the result is 0 for elements that are not on the diagonal. Therefore, $F_N F_N^H = F_N^H F_N = I$.

1.4.2 Windowing

The definition 1.1 and other methods consider signals of infinite length. However, all recordings in real life, the length of DFT that can be numerically computed, etc., are of course finite. And we may even want to artificially split the recording into shorter chunks, for example because we are interested in some properties that are changing over time[4]. We do this by multiplying the signal with a window function, which is a function that is nonzero on some interval (usually 100 to 10000 samples long) and zero elsewhere. This will extract a piece of the signal which we can further process. Even the naive approach of cutting the signal into pieces and computing DFT from each of them without any preprocessing is in fact an application of a rectangular window.

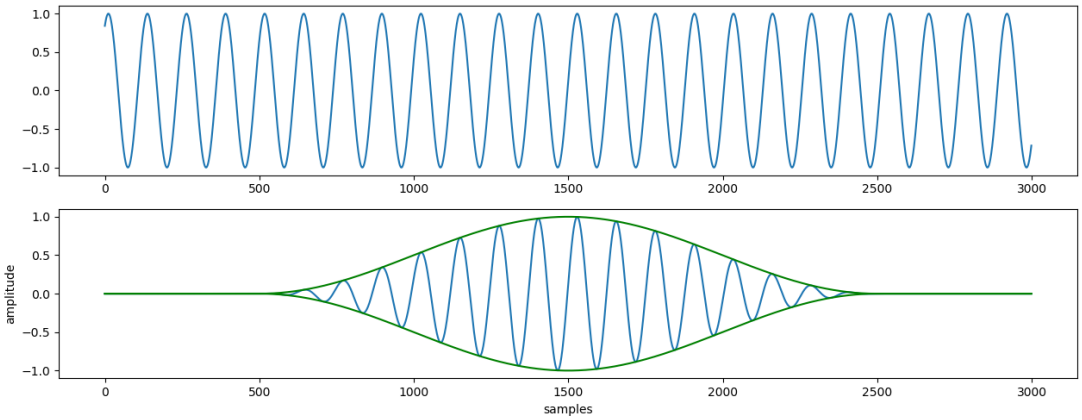


Figure 1.2: Signal and a portion of it selected by a window.

[4]Splitting the signal into overlapping chunks and computing DFT for each of them is called short-time Fourier transform (STFT).

This process will necessarily create some artifacts and choosing the shape and length of the window is a compromise between various parameters.

Windowed STFT as a bank of bandpass filters

STFT of signal s_n windowed by window w_n shifted by k (so we can select different portions of the signal) is

$$\mathrm{STFT}(s)_{k,f} = \sum_{n=0}^{N-1} w_n \underbrace{s_{n+k} e^{-\frac{2\pi i f n}{N}}}_{\text{downconversion}}. \tag{1.13}$$

Instead of applying a window and then taking the DFT, this can also be seen as first shifting the frequency f we are currently analyzing to zero (c.f. (1.3)) and filtering the result with a filter w_n (c.f. (1.5))[5]. This means that the window function should behave as a low-pass filter, so we get the signal which we have shifted to zero frequency, but preferably not much interference from the other frequencies. As has been shown in 1.3.1, a moving average filter, which is in fact a rectangular window, behaves as such. We define two frequently used window functions and show their frequency response for comparison below.

$$\begin{aligned}
\mathrm{Hann}(N)_k =& 0.5 - 0.5\cos\frac{2\pi k}{N} \\
\mathrm{Nuttall}(N)_k =& 0.3635819 - 0.4891775\cos\frac{2\pi k}{N} + 0.1365995\cos\frac{4\pi k}{N} - \\
&- 0.0106411\cos\frac{6\pi k}{N}
\end{aligned}$$

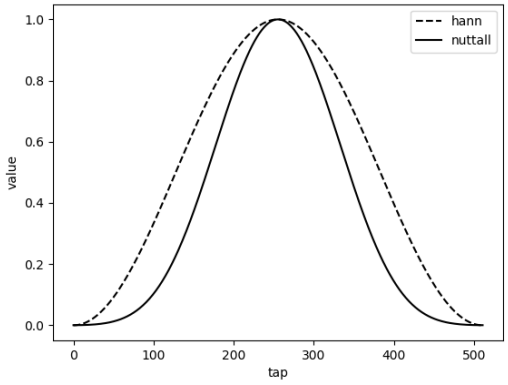


Figure 1.3: Plot of Hann and Nuttall window.

[5]It is in fact filtered by $\rho\, w_n$ as the convolution has the minus sign, but as windows used in practice are symmetric, it does not really matter.

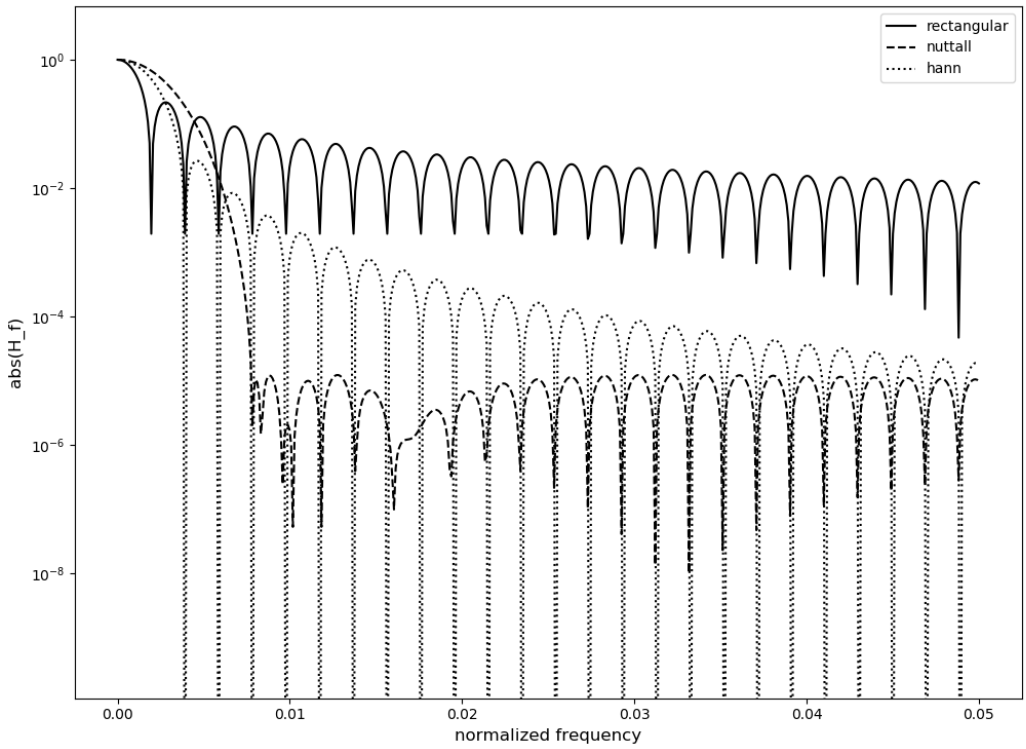


Figure 1.4: Comparison of rectangular, Hann and Nuttall windows with 512 taps.

We can see that the Nuttall window has much better rejection at higher frequencies, but the peak at zero frequency is somewhat broad. This means that we will have much less unwanted noise, but will not be able to discern two peaks that are close to each other. This is a common tradeoff when choosing the window function.

Window and convolution of the spectrum

Another view of windowing is through convolution of the result of the DFT. Pointwise multiplication of the DFT-transformed signal is equivalent to convolution of the original signal and vice versa (we show this in 1.6 for the case of discrete spectrum). Here we multiply the signal with the window, so after DFT is computed, the result looks like the hypothetical spectrum of the original signal convolved with the DFT image of the window, which causes smoothing of the peaks. This is usually called *spectral leakage* of the peaks.

1.5 Crosscorrelation

Crosscorrelation is used to determine similarity and time shift of two signals. Cross-correlation between signals f and g is defined as

$$\mathrm{x}(f,g)_k = \sum_{j=-\infty}^{\infty} f_{j+k}\overline{g_j}.$$

This can be thought of as shifting the signal f over signal g and taking a dot product at each step.

- Crosscorrelation is "convolution with time-reversed complex conjugate", therefore most properties of convolution apply. $\mathrm{x}(f, g) = f * \overline{\rho(g)}$
- Additivity: $\mathrm{x}(f + h, g) = \mathrm{x}(f, g) + \mathrm{x}(h, g)$
- Quasi-symmetry:

$$\mathrm{x}(f, g)_k = \sum_{j=-\infty}^{\infty} f_{j+k}\overline{g_j} \overset{l=j+k}{=} \overline{\sum_{l=-\infty}^{\infty} \overline{f_l} g_{l-k}} = \overline{\mathrm{x}(g, f)_{-k}} \tag{1.14}$$

- Filtering (convolution):

$$\mathrm{x}(a * f, g) = a * f * \overline{\rho(g)} = a * \mathrm{x}(f, g), \tag{1.15}$$

$$\begin{aligned}\mathrm{x}(a * f, b * g) = a * f * \overline{\rho(b * g)} = a * f * (\overline{\rho(b)} * \overline{\rho(g)}) = \\ = f * \overline{\rho(g)} * a * \overline{\rho(b)} = \mathrm{x}(f, g) * a * \overline{\rho(b)}\end{aligned}$$

- For finite-length signals we are usually interested only in the part of the result where nonzero components of both signals fully overlap. This is called the "valid mode".

1.6 Efficient computation of correlation using Fourier transform

Computing valid mode crosscorrelation of two signals f and g of lengths $l_f \leq l_g$ naively requires $(l_g - l_f + 1) \cdot l_f$ complex multiplications, but it can be computed efficiently by applying the convolution theorem

$$\mathrm{x}(f, g) = \mathrm{IFFT}(\mathrm{FFT}(f) \odot \overline{\mathrm{FFT}(g)})[: (l_g - l_f + 1)],$$

where $\odot$ is pointwise complex multiplication, FFT is a fast Fourier transform of length at least l_g (signals are padded with zeros to this length[6]) and $[: (l_g - l_f + 1)]$ extracts only the "valid" part of the result. This requires three FFTs ($\Theta(n \log n)$) and one pointwise multiplication ($\Theta(n)$), so for many values of l_f and l_g it is faster than computing the correlation by definition ($\Theta(n^2)$ worst-case).

First, notice that for signals f and g of lengths $l_f \leq l_g$ (padded with zeros elsewhere), a valid mode crosscorrelation

$$\mathrm{x}(f, g)_k = \sum_{j=-\infty}^{\infty} f_{j+k}\overline{g_j} \quad \text{for } k \in \{0 \dots l_g - l_f\}$$

is equal to computing a cyclic crosscorrelation, where the indices wrap around using a modulo operation

$$\mathrm{x}_c(f, g)_k = \sum_{j=0}^{N-1} f_{(j+k)\%N}\overline{g_j} \quad \text{for } k \in \{0 \dots l_g - l_f\}$$

[6]It might be useful to compute FFT of larger size than strictly needed. FFT algorithms perform best when the transform size is a power of two, so for example computing a FFT of size 2048 is better than computing FFT of size 1901, even though part of the result will be discarded.

for $N \geq l_g$. This is because only the zero part wraps around.

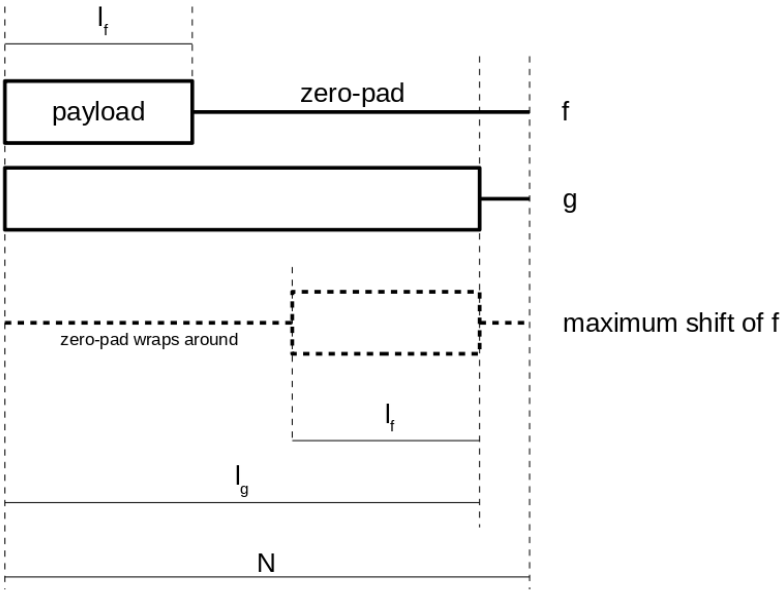


Figure 1.5: Validity of circular convolution

We now claim that

$$\mathrm{x}(f,g) = \mathrm{x}_c(f,g) = \frac{1}{N}\,\mathrm{IFFT}(\mathrm{FFT}(f) \odot \overline{\mathrm{FFT}(g)})[:(l_g - l_f + 1)].$$

Compute DFT of a cyclic correlation:

$$\mathrm{DFT}(\mathrm{x}_c(f,g))_\varphi = \sum_{k=0}^{N-1}\sum_{j=0}^{N-1} f_{(j+k)\%N}\overline{g_j}e^{\frac{-2\pi i\varphi k}{N}}$$

Split the exponential using $e^{\frac{-2\pi i\varphi k}{N}} = e^{\frac{-2\pi i\varphi(k+j-j)}{N}}$ and reorder the sums while substituting $\hat{j} = j + k$ (as both the complex exponential and $(j+k)\%N$ wrap around):

$$\sum_{k=0}^{N-1}\sum_{j=0}^{N-1} f_{(j+k)\%N}e^{\frac{-2\pi i\varphi(k+j)}{N}}\overline{g_j}e^{\frac{2\pi i\varphi j}{N}} = \sum_{\hat{j}=0}^{N-1} f_{\hat{j}}e^{\frac{-2\pi i\varphi\hat{j}}{N}}\overline{\sum_{j=0}^{N-1} g_j e^{\frac{-2\pi i\varphi j}{N}}}$$

Now apply the inverse DFT.

$$\frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{j=0}^{N-1} f_j e^{\frac{-2\pi i\varphi j}{N}}\overline{\sum_{j=0}^{N-1} g_j e^{\frac{-2\pi i\varphi j}{N}}}\right)e^{\frac{2\pi i\varphi k}{N}}$$

1.6.1 Implementation, benchmark

A sample program to test the approach described above can be found in `rtl-tdoa` source directory. The program is called with values of l_f and l_g and the number of repetitions. It generates random data (from uniform $[-0.5, 0.5]$ distribution) and repeatedly computes crosscorrelation by-definition and then using Fast Fourier transform. Correctness of the implementation is checked by comparing the results, and the total computational time for both approaches is printed.

For naive correlation, `libvolk` is used, which provides hand-optimized SSE, AVX and NEON routines for computing complex dot product. For FFT, `FFTW` is used.

Example compilation and usage of fftcorr_benchmark

```
$ make fftcorr_benchmark
$ ./fftcorr_benchmark
Usage: ./fftcorr_benchmark l_f l_g iters_naive iters_fft
Example: ./fftcorr_benchmark 512 1024 1000 1000
$ ./fftcorr_benchmark 512 1024 1000 1000
Maximum difference of the results: 0.000003
Total wall time: naive: 176 ms, FFT: 14 ms
```

The FFT implementation is faster asymptotically, but for small instances or $l_f \simeq l_g$ the naive computation might be faster in terms of wall time. However, for parameters which we use in our radar, the FFT algorithm wins by a great margin, so it is not needed to have two implementations and logic that will choose between them.

2. Time difference of arrival

2.1 Theory

2.1.1 Basics

Consider that two geographically separated receivers with precisely synchronised clocks are receiving the same signal:

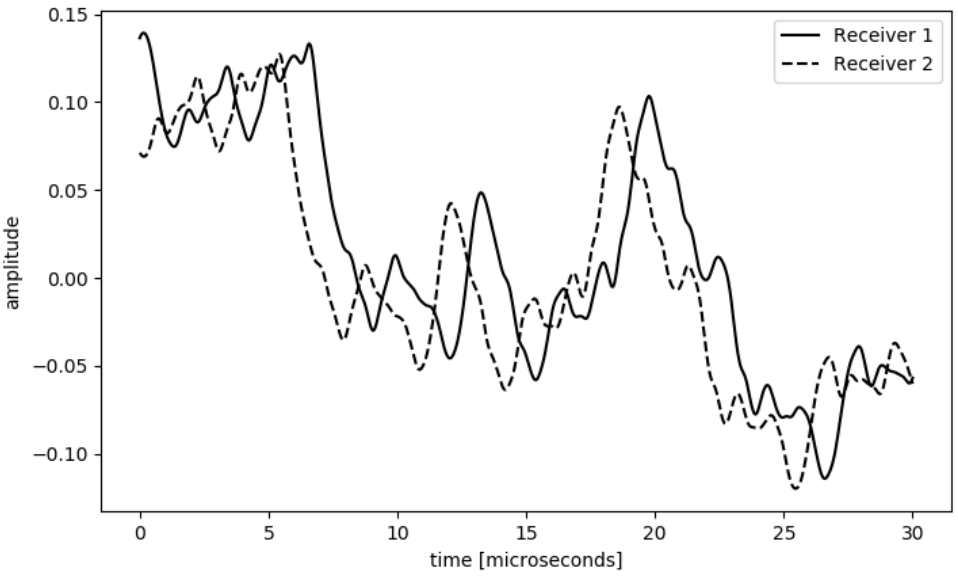


Figure 2.1: The same signal received by two receivers

We see that the Receiver 2 received the signal about $t_{12} = 1.2$ microseconds earlier. Therefore, we conclude that the position of the transmitter must satisfy the equation

$$d_1 - d_2 = t_{12}c, \tag{2.1}$$

where d_1 is the distance between Receiver 1 and the transmitter, d_2 between Receiver 2 and the transmitter, and c is speed of light (and the difference therefore evaluates to 360 meters). Points that satisfy this equation form a hyperbola with foci at our receivers and the length of semi major axis $a = t_{12}c/2$.

2.1.2 Geometry considerations on a round planet

For the sake of simplicity, we suppose that everything is located on the Earth surface. If we were trying to locate the transmitter in a 3-dimensional space (e.g. an airplane), then hyperboloids instead of hyperbolas would be involved. Unfortunately, due to hardware limitations, our device cannot track airplanes. See 2.1.6 for discussion about airplane tracking.

Additionally, we are approximating the local surface of the Earth with a plane as we are building system on the scale of a few kilometers (covering a city and surroundings) where the curvature of the Earth is negligible. Some errors will

probably be introduced by local topography as radio waves of lower (VHF[1] and less) frequencies can "bend" around hills and buildings. We hope that we will mostly encounter direct line of sight cases as both receivers and test targets are on rooftops.

The third and the fourth receiver

Now we get a third reveiver, measure the time differences d_{t13} and d_{t23}, and use equation 2.1 to get two more hyperbolas. Then we compute the intersection of all three hyperbolas and get the location of the transmitter.

Of course in practice all the hyperbolas will not intersect in one point because of measurement errors, but if the errors are small, all individual intersections are close together.

It is useful to add the fourth receiver:

- In rare cases of bad geometry, we can get two distant intersections or near-intersections.

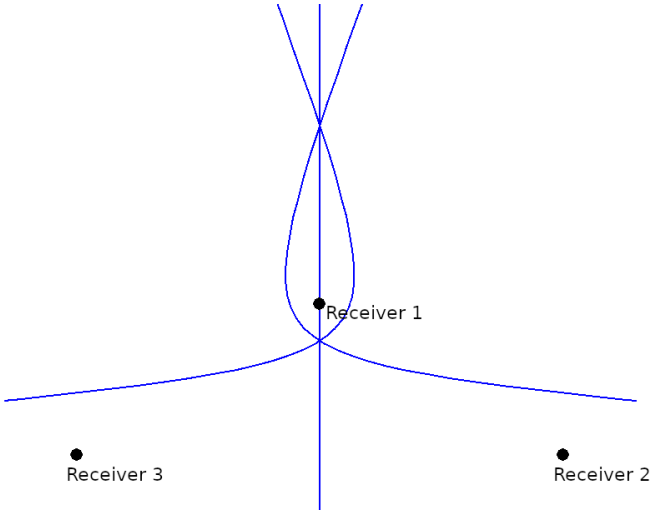


Figure 2.2: A "phantom" secondary solution

- Validating the result and estimating its error. With three receivers we have three intersections, can measure their distance and infer that low distances mean low error, but the third hyperbola is not independent of the first two.

2.1.3 Behavior of DSP operations in a multi-receiver situation

Ideally, we would apply crosscorrelation to the signals received directly from the antenna and learn the time offset t_{12} at which the crosscorrelation gives the maximum value. Unfortunately, it is not possible to directly sample this signal (which might be in the UHF[2] band), at least with commonly available hardware.

A software-defined radio receiver takes the signal from the antenna, applies frequency shift, low-pass filtering and decimation and provides us with this result.

[1] very high frequency, frequency range between 30 MHz and 300 MHz
[2] ultra high frequency, frequency range between 300 MHz and 3 GHz

We can run crosscorrelation on this result. We therefore need to investigate the properties of operations introduced in Chapter 1 in this situation.

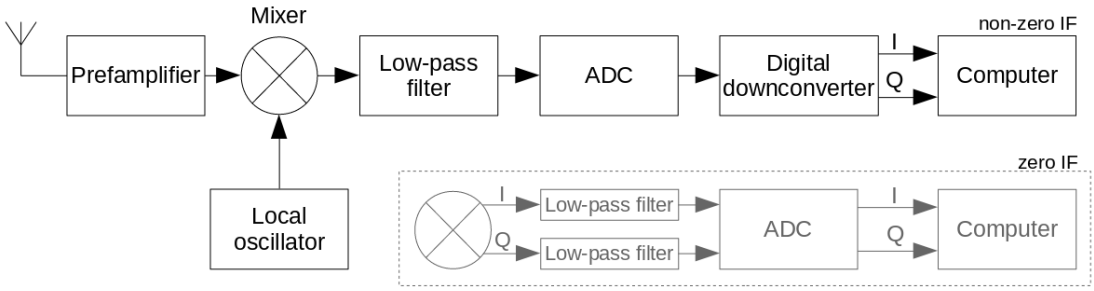


Figure 2.3: A simplified block diagram of a software-defined radio using *non-zero intermediate frequency* (top black), such as newer (R820T-based) rtl-sdrs. The mixer does not convert the received signal to zero frequency, but leaves it at several MHz, and the final shifting is done by DSP. Another approach is to have a mixer producing complex signal centered at zero frequency and sampling it with a quadrature ADC (bottom gray). This was used in older E4000-based rtl-sdrs. For our application the difference between zero and non-zero IF is not important.

Frequency shift

Let the signal a_n be received by the first receiver and shifted by frequency f and the signal b_n received by the second receiver and shifted by frequency g. Additionally, as we have no means to synchronize the phase difference of the two receivers, assume that a_n is shifted by an unknown phase ϕ. Compute crosscorrelation of a_n and b_n:

$$\begin{aligned}\mathrm{x}(a_n e^{2\pi i f n + i\phi}, b_n e^{2\pi i g n})_n &= \sum_{k=-\infty}^{\infty} a_{n+k} e^{2\pi i f (n+k) + i\phi} \overline{b_k e^{2\pi i g k}} \\ &= \sum_{k=-\infty}^{\infty} a_{n+k} \overline{b_k} e^{2\pi i f n + 2\pi i f k + i\phi - 2\pi i g k} \\ = e^{2\pi i f n + i\phi} \sum_{k=-\infty}^{\infty} a_{n+k} \overline{b_k e^{2\pi i k (f-g)}} &= e^{2\pi i f n + i\phi}\, \mathrm{x}(a_n, b_n e^{2\pi i n (g-f)})_n\end{aligned} \tag{2.2}$$

Specially, if $f = g$ (both receivers are tuned to the same frequency), we get $\mathrm{x}(a, b)$, shifted by frequency f plus an unknown phase ϕ. We discard the phase information by taking absolute value of the result. To assure that $f = g$, despite potential frequency error caused by oscillator inaccuracy in the receivers, we introduce a calibration process in 2.2.10 and 2.2.12.

Low-pass filtering

As per (1.15), low-pass filtering the source signal (by convolving it with a FIR filter) is the same as low-pass filtering the result of the correlation. Therefore we expect the result to be low-pass filtered (e.g. peaks in it will be "rounded") but otherwise intact.

Decimation

Intuitively, as the correlation result is now low-pass filtered, we can safely decimate it. Explicitly, let D be the decimation (keep-1-in-d) function and h suitable low-pass anti-aliasing filter. Then the correlation of the filtered and decimated signal is

$$\mathrm{x}(D(a*h), D(b*h)) = D(a*h) * \overline{\rho(D(b*h))} = D(a*h) * D(\overline{\rho(b*h)}), \quad (2.3)$$

which is equal to $D(a*h*\overline{\rho(b*h)})$ if there is no power in the spectrum beyond Nyquist frequency and therefore no aliasing has happened. This will not hold exactly for real-life signals, but the antialiasing filter in front of the ADC should make this error negligibly low. Expanding the equation further, we get $D(a*h*\overline{\rho(b*h)}) = D(\mathrm{x}(a*h, b*h)) = D(\mathrm{x}(a,b)*h*\overline{\rho(h)})$, and because h is a low-pass filter, it is symmetric, real, and approximates a sinc function, and hence convolution with itself is nearly the same filter and therefore the result approximates $D(\mathrm{x}(a,b)*h)$[3].

2.1.4 Dilution of precision

For certain geometry of the receivers and the position of the transmitter, the solution can be ill-conditioned in the sense of numerical stability and robustness to measurement errors. The following picture shows an example of bad and good geometry.

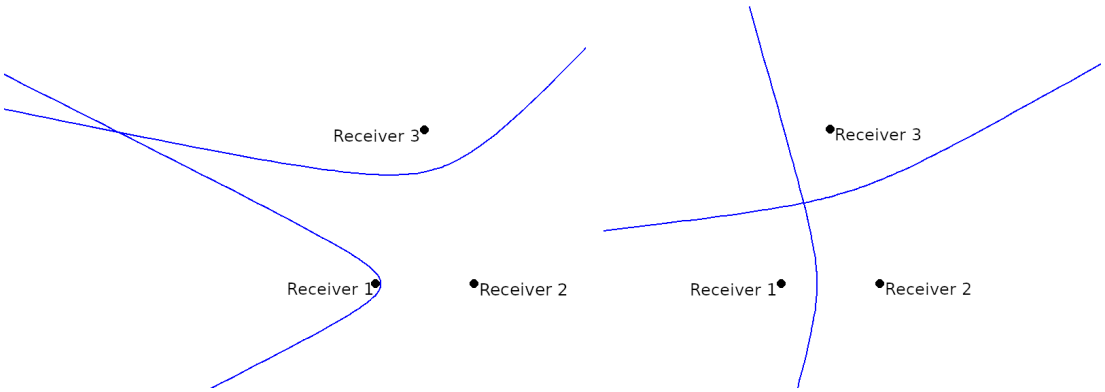


Figure 2.4: An unfavorable (left) and favorable geometry

We define the dilution of precision as DOP $= \frac{\Delta l}{c\Delta t}$, i.e., a highly "diluted" case means that a small change in measured time difference results in huge jump in computed location. We can then evaluate DOP at a point with respect to two receivers by shifting the point by a small ϵ in all directions and observing the minimum change of Δt; the DOP is then $\frac{\epsilon}{c\Delta t}$. For a multi-receiver setup, we evaluate DOP for all pairs of receivers and then pick the second-best value for each point (as to determine the location, two hyperbolas are needed).

Figures 2.5 and 2.6 show the results of evaluating the DOP for each point on a plane. We see that the things we want to measure should be somewhere "in between of" our receivers. When picking up places for our receivers, we

[3]One can also imagine the last step as applying the low-pass filter twice. And low-passing an already low-passed signal should make little difference.

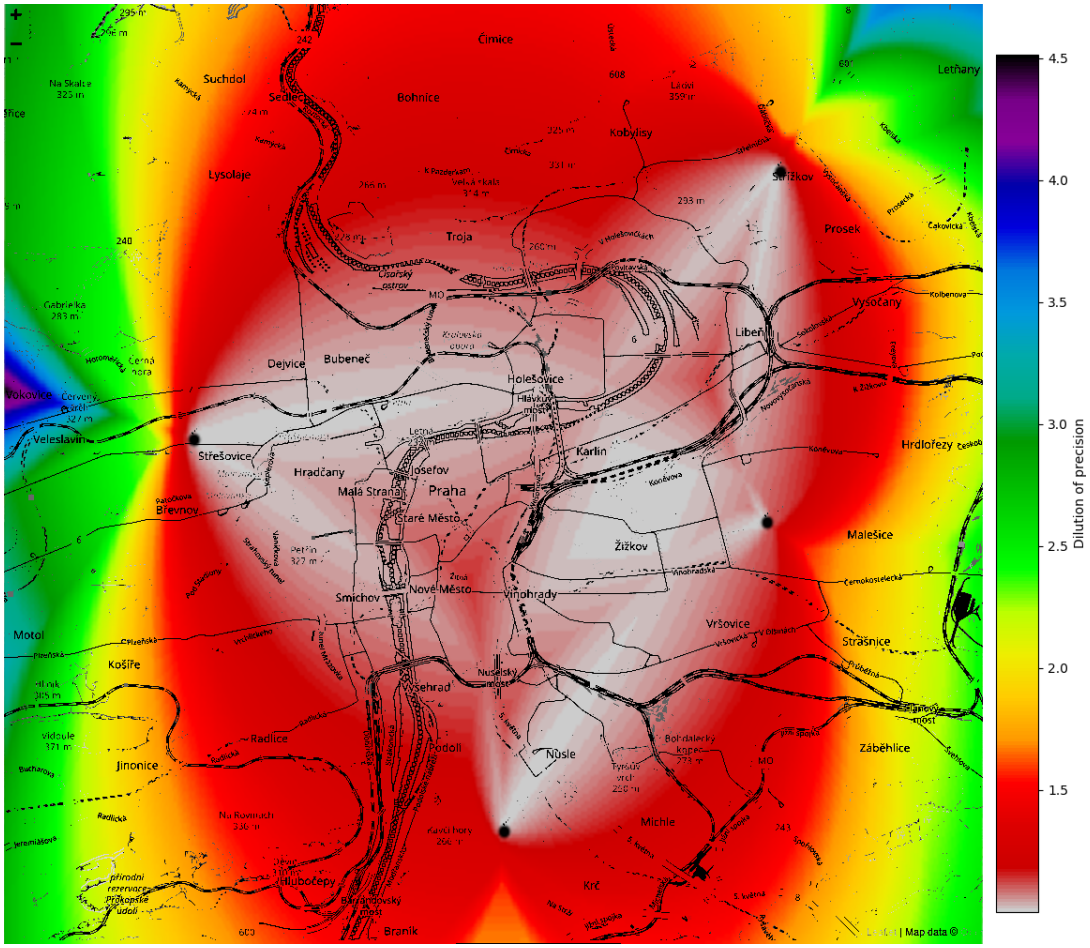


Figure 2.5: Dilution of precision in our system with 4 receivers, superimposed on a map of Prague.

should take this into account, though of course additional constraints like local topography (the receiver should be on an elevated place) or price of rooftop hosting must be considered.

2.1.5 Precise clock synchronization

It is absolutely crucial to have the same time on all receivers, preferably with the precision of 100 ns or better (100 ns corresponds to 30 meters at the speed of light, but because of dilution of precision, larger error in the final computed position of the transmitter is expected). This can generally be achieved in three ways:

Clock distributed via the network

The receivers are connected by some network, be it optical or microwave link, that also carries precise timing information. This is not our case, as everything that is available to connect our receivers without prohibitive expenses is standard Internet, and protocols for time distribution over the Internet (such as NTP) achieve several orders of magnitude worse accurracy.

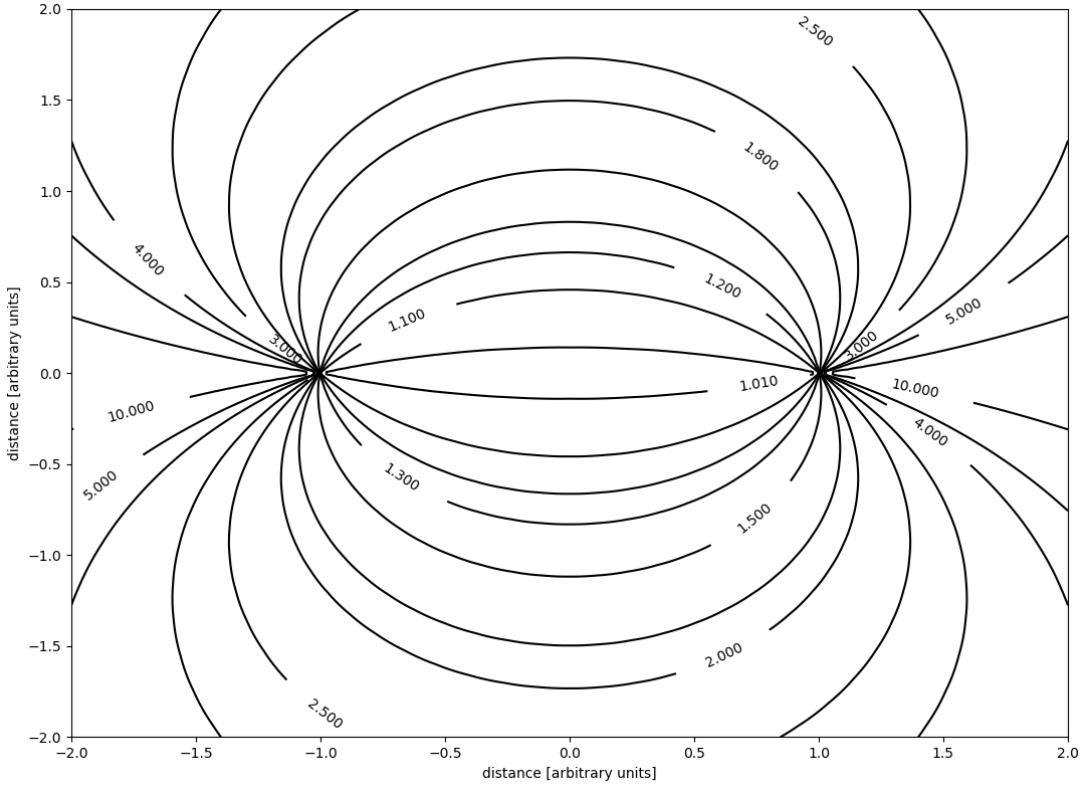


Figure 2.6: Dilution of precision of 2 receivers located at $(1, 0)$ and $(-1, 0)$. For example at the contour with value 2, a timing error of 1 unit (e.g. 0.5 in each receiver in the opposite direction) results in a location error of 2.

GPS

A by-product of acquiring a GPS position is also timing information. The GPS receiver is required to expose the timing signal, usually in the form of "pulse per second" (PPS) pin, where a rising edge is provided exactly at the start of each second. Such receivers can now be obtained for 20 USD or less. However, a problem arises with synchronizing this signal with the SDR input, as most cheap SDRs do not have any input for auxiliary signals. As the first try, one can mix the PPS signal, shaped by a suitable filter, into the signal input of the SDR, or for example short the signal input to ground every second and then look for sudden disappearance of the signal. Unfortunately, it is difficult to precisely detect the sample (or even do it with subsample resolution) when this event happened. We think this might be solved by using the PPS signal to derive some sequence that is designed for easy synchronization (such as Gold code) and then lock onto this sequence, but we finally decided not to use GPS as a time source.

Reference transmitter

A reference transmitter with known location (which are ubiquitous in populated areas) can be used as a source of shared time information. The only requirement is either the ability to receive the reference and the target at once (which in most cases requires special dual-channel hardware) or to quickly retune between these two. Fortunately it turns out that even the consumer-grade rtl-sdr device can retune fast enough before the clocks drift away. We have selected this approach

for our work.

The reference transmitter should have line-of-sight coverage, the transmitted signal should not have any repeating patterns (as we can mistakenly "lock" the timing synchronization to a wrong copy of that pattern) and should be wideband so the crosscorrelation function is sharp and well-defined. The line-of-sight condition is generally satisfied by public broadcasting transmitters (TV and radio), and the latter two conditions are met by the digital ones (DVB-T and DAB, self-similarity is removed by compression and whitening of the data stream). We use DAB in our experiments as it has convenient bandwidth of 1.5 MHz (our SDR has 2 MHz). DVB-T has 8 MHz and selecting only a quarter of the DVB-T multiplex while not yet fully synchronized seemed to be a bit difficult to implement correctly (though a naive filtering seems to work pretty well in practice too).

2.1.6 Previous work

Mutability MLAT server

Airplanes transmit radio beacons with their number, type and other information via a protocol called ADS-B. Thanks to this, the radar of flight control can display not only bright spots where it detects airplanes, but also anotate them with these metadata. Some airplanes, but not all, also transmit position. (private planes probably tend to limit broadcasting of position because of popularity of webpages like FlightRadar24, which receive exactly this data from the air and "spy" on them)

mlat-server[4] aggregates these data with additional information "at which sample the message started" (captured with rtl-sdr and a special ADS-B receiver software `mlat-client`). When at least one airplane with known location appears, mlat-server uses it to resolve timing differences between the receivers and computes positions of sources of all the other messages.

This demonstrates that rtl-sdr has clock stable enough to be used for multilateration even in the "receive known signal ... some delay ... receive target signal" scenario. However, by design, this program works only with airplanes.

Mlátička

Mlátička [5]was a maturita work by Josef Gajdůšek aimed to create a TDOA multilatelator for generic signals; author of this thesis was the supervisor of this work. The first approach was GPS (described in 2.1.5) with shorting ADC input to ground. Unfortunately, it was impossible to obtain reasonable resolution of the timing signal this way. Josef then proceeded to build his own SDR with auxiliary signal input, which will provide timestamps in the data stream. Due to hardware problems and the lack of time, Mlátička never produced successful output.

TDOA evaluation

A proof-of-concept MATLAB implementation of TDOA for rtl-sdr accompanied by a short article can be found in [6]. The authors used the retune-to-reference-transmitter approach.

Blitzortung

The project Blitzortung.org uses Time Difference of Arrival of very low frequency (VLF) radio waves to locate lightning strikes. They use GPS for time synchronization and custom hardware based on common microcontrollers for recording.

Tamara family

For the sake of completeness, we must mention a product family produced by Czechoslovak Tesla since the 1960s: Kopáč [short for Korelační Pátrač, meaning Correlation Locator], Ramona, Tamara and VERA, a series of TDOA multilaterators using first analogue processing and then digital computers. VERA is still in active development and can be considered a very up-to-date system, tracking both airplanes and terrestrial emitters and supporting wide range of signals. Their products are, however, not available for general public.

2.2 Implementation

We have implemented a TDOA multilaterator using retune-to-reference approach and ran it with four receivers located in Prague.

2.2.1 Hardware and software

The rtl-sdr radio

rtl-sdr is a common name referring to a class of USB dongles originally sold for reception of TV broadcast. They are based on Realtek RTL2832U DVB-T demodulator, which has an interesting feature: it can be switched into a mode in which raw samples are sent over the USB, thereby working as a universal software-defined radio. Although the parameters and overall quality of rtl-sdr devices are rather low, they have seen widespread adoption because of their price (starting at 8 USD), which is at least an order of magnitude lower than the usual price of competing devices.

The Realtek RTL2832U chip is the ADC and DSP processor (parts labelled "ADC" and "Digital downconverter" in Figure 2.3). A RF frontend — a device containing a preamplifier, oscillator, mixer and a low-pass filter — needs to be added. Older rtl-sdrs used Elonics E4000 and Fitipower FC0012, but since 2015, Rafael R820T is the de-facto standard.

Parameters of rtl-sdr with R820T:

- Frequency range: 24 to 1750 MHz
- Bandwidth: approx. 2 MHz
- ADC sample rate: 2.4 MS/s (complex)
- ADC resolution: 8 bit (each sample is composed of 8-bit real and 8-bit imaginary component)
- Oscillator with no temperature compensation. We have observed initial errors between −20 and +70 ppm, and this initial error changes by another 3 to 10 ppm when the device heats up during normal operation.

Other hardware used

As antennas, we have used an unbranded telescopic TV dipole and several HAM and professional scanner antennas (Diamond X-30 N and Discone SD-1300 U). The signal from the TV antenna was rather weak, requiring setting of high gain in the radio preamplifier. On the other hand, both professional antennas performed reasonably well. Unfortunately, this means that the price of the high-quality antenna is bigger than the price of the rest of the hardware.

The data from each radio is processed by a generic computer, in our case two Raspberry Pis, an Orange Pi and an x86 laptop were used. This diversity enabled us to ensure that no timing is dependent on specifics of the USB controller. The computers run Debian GNU/Linux and don't use any special configuration, e.g. no real-time kernel patches.

The nodes are connected to an OpenVPN network because it was difficult to get public IP address or IPv6 connectivity on some of the rooftops. OpenVPN also provides somewhat secure environment.

The software is written in Python with C calls via CFFI where low-level interaction with the radio driver or high performance is needed. The controlling server communicates with nodes by sending JSONs over HTTP API. The user can interact with the software using a CLI and the results of the measurements are generated as HTML and displayed in a browser.

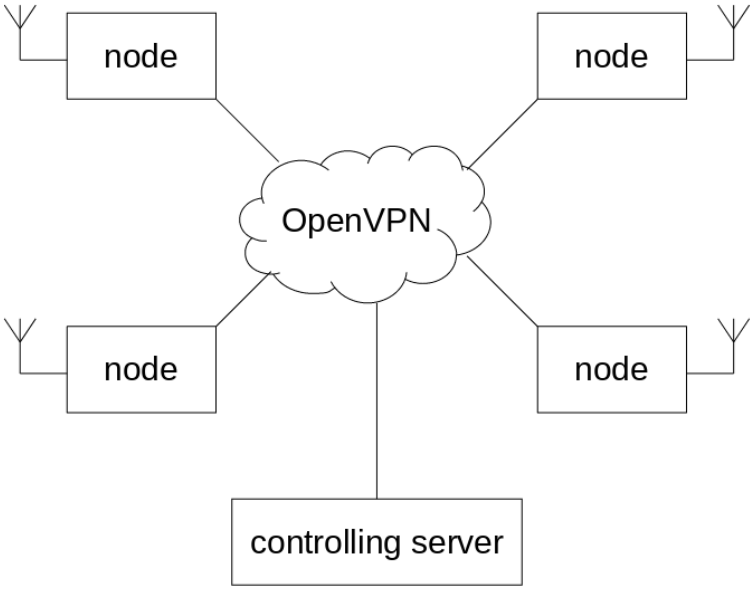


Figure 2.7: Our TDOA system architecture

2.2.2 Feasibility

First, the radio needs to be retuned without losing samples (or at least the number of samples lost must be known) — otherwise, it will not be possible to tune to a known transmitter to acquire timing information and then retune to the target transmitter and use this timing information to measure it. We have conducted a simple experiment: two rtl-sdrs were connected to the same signal source using an antenna splitter. Both were tuned to some frequency, crosscorrelation was computed and the offset of the largest peak was noted. Then the SDRs were

retuned and crosscorrelation was computed again. We then checked that the offset of the largest peak is the same.

Next, the clock in the radio needs to have enough short-term stability (over the course of several seconds while the measurement is being taken) to "hold" the timing information during retuning. From repeatedly running kalibrate-rtl, a program which compares local clock with the clock of GSM network, it is known that this stability is better than 0.5 ppm, as the main source of inaccurracies is thermal drift and the temperature of the device does not change rapidly. Therefore if the measurement and retuning will take 0.5 seconds, the precision should be better than $c \cdot 0.5\text{ppm} \cdot 0.5\text{s} = 75$ meters (multiplied by some DOP factor) which seems somewhat usable. And if it turns out to be insufficient, a precision oscillator can be used, though this increases the price.

And finally, the radio must be able to retune fast enough to make the measurement possible, and this retuning must happen on all radios at the same moment. This can be achieved by writing a custom recorder using the librtlsdr C API.

2.2.3 PLL dithering

An important feature of rtl-sdr worth noting is a PLL dithering. Unfortunately not everything is known about it as detailed datasheets of chips inside rtl-sdr are not publicly available. It happens when the desired frequency is not an integer multiple of the $\text{base\_clock} \cdot 2^{-\text{counter\_resolution}}$. It looks as if when the PLL could not generate the frequency directly, it resorts to randomly flipping the least significant bit to approximate it on average. Of course this has detrimental effects on any measurement that requires precise timing or phase control. Fortunately, it can be turned off.

2.2.4 The measurement process

More details on each step are given in the following sections.

The following steps are executed by all nodes once:

- Synchronize clock via NTP (2.2.5)
- Power on the radio, wait several minutes for the parameters to stabilize (mostly due to thermal drift during the warmup) (2.2.6)
- Pick a transmitter with known location to be used as timing reference
- Measure error of local oscillator by listening to a nearby GSM BTS (2.2.7)
- Tune to the reference transmitter, adjust gain (2.2.8)

The following steps are executed by all nodes for every target we want to measure:

- Tune to the target transmitter, adjust gain
- Measure error of local oscillator via GSM again
- Record 0.5 seconds of the target, then 0.5 seconds of the reference, and then 0.5 seconds of the target again (2.2.9)

Then the recorded data are downloaded to the controlling server and further processing is done for every pair of nodes:

- Resample and frequency-shift the recording to compensate for oscillator errors according to the numbers from GSM calibration (2.2.10)
- Acquire coarse synchronization by computing series of very long correlations (2.2.11)
- Compute detailed correlations (2.2.12)
- Estimate oscillator errors from these correlations
- Resample and frequency-shift the recording to compensate for oscillator errors according to this new estimation
- Compute detailed correlations again
- Filter these detailed correlations to reduce noise (2.2.15)
- Use these correlations to compute the time difference between the reference transmitter and the target transmitter
- Subtract distance to the known reference transmitter to obtain time difference of arrival of the target signal
- Plot the corresponding hyperbola to a map (2.2.14)

2.2.5 Coarse synchronization with NTP

The nodes need to be at least coarsely synchronized when the measurement is taken — for example because we are retuning between the reference and the target transmitter and this retuning needs to happen in a reasonable amount of time so the unstable crystal in the radio will not diverge too much. We use public NTP servers on the Internet to achieve this synchronization.

Another possible source of timing errors might be the computer system taking the measurement itself. The data are read though USB and asynchronous libusb callback is issued once a reasonably-sized chunk is transferred. Stock Linux with no real-time patches is used, which means these callbacks can arrive arbitrarily late. We record time when the callback is issued and then linearly extrapolate the exact time at which each sample has been taken based on the sample rate.

In practice, this gives us a timing error of about 20 milliseconds, which is sufficient for our application.

We have noticed problems with OpenNTPD after suspending to RAM, so when using laptops or other suspending computers, so after resume, check that the time is correct with `ntpdate -q` *and possibly restart the daemon.*

2.2.6 Power on and then sample continuously

When the radio is started, the AD converter, USB communication and other hardware begin to consume more power and the entire device gradually heats up. This induces thermal drift of the oscillator. And when the device is stopped, it begins to cool down. Therefore, it is important to open the device, continuously read data at a sustained rate (discarding them when they are not needed) and wait a few minutes until the temperature stabilizes.

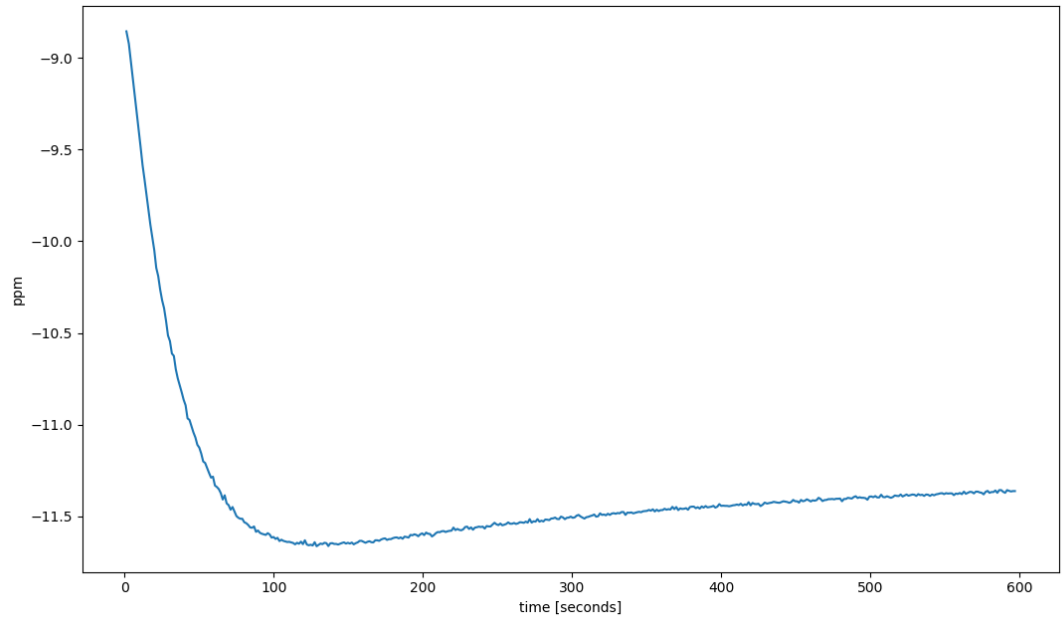


Figure 2.8: Crystal error in parts per million for the first 10 minutes after power on. Sufficient stability is achieved after about 2 minutes.

2.2.7 Calibration

After the warmup stage, we need to determine how big the current error is. This can be done by receiving some signal with known frequency. GSM (2G mobile phones) is pretty suitable for this task: base stations have timing derived from GPS timing reference, the GSM-900 band lies in the frequency range of rtl-sdr, and GSM signal is ubiquitous.

The process is detailed in [7]. The physical GSM channel is split into multiple logical channels using a time division multiplex. Most of these channels are used to carry data, but one of them, called Frequency Correction Channel (FCCH), transmits all zeros, without any whitening. When zeros are modulated by GMSK[4], a specific tone at an exact frequency appears. This tone can be detected and its frequency measured.

We have used the kalibrate-rtl program which implements this tone detection. Unfortunately it requires exclusive access to the device. We have created a wrapper called kalibrate-everything that reads samples from standard input and feeds them to the algorithm in kalibrate-rtl. The calibration process is as follows:

- Pick a GSM base station (from some public database of BTSs or by a spectrum scan) with frequency f_g
- Record several seconds of signal at frequency f_g
- Find the FCCH tone in the signal, measure its frequency and compute the offset $f_o = f_{\text{measured}} - f_g$ from the correct frequency
- Compute $e = \frac{f_o}{f_g}$, the relative error of our oscillator. (for practical reasons, e is often multiplied by 10^6 and then called ppm (parts-per-million) error)

[4]Gaussian minimum-shift keying, the modulation scheme used in GSM

Figure D: Simplified R820T Block Diagram

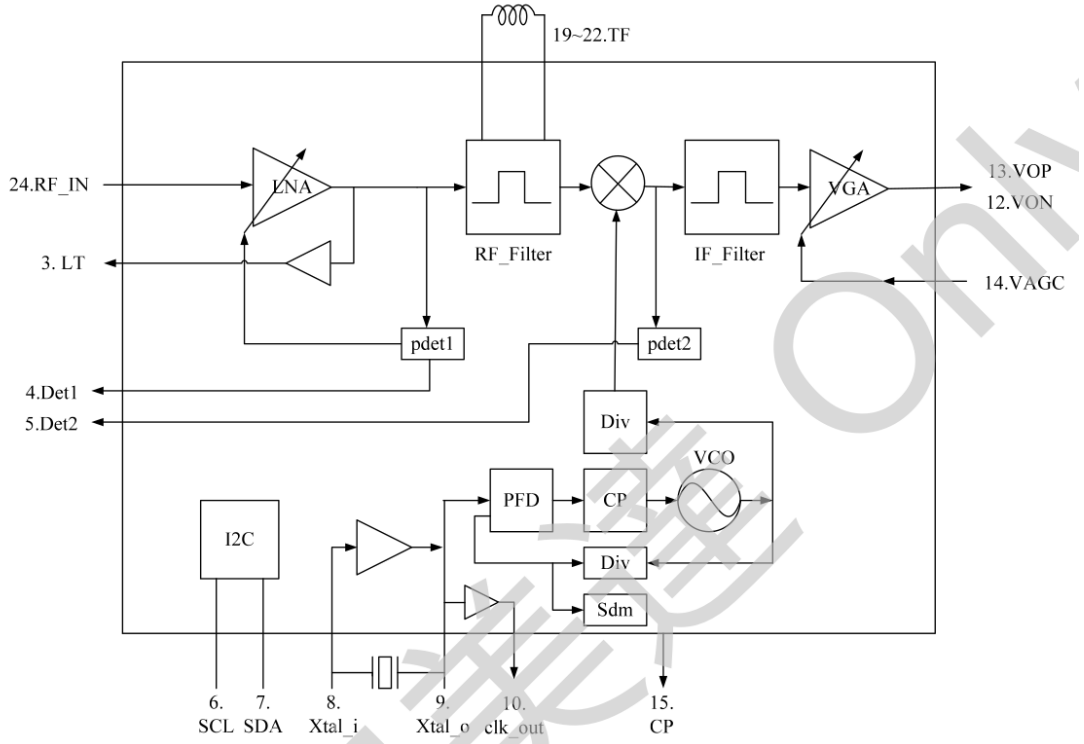


Figure 2.9: R820T block diagram from datasheet[8], showing the three gain stages

2.2.8 Adjusting the gain

To obtain recording with high signal-to-noise ratio, gain of the radio has to be set correctly. Too low gain will result in unused dynamic range of the ADC, too high gain will cause distortions in the analog part and clipping in the ADC. This is even more important in rtl-sdr than in more expensive radios, as rtl-sdr has only 8-bit output and hence low dynamic range.

The R820T chip has three configurable amplifiers: the input "Low Noise Amplifier" (0 to 33 dB range), the Mixer amplifier (0 to 16 dB range) and the output "Variable Gain Amplifier" (-5 to 37 dB range). Each of them offers 16 levels. LNA and Mixer support autogain. The autogain takes about 50 milliseconds to settle after retuning, depending on whether we are tuning between very strong and very weak signal, or the levels of both signals are similar. This delay is still fine for our usecase, as we make the retuning after 500 ms.

Gains in the original librtlsdr

The original librtlsdr uses the following heuristics and hardcoded constants to adjust gain:

- If the gain is set to manual, VGA is set to a fixed gain of 16 dB. LNA and Mixer gains are then ramped up together until the requested gain is achieved.
- If the gain is set to automatic, VGA is set to a fixed gain of 26 dB and LNA and mixer are set to autogain.

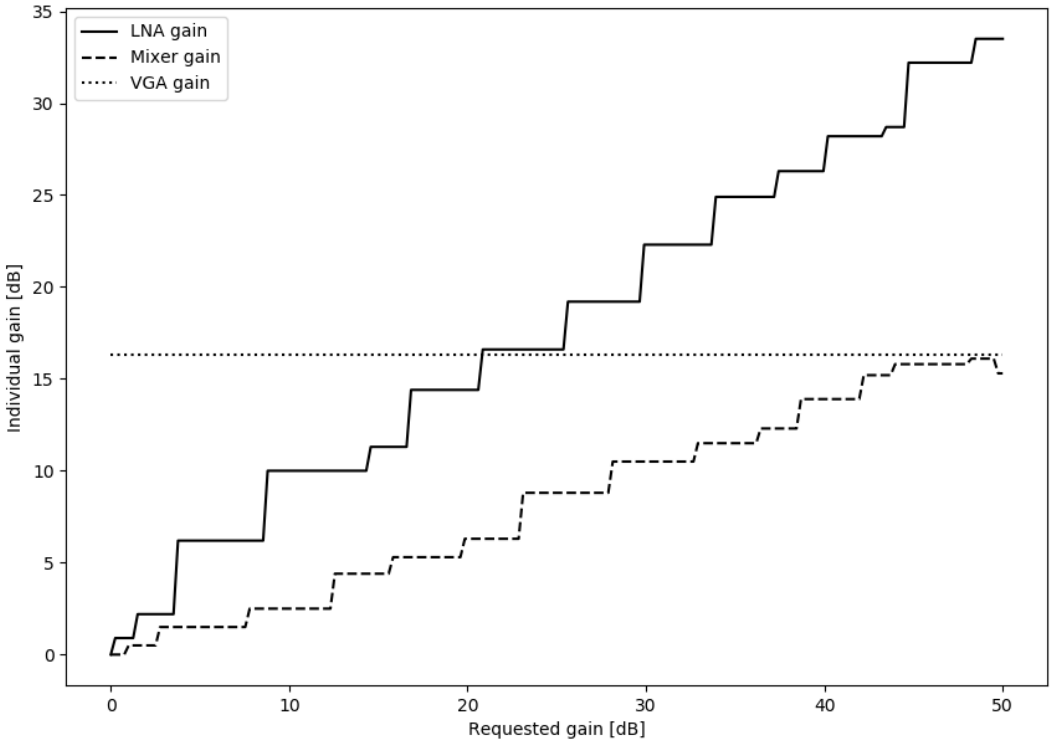


Figure 2.10: Requested gain vs. actual gain settings in librtlsdr.

One can sometimes obtain better results by using a more elaborate approach.

Determining the right gain

For autogain, VGA setting of 26 dB is almost always (in our case with strong broadcast transmitters) too much and results in clipping. We simply exhaustively try all 16 possible values and choose the largest one where clipping does not occur yet.

In some rare cases, usually when there is a strong narrowband transmitter nearby (in the frequency sense), autogain does not work well. Therefore, we introduce a utility function: we know the frequency and bandwidth of the transmitter we want to measure, so we record a short sample, compute power spectrum of it, and define "fitness" as (mean power of the target transmitter) − (mean power of everything else). *As the first attempt we have used the difference between maximum and minimum in the spectrum (except the edges where the transition band of the antialiasing filter attenuates the signal), but this has sometimes failed due to very narrow high peaks that occur in the spectrum — probably a harmonic of some digital signal in the radio or processing computer. This can be fixed by considering the difference between the 10^{th} and the 90^{th} percentile. However, in practice, both the difference-of-percentiles and the difference-of-means seem to work equally well.*

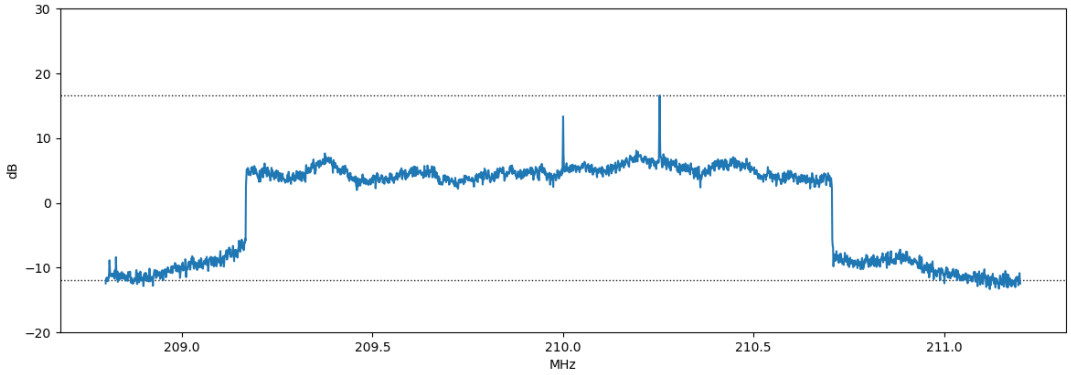


Figure 2.11: Example of a narrowband interference and the maximum and minimum lines when receiving a DAB signal.

We considered the following methods to maximize the fitness:

- **Exhaustive search** of all $16^3 = 4096$ possible settings. We can reduce the search space to about half the size by excluding nonsense values like setting LNA and Mixer to 0 and then VGA to the maximum. The straightforward implementation can try one gain setting in about half a second, but this could be reduced by changing the way how buffer management in librtlsdr works. The inevitable time for one try seems to be 25 ms which takes the setting itself plus about 20 ms to take a reasonable sample of the result. So the search may take about two minutes after some additional work, but we have decided it is not worth it.

- **Entropy-based black-box optimization** by scikit-optimize[9]. This approach samples the function at several points, fits it with a machine-learning model of some kind and then tries to estimate where extrems of the function lie based on the model. It was unfortunately prone to hitting local maxima in the utility function. Additionally, on low-end nodes like Orange Pi the construction of the machine learning model is slow.

- **Random sampling**. This works surprisingly well and picking the best of 30 random samples usually yields usable result.

We have implemented two methods and the user can choose between them when requesting a measurement. The `adcrange` method enables autogain and tries all 16 values of VGA gain. The `random` method does the same, then turns autogain off and tries 30 random settings. Fitness function is then evaluated for both the autogain and random samples and the best one is chosen. Additionally, the recorder can be manually advised to change the obtained gains by a given offset.

2.2.9 Recording

The timing uncertainty of HTTP requests from the controlling server is assumed to be higher than the uncertainty of NTP time. Therefore when a recording is to be done, the server generates a timestamp a few seconds in the future and tells the nodes "recording will start from this timestamp". At this timestamp, nodes

tune to the target frequency, wait 0.5 seconds, tune to the reference frequency, wait 0.5 seconds, tune back to the target frequency and wait 0.5 seconds.

The reason for recording "more" from the target than from the reference is that one of course picks a strong good transmitter as a reference (and hence only a small sample is required to get high precision reading), whereas the target transmitter is given and the quality may be poor (both because the signal strength is low and the signal is narrowband or has poor autocorrelation function). And we use the target-reference-target scheme (instead of simpler long recording of the target and then short of the reference) in hope to minimize the mean duration between the recordings of the target and the reference to avoid clock drift of the receiver.

2.2.10 Resampling and frequency shifting

Data retrieved from the node contain 1.5 seconds of recording (as described in 2.2.9) and information about oscillator error (obtained in 2.2.7). This error manifests itself in two ways: the sample rate of the recording is wrong and the frequency at which the recording has been taken is also wrong.

First, we remove DC offset from the data by computing the mean of all samples except the ones acquired during retuning and subtracting it from all samples.

Then, we use a fractional resampler to correct the sample rate. The fractional resampler works by precomputing a number of filters (128 in our case) that represent an impulse sampled with a lag of $0/128, 1/128 \ldots 127/128$ samples (the first filter is therefore a single number 1) and then performing Algorithm 1. This will not produce an exact result for resampling ratio that is not an integer multiple of 1/128, however, as the sampled shifted impulse changes its values continuously, it should give us a reasonable approximation.

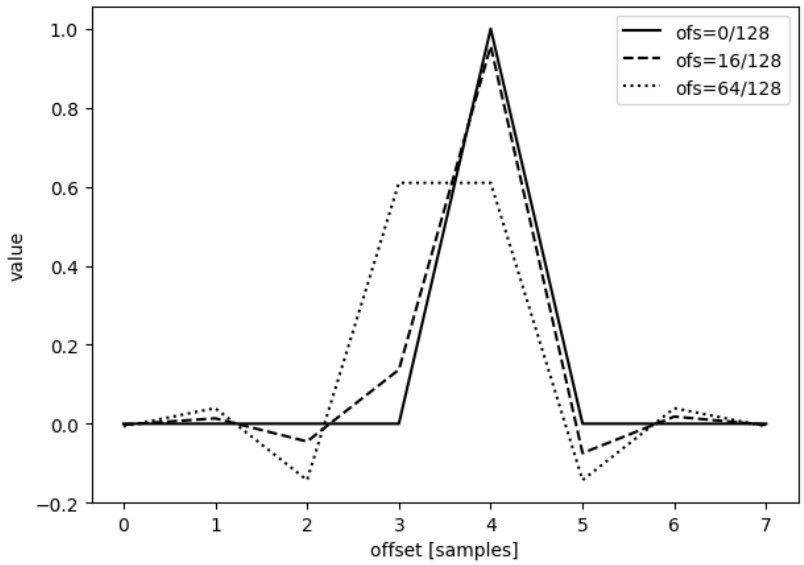


Figure 2.12: An impulse sampled with various fractional delays

```
                  Algorithm 1: Fractional resampler
                  Implementation: xcutil.c:resample

INPUT:
  resample_rate ... how many input samples to use for one output
                       sample (real number)
  bank ............ filterbank of N filters each of length L,
                       representing 0 to (N-1)/N sample lag
  in .............. input signal

OUTPUT: signal resampled by resample_rate

input_offset = 0.0
while input is not fully consumed:
  integer_position = floor(input_offset)
  filter_to_use = round((1-(input_offset-integer_position))*N)

  yield dot_product(in[integer_position:integer_position+L],
                    bank[filter_to_use])

  input_offset += resample_rate
```

The tricky part is how to generate these filters so they have unit gain at most frequencies and a linear phase. We have used the `gen_interpolator_taps` tool that comes with GNU Radio.

Finally, we compute the frequency correction. The frequency offset is $f_o = ef$ (where e is the error computed in 2.2.7 and f is the frequency we are listening to) and as we model frequency shift as rotation with complex exponential (in detail described in [10]), we compute

$$\text{corrected}_n = \text{input}_n \, e^{2\pi i n f_o / f_s},$$

where f_s is the sample rate.

Note that the frequency f is different for the target and for the reference transmitter.

Builtin frequency correction in rtl-sdr

The relative error can be passed to the SDR driver and it automatically compensates it:

- The Realtek RTL2832U chip has a register where the error can be written and the internal clock generator which provides clock for the ADC then changes its frequency according to this. (correction of the sample rate)
- When retuning, the driver instructs the tuner to tune to frequency $f' = (1 + e)f$ instead of f. (correction of the frequency)

There is a problem with this approach: both these "corrected" frequencies are only approximate (due to limited precision of PLLs, c.f. 2.2.3). When this correction is not used, the relative errors of sample rate and frequency are the

same, as they are both derived from one crystal oscillator. When this correction is set differently on multiple SDRs, we break this relationship. And as we describe in 2.2.12, we can precisely measure sample rate errors, but not frequency errors. When the relative error is the same, we can compensate both.

Therefore, we need to set this compensation to the same value on all nodes and do all the corrections during postprocessing. We can for example set it to zero or to the mean error of all nodes.

2.2.11 Coarse synchronization by long correlations

Now we have two files sampled at approximately the same rate and from approximately the same frequency (the error of GSM calibration is about 1 part per million), but with up to 20 ms time offset (the imprecision of NTP and the USB stack). We will compute long correlations of the part where we were tuned to the reference transmitter:

```
                Algorithm 2: Coarse synchronization
              Implementation: xcutil.c:compute_alignment

INPUT:
  in1, in2 ............... recordings of the reference from 2 receivers
PARAMETERS:
  winlen = 256*1024 ...... length of the sliding window
  range = 40*1024 ........ maximum absolute time difference of signals
  decimation = 32*1024 ... how far apart to lay the sliding windows
  S = 16 ................. final smoothing of the result

OUTPUT: time shift between in1 and in2

acc = array of zeros from -range to +range

for i in range(winlen/2, len(in1)-winlen/2, decimation):
  w1 = in1[i-winlen/2:i+winlen/2]
  w2 = in2[i-winlen/2+range:i+winlen/2-range]

  result = valid mode correlation of w1 and w2

  offset = argmax(abs(result))
  acc[offset]++

smooth acc by a moving average filter of length S

return argmax(acc)
```

- The parameter `winlen` is chosen so that the correlation can be evaluated efficiently with radix-2 FFT.
- The smoothing is important because the GSM correction has accurracy of about 1 ppm and the position of the correlation peak may therefore drift by a few samples over the course of the recording lasting a few seconds.

Now we low-pass filter the reference signal to its original bandwidth (e.g. 1.5 MHz for DAB, no filtering for DVB-T as the bandwidth of DVB-T is greater than the bandwidth of rtl-sdr) and discard the beginning of one of the recordings (depending on the sign of the result of Algorithm 2) so that they align.

2.2.12 Computing fine correlations

Then we compute much smaller correlations (with range, say, 128 samples, and window size several thousands). We try to shift the result of each correlation by a fraction of a sample to obtain subsample resolution (one sample at 2 MS/s is 150 meters, we want higher accuracy, and (1.15) tells us shifting the result is the same as shifting the input signal if we use convolution for shifting). We pick the offset and shift with the maximum absolute value.

```
                  Algorithm 3: Fine correlations
             Implementation: xcutil.c:compute_correlations

INPUT:
  in1, in2 ............ input signals from 2 receivers
  bank ................ filterbank of N filters representing
                          0 to (N-1)/N sample lag
PARAMETERS:
  winlen = 8192 ....... length of the sliding window
  range = 128 ......... maximum time difference of signals
  decimation = 2048 ... how far apart to lay the sliding windows

OUTPUT: correlation peaks, their fitness and a visualisation of them

image = empty image (grayscale floatmap)
for i in range(winlen/2, len(in1)-winlen/2, decimation):
  w1 = in1[i-winlen:i+winlen]
  w2 = in2[i-winlen+range:i+winlen-range]

  result = valid mode correlation of w1 and w2

  for each f in range(0, N):
    tmp = filter result with bank[f], thereby shifting it by a fra-
                                                        ction of a sample
    tmp = abs(tmp)
    peak_height = max(tmp)
    peak_pos = argmax(tmp)-f/N
    fitness = peak_height/mean(tmp)
  pick peak_pos and fitness with maximum peak_height from the above

  yield peak_pos, fitness

  tmp = upsample result, so the image has reasonable width
  tmp = abs(tmp)
  add tmp as a new row to the image

normalize each row of image to [0,1] as magnitudes of correlations
```

```
  vary wildly
return image
```

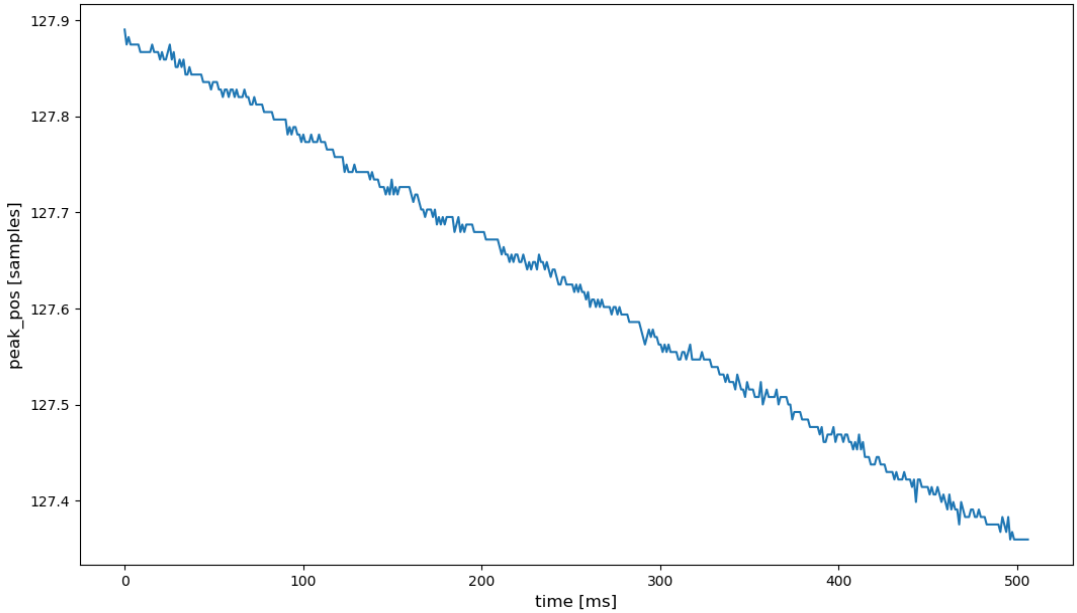


Figure 2.13: The resulting `peak_pos` list

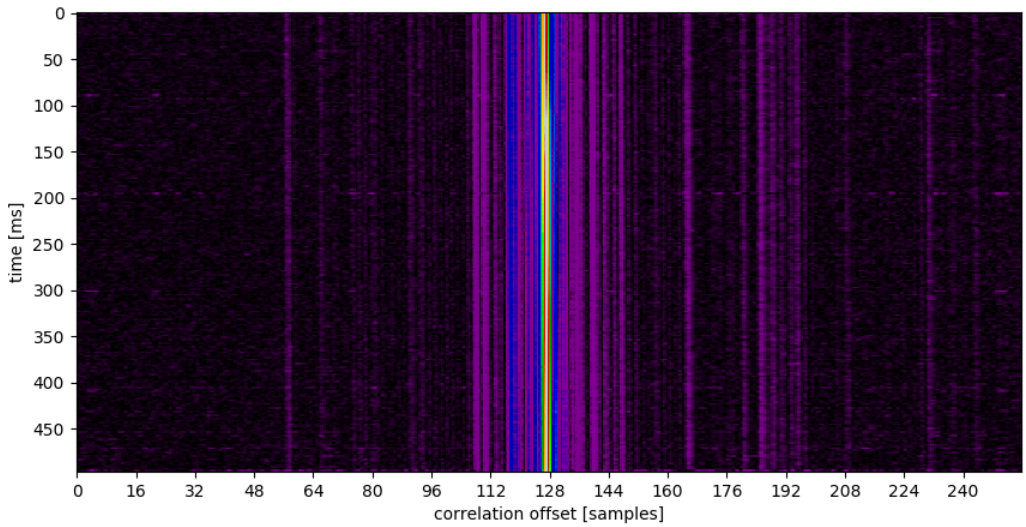


Figure 2.14: The resulting image

The plot of the computed `peak_pos` values and the returned image can be seen in Figures 2.16 and 2.14. Notice that the correlation maximum is drifting. This is caused by the two signals being sampled at different sampling rates (and, as the radio has only one clock source, also with some frequency offset) — which means that our correction in 2.2.10 was not exact. We fit the data with a line using least-square method and use the resulting slope as an additional error factor, reflecting both the incorrect sample rate and the incorrect frequency.

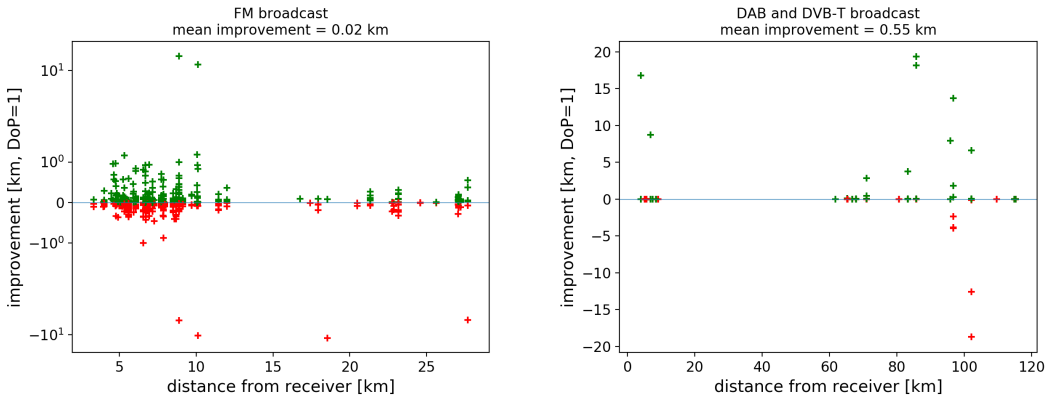


Figure 2.15: Scatter plot of improvement in accuracy after the additional error factor has been corrected (the left plot has log axis).

It turned out that the approximation of the fractional delay filters is not precise enough and the filters tend to lower the amplitude around the shift of 1/2 of a sample. This problem can be seen on a simple synthetic example of shifting a sine wave, where the peak at $\frac{\pi}{2}$ drops to about 0.9998. This problem is negligible on signals with sharp correlation peaks (such as the ones produced by DAB and DVB-T, which we use for synchronization), but a bias against shifting is noticeable with low-bandwidth signals where the peak is smooth. On the other hand the smooth peaks suffer from noise in practice, so we see the peak with different shifts as it randomly shifts with the noise; and as we make linear repression and averaging of thousands of individual correlations, the errors "average out". Another approach would be to fit the points around the peak with parabolic, Gaussian or other curve (which on the contrarydoes not work well in the case of sharp peaks) or use some more complicated resampler. However, it looks like the limitations in precision of our system do not stem from this biasing issue.

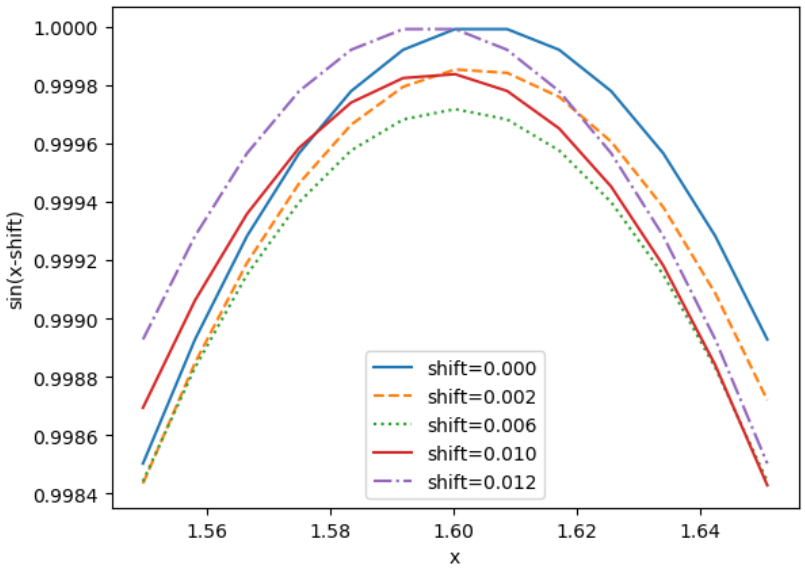


Figure 2.16: Shifting a sine wave (zoomed on the very top near $\frac{\pi}{2}$)

2.2.13 Extracting the time difference

We repeat steps 2.2.10 through 2.2.12 using the obtained additional error factor. This compensation proved to be very helpful in cases of low SNR conditions. Additionally, we compute the fine correlations not only over the reference, but also over the target. The resulting image now looks like this:

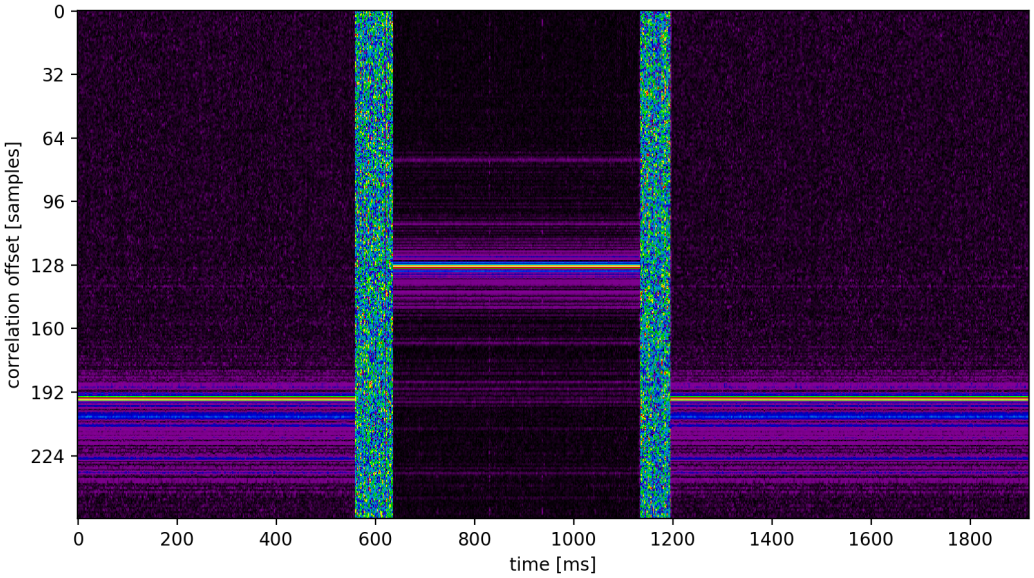


Figure 2.17: The resulting image with both reference and target (rotated)

We see that we have listened for about 550 ms to one transmitter, then there is some noise while the radios were retuning, then 500 ms of another transmitter and then again the first one. We can see that the time difference of arrival between these two transmitters is about 66 samples (8.8 kilometers when sampled at 2.25 MS/s), which indeed is the correct result[5]. To determine this result algorithmically, we fit the lines (the target and the reference) using linear regression and then calculate their distance along the offset axis (the vertical axis on Figure 2.17, horizontal on 2.14). By this, we measure the time difference $t_{\text{tgt}} - t_{\text{ref}}$ similar to 2.1,

$$d_{t,1} - d_{t,2} - (d_{r,1} - d_{r,2}) = (t_{\text{tgt}} - t_{\text{ref}})c. \tag{2.4}$$

We know the positions of the reference and of our receivers and we can therefore compute $d_{r,1}$ and $d_{r,2}$ for example by Haversine formula[11]. We compute the resulting difference in distance,

$$\Delta L = d_{t,1} - d_{t,2} = (t_{\text{tgt}} - t_{\text{ref}})c + (d_{r,1} - d_{r,2}). \tag{2.5}$$

2.2.14 Plotting hyperbola to a map

Having measured the ΔL, we need to plot points that satisfy the result into the map. Let the Receiver 1 be located at coordinates $(\text{lat}_1, \text{lon}_1)$ (in degrees) and the Receiver 2 at $(\text{lat}_2, \text{lon}_2)$. As symmetries would complicate the explanation, assume that $\text{lon}_1 < \text{lon}_2$ and $\Delta L > 0$.

[5]The location of these transmitters can be found on the website of local regulatory office and the distance then simply measured on a map.

We first create a tangent plane to the Earth surface, with cartesian coordinates and zero exactly in the middle between our receivers. We define a function mapping geographical coordinates to our grid with unit kilometers[6]:

```
Ce = circumference of the Earth [km]

# Get the center of projection
shift_lat = (lat1+lat2)/2
shift_lon = (lon1+lon2)/2

# Scale the longitude, as the length of a given circle of latitude
#  is shorter when not on the equator.
lon_scale = cos(radians(shift_lat))

def spherical2dist(lat, lon):
  # Return horizontal and vertical coordinates in our grid
  dx = (lon - shift_lon)/360 * lon_scale * Ce
  dy = (lat - shift_lat)/360 * Ce
  return dx, dy
```

Recall the definition and basic properties of a hyperbola[12]:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \tag{2.6}$$

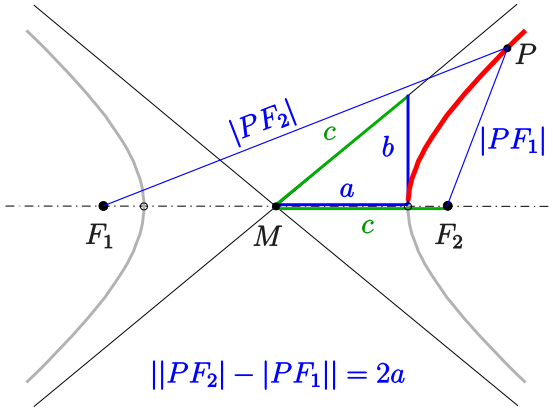


Figure 2.18: A hyperbola. M is center of our coordinates, F_1, F_2 our receivers, P possible location of the target transmitter.

The distance difference is exactly our ΔL, $2a = \Delta L$. Additionally, the coordinates of point M are $[0, 0]$. We also know $c = \text{distance}(F_1, F_2)/2$ and therefore b from the right-angle triangle abc. Solve the equation (2.6) for y:

$$\begin{aligned} y^2 &= \frac{x^2 b^2}{a^2} - b^2 = \frac{b^2(x^2 - a^2)}{a^2} \\ y &= \pm b \frac{\sqrt{x^2 - a^2}}{a}. \end{aligned} \tag{2.7}$$

[6]If one needs to make this work even when the 180 East/West meridian lies between the transmitters, and near the poles, where the longitude distortion is extreme, more complicated calculations would be needed.

We can now compute the points of the hyperbola by substituting numbers from a to some large value for x in (2.7).

However, the real geometry probably does not look like Figure 2.18, that is, F_1 and F_2 do not lie on the X axis. We therefore need to rotate the points by angle $\Theta = \arctan((F_{2,y} - F_{1,y})/(F_{2,x} - F_{1,x}))$[7] around the center M (which is conveniently chosen to be $[0, 0]$).

Finally, we apply the transform inverse to `spherical2dist` and obtain a list of latitudes and longitudes, which we can import into virtually any geographical software (for example, GpsPrune).

How far apart should one lay the x coordinates evaluated by (2.7)? That depends on the shape of the hyperbola — whether it is almost line-like ($\Delta L \simeq 0$), or very curved ($\Delta L \simeq \pm 2c$). We use a simple adaptive algorithm, starting with an x increment of 10 meters, computing the point, and then if it's too far away from or too near to the previous point, changing the increment size accordingly. We also lay the points closer to each other near the vertex, as the hyperbola is more curved there, so higher resolution is necessary.

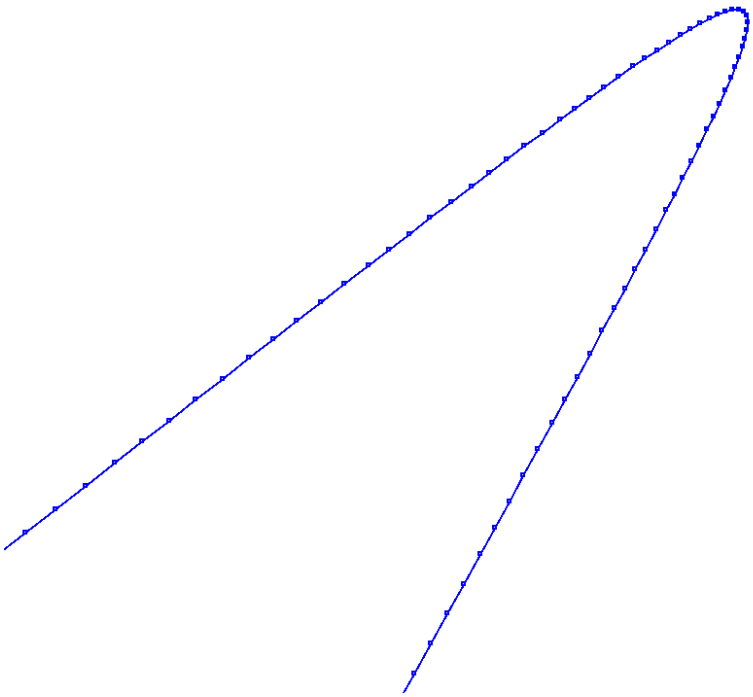


Figure 2.19: The generated hyperbola.

2.2.15 Fixing non-optimal correlation functions

Unfortunately, the result is not always as good as shown in Figure 2.17. The following factors may cause the correlation function to be noisy:

1. Low signal intensity, as the transmitter is far away, has low power, has directional antenna that is pointing away from our receiver or there is a hill or a building between the transmitter and the receiver. The problem here is that we need good signal on at least three receivers for full location and

[7] arctan2 in the general case without the $\text{lon}_1 < \text{lon}_2$ assumption.

the receivers should preferably be far from one another to minimize the dilution of precision (2.1.4). This makes transmitters with directional/sector antennas (as are for example most BTSes of cellular networks) difficult to locate.

2. The signal is narrowband. Intuitively, as we are computing $\mathrm{x}(f, g) = \mathrm{IFFT}(\overline{\mathrm{FFT}(f)} \odot \mathrm{FFT}(g))$, we want the spectra of the signals to be wide and flattop, as inverse Fourier transform of such functions is a narrow, sharp pulse.

3. The signal has poor autocorrelation function. An example of this is FM broadcast transmitting speech: during pauses between words, the only payload being transmitted is a 19kHz pilot tone (and low-bitrate RDS data), making the signal self-similar every $1/19\mathrm{kHz} = 53\mu\mathrm{s}$. Certain phonemes or even musical instruments cause similar problems which can be seen in the Figure 2.20.2. On the other hand, digital transmission modes use various compression, entropy coding and spread-spectrum methods making them look "noise-like" and therefore this problem is usually not present.

4. There is interference from another transmitter. Poor receiver selectivity may result in signal from another nearby transmitter mixing with our target, making us measure difference between other transmitters than we think we are measuring.

The problem (1) has no solution known to us except investing more in hardware, for example using high-gain antennas and better radios and getting many more receivers so at least three of them always have good reception. An approach to this is described in [13], which analyzes precisely locating mobile phones using cellphone towers. Given that modern BTSes are implemented using software-defined radios, creating such system might even be possible by a single software update, at least for frequency bands where the BTSes are able to receive signals.

Regarding problems (2) and (3), literature suggests manipulating the input signal to obtain a better correlation result. This approach is tested in 2.2.16, but with no noticeable improvement.

We deal with the problem (4) in the user interface of our program: spectra of the signal are displayed in measurement report and should an interference happen (which can be usually seen from the spectrum), the user is expected to modify the settings of the recording to try to alleviate it, for example by slightly changing the baseband frequency in order to tune farther away from the interference source or lowering the gain.

The image generated by Algorithm 3 is handy for debugging the process. As we can see in 2.20, the quality of real-life correlation functions varies wildly and a simple linear regression of the peaks as presented in 2.2.12 is not enough to determine the correct result. We have devised the following heuristics:

- Compute fitness of each peak as the ratio of its height and a mean value of the rest of the correlation.
- Discard peaks with fitness lower than certain threshold. This fixes the situation of FM broadcast briefly transmitting silence (and hence producing a "bad" correlation), certain synchronization marks found in DAB multiplex

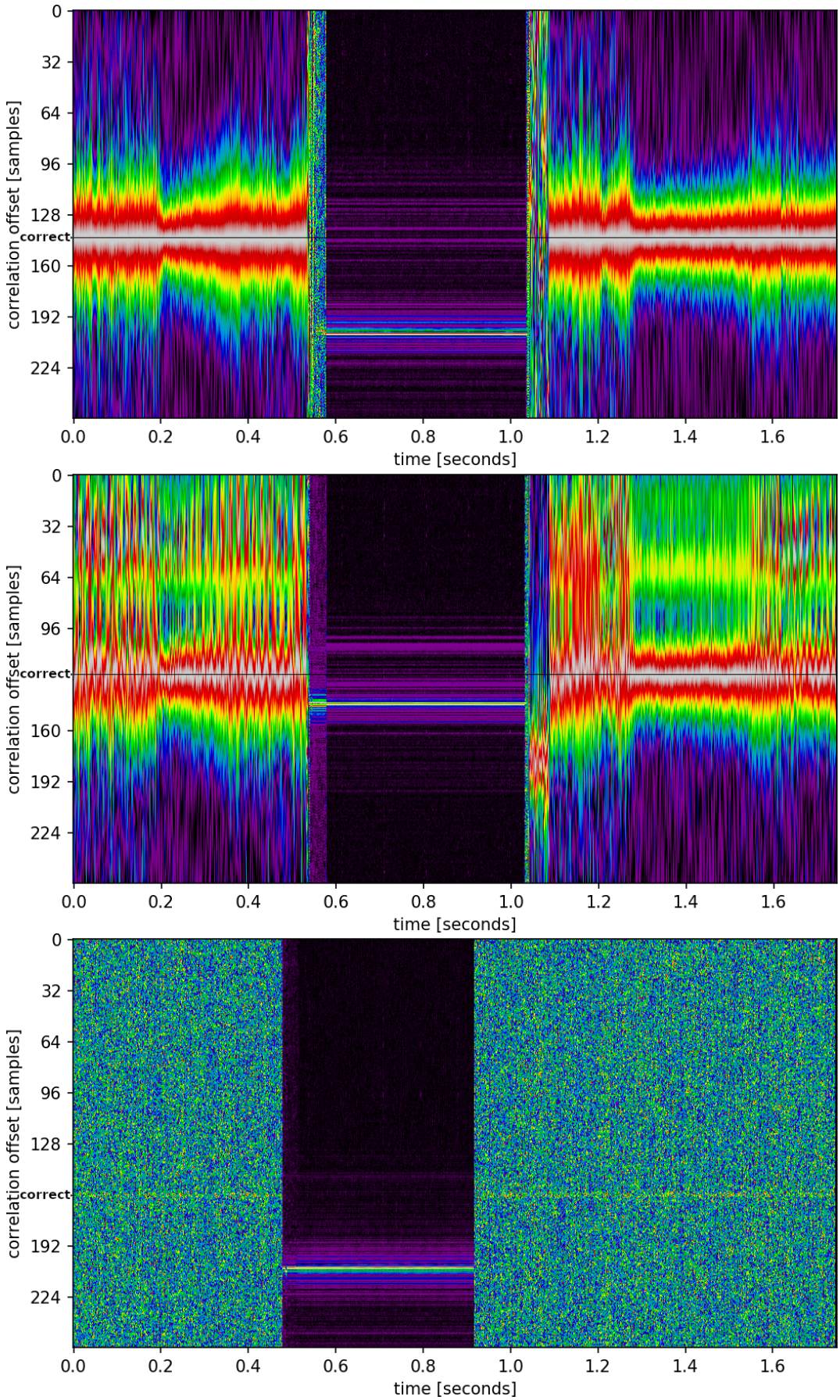


Figure 2.20: Examples of mediocre and bad correlation functions, computed by Algorithm 3. The correct position of the peak is marked. The first two are FM broadcast, the last one DAB radio with low SNR.

or just some brief dropout in the signal. We have chosen to sort peaks by fitness and drop the first decil (a hyperparameter that can be tuned).

Now there are still peaks that are randomly "off", and even if they were uniformly distributed, they would contribute with a nonzero mean, because the correct result is not exactly in the center and therefore there would be more spurious peaks on one side of it than on the other. We also cannot discard everything farther than a certain distance (e.g. two standard deviations) from mean or median, as these are biased for the same reason. We will use the approach already introduced in 2.2.11: we accumulate positions of the peaks, smooth the result with a moving average filter of length 3 (can be a lot shorter than in Algorithm 2, as we have performed precise correction of sample rate) and discard everything that is farther than 2 samples from argmax of this smoothed function. Finally, we perform linear regression on the result.

This algorithm is able to extract the correct result with error of less than 0.5 samples from all three examples presented in Figure 2.20.

2.2.16 Prefiltering and whitening

The authors of [14] define a general method for normalizing the spectrum product ($\mathrm{FFT}(f) \odot \overline{\mathrm{FFT}(g)}$) before computing the inverse transform. With the right normalization, we might be able to attenuate the frequencies in the signal which cause the correlation function to be poor and amplify the useful ones. As an example, in [15], a signal with strong power-line interference (60 Hz) is given, resulting in crosscorrelation function with major peaks every 1/60 s. The normalization then attenuates this frequency and correlation of other components of the signal becomes visible. On the other hand, the authors of [16] have tested one of the popular filters (SCOT, introduced in the next paragraph) and found out that it does not perform well in a case that is potentially similar to our scenario:

> For the case of a narrow band Gaussian noise signal embedded in white Gaussian noise, the standard cross correlation method of time delay estimation has shown significantly better performance than the SCOT method.

Let us define $S_{a,b} = FFT(a)\overline{FFT(b)}$ and the generalized crosscorrelation method as

$$\mathrm{GCC}(f,g)_k = \mathrm{IFFT}(\psi \odot S_{f,g}), \tag{2.8}$$

where ψ is the frequency weighting.

We have tested the following values for ψ suggested in [14]:

- *(The Smoothed Coherence Transfom (SCOT))* $\psi = 1/\sqrt{S_{f,f}S_{g,g}}$
- *(The Eckart Filter)* $\psi = |S_{f,g}|(S_{f,f} - |S_{f,g}|)(S_{g,g} - |S_{f,g}|)$
- *(The HT Processor (HT))* $\psi = |S_{f,g}|/|S_{f,f}S_{g,g}|(1 - |\frac{S_{f,g}^2}{S_{f,f}S_{g,g}}|)$.

For evaluation, we took 0.7 seconds long samples (consistent with the length of our measurement process described above) of the following signals, all of them sampled at 2.25 MS/s:

- Artificially generated white noise with bandwidth 140 kHz.
- A recording of FM broadcast transmitting music with bandwidth 140 kHz.
- The same broadcast, transmitting spoken word (so intervals with silence are present).
- The DAB broadcast with bandwidth 1.5 MHz.

First, we have worked with a simulation:

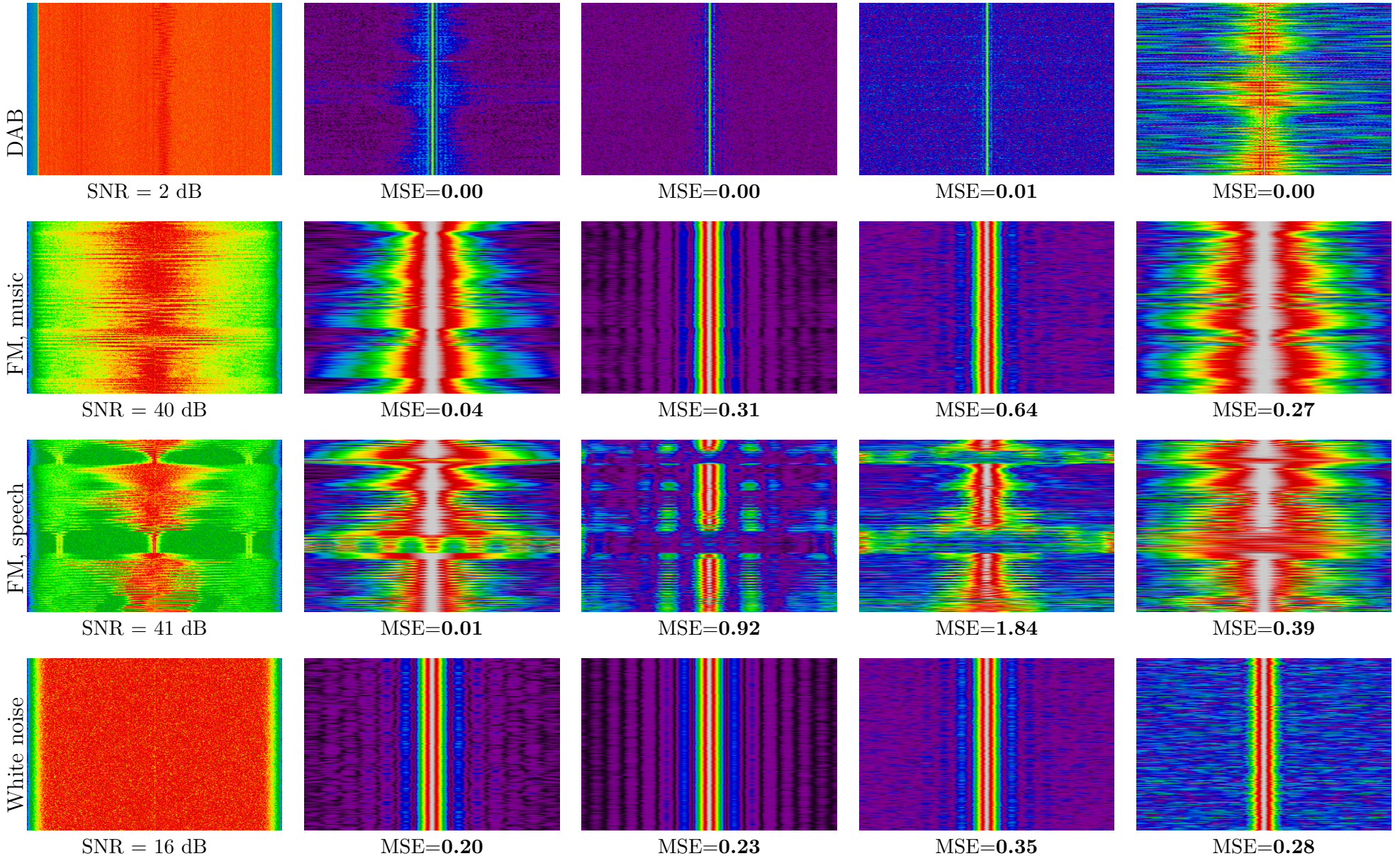
- The signal is duplicated,
- a different realization of white noise is added to each copy; the amplitude of the noise is manually selected so the crosscorrelation function without filtering resembles real crosscorrelation functions such as the one shown in Figure 2.20.1,
- the resulting SNR is computed as the 90^{th} percentile of the power of the signal minus the minimum power of the noise floor,
- multipath distortion is simulated by filtering one of the copies with filter $[0.2, 0, 0, 0, 0, 1, 0, 0, 0, 0.1]$,
- filtered crosscorrelation for each of the evaluated ψ is computed, with window of length 4096 samples and decimation 1024, and with the statistical postprocessing described in 2.2.15,
- mean square error (the square of the distance to the correct result) of the result is computed.

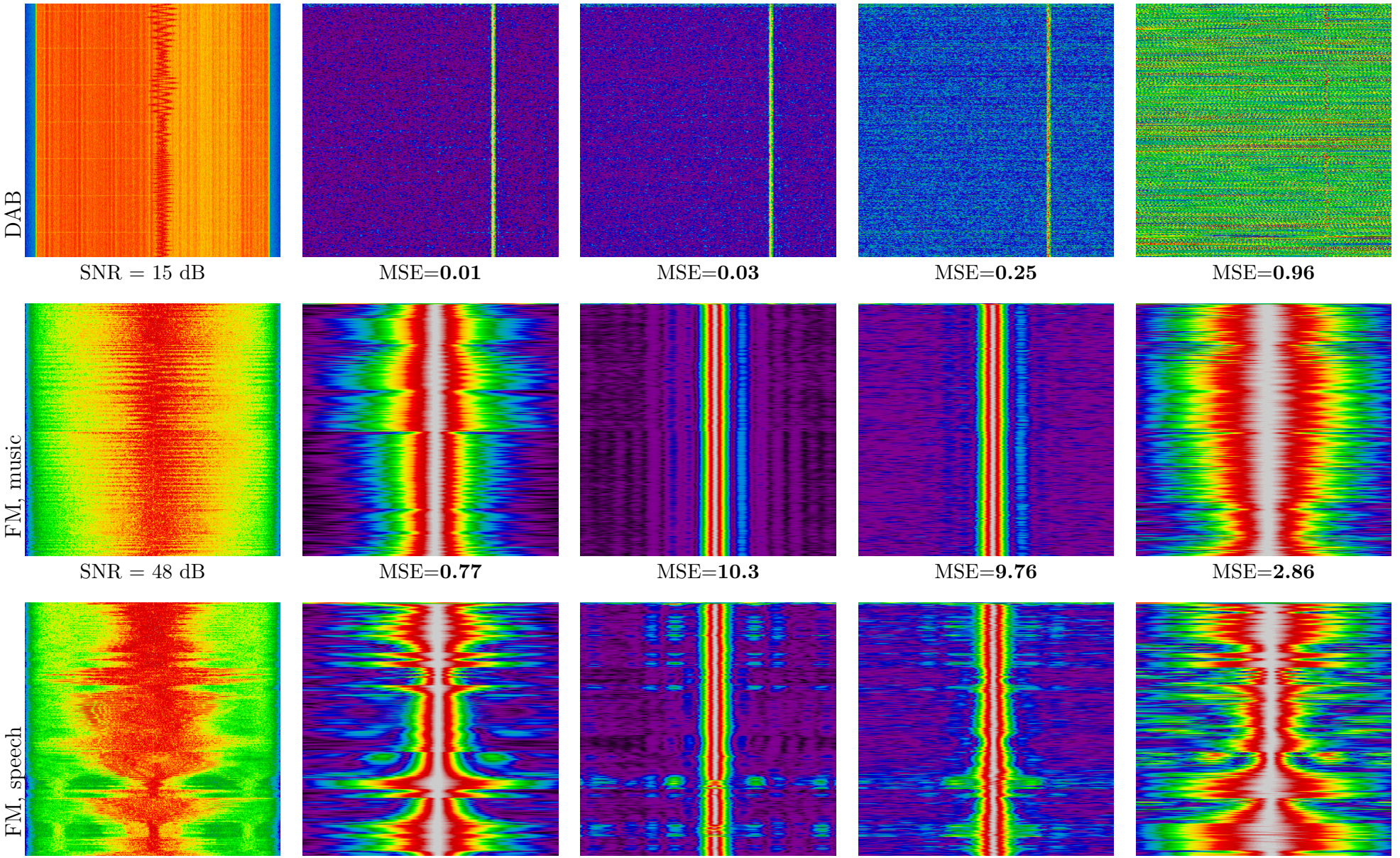
Then we did the same process, but without adding the noise and multipath distortion, with real signals received from two of our nodes.

One improvement can be made: the signal does not occupy the whole bandwidth (for example the 140 kHz FM broadcast is less than one tenth of 2.25 MHz provided by our radio), and the rest is probably just noise (or worse, residual signals from other transmitters left in place by imperfect filtering). We therefore force ψ to zero everywhere except for the range where the signal is assumed to be present.

The results are shown in the following tables[8]. The filtered correlation functions look somewhat "sharper", but unfortunately the error of the position of peaks is even worse than in the unfiltered case.

[8]The first column of each table shows STFT of the signal, with time on the vertical axis and frequency on the horizontal; the STFT is zoomed in frequencies so only the signal is shown (otherwise there would be huge margins). The other columns show results of crosscorrelation when applying various filters ψ. Time offset is on the horizontal axis, just like in Figure 2.14.


(Simulation) Waterfall
No whitening
SCOT
HT
Eckart
DAB
SNR = 2 dB
MSE=0.00
MSE=0.00
MSE=0.01
MSE=0.00
FM, music
SNR = 40 dB
MSE=0.04
MSE=0.31
MSE=0.64
MSE=0.27
FM, speech
SNR = 41 dB
MSE=0.01
MSE=0.92
MSE=1.84
MSE=0.39
White noise
SNR = 16 dB
MSE=0.20
MSE=0.23
MSE=0.35
MSE=0.28


(Real data) Waterfall
No whitening
SCOT
HT
Eckart
DAB
SNR = 15 dB
MSE=0.01
MSE=0.03
MSE=0.25
MSE=0.96
FM, music
SNR = 48 dB
MSE=0.77
MSE=10.3
MSE=9.76
MSE=2.86
FM, speech
SNR = 54 dB
MSE=1.20
MSE=0.79
MSE=1.70
MSE=1.25

2.2.17 Dealing with Single Frequency Networks

DVB-T and other digital television and radio standards allow multiple transmitters to transmit on the same frequency at the same time. Usually one region of the size of a few thousands square kilometers is covered by a few transmitters, all operating in one SFN. This saves the frequency spectrum and increases signal strength. A few tricks must be employed for this to work:

- The transmitters must all transmit the signal at precisely the same time, so buffering and synchronized time must be used.
- The modulation scheme must use slow enough symbol rate, as the symbols from each transmitter, despite being transmitted at the same moment, arrive delayed by various amount of time due to different distances and thus different propagation delays. These delays are of course different on different places around the covered area. For example if the transmitters are 50 kilometers apart, any rate over $1/(50\text{km}/c) = 6000\text{Symbols/s}$ will result in complete overlap of consecutive symbols. DVB-T deals with this by using an OFDM[9] with thousands of carriers, each transmitting a separate stream of symbols at a very slow rate.
- Moreover, to prevent intersymbol interference and crosstalk altogether, a small delay with silence (called guard interval) should be inserted after each symbol. And as the signal should not have sudden changes in its power, it should not be such that all carriers present a symbol and then are all switched off; the carriers must be shifted by a fraction of symbol relatively to each other, so some transmit and some are silent at each point of time.
- The modulation scheme must be robust to narrowband interference. The radio waves from different transmitters may produce both constructive and destructive interference. And as the television signal is wideband and interference patterns depend on frequency, it is expected that across the channel some frequencies would be attenuated. In DVB-T, this means that a few carriers from the OFDM would have signal level unusable for decoding. Agressive error correction and bit scrambling are employed so the original bitstream can be reconstructed. Overall, the effects of this interference should be positive, which is called SFN gain.

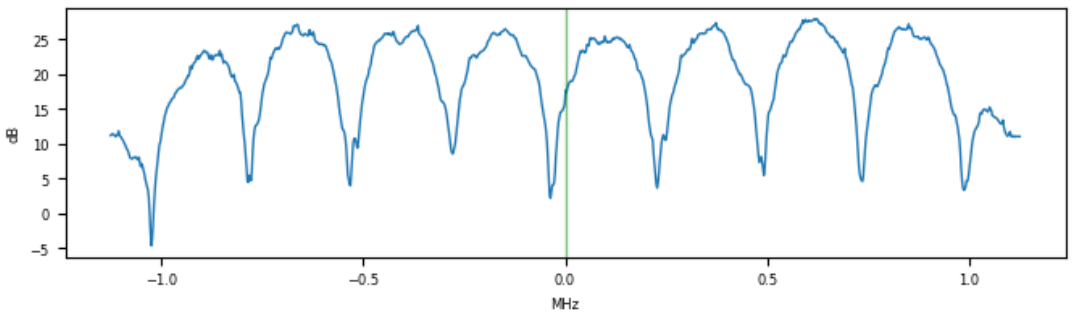


Figure 2.21: A part of spectrum of a received DVB-T SFN showing narrowband destructive interference.

[9] orthogonal frequency division multiplex

From the perspective of TDOA, the SFN is seen as multiple discrete correlation peaks, each corresponding to one transmitter. Fortunately, the approach described in 2.2.15 allows us to separate one peak. We compute the time difference using the separated peak, then remove it (simply force the crosscorrelation function to zero at that offset plus some safety padding), take the next peak and so on. We then draw multiple hyperbolas, one for each peak. The result looks messy and there are some false intersections, but that is probably everything we can do. It may help to compute the solution in 3D and discard the solutions that are not on the Earth surface.

3. Angle of arrival

Another approach to emitter location is determining the direction from which the signal is coming. The transmitter then lies on the ray defined by the location of the receiver and this angle. Then we can either use multiple receivers and find an intersection of their rays, or move in the direction of the ray, finally approaching the transmitter.

The advantage of AoA over TDoA is that while TDoA requires at least two synchronized receivers to make a measurement, an AoA measurement can be made without any remote support, and in case of non-moving transmitters, the other rays needed for intersection can be cast anytime in the future.

AoA also has different behavior of errors: nearby targets will be located with lower spatial error, whereas TDoA is expected to have similar error on the entire covered area (given the strength of the signal is sufficient and the dilution of precision is not very high). In the "approaching the transmitter" use-case, the error of AoA is therefore expected to go to zero during the process and even measurement with huge errors can lead to a satisfiable outcome.

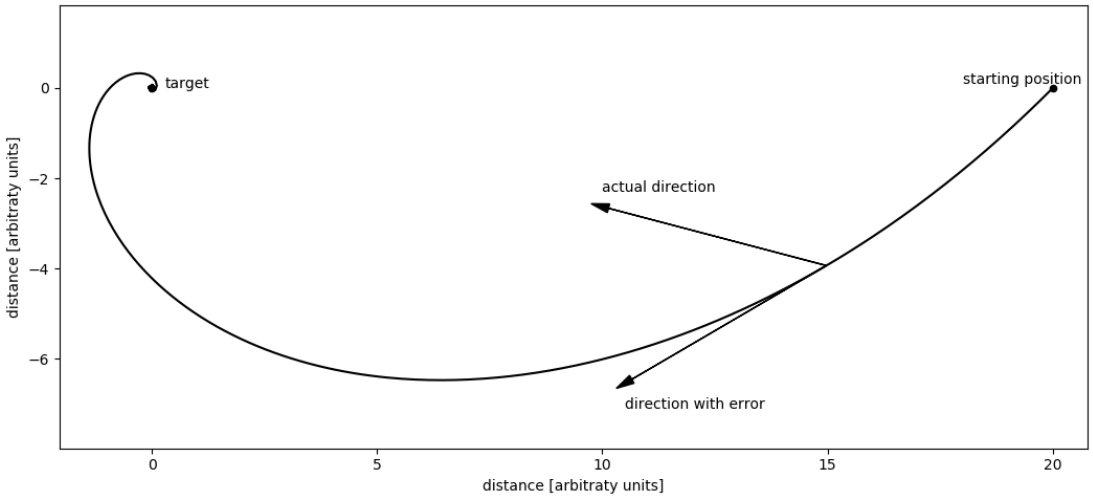


Figure 3.1: Approaching the transmitter while making a systematic error of 45° in the angle of arrival estimation.

On the other hand, AoA can fail completely in presence of reflections and multipath propagation (e.g. measuring near big buildings, hills or in valleys). And regarding complexity of the setup, while we managed to build TDoA with off-the-shelf hardware without any modifications, the methods for AoA which we present require some modifications and custom hardware.

3.1 Directional antenna

Several geometries of antennas exhibit strong maximum (Yagi antenna, log-periodic antenna, parabolic dish antenna) or minimum (magnetic loop antenna) in some direction of their radiation pattern. By rotating the antenna and finding the angle at which the intensity of the signal is maximal/minimal we get the target direction.

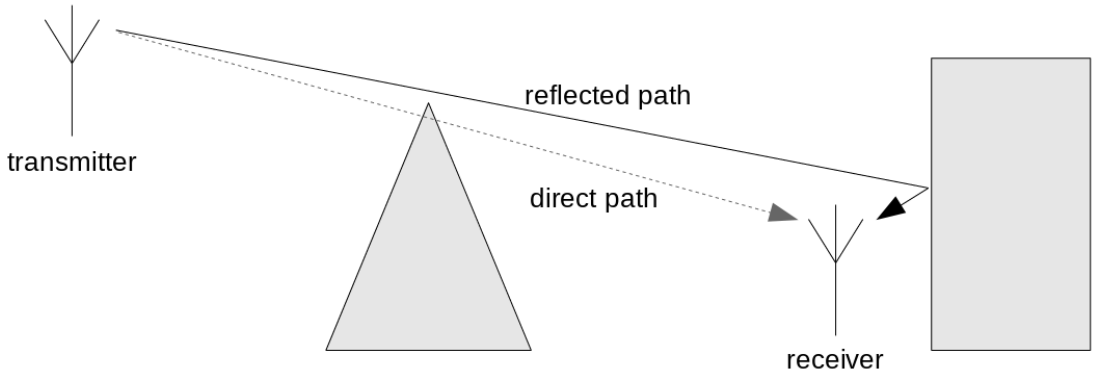


Figure 3.2: In case of blocked direct path and presence of reflections, TDoA will give an inaccurate result (the reflected path is a bit longer than the direct path), whereas AoA will give a completely wrong result (the reflected signal comes from the exactly opposite direction).

Advantage of this approach is that the directional antennas "amplify" the target signal or at least reject interference from the other directions, so even distant or low-power targets can be received reliably. Disadvatages are two: first, the antenna with sharp directional pattern is usually calculated and manufactured to operate within a narrow frequency band only and parameters on other frequencies are worse; second, this requires a rooftop mount with space for physically rotating antenna and a motorized rotator. This mechanical part must either withstand outside weather conditions or be housed in a bulky radome.

3.2 Antenna switching

Consider a transmitter transmitting an unmodulated sine wave and a receiver with two antennas less than $\lambda/2$ apart (where λ is the wavelength of the signal) which can be instantaneously switched using an antenna switch. Upon switching, a sudden jump in phase of the wave will be observed. This phase difference will be maximal when the antennas are in line with the transmitter and zero when the distance from the transmitter to both antennas is the same. This leads to two (or in some implementations four) symmetric solutions; to disambiguate them, a third antenna is added.

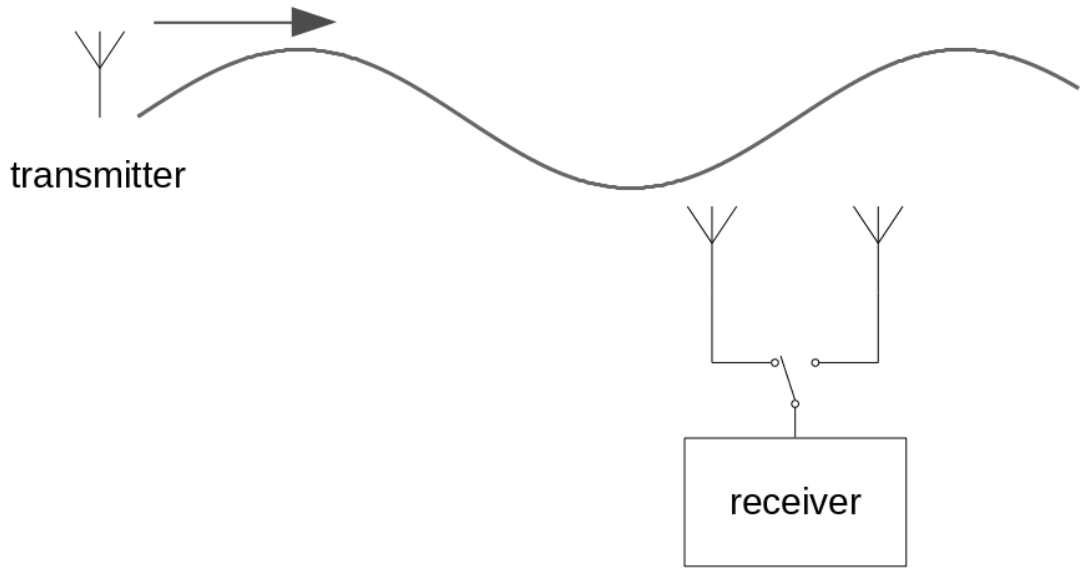


Figure 3.3: A setup with antenna switching

In reality, the signal is modulated and most modulations are shifting the phase, for example PSK[1] or FM (where immediate changes in frequency demonstrate themselves as changes in phase). We therefore do not know how much of the phase change during antenna switch comes from the switch itself and how much of it comes from the transmitter changing its phase/frequency because of the modulation. This can be sorted out by making thousands of switches over a few seconds and averaging the result — the phase changes of the remote transmitter are likely to be uncorrelated with our switching and will average out to zero.

Another problem arises when implementing this approach on a SDR without auxiliary signal input (c.f. 2.1.5), as we need to determine the moment when the switch happened to compare the phase before and after. Authors of [17] and [18] deal with this by switching at a known frequency of a few kHz, demodulating the signal with a quadrature[2] demodulator (jumps in phase produce peaks in the demodulated signal) and then selecting this known frequency with a narrow band-pass filter. The amplitude of the filtered signal corresponds to the height of the impulses, which corresponds to the magnitude of phase discontinuities in the input signal.

Then, as we want to disambiguate the symmetric solutions, we need to employ a third antenna, make a few switches between antenna #1 and #2 and then between #2 and #3 and again need to somehow infer from the recorded samples which impulses belong to which pair of antennas. As mentioned earlier, the stream from the radio is only coarsely synchronized with the "outside world" including the antenna switcher. Therefore, we either need to put the transitions between the pairs of antennas far apart enough, or perform the switching in some unique pattern which we can later recognize.

The next problem is that we cannot receive the target (as we are perhaps interested in the contents of the transmission) and make the measurement at

[1]Phase Shift Keying

[2]Regarding the nomenclature of "FM" and "quadrature" demodulator: under FM demodulator we understand a quadrature demodulator followed by a deemphasis filter. For this application the distinction is not important, though.

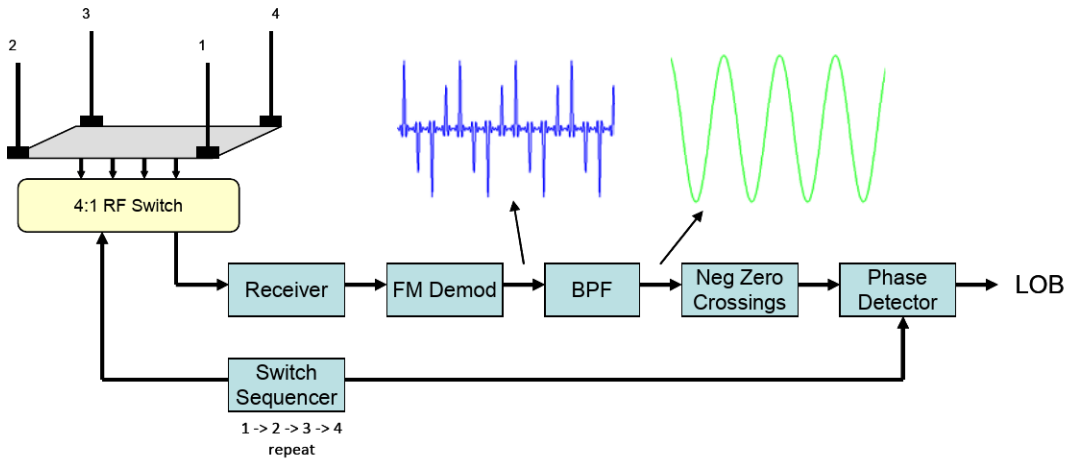


Figure 3.4: A block diagram of the device presented in [18].

once, as these jumps in phase will usually make successful reception impossible. This can be fixed by either getting a second radio which will not have the antenna switched or maybe by somehow cancelling the jumps in phase once they are detected.

And, finally, as the recording incorporates the switching signal, one of the main advantages of software-defined radio — that the "raw" recording is saved and can be processed later by various means — is lost as the switching sequence must be decided prior to the recording and cannot be changed later.

There is one particular use-case where these problems manifest very strongly: a network using TDMA[3] where we want to locate individual transmitters.

- The transmission happens only in one timeslot out of N (usually 4 or 8) plus there is spacing between the timeslots and so there will not be enough signal to perform the averaging of the phase discontinuities.
- The timeslosts are short and the narrow band-pass filter for filtering the results is longer (in the sense of number of taps) than the timeslot.
- The transmission is short and so there is no time to perform the changing of antenna pairs.
- To detect where the timeslot starts, the signal has to be correctly decoded. As the start of the timeslot is a precise time information, the add-on radio needs to be synchronized to the radio taking the measurement, which poses additional difficulty.

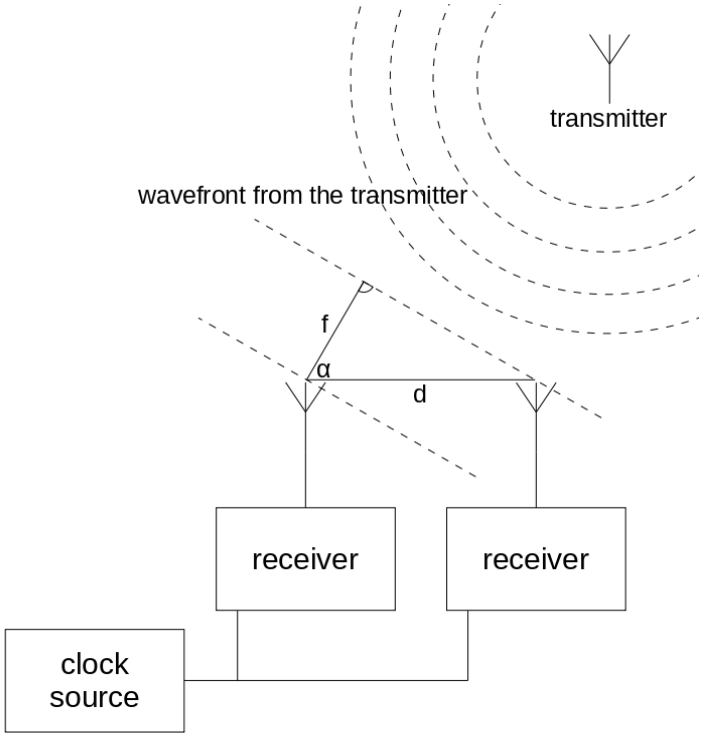
We have implemented an early proof-of-concept prototype of this approach, but after facing the problems discussed above, we have decided to abandon it.

3.2.1 Coherent receivers

We will get rid of the antenna switcher by connecting each antenna to a separate receiver and then feeding all the receivers from a common clock source. This way, the individual streams will be synchronous and we can then simply compare the phases of the received signals.

[3]Time Division Multiple Access

Geometry of coherent receivers



Assuming the transmitter is far away (distance to the transmitter $\gg d$) and $d < \lambda/2$, the coming wavefront generates a phase difference between the signals from the two receivers $\phi = f/\lambda$ (in units "fraction of a unit circle" rather than radians). From this we have

$$\begin{aligned} \cos\alpha &= \frac{f}{d} = \frac{\phi\lambda}{d}, \\ \alpha &= \arccos\frac{\phi\lambda}{d}. \end{aligned} \tag{3.1}$$

3.2.2 Implementation

We have desoldered oscillators from all the involved rtl-sdrs except one and connected pins number 9 (the clock input) of the R820T chips on all radios together. Now all the SDRs sample synchronously and by using crosscorrelation we can determine the initial offset caused by different moment of initialization. However, their phase offset (ϕ in (2.2)) is not the same because their PLLs[4] acquired lock at different times, and we cannot use the "transmitter with known location" trick because retuning to different frequency unlocks and re-locks the PLL and finally we end up with a different unknown offset. The solution used by [19] is to connect the SDRs to a common noise source, compute crosscorrelation (or just a dot product if the sample offset has already been corrected) and use the phase of the result as the correction factor. High-frequency high-bandwidth noise can be obtained by tapping the USB data lines with a small capacitor. The switching is then done by SA630 switches. One switch is used for each radio (switching

[4]phase-locked loop

between antenna and a common line) and another switch is used to switch this common line between ground and USB noise. This is done to reduce crosstalk and improve noise isolation because SA630 has the worst-case isolation of only 24 dB.

We have created a printed-circuit board for three SDRs.

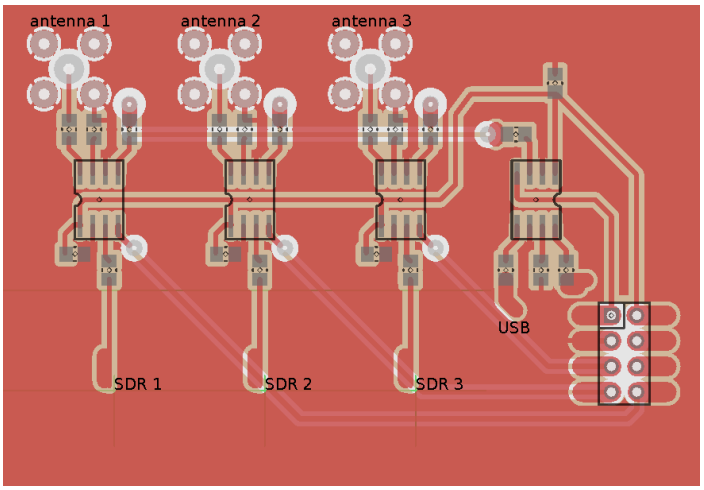


Figure 3.5: A PCB with 4 switches for 3 SDRs.

The length of the trace feeding noise is not the same to all SDRs. And when connecting the antennas, the lengths of the coaxial cables may slightly differ. Therefore we introduce a calibration process, where the user points each pair of the antennas perpendicular to a known signal source ($\alpha = 90°$, the phase difference should be zero) and the measured phase is saved. When making a real measurement, this saved phase is subtracted from the measured phase.

The quality of the RF part

Figure 3.6: The default rtl-sdr antenna

The first prototype used a hand-drawn single-sided PCB and the antennas that come with rtl-sdr. We have immediately noticed that even with switches switched

to the noise source, there is a huge leak of signals from the outside. It had to be fixed in several ways:

- The board is now double-sided, with the bottom side (filled mostly with ground plane) acting as shielding. The board is made by photo-printing for higher precision.
- The sides of the board are wrapped in a copper foil, shorting ground planes of the top and the bottom side.
- The device is put in a metal box for shielding.

The second problem was with antennas. We have noticed that the measured phase shift depends on the shape and twisting of the coaxial cables that feed the antennas. We hypothesize that this is caused by the antennas having absolutely no symmetrization, so the incoming radio wave resonates over the cable shielding. Replacing it with a ground-plane antenna fixed the problem and the readings then seemed reasonable.

Dilution of precision

Similarly to 2.1.4, the result is unstable near the maxima and minima of the phase difference (intuitively, the absolute value of the derivative of the arccos function is high near the edges). As we have three antennas and therefore get three phase differences, we can discard the one that is neares to the edge and compute the result using the remaining two.

Remarks regarding the electronics design

We need a logic signal controllable from the computer to control the antenna switches. Due to the shortage of exposed GPIO[5] pins on modern computers, we can either use the integrated I2C bridge in rtl-sdr and an I2C logic expander, or just put a cheap Arduino clone aside. We have chosen the second option for simplicity.

USB cannot deliver enough power for three radios. An external power supply had to be added.

[5]General-Purpose Input/Output

Conclusion

TDOA

Can a TDOA multilaterator be built using consumer-grade hardware? Apparently, after implementing the described quirks to overcome unstable clock source in the radios, yes. And surprisingly for us, the main limitation was not precise timing synchronization, but poor autocorrelation functions of some signals (and low SNR and selectivity of the radio used).

Future work

How could be our work further improved?

- Upgrading to better SDRs with higher dynamic range and better robustness to interference from strong transmitters might allow locating even weak transmitters and transmitters that are not on elevated places — for example uplink channel from mobile phones and walkie-talkies could be traced. This should be relatively easy to implement, as almost all logic dealing with specifics of rtl-sdr is concentrated in `rtl.c`, which could be re-implemented for other SDRs.
- Additionally, with an SDR that is able to receive frequencies in the microwave range, it might be possible to receive pulses from airborne radars and therefore passively track airplanes with a secondary responder turned off (but radar turned on).
- The speed of the software could be improved: the correlator could run in multiple threads and generation of the plots could be more efficient (currently, most of the time is spent in generating the images). The complete report currently takes about a minute to generate on a middle-end computer.
- A more precise resampler may improve subsample accurracy.

AOA

Implementing AOA direction finder proved challenging and difficult to debug with respect to RF design. Despite the described efforts, the measured angle sometimes goes completely wrong for a while. This may be caused by spurious radio reflections in the environment and inside the device. Future work should concentrate on having matched impedance along the entire signal path and overall improvement of the physical design.

Bibliography

[1] James E. Palmer, H. Andrew Harms, Stephen J. Searle, and Linda M. Davis. DVB-T Passive Radar Signal Processing. *Trans. Sig. Proc.*, 61(8):2116–2126, April 2013.

[2] Fadel Adib, Chen-Yu Hsu, Hongzi Mao, Dina Katabi, and Frédo Durand. Capturing the human figure through a wall. *ACM Trans. Graph.*, 34(6):219:1–219:13, October 2015.

[3] David Klusáček. New Methods in Statistical Speech Recognition. `https://dspace.cuni.cz/handle/20.500.11956/41647`, 2012. Accessed: 2018-04-27.

[4] Mutability Ltd. Mode S multilateration server. `https://github.com/mutability/mlat-server`, 2015. Accessed: 2018-12-20.

[5] Josef Gajdůšek. Pasivní radiolokátor. `https://maturita.atx.name/`, 2017. Accessed: 2018-12-20.

[6] Stefan Scholl. Experimental Matlab Scripts for Evaluation of a TDOA System based on RTL-SDRs. `https://github.com/DC9ST/tdoa-evaluation-rtlsdr`, 2017. Accessed: 2018-12-21.

[7] G. N. Varma, U. Sahu, and G. P. Charan. Robust frequency burst detection algorithm for GSM/GPRS. 6:3843–3846 Vol. 6, Sep. 2004.

[8] Rafael Microelectronics, Inc. R820T High Performance Low Power Advanced Digital TV Silicon Tuner Datasheet, 2011.

[9] Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. scikit-optimize/scikit-optimize: v0.5.2, March 2018.

[10] Jan Hrach. Frequency Spectrum Monitoring System. `https://dspace.cuni.cz/handle/20.500.11956/80128`, 2016.

[11] Wikipedia contributors. Haversine formula — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=892532744`, 2019. [Online; accessed 2-May-2019].

[12] Wikipedia contributors. Hyperbola — Wikipedia, the free encyclopedia, 2019. [Online; accessed 2-May-2019].

[13] George A. Mizusawa. Performance of hyperbolic position location techniques for code division multiple access. 1996.

[14] C. Knapp and G. Carter. The generalized correlation method for estimation of time delay. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(4):320–327, August 1976.

[15] G. C. Carter, A. H. Nuttall, and P. G. Cable. The smoothed coherence transform. *Proceedings of the IEEE*, 61(10):1497–1498, Oct 1973.

[16] K. Scarbrough, N. Ahmed, and G. Carter. An experimental comparison of the cross correlation and SCOT techniques for time delay estimation. In *ICASSP '80. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 807–810, April 1980.

[17] WA2EBY Mike Kossor. A Doppler Radio-Direction Finder. 1999.

[18] RasHAWK: Distributed EM Situational Awareness Based on Raspberry Pi and REDHAWK. 2014. Accessed: 2018-12-20.

[19] tejeez. Synchronized RTL-SDR receivers and direction finding. `https://github.com/tejeez/rtl_coherent`, 2015. Accessed: 2018-12-28.

A. TDOA: user guide

A.1 Installation

The software is divided into two parts: the recorder, which must be installed on the nodes, and the correlator, which must be installed on the central server.

We will be cloning the software from git repositories; snapshots of the repositories can be found in the supplementary material.

List of dependencies required: `volk`, `fftw3`, `python3`, `numpy`, `scipy`, `cffi` and `bottle`. The controlling server additionally requires `sqlite3` and `matplotlib`.

A.1.1 Recorder

First, the patched rtl-sdr (which allows setting of gains (2.2.8) and disables dithering (2.2.3)) needs to be installed. Refer to the official rtl-sdr documentation for more details.

```
$ git clone https://jenda.hrach.eu/p/rtl-sdr && cd rtl-sdr
$ git checkout no_dither
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/rtl-sdr ..
$ make && sudo make install
```

Next we need kalibrate-everything (2.2.7).

```
$ git clone https://jenda.hrach.eu/p/kalibrate-everything
$ cd kalibrate-everything
$ make && sudo make install
```

Finally we compile the recorder.

```
$ git clone https://jenda.hrach.eu/p/rtl-tdoa && cd rtl-tdoa/recorder
$ make RTLSDR_PATH=/opt/rtl-sdr
```

A.1.2 Correlator (the controlling server)

```
$ git clone https://jenda.hrach.eu/p/rtl-tdoa && cd rtl-tdoa
$ make
```

We also need to initialize an empty database.

```
$ cat schema.sql | sqlite3 tdoa.sqlite
```

A.2 Configuration

All the data and configuration is held in a SQLite database `tdoa.sqlite`. First, you will need to configure your nodes. Open table `nodes` and add a record for every node:

- name: some human-readable ID
- url: the API endpoint under which the node is available, e.g. `http://127.0.0.1:8080`
- lat,lon: location of the node
- gsmfreq: frequency of a nearby BTS (used for calibration); use for example `kalibrate-rtl` with the `-s` (scan) parameter to find one
- device_index: index of the radio to be used; if you have only one radio connected, enter `0`
- ppm: initial error of the oscillator
- samplerate: sample rate, must be a multiple of 250000. 2000000 or 2250000 is recommended. Currently all nodes in the system need to have the same sample rate.

Next, you need to configure transmitters in the `ctu` table. You must add at least one transmitter to be used as a reference, but we recommend adding more transmitters as locating a known transmitter will show you useful debug info. Try searching the website of your local regulatory authority; maybe they release a table that can be conveniently imported. The following fields are important:

- loc,name: human-readable description of the transmitter location and name
- freq: frequency
- type: the type of the transmitter, e.g. `DAB`. The `bandwidths` table must contain the bandwidth of it.
- lat,lon: location of the transmitter

A.3 Taking a measurement (CLI version)

The program for controlling the nodes, `tdoa.py`, can be either used on a command line, or imported as a Python module. We will use the command-line interface here. Run `./tdoa.py help` for a description of available commands.

Start the recorder (`./record.py`) on all nodes. By default it listens to the world; add an IP address as an optional argument to limit listening only on this address. Then try running `ping` on the master server:

```
$ ./tdoa.py ping
```

You should receive `pong` from all nodes. The next thing we should do is to start the radios on all nodes:

```
$ ./tdoa.py start
```

To run more commands in one invocation, separate them with comma. Let's make a recording at 100.5 MHz using the DAB transmitter ID 1925 (ID from the table `ctu`) at 227.36 MHz as a reference:

```
$ ./tdoa.py calibrate, optimize_gain 227360000 adcrange,
  optimize_gain 100500000 adcrange, calibrate, record 1925 100500000
```

An ID of the recording will be returned. We can now compute correlations, for example for a transmitter ID 1633 at 101.1 MHz (i.e., an offset of 600 kHz) with bandwidth 140 kHz:

```
$ ./tdoa.py correlate <the_returned_id> 600000 140000 -g 1633
```

You can run multiple correlations for one recording — for example should there be another transmitter at 100.7 MHz, we would also run `./tdoa.py correlate the_id 200000 140000`.

To view the measurement protocols, view `protocols/index.html` in your browser.

A.4 Taking a measurement (library version)

To use the TDOA client as a Python 3 module, import and instantiate the `tdoa` object.

```
from tdoa import tdoa
t = tdoa()
```

There are functions with the same name and similar parameters as the commands described in A.3:

```
t.ping()
t.start()
t.calibrate()
t.optimize_gain(reference_freq, "adcrange")
t.optimize_gain(target_freq, "adcrange")
record_id = t.record(reference_id, target_freq)
t.correlate(record_id, offset, bandwidth, ["-g", gold_id])
```

Refer to docstrings in the module for a more elaborate parameter description. The sample script `collect_stuff.py` uses the library to locate all transmitters selected from the database.

B. TDOA: technical information

B.1 Locking

Commands to the nodes should not be issued by two clients at once, as one might instruct the node to tune to one frequency and another to some other frequency. We prevent accidental execution of two clients (on one machine) by creating a lockfile; no attempts are made to synchronize concurrent clients from different machines, as the protocol has been designed out of the neccessity of master-slave communication and we do not consider multi-master situation an important use-case.

The CLI client handles locking automatically. Users of the Python library should call `acquire_lock` and `release_lock`.

B.2 Playing with gain

The tool `recorder/gain.py` allows you to manually set all gains of rtl-sdr and display the resulting spectrum and histogram in real time. Run `gain.py <freq in MHz> <error in ppm>` and then press `w`/`e`, `s`/`d` and `x`/`c` to adjust gains, `a` to switch autogain, `q` to quit. The tool prints current settings and esimated SNR (as the difference of minimum and maximum in the spectrum).

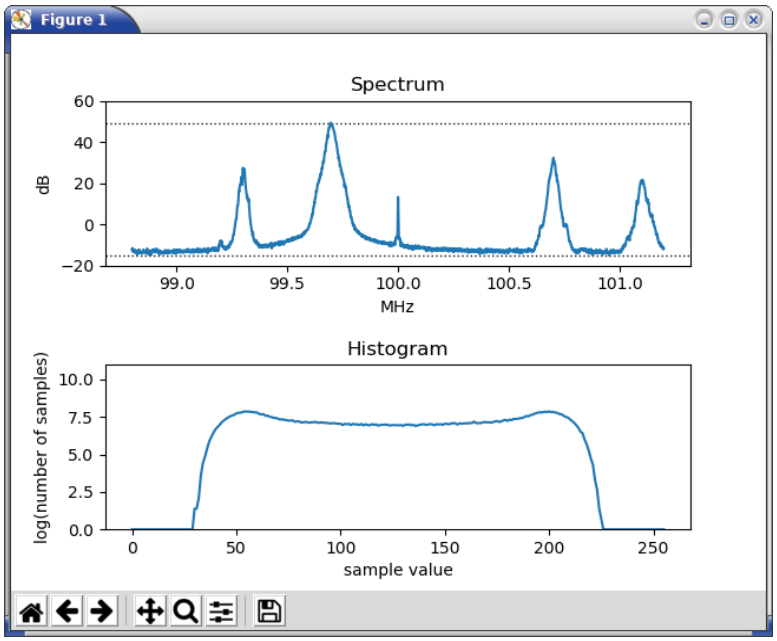


Figure B.1: gain.py

B.3 Recorder API

The recorder has the following API endpoints used by the `tdoa.py` tool. To make a recording they should be called in this order, though of course it is not needed to call `/ping` and `/start` repeatedly.

/ping

Test that the node is alive.

parameters	none
returns	`pong (running)` or `pong (not running)` depending on whether a `/start` command has already been issued

/start

Initialize and start the SDR.

parameters	`data`: a JSON dictionary with fields `device_index`, `ppm`, `samplerate` and `gsmfreq`
returns	`OK` or `Device already running`

/calibrate

Determine the oscillator error by listening to a GSM base station.

parameters	none
returns	decimal number: the measured error in parts per million

/setgain

Determine the correct gain

parameters	`data`: a JSON dictionary with fields `freq` (the center frequency), `method` (the string `adcrange`, `force` or `random`; the methods `adcrange` and `random` are described in 2.2.8); for the method `random` `flist`, a frequency list where the utility function should be evaluated; for method `force` a list of four integers, the first three are SDR gains (0 to 15), the last one 0 or 1 means autogain
returns	a JSON list of four ints: the three gains and autogain status

Frequency list format

List of integers in the form of `offset_freq1`, `bandwidth1`, `offset_freq2`, `bandwidth2`, `...` For example `200000 100000 -400000 100000` means two 100 kHz signals, one at +200 kHz and one at -400 kHz.

/gaincache

Retrieve the determined gains.

parameters	none
returns	dictionary mapping frequency to list of 4 integers (gain settings + autogain)

/record

Perform a recording of target-reference-target. `/setgain` has to be called for both of these frequencies prior to `/record`.

parameters	`data`: a JSON dictionary with fields `tt` (the UNIX timestamp of the start of the recording), `reference` (reference frequency), `target` (the target frequency) and `ppm` (the oscillator error)
returns	raw samples from the rtl-sdr ADC (approx. 8 MB of data)

B.4 Database schema

Terminology

Recording is the event when all nodes in the system record target-reference-target from a given timestamp. *Measurement* is the computed time difference between a pair of nodes ($t_{\text{tgt}} - t_{\text{ref}}$ from (2.4)).

Table: recordings

One row for each recording.

id	primary key
date	when the recording was started
reference_id	foreign key `ctu(id)`, the reference transmitter used
target_freq	the frequency of the target
comment	optional text comment

Table: files

One row for each file, multiple files for each recording (e.g. 4 files in a system with 4 nodes).

id	primary key
recording	foreign key `recordings(id)`
filename	the file where raw data of the recording are stored
lat, lon	location of the node where the data have been recorded
samplerate	sample rate of the recording
ppm	frequency error of the recording
meta	arbitrary metadata, currently the determined gain

Table: measurements

One row for each pair of files (e.g. 6 rows in a system with 4 nodes).

id	primary key
file1	foreign key `files(id)`, the first file entering the crosscorrelation
file2	foreign key `files(id)`, the second file entering the crosscorrelation
offset	frequency offset of the target
delta	the computed TDOA value
targetsd	standard deviation of the correlation peaks positions of the target
referencesd	standard deviation of the correlation peaks positions of the reference
golddelta, goldlat, goldlon	the correct values if known (e.g. we are locating a known transmitter to test the accuracy of the system)
sfn	number of correlation in a single frequency network (0 if this is not a SFN)
log	text log of the correlator

C. TDOA: Results

We have measured all commercial VHF and UHF broadcasting TV and radio stations for which a reasonable reception in Prague can be obtained: all FM broadcast stations in Prague and a few surrounding cities (such as *Kralupy nad Vltavou*) and strong DVB-T/DAB transmitters around Bohemia (such as *Ještěd*, *Milešovka*, *Ústí nad Labem* and *Pardubice-Krásné*). Even in these cases, despite the extreme dilution of precision, the results are not completely off. On the other hand, FM broadcasting suffers from inferior crosscorrelation function; about half of the locations are successfully determined, the other half fail completely. We believe this may have multiple causes:

- multiple transmitters in the same band, leading to interference due to poor selectivity of our receiver
- the analogue and low-bandwidth nature of the signal
- bending and reflections of lower frequencies which FM broadcast uses (88 to 108 MHz compared to 200 MHz for DAB and 500 to 750 MHz for DVB-T)

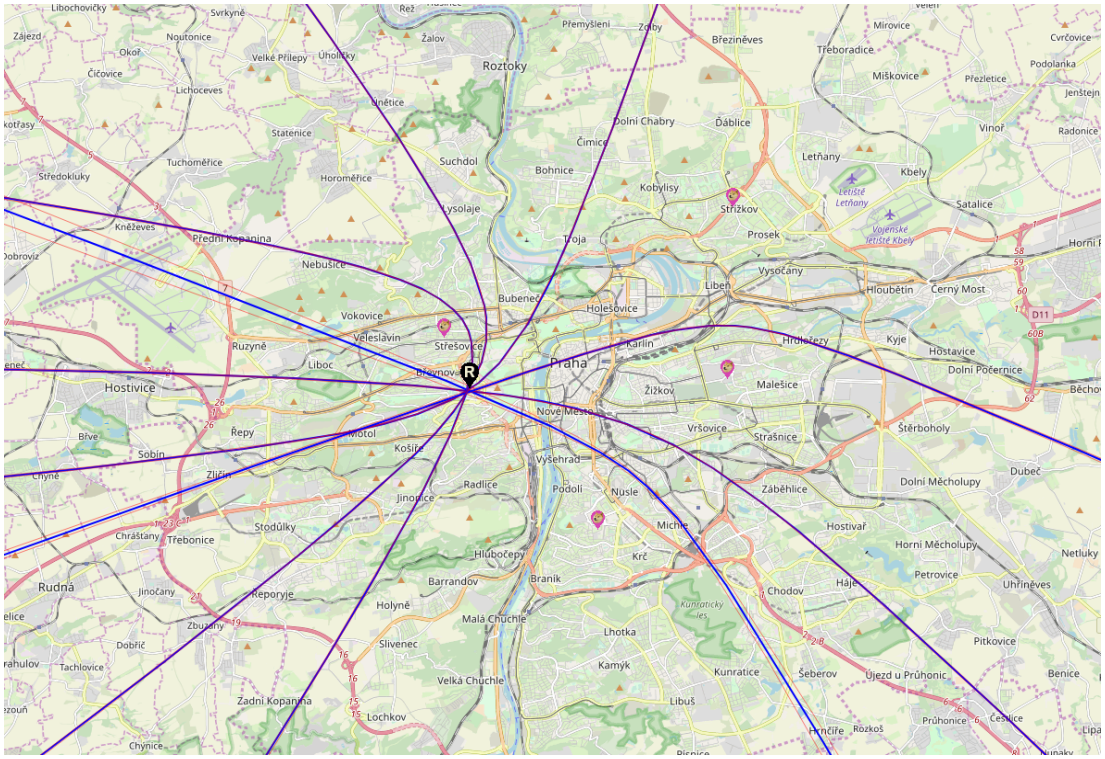


Figure C.1: Example of measuring a DAB transmitter at 1.5 GHz at Praha-Strahov (1 kW ERP).

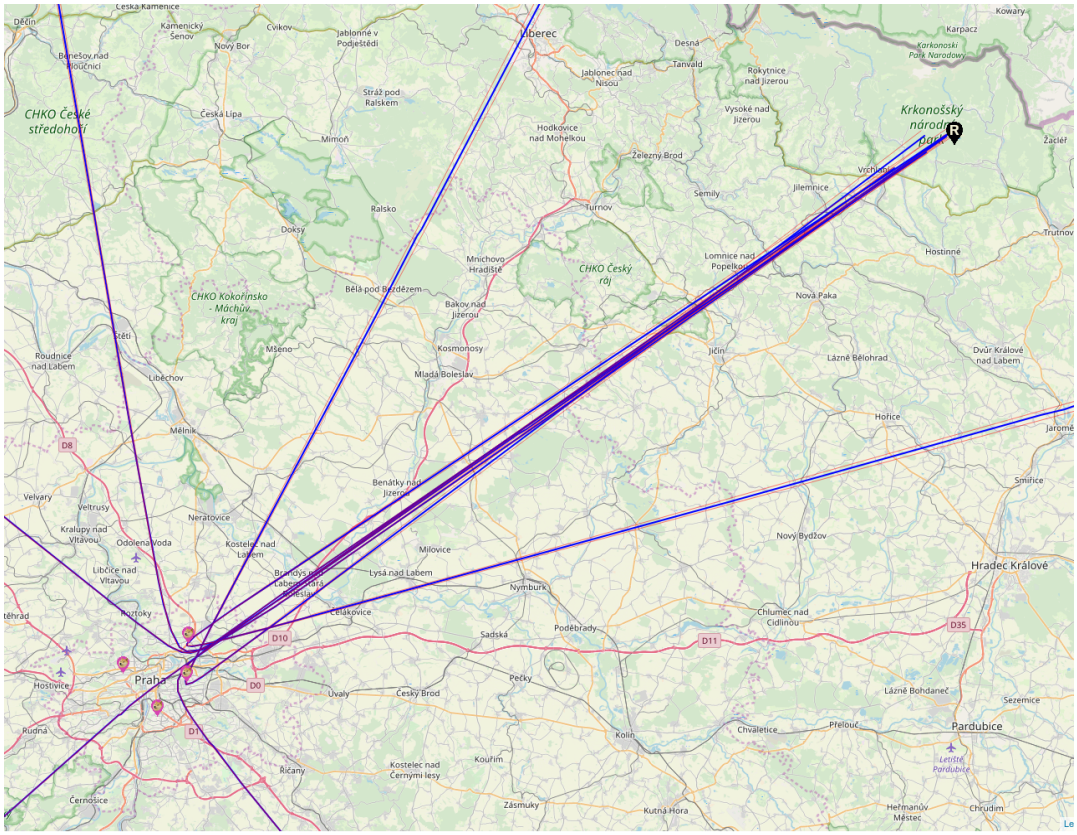


Figure C.2: Example of measuring a DVB-T transmitter in Krkonoše mountains from Prague (110 km, 100 kW ERP).

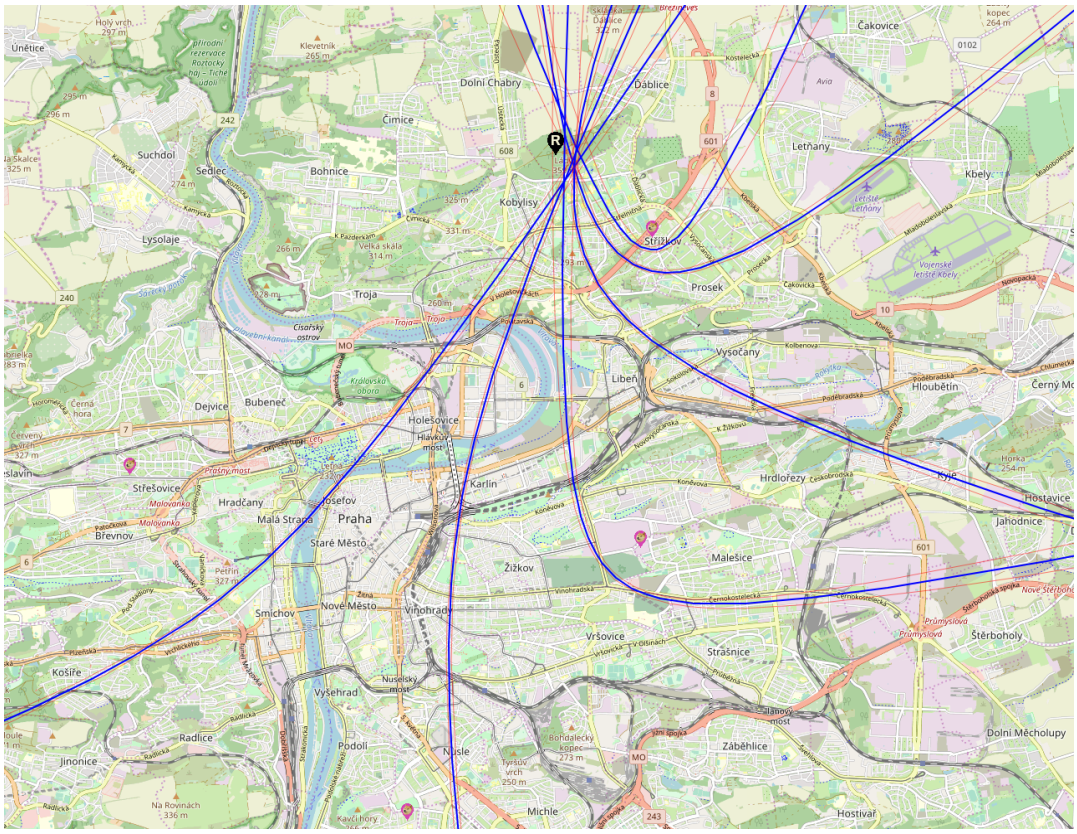


Figure C.3: Example of measuring a FM transmitter in Prague (1 kW ERP).

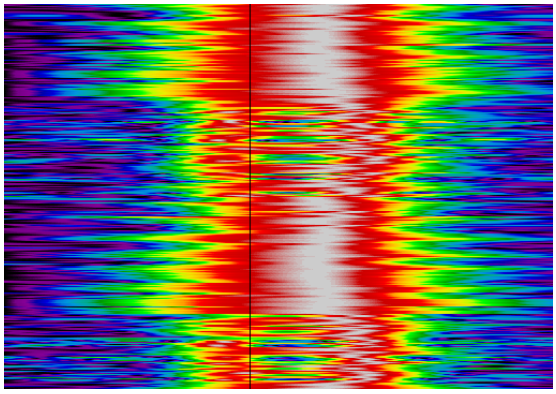


Figure C.4: Example of a crosscorrelation failing completely. The correct result is marked as a black line.

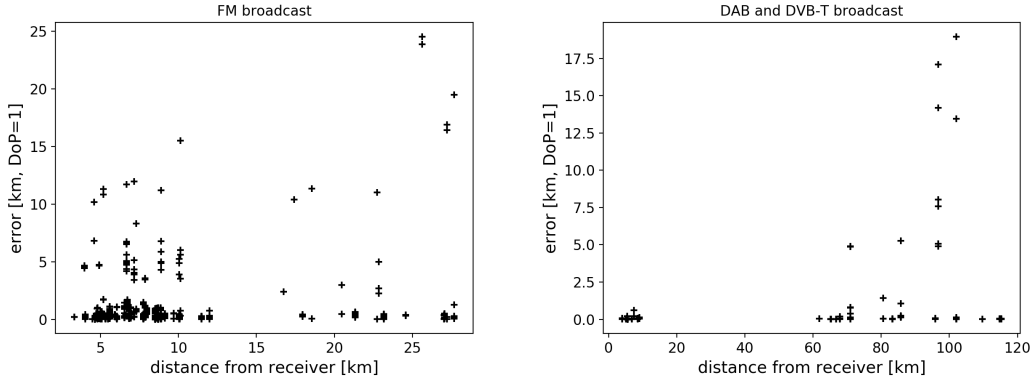


Figure C.5: Scatter plot of error and distance between the target and the receiver across all our measurements, recomputed as if DoP was 1.

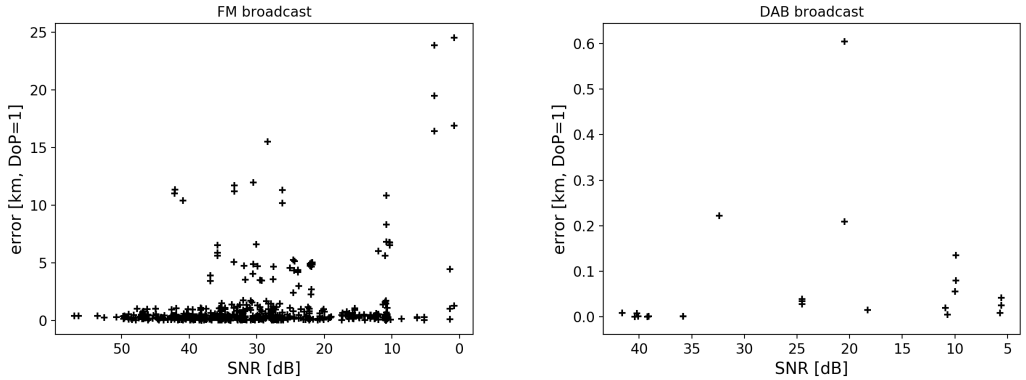


Figure C.6: Scatter plot of error and SNR (as the 90$^{\text{th}}$ percentile of target power − noise floor).

The measurement protocols generated by our software are available in electronic form at `https://popelka.ms.mff.cuni.cz/~hrach/tamarka/` and in the electronic attachment to this thesis. Futhermore, an example one is shown below.

TDOA measurements at 102900 kHz

Legend

Spectrum:

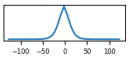
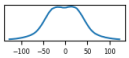
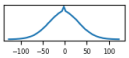
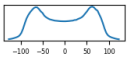
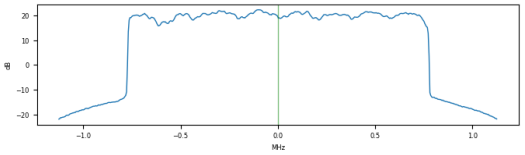
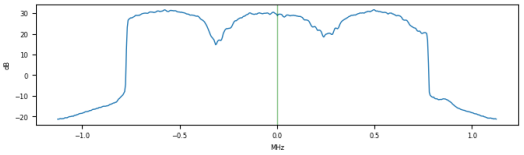
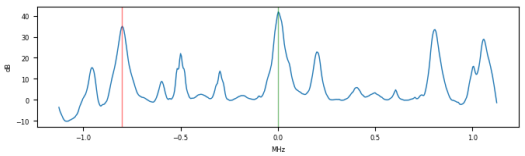
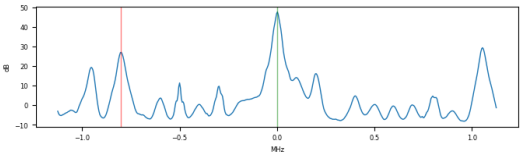
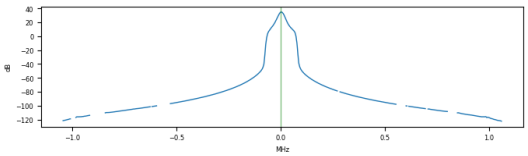
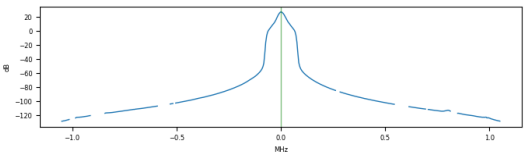
- Green: zero offset
- Red: target signal

Waterfall:

Correlation function:

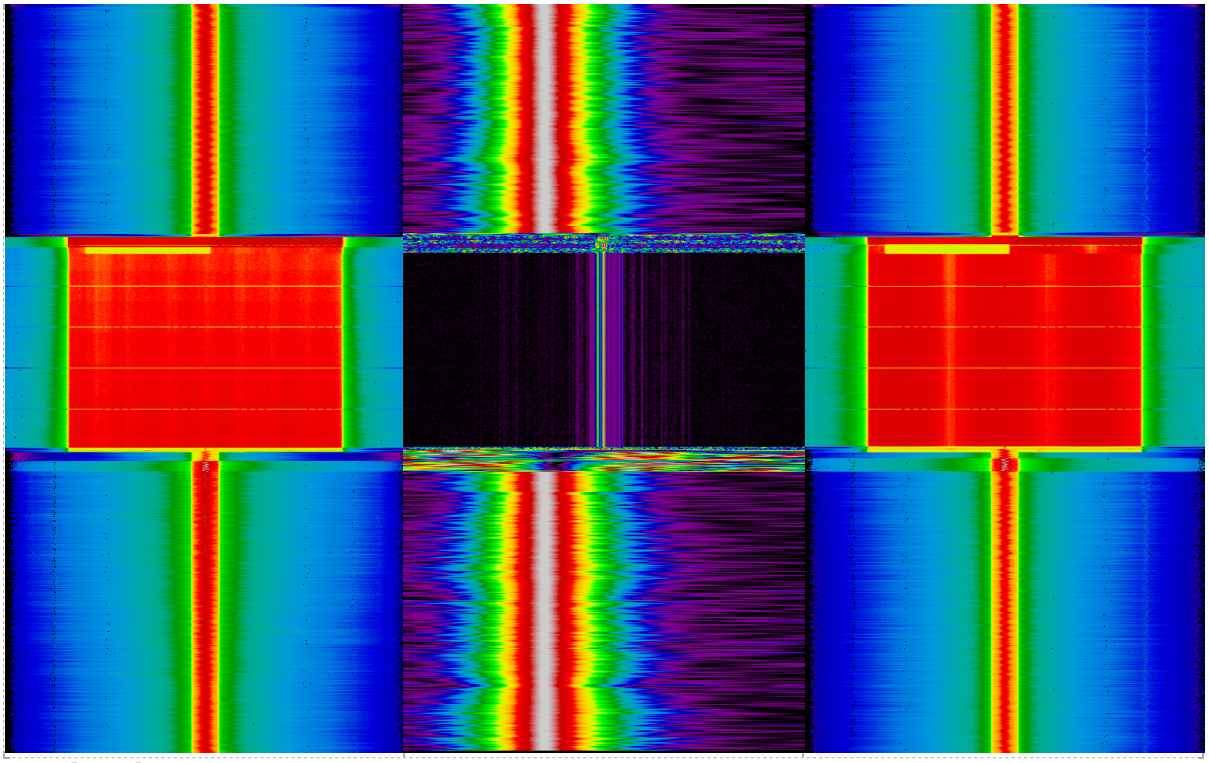
- Mouse over to show the fitted result
- White (on the very left): this correlation has been used (no retuning && sufficient fitness)
- Yellow on the left: fitness function (grows to the right)
- Cyan lines: lines fitted through the reference and target transmitters
- Yellow dots: correlation peaks that were used
- Red line: gold result (if the position of the target is known)

You will find the map at the bottom of the page. (jump there)

correlation 4286	**2019-04-20T23-10-33_103700000_pankrac**	**2019-04-20T23-10-33_103700000_brmlab**
gain	[9, 9, 9, 1]	[6, 6, 6, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	38.165 stddev = 0.005 reference, 0.489 target [samples]; known correct: 41.884	

Waterfall 1	**Correlation function**	**Waterfall 2**

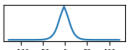
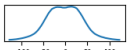
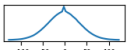
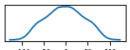
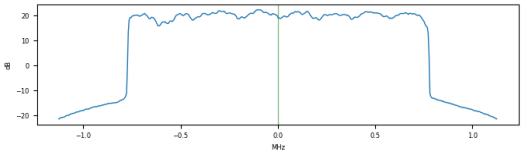
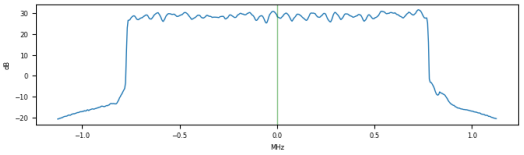
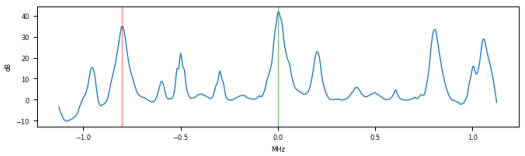
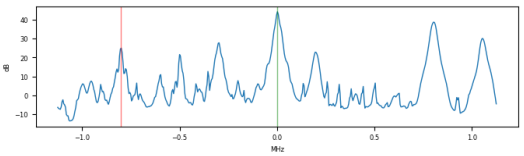
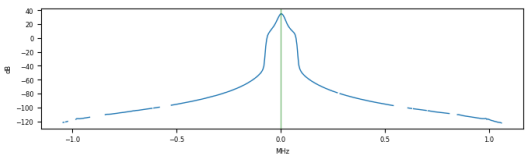
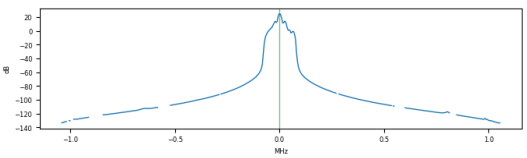
TDOA measurements at 102900 kHz



Correlator log

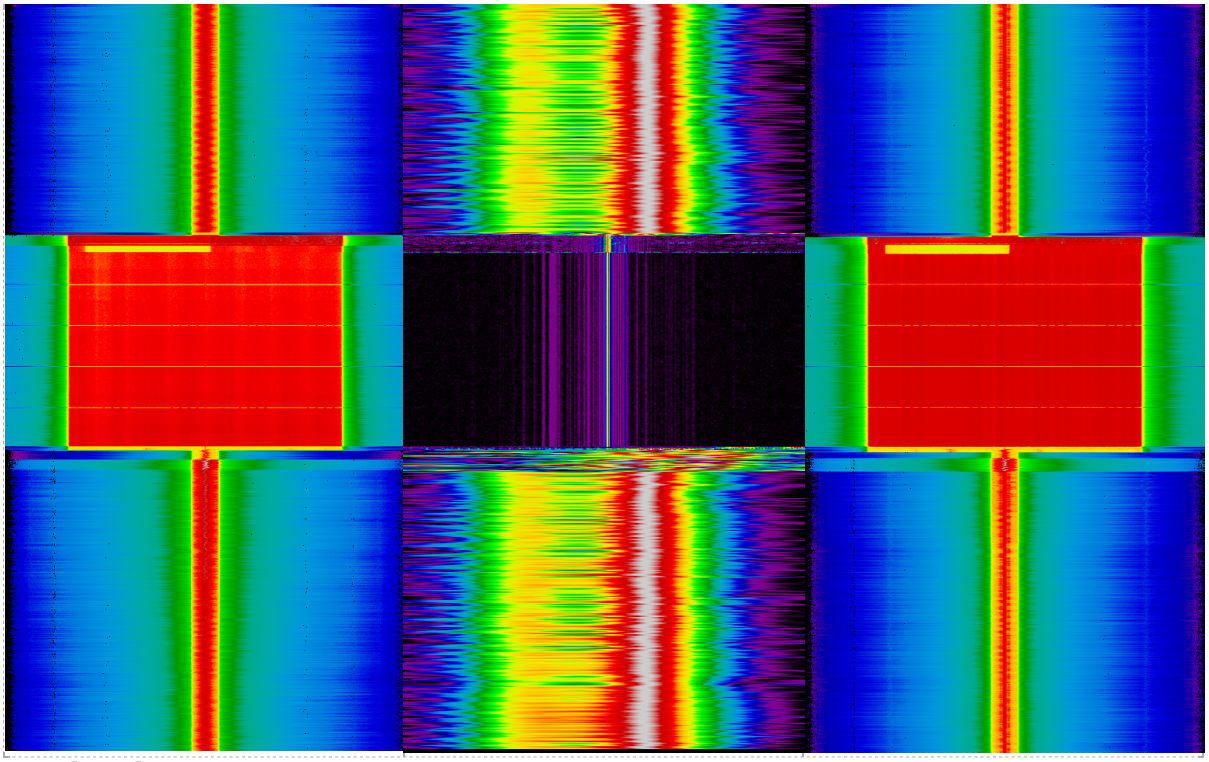
```
2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_brmlab    (pass 1)
reference error = -0.355672 ppm, target error = -0.028515 ppm
Measured dist: 37.821271    target fitness: 2.421252  stddevs: 1.697929 target, 0.096580 reference
gold result: 41.883851

2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_brmlab    (pass 2)
snr: 35.1 33.4 dB
reference error = -0.006011 ppm, target error = -0.034774 ppm
Measured dist: 38.164853    target fitness: 2.428682  stddevs: 0.488949 target, 0.004685 reference
gold result: 41.883851
```

correlation 4287	**2019-04-20T23-10-33_103700000_pankrac**	**2019-04-20T23-10-33_103700000_brevnov**
gain	[9, 9, 9, 1]	[7, 7, 7, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	-24.822 stddev = 0.004 reference, 1.701 target [samples]; known correct: -23.634	

Waterfall 1	**Correlation function**	**Waterfall 2**

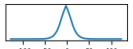
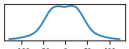
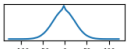
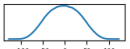
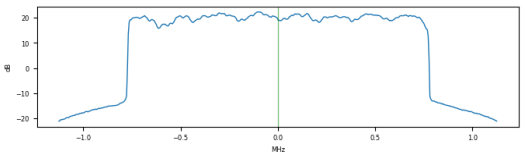
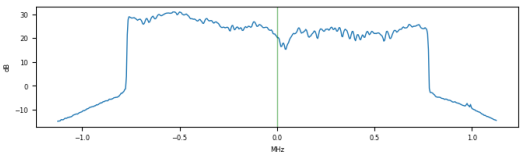
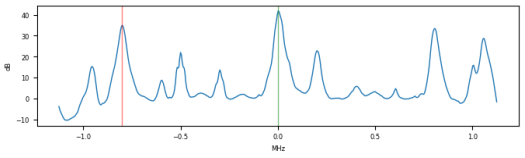
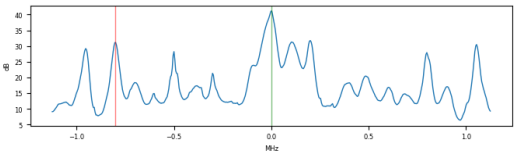
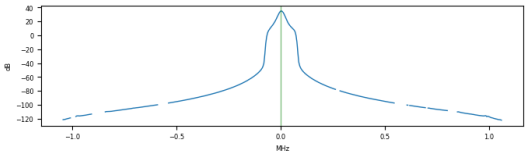
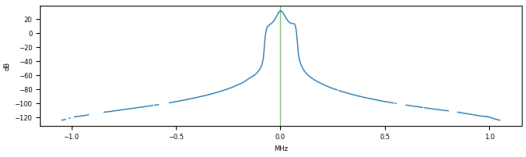
TDOA measurements at 102900 kHz



Correlator log

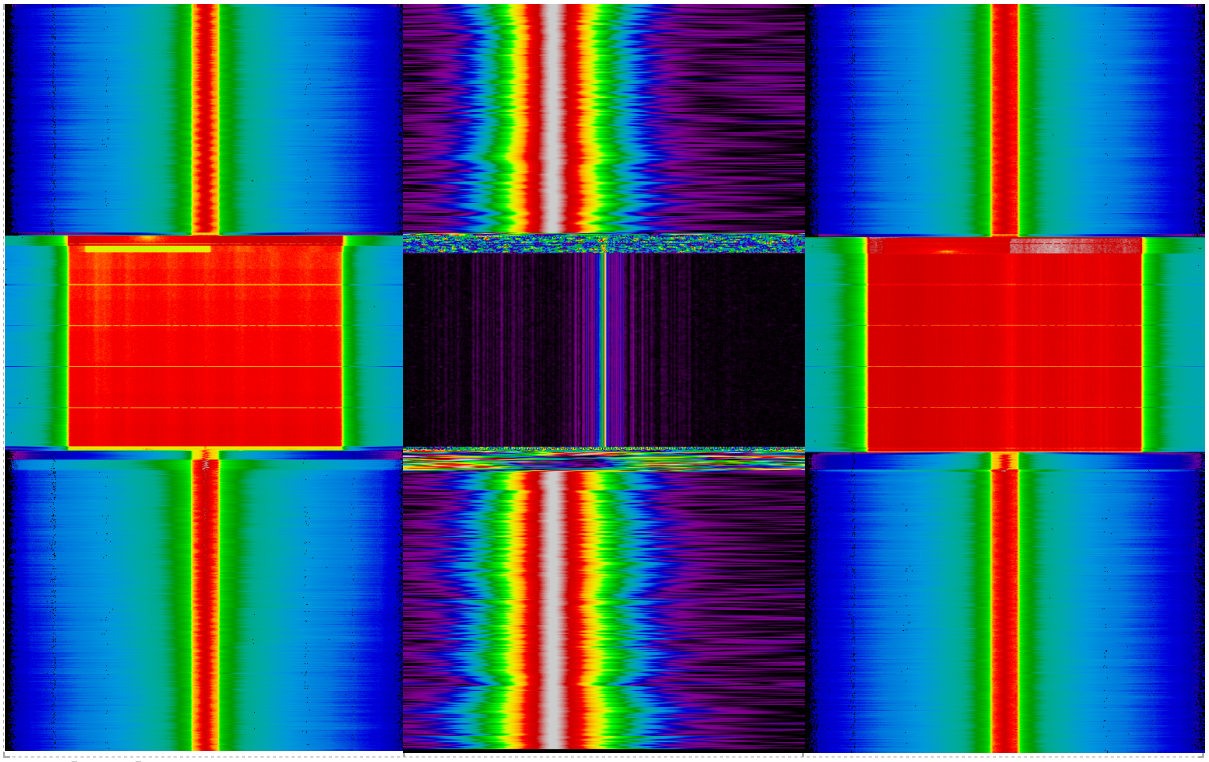
```
2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_brevnov    (pass 1)
reference error = -0.023190 ppm, target error = -0.101945 ppm
Measured dist: -24.880483    target fitness: 1.649411  stddevs: 1.710155 target, 0.007583 reference
gold result: -23.634450

2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_brevnov    (pass 2)
snr: 35.1 27.7 dB
reference error = 0.000075 ppm, target error = -0.122573 ppm
Measured dist: -24.821637    target fitness: 1.650235  stddevs: 1.701024 target, 0.003787 reference
gold result: -23.634450
```

correlation 4288	**2019-04-20T23-10-33_103700000_pankrac**	**2019-04-20T23-10-33_103700000_luna**
gain	[9, 9, 9, 1]	[12, 12, 12, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	33.116 stddev = 0.007 reference, 0.516 target [samples]; known correct: 34.954	

Waterfall 1	**Correlation function**	**Waterfall 2**

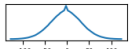
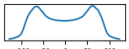
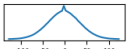
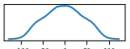
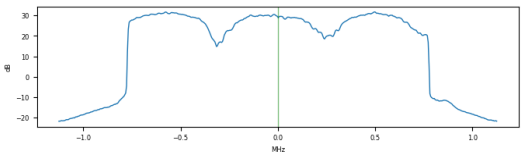
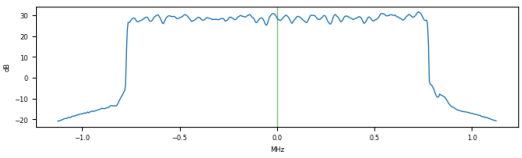
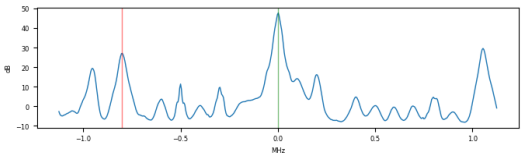
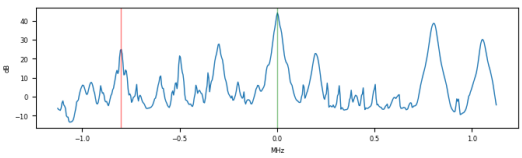
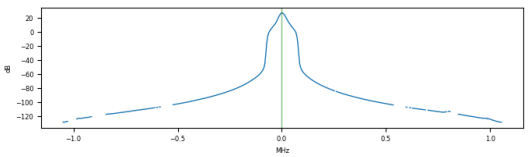
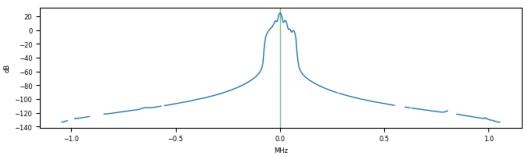
TDOA measurements at 102900 kHz



Correlator log

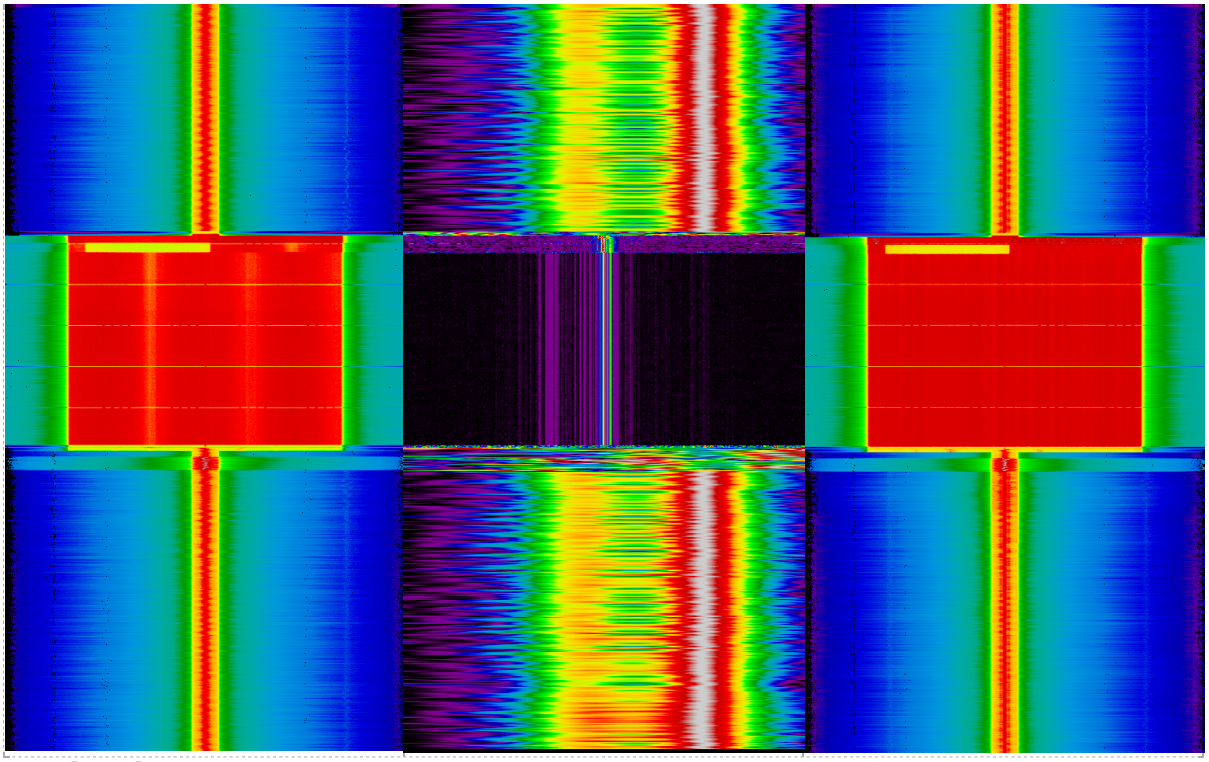
```
2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_luna    (pass 1)
reference error = 0.318073 ppm, target error = 0.071670 ppm
Measured dist: 33.467982    target fitness: 2.393485  stddevs: 1.390640 target, 0.086432 reference
gold result: 34.954401

2019-04-20T23-10-33_103700000_pankrac <--> 2019-04-20T23-10-33_103700000_luna    (pass 2)
snr: 35.1 22.2 dB
reference error = -0.015208 ppm, target error = 0.005130 ppm
Measured dist: 33.115504    target fitness: 2.397069  stddevs: 0.516244 target, 0.007361 reference
gold result: 34.954401
```

correlation 4289	**2019-04-20T23-10-33_103700000_brmlab**	**2019-04-20T23-10-33_103700000_brevnov**
gain	[6, 6, 6, 1]	[7, 7, 7, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	-64.254 stddev = 0.005 reference, 1.577 target [samples]; known correct: -65.518	

Waterfall 1	**Correlation function**	**Waterfall 2**

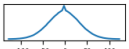
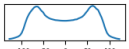
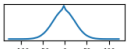
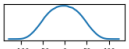
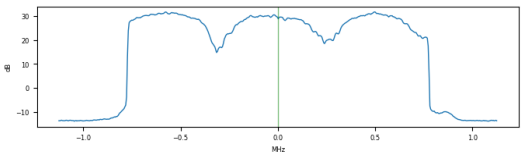
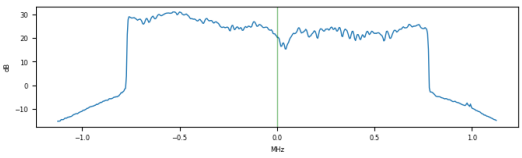
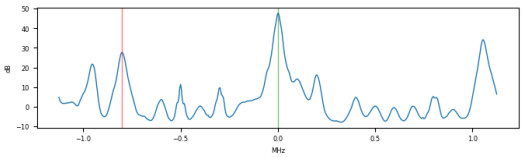
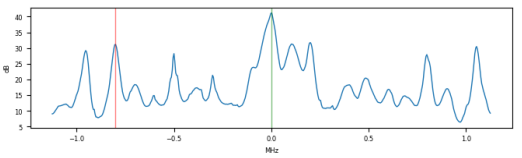
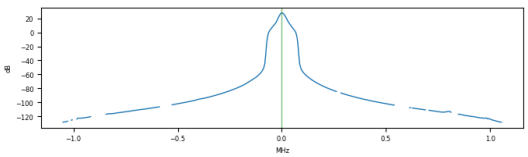
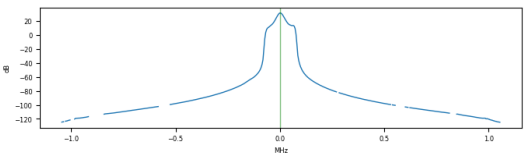
TDOA measurements at 102900 kHz



Correlator log

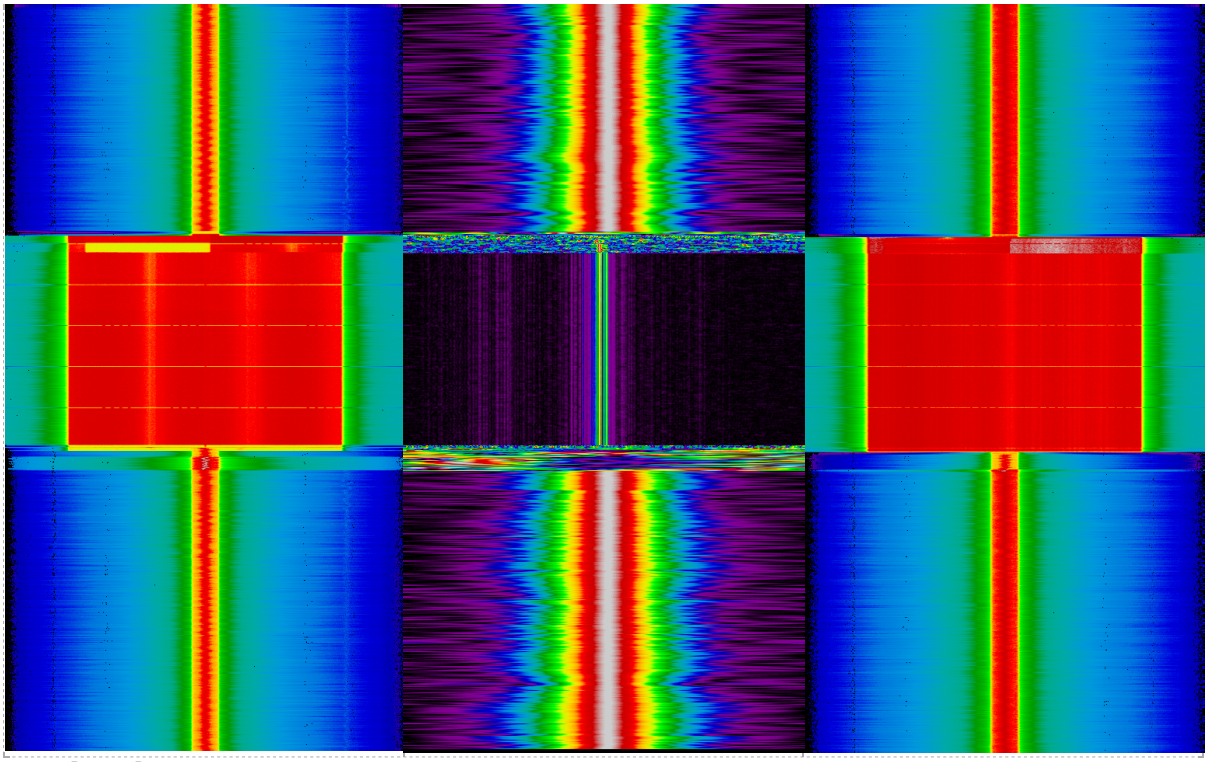
```
2019-04-20T23-10-33_103700000_brmlab <--> 2019-04-20T23-10-33_103700000_brevnov    (pass 1)
reference error = 0.346987 ppm, target error = 0.025114 ppm
Measured dist: -63.724084    target fitness: 1.695357  stddevs: 2.040406 target, 0.094202 reference
gold result: -65.518301

2019-04-20T23-10-33_103700000_brmlab <--> 2019-04-20T23-10-33_103700000_brevnov    (pass 2)
snr: 33.5 27.7 dB
reference error = -0.011536 ppm, target error = -0.036625 ppm
Measured dist: -64.254136    target fitness: 1.698083  stddevs: 1.577111 target, 0.005198 reference
gold result: -65.518301
```

correlation 4290	**2019-04-20T23-10-33_103700000_brmlab**	**2019-04-20T23-10-33_103700000_luna**
gain	[6, 6, 6, 1]	[12, 12, 12, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	-4.996 stddev = 0.011 reference, 0.469 target [samples]; known correct: -6.929	

Waterfall 1	**Correlation function**	**Waterfall 2**

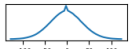
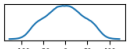
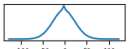
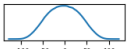
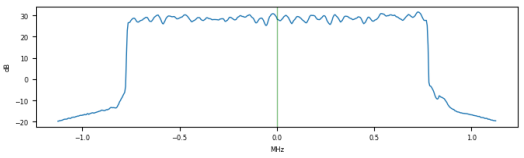
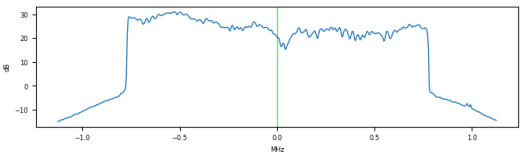
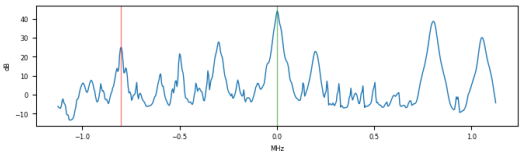
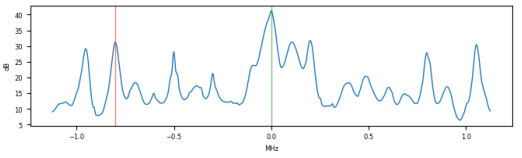
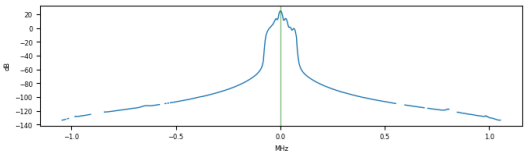
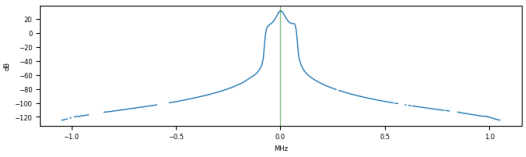
TDOA measurements at 102900 kHz



Correlator log

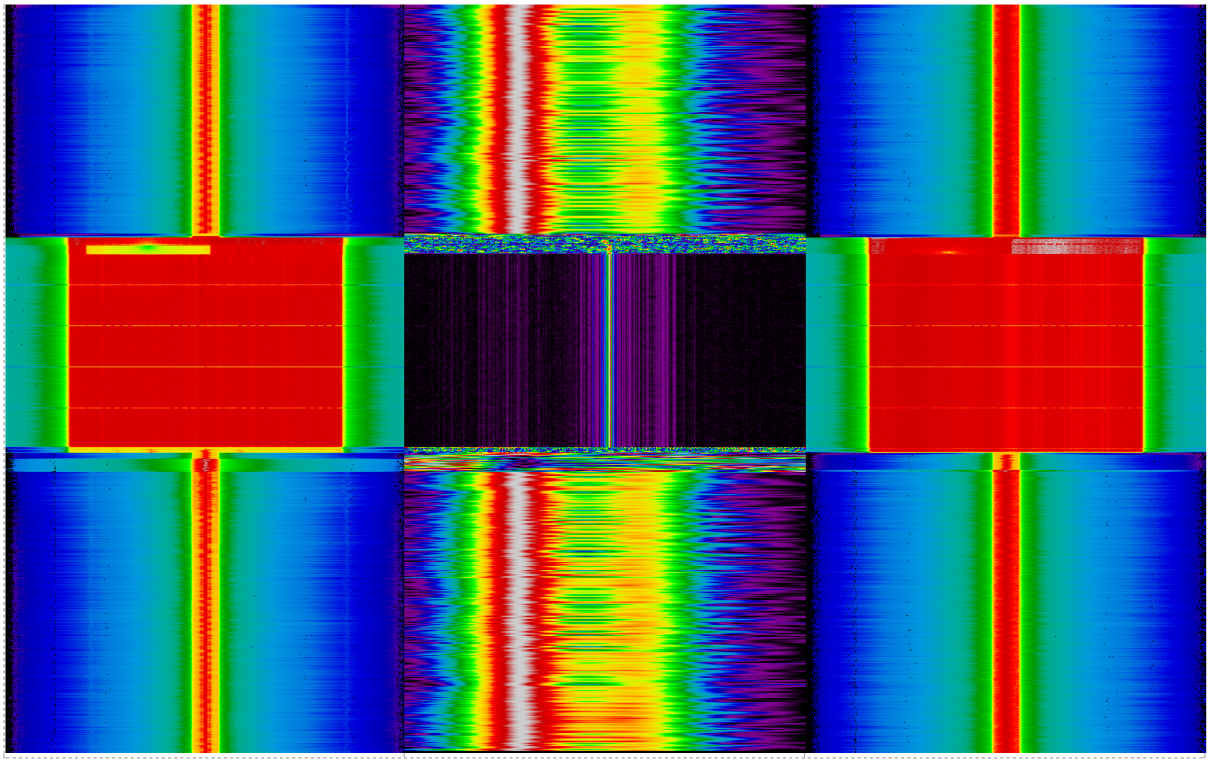
```
2019-04-20T23-10-33_103700000_brmlab <--> 2019-04-20T23-10-33_103700000_luna    (pass 1)
reference error = 0.707390 ppm, target error = 0.011993 ppm
Measured dist: -3.425755    target fitness: 2.486794  stddevs: 2.967850 target, 0.191185 reference
gold result: -6.929450

2019-04-20T23-10-33_103700000_brmlab <--> 2019-04-20T23-10-33_103700000_luna    (pass 2)
snr: 34.1 22.2 dB
reference error = -0.034429 ppm, target error = -0.063983 ppm
Measured dist: -4.996376    target fitness: 2.489684  stddevs: 0.468869 target, 0.010827 reference
gold result: -6.929450
```

correlation 4291	**2019-04-20T23-10-33_103700000_brevnov**	**2019-04-20T23-10-33_103700000_luna**
gain	[7, 7, 7, 1]	[12, 12, 12, 1]
samplerate	2250000	2250000 Hz
histogram	ref:  tgt: 	ref:  tgt: 
ref. spectrum		
tgt. spectrum		
tgt. filtered		
target_freq	103700000 Hz	
reference	DAB Final Lic: VYSÍLACÍ SÍŤ A, PRAHA MESTO, 227360 kHz	
result	58.700 stddev = 0.006 reference, 1.654 target [samples]; known correct: 58.589	

Waterfall 1	**Correlation function**	**Waterfall 2**

TDOA measurements at 102900 kHz



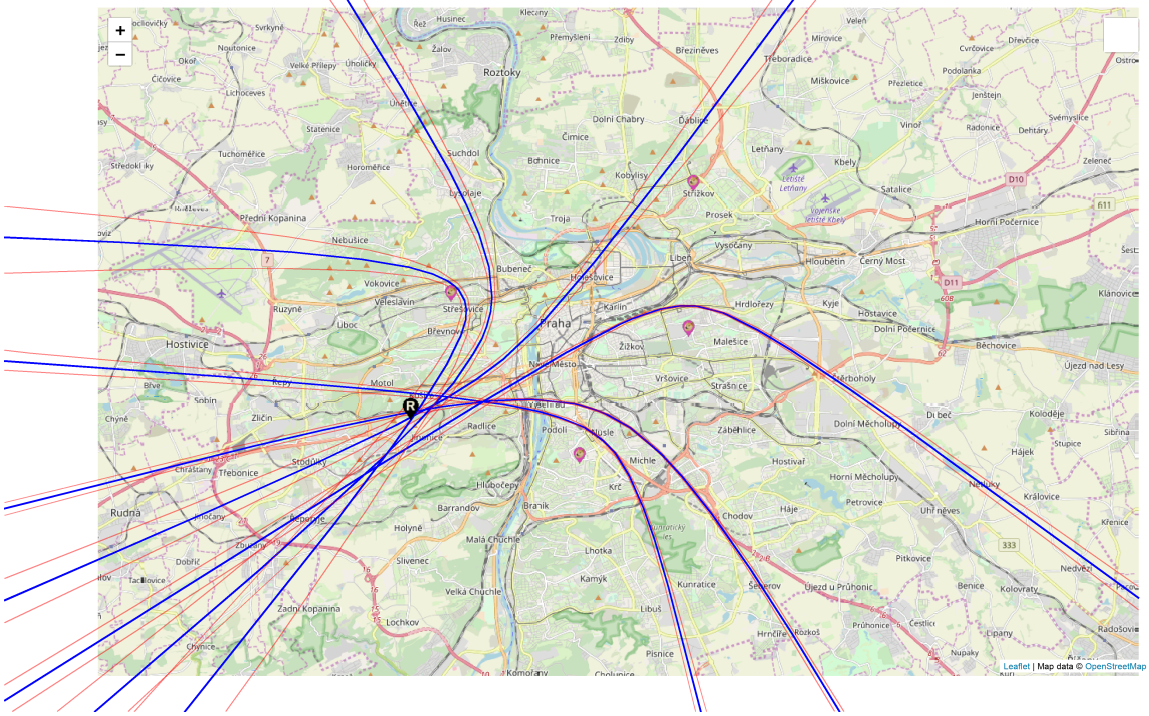
Correlator log

```
2019-04-20T23-10-33_103700000_brevnov <--> 2019-04-20T23-10-33_103700000_luna    (pass 1)
reference error = 0.336665 ppm, target error = 0.104572 ppm
Measured dist: 59.270926    target fitness: 1.663624  stddevs: 2.179734 target, 0.091118 reference
gold result: 58.588851

2019-04-20T23-10-33_103700000_brevnov <--> 2019-04-20T23-10-33_103700000_luna    (pass 2)
snr: 27.7 22.2 dB
reference error = -0.011399 ppm, target error = 0.150937 ppm
Measured dist: 58.699789    target fitness: 1.671994  stddevs: 1.653875 target, 0.006493 reference
gold result: 58.588851
```

Map

- - receivers
- - real position (if known)
- blue line - measurement hyperbola
- pale blue line - measurement hyperbola of measurement with high standard deviation
- red line - standard deviation error bar



D. AOA: user guide

List of dependencies required: `volk`, `python3`, `numpy`, `scipy` and `pygame`.

First, we need a patched rtl-sdr, the same that has been used in Appendix A.

```
$ git clone https://jenda.hrach.eu/p/rtl-sdr \&\& cd rtl-sdr
$ git checkout no_dither
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/rtl-sdr ..
$ make && sudo make install
```

Next, clone and compile the program:

```
$ git clone https://jenda.hrach.eu/p/rtl-aoa \&\& cd rtl-aoa
$ make RTLSDR_PATH=/opt/rtl-sdr
```

Edit the `common` section of the configuration file `sdr.ini` to your needs.

Run `./rtl-aoa.py /dev/ttyUSB0` (or other device node assigned on your computer). After the initial phase is measured, you should see the following window:

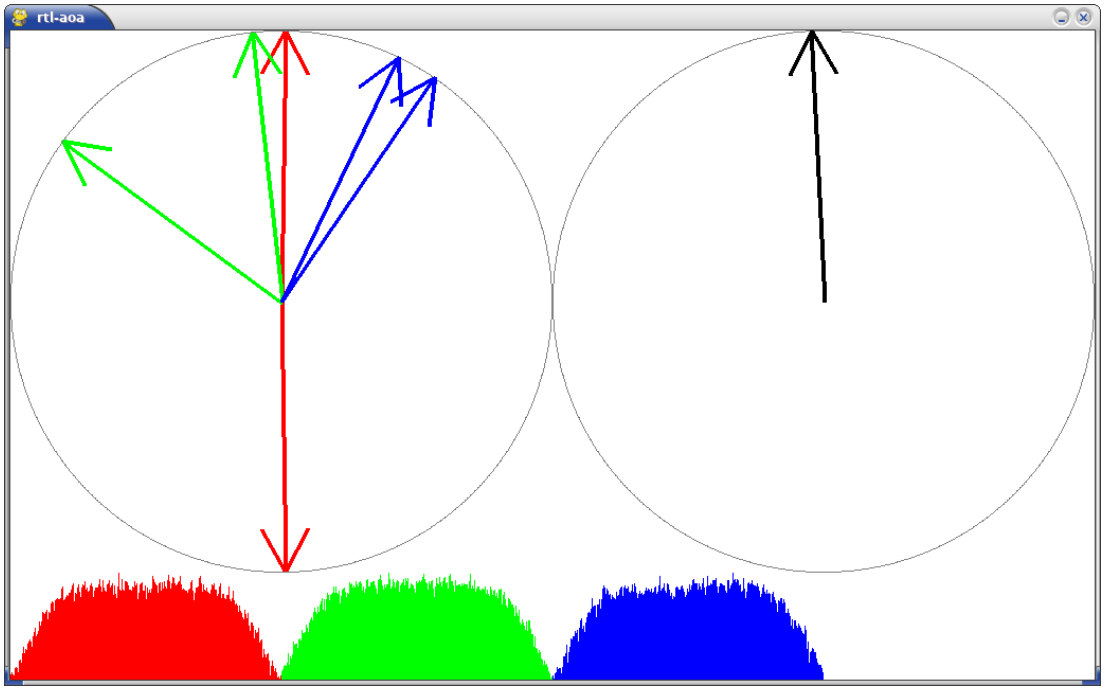


Figure D.1: rtl-aoa.py. The left dial shows angles from each antenna pair, the right dial shows the final computed direction, and the graphs on the bottom show spectrum of the signal received by all three radios.

Press `c` to enter the calibration mode. The measured phase offsets will be written to the standard output. Consecutively align each antenna pair perpendicularly with the signal source and note the measured phase. Then write these phases to `sdr.ini` and restart the program.