GNU/Linux Based TDOA Localization Using COTS Hardware

Georg Schiesser
OpenTech EDV-Research GmbH [1]
Lichtenstein Str 31, 2130 Mistelbach, AUSTRIA
e0307201@student.tuwien.ac.at

Abstract

The goal of this project is to determine the position of a sound source using an array of four microphones, a good quality audio card and a GNU/Linux based PC. The microphones are located at exactly known positions somewhere in the room. Time-Difference-Of-Arrival (TDOA) localization is based on the fact that the sound signal is received at the microphones with relative time-differences, depending on the distance between microphone and sound source and the speed of sound in air (c $\approx$ 343 m/s). At present either a frequency generator or a test-signal are used as the sound source. Later it will be possible to use other sound sources like clapping hands but recent measurements showed that this is difficult because of nonlinear recording effects like echo and distortion. Up to now the audio signals are recorded using ALSA. In general every tool can be used to record audio. One of the project's goals is to use the real-time audio toolkit RAT in a later version to guarantee minimal latencies in the recording interface. The calculation of the TDOA values between the four audio streams is based on an iterative cross-correlation algorithm optimized for low latencies. Every channel is compared with every other channel at different time lags ranging from -WINDOW to +WINDOW. The maximum value indicates the time difference at which two channels are correlated. Traditional algorithms are based on FFT or other transformations, but these techniques cannot be used if the TDOA values must be updated after every sample to keep the latencies at the minimum. In general TDOA localization can be thought of calculating the intersection of multiple hyperboloids. The equation system is too complex to be solved exactly. So the position of the sound source is approximated using Newton's method of multidimensional-root-finding. The algorithm is based on the GNU Scientific Library (GSL).

1 Introduction

The aim of this project is to implement a TDOA localization in GNU/Linux [2] [3] to determine the position of a sound source with multiple microphones and a good soundcard.

I am not sure but i think that one motivation for this project was to get rid of the flies in my room in summer. I thought that it would be nice if there was a servo-controlled 10MW laser-beam which could burn all the flying insects. I quickly realized that using image-recognition to localize the position of a $1cm^3$ object is not possible. So i decided to try TDOA localization of the moving wings.

There are a lot of papers about TDOA localization available, but most of the work is proprietary and not available to the public, especially in the field of Passive Surveillance [4] [5] [6] [7].

So the main motivation for this project is to implement the basic functionalities using mainstream hardware and a GNU/Linux system provinding results under GPL and FDL [8] [9].

TDOA localization can be used in various fields like robotics, automation, and multimedia.

But keep in mind that all localization stuff done is based on the speed of sound in air. If you want to determine the position of your favorite radio station, then you will need other hardware than a soundcard.

2 Recording Interface

Although recording is a very important part of the project, I will only describe the most important things because there is a lot of documentation available about how to get a sound signal into your PC.

Up to now recording has been done using only one multi-channel soundcard. Later there will be the possibility to record the audio signals with distributed nodes and timestamp synchronisation. Distributed data aquisition is much more difficult, especially if no external trigger is available. Recent investigations showed that there are hard-real-time linux drivers available, for example R.E.D.D. [10] [11] and RTnet [12]. They could be used to implement a global-time protocol to synchronize distributed data aquisition nodes.

2.1 Hardware

The basic hardware setup consists of a standard PC speaker, a microphone-array and a good quality sound card.

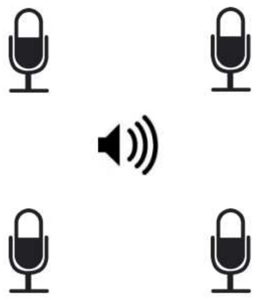

FIGURE 1: *Hardware Setup*

The speaker is used to play a test-signal to make the localization easier. Four microphones are placed at exactly known positions somewhere in the room. A Terratec Phase88 PCI sound card is used to record the signals.

Keep in mind that some kinds of microphones need phantom power and must be pre-amplified before you can connect them to the soundcard.

2.2 Software

Up to now recording has been done using the well-known ALSA tools [13]. After configuring the soundcard using `alsamixer` you can start recording.

```
# arecord --file-type=raw   \
          --channels=4      \
          --format=S32_LE   \
          --rate=96000      \
          --duration=1 > data.raw
```

Note that `envy24control` [14] can be used to configure the non-standard functionalities of Terratec soundcards based on the **envy** chipset. Furthermore it can be very usefull to install the linux sound exchange tool `sox` [15] if you want to apply filters or change the data format.

At present all data is written into a file before starting signal processing because it is easier to verify the measurements and the TDOA localization has a very high cpu-usage.

2.3 Test-Signal

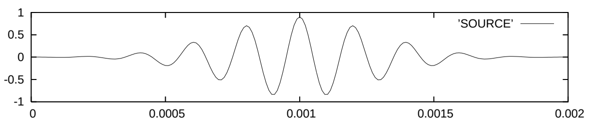

FIGURE 2: *Test-Signal*

During development a test-signal was used to test the cross-correlation algorithm. The test-signal should be a short pulse with a constant period which must be high enough to produce unique positions within a specified area. Rectangular pulses are not appropiate because they consist of too much frequencies, especially high frequencies are a big problem. Pure sine waves don't work because they would yield multiple possible positions during cross-correlation. The test-signal was generated using a sine wave multiplied with a gaussian curve as shown in figure 2.

3 Cross-Correlation

This section is about how to determine the time-differences-of-arrival (TDOAs) by calculating the cross-correlation of the four audio streams.

3.1 Basics

In signal processing, the cross-correlation is a measure of similarity of two signals, commonly used to find features in an unknown signal by comparing it to a known one. It is a function of the relative time between the signals. For discrete functions f_i and g_i the cross-correlation is defined as

$$(f \star g)_i \equiv \sum_j f_j \, g_{i+j} \tag{1}$$

where the sum is over the appropriate values of the integer j. [16]

3.2 Optimization

We want to keep the latencies at a minimum, so we must update the cross-correlation after every sample. Furthermore we assume that the TDOAs must be in the range of $-WINDOW$ to $+WINDOW$ samples. Using the standard algorithm we would have to calculate about

$$SAMPLERATE * (2 * WINDOW)^2 \tag{2}$$

multiplications per second. The most important part of the implementation is that we only update the cross-correlation instead of calculating it.

```
c[N] += MULTIPLY(a_new, b_new);
c[N] -= MULTIPLY(a_old, b_old);

for (i = 1; i < N; i++)
{
  c[N + i] += MULTIPLY(b_new, a[i]);
  c[N + i] -= MULTIPLY(b_old, a[N + i]);

  c[N - i] += MULTIPLY(a_new, b[i]);
  c[N - i] -= MULTIPLY(a_old, b[N + i]);
}
```

Using the modified algorithm we can reduce the number of multiplications per second to

$$SAMPLERATE * 2 * WINDOW \tag{3}$$

The maximum's index of the cross-correlation now specifies the TDOA at which the two signals match. Furthermore we can define the fitness of the solution by the maximum itself.

3.3 Redundancy

The cross-correlation must be calculated between every pair of audio streams. Using four microphones we have to calculate a total number of 6 time-differences between AB, AC, AD, BC, BD and CD. In general we are only interrested in the TDOAs AB, AC and AD, but recent measurements showed that calculating the TDOAs is the most critical part and should be done in a redundant way to minimize the error.

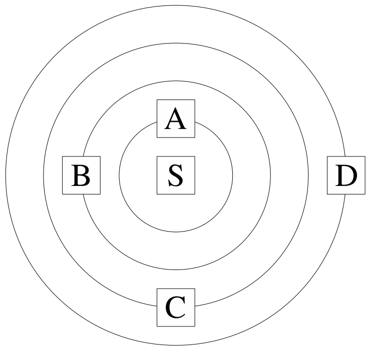

FIGURE 3: *Redundancy*

$$ABC := AB + BC - AC = 0 \tag{4}$$
$$ABD := AB + BD - AD = 0 \tag{5}$$
$$ACD := AC + CD - AD = 0 \tag{6}$$

As this application shows unfavorable error propagation some form of error detection is needed, this equation system can be used to determine the errornous part of the calculated TDOAs or in case of "out of bounds errors" drop the result entirely.

3.4 Runtime Optimization

The calculation of the time-differences between each channel is very cpu-intensive. After each sample the cross-correlation must be updated to determine the time-difference of arrival. There are several ways to optimize for runtime.

- Samplerate: Reducing the samplerate, for example from $96kHz$ to $48kHz$, would of course speed up the algorithm dramatically. But keep in mind that only using half of the time-resolution would also reduce the accuracy of the cross-correlation.

- Samplesize: The soundcard is able to record at $96kHz$ and $24bit$ sample resolution. During the cross-correlation the input samples are multiplied with other input samples. This is done $WINDOW$ times. Assuming that we have a $WINDOW$ of 512 samples, we would at least need $58bit$. So either the samplesize must be reduced to do $32bit$ calculations or you have do it the $64bit$ way. During the development of this project both the $32bit$ and the $64bit$ implementation have been tested and the measurements showed that only using $11bit$ input samplesize to do $32bit$ calculations is enough. There are only minor differences in the results of the two versions.

- Windowsize: Most of the measurements have been done using a $WINDOW$ of 512 samples. The $WINDOW$ depends on the samplerate, the microphone locations and the speed of sound.

One of the most recent feature is the input trigger. It is used to avoid calculating the cross-correlations if all input channels are silent. During this period no calculation is done. The cross-correlation algorithm only starts when the input volume of one one channel increases above the trigger-level.

So the cpu-usage can be reduced most of the time when all input channels are low. Furthermore this optimization does not have any impact on the accuracy of the cross-correlation.

3.5 Example

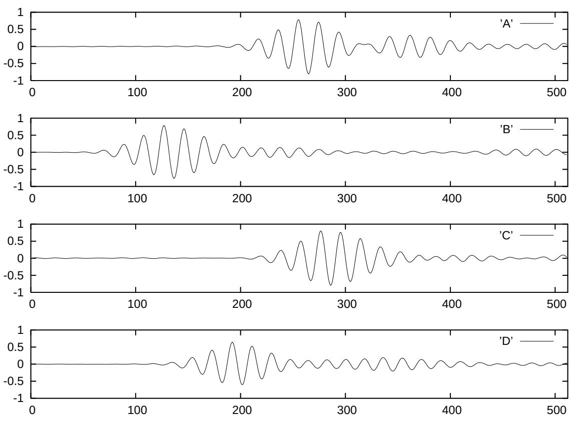

FIGURE 4: *TDOA in Samples*

This measurement was done using the test-signal and four microphones. In figure 4 you can see the channes A, B, C, and D recorded by the soundcard. The signal from the first channel A is disturbed and contains echo from a wall, which should be avoided because the echo signal can cause wrong cross-correlation results.

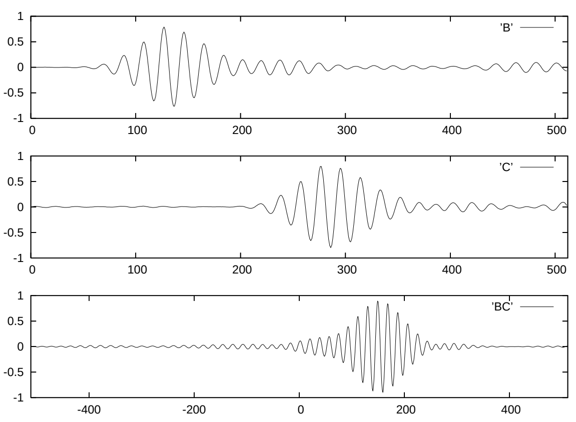

FIGURE 5: *Channel B and C and the Cross-Correlation BC in Samples*

In this example the cross-correlation is calculated between channel B and C. The time-difference-of-arrival between the channels is about 150 samples. Figure 5 shows the two channels and the cross-correlation with a maximum value at the index 150 which indicates the TDOA between the channels.

4 Position approximation

4.1 Multilateration

Multilateration, also known as hyperbolic positioning, is the process of locating an object by accurately computing the time difference of arrival (TDOA) of a signal emitted from the object to three or more receivers. It also refers to the case of locating a receiver by measuring the TDOA of a signal transmitted from three or more synchronised transmitters. [17]

4.2 Principle

If a pulse is emitted from a platform, it will arrive at slightly different times at two spatially separated receiver sites, the TDOA being due to the different distances of each receiver from the platform. In fact, for given locations of the two receivers, a whole series of emitter locations would give the same measurement of TDOA. Given two receiver locations and a known TDOA, the locus of possible emitter locations is a hyperboloid (a surface approximately shaped like two cones joined at the origin). In simple terms, with two receivers at known locations, an emitter can be located onto a hyperboloid. Note that the receivers do not need to know the absolute time at which the pulse was transmitted - only the time difference is needed.

Consider now a third receiver at a third location. This would provide a second TDOA measurement and hence locate the emitter on a second hyperboloid. The intersection of these two hyperboloids describes a curve on which the emitter lies.

If a fourth receiver is now introduced, a third TDOA measurement is available and the intersection of the resulting third hyperboloid with the curve already found with the other three receivers defines a unique point in space. The emitter's location is therefore fully determined in 3D. [17]

4.3 Problem Statement

For two 3D points, $P = (p_x, p_y, p_z)$ and $Q = (q_x, q_y, q_z)$, the distance $\|P - Q\|$ is computed as

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \qquad (7)$$

The relationship between the microphone positions and the position of the sound source is defined as

$$\|P - P_i\| - \|P - P_0\| - c * TDOA_i = 0 \qquad (8)$$

where P is the unknown position of sound source and P_i is the position of the i-th microphone. $TDOA_i$ denotes the time difference of arrival between the microphones P_i and P_0.

Using four microphones the position of the sound source P can be computed as

$$\|P - P_1\| - \|P - P_0\| - c * TDOA_1 = 0 \tag{9}$$
$$\|P - P_2\| - \|P - P_0\| - c * TDOA_2 = 0 \tag{10}$$
$$\|P - P_3\| - \|P - P_0\| - c * TDOA_3 = 0 \tag{11}$$

The equation system is too complex to be solved exactly. Instead, an approximation algorithm is used to determine the position of the sound source. In this project I decided to use the GNU Scientific Library [18] to implement the position approximation.

4.4 Root-Finding

The most common example of root-finding is the newton iteration defined as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{12}$$

Multidimensional Root-Finding is similar to onedimensional root-finding [19]. It requires the simultaneous solution of n equations, f_i, in n variables, x_i, $f_i(x_1, ..., x_n) = 0$ for $i = 1...n$.

In general there is no way of knowing whether any solutions exist. All algorithms proceed from an initial guess using a variant of the Newton iteration,

$$J\Delta x = -f(x) \tag{13}$$

where x, f are vector quantities and J is the Jacobian matrix $J_{ij} = \partial f_i / \partial x_j$. [18]

4.5 Implementation

At first an initial starting position is choosen. The initial guess is based on TDOA values. Then one newton iteration is executed to calculate a new approximation of the source position. The distance between the current and the previous position yields the accuracy of the current approximation. This is done until the accuracy is below a given tolerance or the maximum number of iteration has exceeded.

5 Conclusion

Although this first prototype shows that TDOA localization with standard Linux is in priciple possible, it also showed a number of limitations both in the algorithmic as well as at the system level. The main issues emerging are:

- precision of the sampling system
- error propagation and resulting numerical effects
- computational resource demands

Due to these problems we decided to continue this project based on a distributed real-time Linux system, outlined below.

Further it can be concluded that even if the final system has hard real-time demands and extensive numerical work to do, mandating distributed resources, prototyping of real-time systems with COTS based systems using mainstream Linux is reasonably possible even in systems with very sensitive signals. Both development and initial verification of the algorithms were significantly simplified by building on mainstream Linux due to debugging capabilities and lack of the RT-Linux inherent constraints (i.e. library access).

Aside from the technical problems a design limitation emerged, in a TDOA system an automated initial calibration method is almost mandatory, measuring the distances of emitter and receiver is not only a painstacking procedure but it is very hard to precisely measure the distances as the point of emission is generally not easily accessible (i.e. what point to you take on a speaker? what point to you take on the microphone?).

6 Future Work

Usage of standard audio devices introduces some uncertainty with respect to the processing of audio interrupts. In mainstream Linux interrupts can be off for a significant time limiting the accuracy of positional calculation. For this reason future works are planed to utilize the hard real-time variants of Linux - in our case RTLinux/GPL [20]. To make the problem more interesting and to improve the scalability (currently a high end number cruncher is a requirement - which does not always fit the demands for hard real-time too well)

6.1 RTLinux recording interface

RTLinux/GPL is a POSIX PSE51 driven RTOS on top of Linux, it is not actually a typcial audio platform, but provides a basic deterministic execution context on which a TDOA system can be implemented with increased precision of sampling. Unfortunately there are some limitations of this kernel-space real-time Linux implementation, notably the inability of directly accessing user-space libraries like the GNU Scientific library, thus a change of design is needed providing a data interface from kernel space, in which the RT-threads execute, to user-space in which the non time-critical calculations are executed. RTLinux/GPL provides fifos and shared memory as basic means of interfacing to user-space, on top of this a sound driver would need to be implemented to access the sound cards as the standard Linux drivers are not RT-safe. Instead we decided to utilize a previous project, the Real-Time Audio Toolkit RAT.

6.2 Real-Time Audio Toolkit RAT

The Real-Time Audio Tools [21] implemented by Arvid Staub provides a low level driver for high-end audio cards (TERRATEC Phase88) and an elaborate interface for channel management and data routing. The basic concept is to abstract each channel of a multichannel card to a logical channel and then describe the data route that the respective data items are to take.

Though originally designed for audio work and not as general data sampling interface RAT has the advantage that it was designed to deliver data to user-space from the very beginnning, basically for hard-disk recording and playback - thus this fits well with the modfified design that executes the numerically intensive parts of the TDOA problem in user-space while guaranteeing deterministic sampling.

6.3 Global Time Synchronisation

To implement a distributed system there are two main demands:

- Global time
- Real-time communication

The two are to a certain extent coupled. For the real-time communication the Real Time Ethernet Drivers [11] in the latest variant [22] are to be used. The decision of how to implement global time, TDM based or via time-diffusion, has not been found yet as detailed evaluation is still pending. So the next steps for the global time will require some prototyping and benchmarking to find a technology that can work efficiently with COTS hardware components under RTLinux/GPL.

6.4 Distributed Data Aquisition

Obviously a distributed system for TDOA will require a system level knowledge of the data, further more if the responsiveness of the system should be sufficiently (i.e. to trace moving objects) the distribution of data will be a crucial point, as noted above the Real-Time Ethernet Device Drivers are to be used for the communication over dedicated ethernet links (point-to-point for now). This will introduce a substantial latency in the overall data aquisition compared to single node data aquisition thus compensation based on precise global time is mandatory.

6.5 Generalized Cross-Correlation

In signal processing it is common to use the generalized cross-correlation to determine the time-difference between noisy signals. This could be another way to optimize the accuracy of the cross-correlation.

6.6 TDOA verification using Neural Networks

Up to now simple condition statements have been used to throw away bad TDOA values calculated by the cross-correlation. Instead, the calculated TDOA values could be verified using neural networks.

6.7 Multidimensional Minimization

During development we had some problems concerning the convergence of the approximation algorithm. This problems could be due to the fact that the error propagation in the overall system is very unfavorable, especially if one assumes general signals and not the test-signals only, it showed that increasing the bit-width of the samples would more or less have no effect due to the integrative nature of the cross-correlation, though an increase of sampling frequency seems promising furthermore an uncertainty with respect to the effective jitter of the signal needs clarification. To clarify or rather confirm that the bit width of the samples is not relevant we ran measurements with reduced bit width (down to 11bit samples) which showed no significant change in the outcome. The second possibility, to increase the sampling frequency, seems limited with COTS components (though a factor of two or four seems realistic, but hardly much more). To improve the jitter of the sampling system a switch from a GPOS (mainstream Linux) to a RTOS (RTLinux/GPL) is proposed.

All these changes together never the less mandate searching for better methodologies than the newton approximation for the convergence estimation. The newton approximation has the severe limitation that a badly selected starting point will in the worst case not converge at all, but in any case the execution time of this method is not known. Possible alternatives under concideration include Multidimensional Minimization based on the GNU Scientific Library.

References

[1] OpenTech, *OpenTech - RTLinux Support in Europe*, 2006-09-14, `http://www.opentech.at/`

[2] GNU, *GNU's Not Unix! - Free Software, Free Society*, 2006-09-15, `http://www.gnu.org/`

[3] Linux, *The Linux Kernel Archives*, 2006-09-15, `http://www.kernel.org/`

[4] Briukhov, Kalinichenko, Zakharov, Panchuk, Vitkovsky, Zhelenkova, Dluzhnevskaya, Malkov, Kovaleva, 2005, *Information Infrastructure of the Russian Virtual Observatiry (RVO)*, Russian Academy of Sciences.

[5] Roke, *Vigilance - wide area multilateration system*, 2006-09-15, `http://www.roke.co.uk/vigilance/`

[6] Baniak, Baker, Cunningham, Martin, 1999, *Silent Sentry - Passive Surveillance*, Lockheed Martin Mission Systems.

[7] ERA Radar Technology, *Multilateration Surveillance Radar P3D*, 2006-09-15, `http://www.era.cz/en/msr-3d.shtml`

[8] GNU, *What is Copyleft?*, 2006-09-15, `http://www.gnu.org/copyleft/`

[9] FSF, *Free Software Licensing*, 2006-09-15, `http://www.fsf.org/licensing`

[10] Sourceforge, *REDD: RTLinux Ethernet Device Drivers*, 2006-09-14, `http://redd.sourceforge.net/`

[11] Sergio Perez, Joan Vila and Ismael Ripoll, 2003, *Building Ethernet Drivers on RTLinux-GPL*, Proceedings of the 5th Real-Time Linux Workshop, pp279–286.

[12] Jan Kiszka, Bernardo Wagner, Yuchen Zhang, Jan Broenink, 2005, *RTnet - A Flexible Hard Real-Time Networking Framework*, 10th IEEE International Conference on Emerging Technologies and Factory Automation.

[13] ALSA, *Advanced Linux Sound Architecture*, 2006-09-15, `http://www.alsa-project.org/`

[14] ALSA OpenSRC, *Envy24Control*, 2006-09-15, `http://alsa.opensrc.org/?page=Envy24Control`

[15] Sourceforge, *SoX - Sound eXchange*, 2006-09-15, `http://sox.sourceforge.net/`

[16] Wikipedia, *Cross-correlation*, 2006-09-01, `http://en.wikipedia.org/wiki/Cross-correlation`

[17] Wikipedia, *Multilateration*, 2006-09-17, `http://en.wikipedia.org/wiki/Multilateration`

[18] Gnu, *GSL - GNU Scientific Library*, 2006-09-15, `http://www.gnu.org/software/gsl/`

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, 1992, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, ISBN 0-521-43108-5.

[20] RTLinux/GPL , *Real-Time Linux*, 2006-09-27, `http://www.rtlinux-gpl.org/`

[21] Arvid Staub, 2004, *RAT - Real-Time Audio Tools*, Proceedings of the 6th Real-Time Linux Workshop, pp57–63.

[22] Florian Bruckner, 2006, *Realtime Ethernet Device Drivers*, Proceedings of the 8th Real-Time Linux Workshop, tba.