Providing Common Time and Space in Distributed AV-Sensor Networks by Self-Calibration

R. Lienhart[1], I. Kozintsev[1], D. Budnikov[2], I. Chikalov[2], and V. C. Raykar[3]

[1] Intel Research, Intel Corporation, 2200 Mission College Blvd, Santa Clara, CA 95052, USA Rainer.Lienhart@intel.com
[2] Intel Research, Intel Corporation, Turgeneva st., 30, Nizhny Novgorod, Russia
[3] Perceptual Interfaces and Realities Lab., University of Maryland, College Park, USA

Abstract. Array audio-visual signal processing algorithms require time-synchronized capture of AV-data on distributed platforms. In addition, the geometry of the array of cameras, microphones, speakers and displays is often required. In this chapter we present a novel setup involving network of wireless computing platforms with sensors and actuators onboard, and algorithms that can provide both synchronized I/O and self-localization of the I/O devices in 3D space. The proposed algorithms synchronize input and output for a network of distributed multi-channel audio sensors and actuators connected to general purpose computing platforms (GPCs) such as laptops, PDAs and tablets. IEEE 802.11 wireless network is used to deliver the global clock to distributed GPCs, while the interrupt timestamping mechanism is employed to distribute the clock between I/O devices. Experimental results demonstrate a precision in A/D D/A synchronization precision better than 50 μs (a couple of samples at 48 kHz). We also present a novel algorithm to automatically determine the relative 3D positions of the sensors and actuators connected to GPCs. A closed form approximate solution is derived using the technique of metric multidimensional scaling, which is further refined by minimizing a non-linear error function. Our formulation and solution accounts for the errors in localization, due to lack of temporal synchronization among different platforms. The performance limit for the sensor positions is analyzed with respect to the number of sensors and actuators as well as their geometry. Simulation results are reported together with a discussion of the practical issues in a real-time system.

Key words: Distributed Sensor Networks, Self-Localizing Sensor Networks, Multichannel Signal Processing

1 Introduction

Arrays of audio/video sensors and actuators such as microphones, cameras, loudspeakers and displays along with array processing algorithms offer a rich set of new features for emerging applications. Until now, array processing required expensive dedicated multi-channel I/O cards and high-throughput computing systems to process multiple channels on a single machine. Recent advances in mobile computing and communication technologies, however, suggest a novel and very attractive platform for implementing these algorithms. Students in classrooms and co-workers at meetings are nowadays accompanied by several mobile computing and communication devices with audio and video I/O capabilities onboard such as laptops, PDA's, and tablets. In addition, high-speed wireless network connections, like IEEE 802.11a/b/g, are available to network those devices. Such ad-hoc sensor/actuator networks can enable emerging applications that include multi-stream audio and video, smart audio/video conference rooms, meeting recordings, automatic lecture summarization, hands-free voice communication, speech enhancement and object localization. No dedicated infrastructure in terms of the sensors, actuators, multi-channel interface cards and computing power is required. Multiple GPCs along with their sensors and actuators co-operate on providing transparent synchronized I/O. However, there are several important technical and theoretical problems to be addressed before the idea of using those devices for array DSP algorithms can materialize in real-life applications.

Figure 1 shows a schematic representation of our proposed *distributed computing platform* consisting of N GPC platforms(e.g., laptops). Each GPC is equipped with audio sensors (microphones), actuators (loudspeakers), and wireless communication capabilities. Given this setup, one of the most important problems is to provide a common reference time to a network of distributed computers and their I/O channels. A second important problem is to provide a common 3D coordinate system for the locations of the sensors and actuators. Solutions to both problems will be presented.

2 Providing Common Time

To illustrate the importance of time synchronization we implemented a Blind Source Separation (BSS) algorithm published in [2]. In the simplest setting two sound sources are separated using the input of two microphones, each connected to a different laptop. However, without synchronization of A/Ds the BSS algorithm failed to perform separation. Figure 2 demonstrates how a difference of only a few Hz in audio sampling frequency between two channels (laptops) impacts source separation. On the x-axis the sampling difference in Hz between two audio channels at about 16 kHz is shown against the achieved signal separation gain by BSS on the y-axis. As can be seen in Figure 2, a difference of only 2 Hz at 16 kHz reduces the signal separation gain from

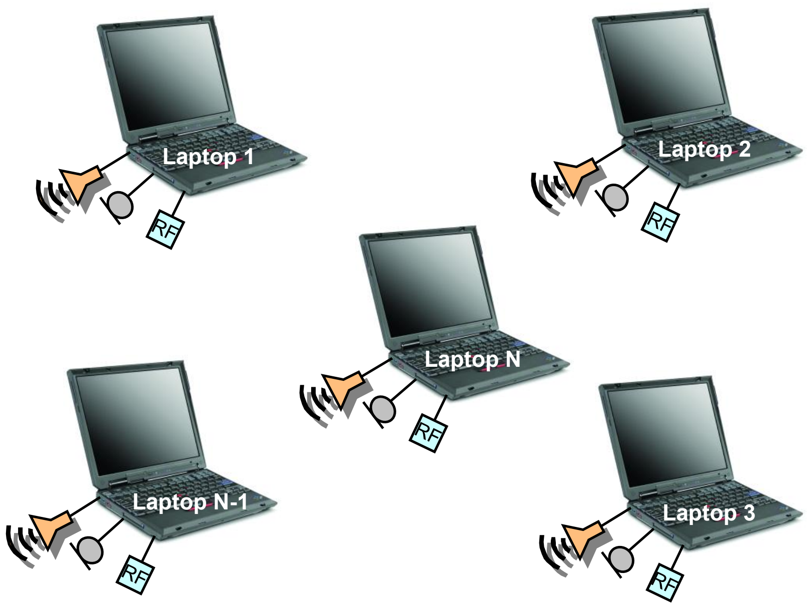

Fig. 1. Distributed computing platform consisting of N general-purpose computers along with their onboard audio sensors, actuators and wireless communication capabilities.

8.5 dB to about 2 dB only. In real life the difference in sampling frequency can be even higher as we illustrate in Table 1. BSS is not the only algorithm that is extremely sensitive to sampling synchronization. Other applications that require similar precision of time synchronization between channels are acoustic beamforming and 3D audio rendering.

Table 1. Audio sampling rates of several laptops.

Laptop	Dell Inspiron 7000	IBM ThinkPad T20	IBM ThinkPad 600E	IBM ThinkPad T23
Sampling rate, Hz	16001.7	16003.6	16001.8	16009.5

2.1 Related Work

The problem of time synchronization in distributed computing systems has been discussed extensively in the literature in the context of maintaining clock

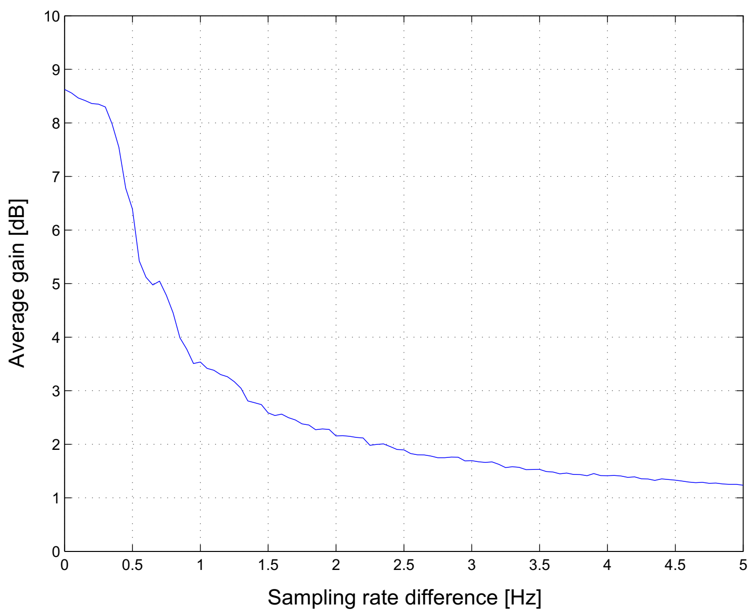

Fig. 2. Sensitivity of acoustic source separation performance to small sample rate differences. Channel 1 is assumed to sample at 16 kHz, while channel 2 is assumed to sample at $16000 + x$ Hz. Signal separation gain is calculated for the Blind Source Separation algorithm in [2].

synchrony throughout large geographic areas. Each process exchanges messages with its peers to determine a common clock. Seminal works have been reported in [6] and [9]. However, the results provided there can not be applied directly to our problem, since the precision of time synchronization is too low. NTP, the Network Time Protocol, currently used worldwide for clock synchronization in the best case achieves synchronization in the range of milliseconds - 2 to 3 orders of magnitudes higher than the microsecond resolution needed for our application scenarios. The Global Positioning system (GPS) provides a much higher clock resolution. Its reported time is steered to stay always within one microsecond of UTC (Coordinated Universal Time). In practice, it has been within 50 nanoseconds. With the Standard Positioning Service (SPS) a GPS receiver can obtain a time transfer accuracy to UTC within 340 nanoseconds (95% interval). GPS, however, only works reliably outdoors and thus does not completely fit our application scenario. There is also some recent work on synchronization in wireless sensor networks. In [10, 1], the reference-broadcast synchronization method is introduced. In this scheme, nodes send reference beacons to their neighbors based on a physical broadcast medium. All nodes record the local time at which they receive the broadcasts (e.g., by using the RDTSC instruction of the Pentium® processor family; the Read-Time Stamp Counter counts clocktics since the processor was started). Based on the exchange of this information, nodes can translate each other's clock. Although promising, the worst case performance of 150μs reported in [10] is too high for our application scenario. Our system is similar in spirit but we rely on additional processing to reduce errors in estimation of synchronization parameters.

In general, all clock synchronization algorithms studied in the literature only address the problem of providing a common clock on distributed computing platforms. They do not address how the I/O can be synchronized with the common clock (we proposed one solution in [7]). In other words, even under the assumption of a perfect clock on each platform, there is still a mechanism required to link the common clock to the data in the I/O channels. On a GPC this is a challenge in itself and we address this problem in this chapter.

2.2 Problem Formulation

We tackle the problem of distributed I/O synchronization in two steps: (1) the local CPU clocks of the GPCs are synchronized against a global clock (inter-platform), and (2) I/O is synchronized against the local clocks and thus also against the global clock (intra-platform). In the experimental results, one of the CPU clocks will arbitrarily be chosen as the global clock.

Each GPC has a local CPU clock (e.g., RDTSC). Let $t_i(t)$ denote the value of this clock on the i-th GPC at some global time t. Assuming a linear model between the global clock and the local platform clock, we get

$$t_i(t) = a_i(t)t + b_i(t), \tag{1}$$

where $a_i(t)$ and $b_i(t)$ are timing model parameters for the i-th GPC. The dependency of the model parameters on global time t approximates instabilities in the clock frequency due to temperature variations and other factors. In practice, these instabilities are in the order of 10^{-5}. In the rest of this section we will omit explicit time dependency to simplify our notations. Similarly, the sampling times of audio A/Ds and D/As on GPC's are approximated as:

$$\tau_i(t_i) = \alpha_i(t_i)t_i + \beta_i(t_i). \tag{2}$$

In this model τ_i is simply the number of samples produced by A/D (or consumed by D/A) converter since the start of the audio I/O. Note that two different timing models are required since the audio I/O devices on a typical PC platform have their own internal clock that is not synchronized to other platform clocks such as the RDTSC.

Given the two timing models above the problem that we address in this section can be formulated as finding $t(\tau_i)$ - the global time stamp of audio sample τ_i. We separate it into two subproblems: finding $\hat{\alpha}_i$ and $\hat{\beta}_i$ such that $t_i(\tau_i) = \hat{\alpha}_i\tau_i + \hat{\beta}_i$ (convert sample number to local time stamp with $\hat{\alpha} = \alpha^{-1}$ and $\hat{\beta} = -\beta/\alpha$) and finding $\hat{a}$ and $\hat{b}$ such that $t(t_i) = \hat{a}t_i + \hat{b}$ (convert value of local clock to global time with $\hat{a} = a^{-1}$ and $\hat{b} = -b/a$).

2.3 Timing relationships on GPC platform

In order to understand the inter and intra platform synchronization methods proposed in here we briefly describe the operations and timing relationships on a typical GPC. Figure 3 shows a processing diagrams of networking and

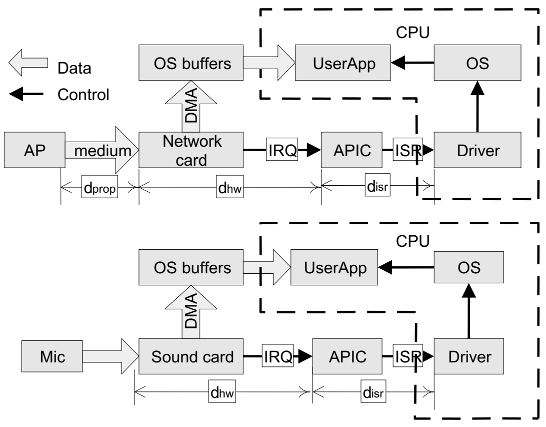

Fig. 3. Network (top part) and audio (bottom part) data and control flows on a typical GPC platform.

audio I/O. Both I/O operations have a very similar structure that can be described by the following sequence of actions (only input path is described):

1. Incoming data is received and processed by a hardware device, and eventually is put into a Direct Memory Access (DMA) buffer. This is modeled in Figure 3 by the delay d_{hw}, which is approximately constant for similar hardware.
2. The DMA controller transfers the data to a memory block allocated by the system and signals this event to the CPU by an Interrupt ReQuest (IRQ). This stage introduces variable delay due to memory bus arbitration between different agents (i.e., CPU, graphics adapter, other DMA's).
3. The interrupt controller (APIC) queues the interrupt and schedules a time slot for handling. Because APIC is handling requests from multiple I/O devices this stage introduces variable delay with standard deviation of around 6 ms and the maximum deviation of 30 ms. Both previous stages are modeled by d_{isr} in Figure 3.
4. The Interrupt Service Routine (ISR) of the device driver is called, and the driver sends notification to the Operating System (OS).
5. The OS delivers a notification and data to the user application(s). This stage has to be executed in a multitasking software environment and this

leads to significant variable delays that depend on CPU utilization and many other factors.

In summary, data traverses multiple hardware and software stages in order to travel from an I/O device to the CPU and back. The delay introduced by the various stages is highly variable making the problem of providing a global clock to the GPCs and distributing it to I/O devices very challenging. It is advantageous to perform synchronization as close to hardware as possible, therefore our solution is implemented at the driver level (during ISR) thus avoiding additional errors due to OS processing.

2.4 Inter-platform synchronization

For the synchronization of CPU clocks over a wireless network we propose to use a series of arrival times of multicast packets sent by the wireless access point (AP). In our current approach we implement a pairwise time synchronization with one node chosen as the master (say $t(t_0) = t_0$). All other nodes (clients) are required to synchronize their clocks to the master . A similar approach was also suggested in [10, 1]. Our solution, however, extends it by introducing additional constraints on the timing model. In order to provide a global clock to distributed platforms that is potentially useful to other applications (e.g., joint stream processing and distributed computations) we impose the clock monotonicity condition to make sure that the global clock is monotonically increasing during model parameter adaptation. In addition we smooth the clock model (a_i and b_i in equation (1)) variation by limiting the magnitude of its updates.

The algorithm consists of the following steps:

1. AP sends next beacon packet.
2. Master node records its local time of packet arrival and distributes it to all other nodes.
3. Client nodes record both their local times of arrival of beacon packets from AP, and the corresponding times received from the master.
4. Clients update local timing models based on the set of local timestamps and corresponding master timestamps.

Let us assume that in Figure 3 the packet j arrives to multiple platforms approximately at the same global time corresponding to local clocks t_i^j ($d_{prop} \approx 0$). The set of observations available on the platforms consist of pairs of timestamps $(\tilde{t}_0^j, \tilde{t}_i^j)$. From Figure 3 we have $\tilde{t}^j = t^j + d_{hw} + d_{isr}$ (we omitted dependency on i) that we further model as $\tilde{t}^j = t^j + d + n$. In this approximation d models all constant delay component and n represents the stochastic component. Given the set of observations $(\tilde{t}_0^j, \tilde{t}_i^j)$ we are required to estimate the timing model parameters $\hat{a}_i$ and $\hat{b}_i$ for all slave platforms. In our experiments a window of 3 minutes is used to estimate current values of $\hat{a}_i$ and $\hat{b}_i$ using the least trimmed squares (LTS) regression [14]. LTS is equivalent to performing least squares fit, trimming the observations that correspond to the largest residuals (defined as the distance of the observed value to the linear fit), and then computing a least squares regression model for the remaining observations. Figure 4 shows comparison of quantiles of residuals with quantiles of normal distribution and Figure 5 plots the histogram of residuals. The distribution appears to be close to Gaussian except for the presence of a few outliers (see Figure 4) that do not fit into a normal distribution. The trimming step is specifically targeted to remove those outliers.

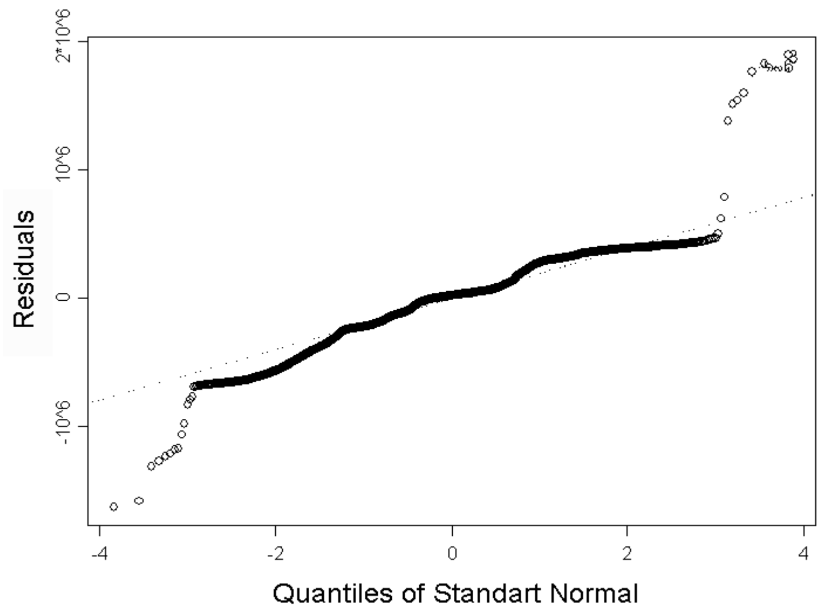

Fig. 4. Comparison quantiles of residuals with quantiles of the normal distribution. Points away from the straight line are treated as outliers and removed during regression.

2.5 Intra-platform synchronization

In order to synchronize the audio clock to the CPU clock we use a similar approach as the one presented in the previous section. The ISR of the audio driver is modified to timestamp the samples in the OS buffer using the CPU clock to form a set of observation pairs $(\tilde{t}_i^j, \tau_i^j)$, where j now represents the index of an audio data packet.

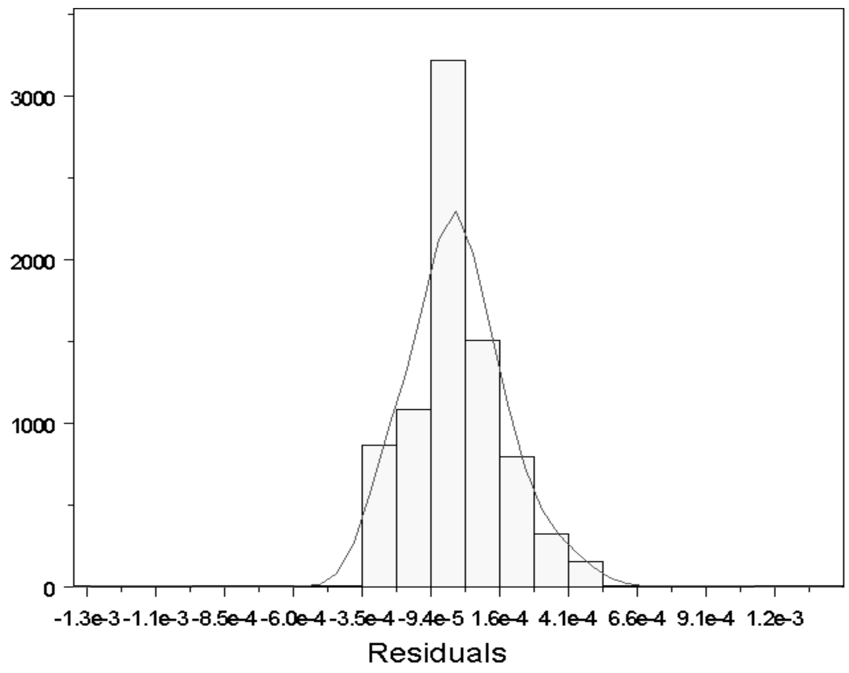

Fig. 5. Histogram of residuals and the normal probability density function.

Following our model in Figure 3 we have $\tilde{t}^j = t^j + d_{hw} + d_{isr}$ (we omitted dependency on i) that we further represent as $\tilde{t}^j = t^j + d + n$. Except for the fact that the τ^j are available without any noise (it is simply the number of samples processed!) we are back to the problem of determining the linear fit parameters for pairs of observations that we solved in the previous section using the LTS method.

In summary, by using LTS procedure twice both local and global synchronization problems are solved and the audio samples can be precisely synchronized on the distributed GPCs.

2.6 Experimental results

The distributed test system was implemented with several off-the-shelf Intel® Centrino laptops using the following software components (see also Figure 6): (a) A *modified WLAN card driver* timestamps each interrupt, parses incoming packets in order to find all master beacon frames, and stores their timestamp values in a cyclic shared memory buffer. The timestamp values as well as the corresponding message IDs are further accessible through the standard driver I/O interface. (b) A *modified AC97 driver* timestamps ISRs and calculates the number of samples transmitted since the beginning of the audio capture/playblack. The value pair is placed into a cyclic shared memory buffer. (c)

The *synchronization agents* are responsible for synchronizing the distributed system. We have three types of agents: the multicast server (MCS), the master synchronization agent (SAM) and the slave synchronization agent (SAS). The MCS periodically broadcasts beacon packets (short packets with unique ID as the payload). The SAM and SASs use the modified WLAN driver to detect the beacons. The SAM periodically broadcasts its recorded timestamps of beacon arrivals to the SAS devices. Based on SASs' recorded timestamps and the corresponding SAM timestamps, each SAS calculates the clock parameter to convert between the platform clock and the global clock. The clock parameters are placed in shared memory for use by other applications. (d) The *Synchronization API* allows user applications to retrieve the local clock value, access the clock parameters, and convert between the platform and global clock. (e) The *audio API* allows user applications to retrieve pairs of local timestamps and sample numbers, as well as to convert global timestamp values to sample numbers and vice versa. It also provides transparent synchronized capture and playback.

Based on these components a distributed audio rendering system was implemented with three laptops (see Figure 6). The first laptop was used as the MCS. Modified AC97 and WLAN drivers were installed on other two laptops. SAM was started on the second laptop, while SAS were started on the third laptop. The distributed system was instructed through the audio API to synchronously playback a Maximum Length Sequence (MLS) signal on the two synchronized laptops. The line-out signal of both laptops were recorded by a multichannel soundcard. The measured inter-GPC offset was at most 2 samples at 48 kHz (less than 42 μs).

3 Providing Common Space

A common space (coordinate system) can be provided by means of actively estimating the three dimensional positions of the sensors and actuators. Many multi-microphone array processing algorithms (like sound source localization or conventional beamforming) need to know the positions of the microphones very precisely. Current systems either place the microphones in known locations or manually calibrate them. There are some approaches which do calibration using speakers in known locations [15]. We offer here a more general approach where no assumptions about the positions of the speakers are made. Our solution explicitly accounts for the errors in localization due to lack of temporal synchronization among different platforms.

We again refer to Figure 1 showing a schematic representation of a distributed computing platform consisting of N GPCs. For the purpose of performing space localization one of them is configured to be the master. The master controls the distributed computing platform and performs the location estimation. As already described each GPC is assumed to be equipped with

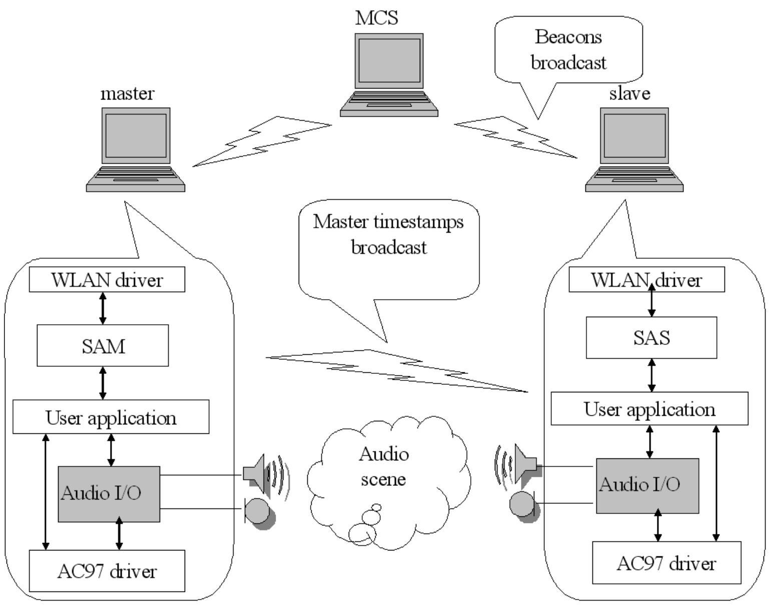

Fig. 6. Distributed audio rendering/capturing system setup

audio sensors (microphones), actuators (loudspeakers), and wireless communication capabilities.

3.1 Related Work

The problem of self-localization for a network of nodes generally involves two steps: ranging and multilateration. The ranging technology can be either based on the Time Of Flight (TOF) or the Received Signal Strength (RSS) of acoustic, ultrasound or radio frequency (RF) signals. The GPS system and long range wireless sensor networks use RF technology for range estimation. Localization using Global Positioning System (GPS) is not suitable for our applications since GPS systems do not work indoors and are very expensive. Also RSS based on RF is very unpredictable [16] and the RF TOF is quite small to be used indoors. [16] discusses systems based on ultrasound TOF using specialized hardware (like motes) as the nodes. However, our goal is to use the already available sensors and actuators on GPCs to estimate their positions. Our ranging technology is based on acoustic TOF as in [15, 11, 4]. Once we have the range estimates the Maximum Likelihood (ML) estimate can be used to get the positions. To find the solution one can assume that the locations of a few sources are known as in [15, 16] or make no such assumptions as in [11, 19].

3.2 Problem Formulation

Given a set of M acoustic sensors (microphones) and S acoustic actuators (speakers) in unknown locations, our goal is to estimate their three dimensional coordinates. Each of the acoustic actuators is excited using a known calibration signal such as maximum length sequences or chirp signals, and the Time of Flight (TOF) is estimated for each of the acoustic sensors. The TOF for a given pair of microphone and speaker is defined as the time taken by the acoustic signal to travel form the speaker to the microphone.

Let $\mathbf{m_i}$ for $i \in [1, M]$ and $\mathbf{s_j}$ for $j \in [1, S]$ be the three dimensional vectors representing the spatial coordinates of the i^{th} microphone and j^{th} speaker, respectively. We excite one of the S speakers at a time and measure the TOF at each of the M microphones. Let TOF_{ij}^{actual} be the actual TOF for the i^{th} microphone due to the j^{th} source. Based on geometry the actual TOF can be written as (assuming a direct path),

$$TOF_{ij}^{actual} = \frac{\parallel \mathbf{m_i} - \mathbf{s_j} \parallel}{c} \tag{3}$$

where c the speed of sound in the acoustical medium [4] and $\| \|$ is the Euclidean norm. The TOF which we estimate based on the signal captured confirms to this model only when all the sensors start capturing at the same instant and we know when the calibration signal was sent from the speaker.

However in a typical distributed setup as shown in Figure 1, the master starts the audio capture and playback on each of the GPCs one by one. As a result the capture starts at different instants on each GPC and also the time at which the calibration signal was emitted from each loudspeaker is not known. So the TOF which we measure from the signal captured includes both the speaker emission start time and the microphone capture start time (See Figure 7 where $\hat{TOF}_{ij}$ is what we measure and TOF_{ij} is what we require). The speaker emission start time is defined as the time at which the sound is actually emitted from the speaker. This includes the time when the play back command was issued (with reference to some time origin), the network delay involved in starting the playback on a different machine (if the speaker is on a different GPC), the delay in setting up the audio buffers and also the time required for the speaker diaphragm to start vibrating The microphone capture start time is defined as the time instant at which capture is started. This includes the time when the capture command was issued, the network delay involved in starting the capture on a different machine and the delay in transferring the captured sample from the sound card to the buffers.

Let ts_j be the emission start time for the j^{th} source and tm_i be the capture start time for the i^{th} microphone (see Figure 7). Incorporating these two the actual TOF now becomes,

[4]The speed of sound in a given acoustical medium is assumed to be constant. In air it is given by $c = (331 + 0.6T)m/s$, where T is the temperature of the medium in Celsius degrees.

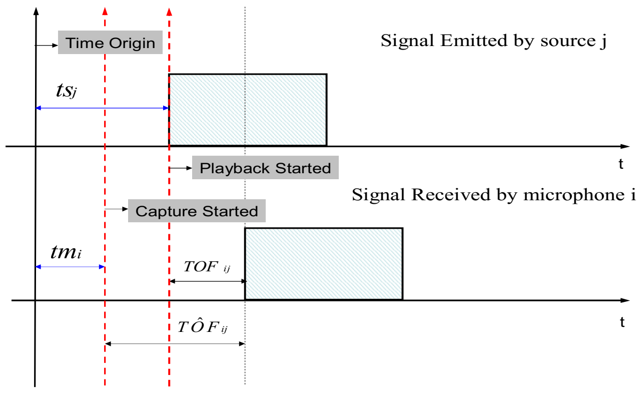

Fig. 7. Schematic indicating the errors due to unknown speaker emission and microphone capture start time.

$$T\hat{O}F_{ij}^{actual} = \frac{\parallel \mathbf{m_i} - \mathbf{s_j} \parallel}{c} + ts_j - tm_i \tag{4}$$

The origin can be arbitrary since $T\hat{O}F_{ij}^{actual}$ depends on the difference of ts_j and tm_i. We start the audio capture on each GPC one by one. We define the microphone on which the audio capture was started first as our first microphone. In practice, we set $tm_1 = 0$ i.e. the time at which the first microphone started capturing is our origin. We define all other times with respect to this origin. We can jointly estimate the unknown source emission and capture start times along with microphone and source coordinates.

In this chapter we propose to use the Time Difference Of Arrival (TDOA) instead of the TOF. The TDOA for a given pair of microphones and a speaker is defined as the time difference between the signal received by the two microphones [5]. Let $TDOA_{ikj}^{estimated}$ be the estimated TDOA between the i^{th} and the k^{th} microphone when the j^{th} source is excited. Let $TDOA_{ikj}^{actual}$ be the actual TDOA. It is given by

$$TDOA_{ikj}^{actual} = \frac{\parallel \mathbf{m_i} - \mathbf{s_j} \parallel - \parallel \mathbf{m_k} - \mathbf{s_j} \parallel}{c} \tag{5}$$

Including the source emission and capture start times, it becomes

$$TD\hat{O}A_{ikj}^{actual} = \frac{\parallel \mathbf{m_i} - \mathbf{s_j} \parallel - \parallel \mathbf{m_k} - \mathbf{s_j} \parallel}{c} + tm_k - tm_i \tag{6}$$

In the case of TDOA the source emission time is the same for both microphones and thus gets cancelled out. Therefore, by using TDOA measurements instead of TOF we can reduce the number of parameters to be estimated.

[5] Given M microphones and S speakers we can have $MS(M-1)/2$ TDOA measurements as opposed to MS TOF measurements. Of these $MS(M-1)/2$ TDOA measurements only $(M-1)S$ are linearly independent.

3.3 Maximum Likelihood (ML) Estimate

Assuming a Gaussian noise model for the TDOA observations we can derive the ML estimate as follows. Let Θ, be a vector of length $P \times 1$, representing all the unknown non-random parameters to be estimated (microphone and speaker coordinates and microphone capture start times). Let Γ, be a vector of length $N \times 1$, representing noisy TDOA measurements. Let $T(\Theta)$, be a vector of length $N \times 1$, representing the actual value of the observations. Then our model for the observations is $\Gamma = T(\Theta) + \eta$ where η is the zero-mean additive white Gaussian noise vector of length $N \times 1$ where each element has the variance σ_j^2. Also let us define Σ to be the $N \times N$ covariance matrix of the noise vector N. The likelihood function of Γ in vector form can be written as:

$$p(\Gamma/\Theta) = (2\pi)^{-\frac{N}{2}} \mid \Sigma \mid^{-\frac{1}{2}} \exp -\frac{1}{2}(\Gamma - T)^T \Sigma^{-1}(\Gamma - T) \tag{7}$$

The ML estimate of Θ is the one which maximizes the log likelihood ratio and is given by

$$\begin{aligned} \hat{\Theta}_{ML} &= \arg_{\Theta} \max F(\Theta, \Gamma) \\ F(\Theta, \Gamma) &= -\frac{1}{2}[\Gamma - T(\Theta)]^T \Sigma^{-1} [\Gamma - T(\Theta)] \end{aligned} \tag{8}$$

Assuming that each of the TDOAs are independently corrupted by zero-mean additive white Gaussian noise [6] of variance σ_{ikj}^2 the ML estimate turns out to be a nonlinear least squares problem (in this case Σ is a diagonal matrix), i.e.

$$\begin{aligned} \hat{\Theta}_{ML} &= \arg_{\Theta} \min[\widetilde{F}_{ML}(\Theta, \Gamma)] \\ \widetilde{F}_{ML}(\Theta, \Gamma) &= \\ \sum_{j=1}^{S} \sum_{i=1}^{M} \sum_{k=i+1}^{M} & \frac{(TDOA_{ikj}^{estimated} - T\hat{D}OA_{ikj}^{actual})^2}{\sigma_{ikj}^2} \end{aligned} \tag{9}$$

Since the solution depends only on pairwise distances, any translation, rotation and reflection of the global minimum found will also be a global minimum. In order to make the solution invariant to rotation and translation we select three arbitrary nodes to lie in a plane such that the first is at $(0, 0, 0)$, the second at $(x_1, 0, 0)$, and the third at $(x_2, y_2, 0)$. In two dimensions we select two nodes to lie in a line, the first at $(0, 0)$ and the second at $(x_1, 0)$. To

[6] We estimate the TDOA or TOF using Generalized Cross Correlation (GCC)[5]. The estimated TDOA or TOF is corrupted due to ambient noise and room reverberation. For high SNR the delays estimated by the GCC can be shown to be normally distributed with zero mean [5].

eliminate the ambiguity due to reflection along Z-axis(3D) or Y-axis(2D) we specify one more node to lie in the positive Z-axis(in 3D) or positive Y-axis(in 2D). Also the reflections along X-axis and Y-axis(for 3D) can be eliminated by assuming the nodes which we fix to lie on the positive side of the respective axes i.e $x_1 > 0$ and $y_2 > 0$. Similar to fixing a reference coordinate system in space we introduce a reference time line by setting $tm_1 = 0$.

3.4 Problem Solution

The ML estimate for the node coordinates of the microphones and loudspeakers is implicitly defined as the minimum of a non-linear function. The solution is same as a nonlinear weighted least squares problem. The Levenberg-Marquardt method is a popular method for solving non-linear least squares problems. For more details on nonlinear minimization refer to [3]. Least squares optimization requires that the total number of observations is greater than or equal to the total number of parameters to be estimated. This imposes a minimum number of microphones and speakers required for the position estimation method to work. Assuming M=S=K, Table 2 lists the minimum K required for the algorithm.

Table 2. Minimum value of Microphone Speaker Pairs (K) required for different estimation procedures (D-Dimension)

$K \geq$	$D = 2$	$D = 3$
TDOA Position Estimation	5	6
TDOA Joint Estimation	6	7

One problem with minimization is that it can often get stuck in a local minima. In order to avoid this we need a good starting guess. We use the technique of metric multidimensional scaling (MDS) [17] to get a closed form approximation for the microphone and speaker positions, which is used as a starting point for the minimization routine. MDS is a popular method in psychology and denotes a set of data-analysis techniques for the analysis of proximity data on a set of stimuli for revealing the hidden structure underlying the data.

Given a set of N GPCs, let X be a $N \times 3$ matrix where each row represents the 3D coordinates of each GPC. Then the $N \times N$ matrix $B = XX^T$ is called the dot product matrix. By definition, B is a symmetric positive definite matrix, so the rank of B (i.e the number of positive eigen values) is equal to the dimension of the datapoints i.e. 3 in this case. Also based on the rank of B we can find whether the GPCs are on a plane (2D) or distributed in 3D. Starting with a matrix B (possibly corrupted by noise), it is possible to factor it to get the matrix of coordinates X. One method to factor B is to use singular value decomposition (SVD) [12], i.e., $B = U\Sigma U^T$ where Σ is a $N \times N$ diagonal matrix of singular values. The diagonal elements are arranged as $s_1 \geq s_2 \geq s_r > s_{r+1} = ... = s_N = 0$, where r is the rank of the matrix B. The columns of U are the corresponding singular vectors. We can write $X^{'} = U\Sigma^{1/2}$. From $X^{'}$ we can take the first three columns to get X. If the elements of B are exact (i.e., they are not corrupted by noise), then all the other columns are zero. It can be shown that SVD factorization minimizes the matrix norm $\parallel B - XX^T \parallel$.

In practice we can estimate the distance matrix D where the ij^{th} element is the Euclidean distance between the i^{th} and the j^{th} GPC. We have to convert this distance matrix D into a dot product matrix B. In order to form the dot product matrix we need to choose some point as the origin of our coordinate system. Any point can be selected as the origin, but Togerson [17] recommends the centroid of all the points. If the distances have random errors then choosing the centroid as the origin will minimize the errors as they tend to cancel each other. We obtain the dot product matrix B using the cosine law which relates the distance between two vectors to their lengths and the cosine of the angle between them. Refer to [13] for a detailed derivation of how to convert the distance matrix to the scalar product matrix.

In the case of M microphones and S speakers we cannot use MDS directly because we cannot measure all the pairwise distances. We can measure the distance between each speaker and all the microphones. However, we cannot measure the distance between two microphones or two speakers. In order to apply MDS, we cluster microphones and speakers, which are close together. In practice, it is justified by the fact that the microphones and the speakers on the same GPC are close together. Assuming that all GPCs have at least one microphone and one speaker, we can measure the distance between the speakers on one GPC and the microphones on the other and vice versa. Taking the average we get an approximate distance between the two GPCs. The position estimate obtained using MDS has the centroid as the origin and an arbitrary orientation. Therefore, the solution obtained using MDS is translated, rotated and reflected to the reference coordinate system discussed earlier. Figure 8 shows an example with 10 laptops each having one microphone and one speaker. The actual locations of the sensors and actuators are shown as 'x'. The '*'s are the approximate GPC locations resulting from MDS. As can be seen the MDS result is very close to the true microphone and speaker locations. Each GPC location got using MDS is randomly perturbed to be used as a initial guess for the microphones and speakers on that GPC. The 'o' are the results from the ML estimation procedure using the perturbed MDS locations as the initial guess. The algorithm can be summarized as follows:

ALGORITHM

Say we have M microphones and S speakers

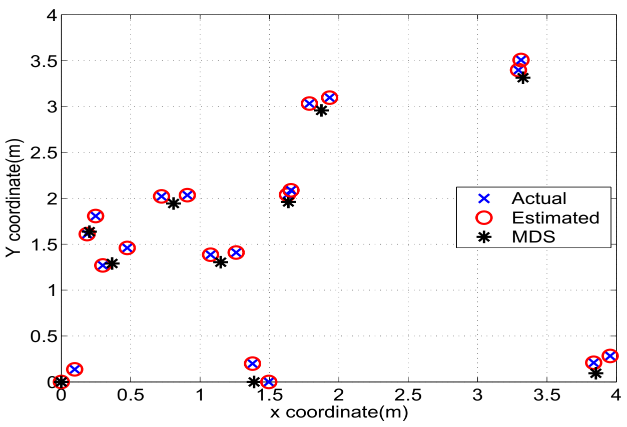

Fig. 8. Results of Multidimensional Scaling for a network consisting of 10 GPCs each having one microphone and one speaker.

- **STEP 0**: *Form a Coordinate system by selecting three nodes: The first one as the origin, the second to define the x-axis and the third to form the xy-plane. Also select a fourth node to represent the positive z-axis.*
- **STEP 1**: *Compute the $M \times S$ Time Of Flight (TOF) matrix.*
- **STEP 2**:
  - *Convert the TOF matrix into an approximate distance matrix by appropriately clustering the closest microphones and speakers.*
  - *Get the approximate positions of the clustered entities using metric Multidimensional Scaling.*
  - *Translate, rotate and mirror the coordinates to the coordinate system specified in STEP 0.*
- **STEP 3**:
  - *Slightly perturb the coordinates from STEP 2 to get approximate initial guess for the microphone and speaker coordinates.*
  - *Set an approximate initial guess for the microphone capture start time*
  - *Minimize the TDOA based error function using the Levenberg-Marquardat method to get the final positions of the microphones and speakers.*

3.5 Analysis

The Cramér-Rao bound (CRB) gives a lower bound on the variance of *any* unbiased estimate [18]. We derived it in [13] for our system leading to the following important observations.

The more microphones and speakers in the network, the smaller the error in estimating their positions as can be seen from Figure 9(a) and 9(b) which

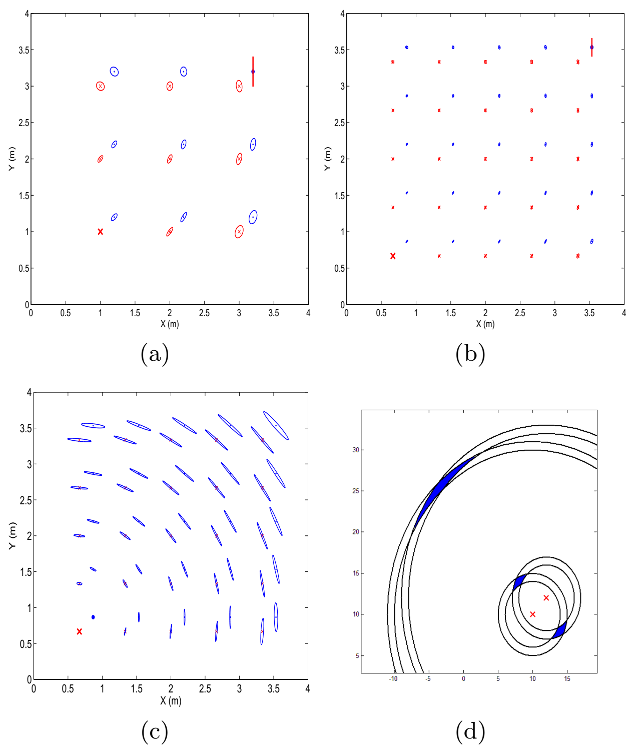

Fig. 9. 95% uncertainty ellipses for a regular 2 dimensional array of (a) 9 speakers and 9 microphones, (b)and (c) 25 speakers and 25 microphones. Noise variance for all cases is $\sigma^2 = 10^{-9}$. The microphones are represented as crosses ($\times$) and the speakers as dots (.). The position of one microphone and the x coordinate of one speaker is assumed to be known (shown in bold). In (c) the known nodes are close to each other and in (a) and (b) they are spread out one at each corner of the grid. (d) schematic to explain the shape of the uncertainty ellipses.

shows the 95% uncertainty ellipses for different number of sensors and actuators. Intuitively this can be explained as follows: Let there be a total of n nodes in the network whose coordinates are unknown. Then we have to estimate a total of $3n$ parameters. The total number of TOF measurements available is however $n^2/4$ (assuming that there are $n/2$ microphones and $n/2$ speakers). So if the number of unknown parameters increases as $O(n)$, the number of available measurements increases as $O(n^2)$. So the linear increase in the number of unknown parameters, is compensated by the quadratic increase in the available measurements.

In our formulation we assumed that we know the positions of a certain number of nodes, i.e we fix three of the nodes to lie in the x-y plane. The CRB depends on which of the sensor nodes are assumed to have known positions. In

Figure 9(c) the two known nodes are at one corner of the grid. It can be seen that the uncertainty ellipse becomes wider as you move away form the known nodes. The uncertainty in the direction tangential to the line joining the sensor node and the center of the known nodes is much larger than along the line. The reason for this can be explained for a simple case where we know the locations of two speakers (see Figure 9(d)). A circular band centered at each speaker represents the uncertainty in the distance estimation. The intersection of the two bands corresponding to the two speakers gives the uncertainty region for the position of the sensor. For nodes far away from the two speakers the region widens because of the decrease in the curvature. It is beneficial if the known nodes are on the edges of the network and as faraway from each other as possible. In Figure 9(b) the known sensor nodes are on the edges of the network. As can be seen there is a substantial reduction in the dimensions of the uncertainty ellipses. In order to minimize the error due to Gaussian noise we should choose the three reference nodes (in 3D) as far as possible.

3.6 Experimental Details and Results

We implemented a prototype system consisting of 6 microphones and 6 speakers. The real-time setup has been tested in a synchronized as well as a distributed setup using laptops. The ground truth was measured manually to validate the results from the position calibration methods.

A linear chirp signal was used to measure the TOF. A linear chirp signal is a short pulse in which the frequency of the signal varies linearly between two preset frequencies. In our system, we used the chirp signal of 512 samples at 44.1kHz (11.61 ms) as our calibration signal. The instantaneous frequency varied linearly from 5 kHz to 8 kHz. The initial and the final frequency was chosen to lie in the common pass band of the microphone and the speaker frequency response. The chirp signal send by the speaker is convolved with the room impulse response resulting in the spreading of the chirp signal.

One of the problems in accurately estimating the TOF is due to the multipath propagation caused by room reflections. The time-delay may be found by locating the peak in the cross-correlation of the signals received over the two microphones. However this method is not robust to noise and reverberations. Knapp and Carter [5] developed the Generalized Cross Correlation (GCC) method. In this method, the delay estimate is the time lag which maximizes the cross-correlation between filtered versions of the received signals [5]. The cross-correlation of the filtered versions of the signals is called as the Generalized Cross Correlation (GCC) function. The GCC function $R_{x_1x_2}(\tau)$ is computed as [5] $R_{x_1x_2}(\tau) = \int_{-\infty}^{\infty} W(\omega)X_1(\omega)X_2^*(\omega)e^{j\omega\tau}d\omega$ where $X_1(\omega)$, $X_2(\omega)$ are the Fourier transforms of the microphone signals $x_1(t)$, $x_2(t)$, respectively and $W(\omega)$ is the weighting function. The two most commonly using weighting functions are the ML and the PHAT weighting. The ML weighting function performs well for low room reverberation. As the room reverberation increases this method shows severe performance degradations.

Since the spectral characteristics of the received signal are modified by the multipath propagation in a room, the GCC function is made more robust by deemphasizing the frequency dependent weighting. The Phase Transform is one extreme where the magnitude spectrum is flattened. The PHAT weighting is given by $W_{PHAT}(\omega) = 1/|X_1(\omega)X_2^*(\omega)|$. By flattening out the magnitude spectrum the resulting peak in the GCC function corresponds to the dominant delay. However, the disadvantage of the PHAT weighting is that it places equal emphasizes on both the low and high SNR regions, and hence it works well only when the noise level is low.

In practice, the sensors' and actuators' three dimensional locations could be estimated with an average bias of 0.08 cm and average standard deviation of 3 cm (results averaged over 100 trials). Our algorithm assumed that the sampling rate is known for each laptop and the clock does not drift. Our initial real time setup integrates the distributed synchronization scheme using ML sequence as proposed in [8] to resample and align the different audio streams. It has now been converted to use the synchronization scheme presented in Section 2. As regards to CPU utilization the TOA estimation consumes negligible resources. If we use a good initial guess via the Multidimensional Scaling technique then the minimization routine converges within 8 to 10 iterations.

4 Conclusion and Outlook

We presented our novel algorithms for self-synchronization of distributed AV-sensor networks in time (i.e., synchronized I/O) with a precision of the order of μs and for self-localization in space (i.e., 3D spatial coordinates) with a precision of the order of several centimeters. These algorithms when implemented in real-life systems can provide a completely new platform for future exciting research in areas ranging from manufacturing to communications, entertainment (especially games), and many more.

Researchers interested in using the common time and space infrastructure are encouraged to contact the authors for a research prototype of the system implemented for laptops with Intel® Centrino^{TM} Mobile Technology.

References

1. Elson J., Girod L., Estrin D. (2000) Fine-grained network time synchronization using reference broadcasts. 5th Symposium on OS Design and Implementation.
2. Fancourt C., Parra L. (2001) The coherence function in blind source separation of convolutive mixtures of non-stationary signals. Proc IEEE Workshop on Neural Networks for Signal Processing, 303–312.
3. Gill P., Murray W., Wright M. (1981) Practical Optimization.
4. Girod L., Bychkovskiy V., Elson J., Estrin D. (2002) Locating tiny sensors in time and space: A case study. Proc. International Conference on Computer Design.

5. Knapp C., Carter G. (1976) The generalized correlation method for estimation of time delay. IEEE Trans. Acoust., Speech, Signal Processing, **ASSP-24(4)**, 320–327.
6. Lamport L., Melliar-Smith P. (1985) Synchronizing clocks in the presence of faults. JACM, **32(1)**, 52–78.
7. Lienhart R., Kozintsev I., Wehr S. (2003) Universal synchronization scheme for distributed audio-video capture on heterogeneous computing platforms. Proc 11th ACM Conf on Multimedia, 263–266.
8. Lienhart R., Kozintsev I., Wehr S., Yeung M. (2003) On the importance of exact synchronization for distributed audio processing. Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing.
9. Mills D. (1991) Internet time synchronization: the network time protocol. IEEE Tran Comm, **39(10)**, 1482–1493.
10. Mock M., Frings R., Nett E., Trikaliotis S. (2000) Clock synchronization for wireless local area networks. IEEE 12th Euromicro Conference on Real-Time Systems (Euromicro RTS 2000), 183–189.
11. Moses R., Krishnamurthy D., Patterson R., (2003) A self-localization method for wireless sensor networks. Eurasip Journal on Applied Signal Processing Special Issue on Sensor Networks, **2003(4)**, 348–358.
12. Press H., Teukolsky S., Vettring W., Flannery B. (1995) Numerical Recipes in C The Art of Scientific Computing. Cambridge University Press, 2 edition.
13. Raykar V., Kozintsev I., Lienhart R. (2003) Self localization of acoustic sensors and actuators on distributed platforms. International Workshop on Multimedia Technologies in E-Learning and Collaboration (WOMTEC).
14. Rousseeuw P. (1984) Least median-of-squares regression. JACM, **79**, 871–880.
15. Sachar J., Silverman H., Patterson W. (2002) Position calibration of large-aperture microphone arrays. Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, 1797–1800.
16. Savvides A., Han C., Srivastava M. (2001) Dynamic fine-grained localization in ad-hoc wireless sensor networks. Proc. International Conference on Mobile Computing and Networking.
17. Torgerson W. (1952) Multidimensional scaling: I. theory and method. Psychometrika, **17**, 401–419.
18. Van Trees H. (2001) Detection, Estimation, and Modulation Theory, **1**. Wiley-Interscience.
19. Weiss A., Friedlander B. (1989) Array shape calibration using sources in unknown locations-a maxilmum-likelihood approach. IEEE Trans. Acoust., Speech, Signal Processing, **37(12)**, 1958–1966.