Time Synchronization for Multi-Modal Target Tracking in Heterogeneous Sensor Networks

Isaac Amundson, Manish Kushwaha, Branislav Kusy, Peter Volgyesi,
Gyula Simon†, Xenofon Koutsoukos, Akos Ledeczi

Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
Email: Xenofon.Koutsoukos@vanderbilt.edu
†Department of Computer Science
University of Pannonia
Veszprem, Hungary

***Abstract*— Heterogeneous sensor networks consisting of resource-constrained nodes as well as resource-intensive nodes equipped with high-bandwidth sensors offer significant advantages for developing large sensor networks for a diverse set of applications. Target tracking can benefit from such heterogeneous networks that support the use of sensors with different modalities. Such applications require tight time synchronization across the heterogeneous sensor network in order to improve both the estimation and real-time performance. In this paper we present a methodology for time synchronization in heterogeneous sensor networks. The synchronization methodology has been implemented as a network service and tested on an experimental testbed demonstrating an accuracy in the order of microseconds over a multi-hop network. In addition, we use the time synchronization method in a multi-modal tracking application for performing accurate sensor fusion of audio and video data collected from heterogeneous sensor nodes and we show that our method improves tracking performance.**

I. Introduction

Wireless sensor networks (WSNs) consist of large numbers of cooperating sensor nodes and have already demonstrated their utility in a wide range of applications including environmental monitoring, transportation, and health care. These types of applications collect sample data by monitoring the environment. In order to make sense of the collected samples, nodes usually send the data through the network to a *sensor-fusion* node where it can be analyzed. We call this process *reactive data fusion*. One important aspect of reactive data fusion is the need for a common notion of time among participating nodes.

In this paper, we consider multi-modal tracking as a reactive data fusion process where several nodes cooperate in estimating the position of a target. Target tracking using multiple sensor modalities is considered to be more robust, more accurate, more compact, and yields more information than using a single sensor modality [1]. A typical multi-modal tracking system consists of heterogeneous sensor nodes with different modalities, such as audio and video, sensing their local environment, and communicating with each other and the sensor fusion node. The local measurements are combined at the sensor fusion node for estimating the position and other characteristics of the target. A target tracking application requires tight synchronization across the heterogeneous sensor network in order to improve both the estimation and real-time performance.

Heterogeneous sensor networks (HSNs) consist of resource-constrained nodes as well as resource-intensive nodes equipped with high-bandwidth sensors such as cameras and are a promising direction for developing large sensor networks for a diverse set of applications [2], [3], [4]. Accurately synchronizing the clocks of all sensor nodes within a heterogeneous network is not a trivial task. Existing WSN time synchronization protocols (e.g. [5], [6], [7], [8], [9]) perform well on the devices for which they were designed. However, these have not been applied to networks of heterogeneous devices.

Our work focuses on achieving microsecond-accuracy synchronization in HSNs. We consider a multi-hop network consisting of Berkeley motes and Linux PCs, a dominant configuration for reactive data fusion applications in HSNs. Mote networks are capable of monitoring environmental phenomena with small energy consumption that allows for extended lifetime. PCs can support higher-bandwidth sensors, such as cameras, and can run intensive processing algorithms on the collected data before routing it to the sensor-fusion node. Time synchronization in both mote and PC networks has been studied independently, however, a sub-millisecond software synchronization method combining these two networks has not yet been developed to the best of our knowledge.

The contribution of the paper is twofold. First, we developed a technique for synchronization between a mote and a PC implemented completely in software. The technique enables the integration of existing protocols for synchronization of mote and PC networks. We have implemented a time synchronization service for reactive data fusion applications based on this methodology and our experimental results show that

we can achieve synchronization accuracy on the order of tens of microseconds. Second, we use the time synchronization method in a multi-modal tracking application. The time synchronization service allows us to perform accurate sensor fusion of audio and video data collected from heterogeneous sensor nodes. Our experimental results show that using an heterogeneous sensor network can improve tracking performance.

Time synchronization in sensor networks has been studied extensively in the literature and several protocols have been proposed. Reference Broadcast Synchronization (RBS) [6] is a protocol for synchronizing a set of receivers to the arrival of a reference beacon. Timestamp Synchronization (TSS) [5] was one of the first synchronization protocols to focus directly on WSNs, proposing the novel concept of time synchronization on-demand. Elapsed Time on Arrival (ETA) [10] provides a set of application programming interfaces for an abstract time synchronization service. RITS [9] is an extension of ETA over multiple hops. It incorporates a set of routing services to efficiently pass sensor data to a network sink node for data fusion. In [11], mote-PC synchronization was achieved by connecting the GPIO ports of a mote and IPAQ PDA using the mote-NIC model. Although using this technique can achieve microsecond-accurate synchronization, it was implemented as a hardware modification rather than in software.

Multi-modal tracking using audio and video sensors has been studied mainly for videoconferencing applications. Pingali et al. [1] describe audio-visual tracking to find a speaker among a group of individuals. They also argue that a joint AV tracking system is more desirable than each single modality. Yoshimi [12] presents a distributed multi-modal tracking system using multiple cameras and microphones to select a single speaker during a meeting. Zotkin [13] presents simulation results of a multi-modal tracking system for smart videoconferencing.

The rest of the paper is organized as follows. Section II describes the time synchronization problem. In Section III, we present our methodology, implementation, and experimental results for time synchronization in heterogeneous sensor networks. Section IV outlines our multi-model tracking system and presents some experimental results. Section V concludes.

II. Time Synchronization

In this section, we formulate the problem using a system model for time synchronization and we describe the sources of synchronization error.

A. System Model

Each sensor node in a WSN is a computing device that maintains its own local clock. Internally, the clock is a piece of circuitry that counts oscillations of a quartz crystal, energized at a specific frequency. When a certain number of these oscillations occur, a *clock-tick* counter is incremented. This counter is accessible from the operating system and its accuracy (with respect to atomic time) depends on the quality of the crystal, as well as various operating conditions such as temperature, pressure, humidity, and supply voltage. When a sensor node registers an event of interest, it will access the clock-tick counter and record a *timestamp* reflecting the time at which the event occurred.

We use the notation t to represent real (or atomic) time, and the notation t_e to represent the time at which an arbitrary event e occurred. Because each node records a timestamp according to its own clock, we specify the local time on node N_i at which event e was detected by the timestamp $N_i(t_e)$. Although the two timestamps $N_i(t_e)$ and $N_j(t_e)$ correspond to the same real-time instant t_e, this does not imply that $N_i(t_e) = N_j(t_e)$; the clock-tick counter on node N_i may still be offset from N_j. Therefore, from the perspective of node N_i, we define the *clock offset* with N_j at real-time t as $\phi_j^i(t) = N_i(t) - N_j(t)$. It may be the case that the offset changes over time. In other words, the *clock rate* of node N_i, $\frac{dN_i(t)}{dt}$, may not equal the ideal rate ($\frac{dN_i(t)}{dt} = 1$). We define the ratio between the clock rates of the two nodes as the *relative rate*, $rr_j^i = \frac{dN_i(t)}{dN_j(t)}$. The relative rate is a quantity directly related to the clock skew, which is defined as the difference between clock rates, and is used in our clock skew compensation methods. We refer to clock offset and clock rate characteristics as the node's *timescale*.

Knowledge of the clock offset and skew can be used to synchronize a pair of nodes. In this work, we use the *timescale transformation approach*. In this approach, references tables are maintained to keep track of the clock offsets and skew between a node and its neighbors. Each reference table is then used to transform a clock value from one timescale to another, providing each node with a common notion of time [14]. In sensor networks, the timescale transformation approach is preferable to physically change the value of the clock-tick counter of a node to match that of another node [14].

B. Sources of Synchronization Error

Synchronization requires passing timestamped messages between nodes. However, this communication has associated message delay, which has both deterministic and nondeterministic components that introduce error into the timescale transformation. We call the sequence of steps involved in communication between a pair of nodes the *critical path*. Figure 1 illustrates the critical path in a wireless connection. The critical path is not identical for all configurations, however, it can typically be characterized by the steps outlined in the figure (for more details, see for example [7], [6], [8]).

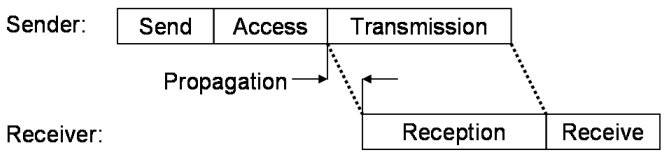


Fig. 1. Critical path

In both the mote and PC networks, the Send and Receive times are the delays incurred as the message is passed between the application and the MAC layer. These segments are mostly

nondeterministic due to system call overhead and kernel processing, and delay can reach hundreds of milliseconds. The Access time is the least deterministic segment in the critical path. It is the delay that occurs in the MAC layer while waiting for access to the communication channel. Depending on network congestion, access time can take longer than one second. The Transmission and Reception times are the delays incurred from transmitting and receiving a message over the physical layer, bit by bit. It can be as high as tens of milliseconds, however, it is mostly deterministic and depends on bit rate and message length. The Propagation time is the time the message takes to travel between the sender and receiver. This delay is highly deterministic and also the least significant; a message can travel up to 300 meters in less than a microsecond.

The mote-PC serial connection is slightly different. The Send and Receive times are similar to wireless communication, however Access time is noticeably different. Unlike wireless networks, in which nodes must compete with a potentially large number of neighbors for access to the channel, a serial connection is only shared by two devices. Furthermore, in duplex mode, simultaneous bidirectional communication is possible, so Access time will not be affected by congestion. However, there will be a delay if the receiver is not ready to accept data. To discover whether data can be sent, the sender sets the Request-To-Send (RTS) pin high on the serial port. If the receiver is able to accept data, it will set the Clear-To-Send (CTS) pin high, and data transfer can begin. Thus, the Access time is the delay that takes place from the time the RTS pin is set by the sender to the time the CTS pin is set by the receiver. The Transmission time starts when the data on the sender is moved in 16-byte chunks to the buffer on the UART, and ends after the data is transmitted bit-by-bit across the serial port to the receiver. Similar to wireless networks, the Propagation time is minimal. The UART on the receiver places the received bits into its own 16-byte buffer. When the buffer is almost full, or a timeout occurs, it sends an interrupt to the CPU, which notifies the serial driver, and the data is transferred to main memory where it is packaged and forwarded to the user-space application.

In addition to the error from message delay nondeterminism, synchronization accuracy is also affected by clock skew when a pair of nodes operate for extended periods of time without correcting their offset. For example, suppose at real-time t, nodes N_1 and N_2 exchange their current clock values at local times $N_1(t)$ and $N_2(t)$, respectively. At some later time, an event *e* occurs that is detected and timestamped by both nodes, and N_1 sends its timestamp $N_1(t_e)$ to N_2. If the clock rates on each node were equal, N_2 would simply be able to take the previously calculated offset and use it to transform the event timestamp $N_1(t_e)$ to the corresponding local time $N_2(N_1(t_e)) = N_1(t_e) + \phi_2^1(t)$. However, because of the clock skew, an attempt to convert $N_1(t_e)$ to the local timescale can potentially result in an error, especially if the interval between the last synchronization and the event was large.

Given the estimated clock offset and relative rate, a timescale transformation, which converts an event timestamp $N_j(t_e)$ from the timescale of node N_j to the timescale of N_i, can be defined as

$$N_i(N_j(t_e)) = N_i(t_s) + rr_j^i(t_s)[(N_j(t_e) - N_j(t_s)]$$

where $N_i(t_s)$ and $N_j(t_s)$ are the respective local times at which nodes N_i and N_j exchanged their most recent synchronization message, s.

III. Time Synchronization in HSNs

This section presents our methodology, implementation, and experimental results for time synchronization in heterogeneous sensor networks.

A. Synchronization Techniques

Our goal is to provide accurate time synchronization to reactive data fusion applications in HSNs. Doing so requires an understanding of how the various components in the network interact. We refer to the combination of these components as a *configuration*. Our configuration consists of Mica2 motes and Linux PCs. There are two physical networks, the B-MAC network formed by the motes and the 802.11 network containing the PCs. The link between the two is achieved by connecting a mote to a PC with a serial connection. This mote-PC configuration is chosen because it is representative of HSNs containing resource-constrained sensor nodes for monitoring the environment and resource-intensive computational nodes.

Mote Network: Existing protocols that minimize synchronization error can be classified as *sender-receiver*, in which one node synchronizes with another, or *receiver-receiver*, in which multiple nodes synchronize to a common event [15]. Several sender-receiver synchronization protocols [7], [8], [9] have been developed for the Berkeley Motes and similar small-scale devices, and provide microsecond accuracy. This is possible because often these embedded devices support operating systems that are tightly integrated with the radio stack, enabling timestamping directly at the network interface, and thus bypassing the majority of nondeterministic message delay in the critical path. For example, in RITS, sender-side synchronization error is essentially eliminated by taking the timestamp and inserting it into the message *after* the message has already begun transmission. On the receiver side, a timestamp is taken upon message reception, and the difference between these two timestamps estimate the clock offset. RITS was shown to synchronize a pair of nodes with an average error of 1.48 μs and an accumulated error of 0.5 μs per hop [9]. The accuracy of a receiver-receiver synchronization protocol such as RBS (discussed below) is comparable to sender-receiver synchronization in a mote network. However, it has greater associated communication overhead, which can shorten the lifetime of the network. Therefore, we selected RITS to synchronize the mote network in our HSN.

PC Network: Synchronization of PC networks has been studied extensively over the past four decades, however, popular sender-receiver algorithms such as the Network Time Protocol (NTP) [16] only provide millisecond accuracy. This is generally acceptable because PC users typically do not require greater synchronization precision for their applications. RBS is a popular receiver-receiver protocol that was found to achieve synchronization accuracy in the order of microseconds over 802.11 networks. Rather than synchronizing clocks to each other, participating nodes timestamp the arrival of a special message broadcast over the network. By exchanging these timestamps, neighboring nodes are able to maintain reference tables for timescale transformation. Receiver-receiver synchronization minimizes error by taking advantage of the broadcast channel found in most networks. Messages broadcast at the physical layer will arrive at a set of receivers within a tight time bound due to the almost negligible propagation time of sending an electromagnetic signal through air. This removes the send and access time nondeterminism from the critical path. The only remaining significant source of nondeterminism is due to the receive time, the majority of which can be removed by timestamping the arrival of the reference beacon in the kernel interrupt function. Because receiver-receiver synchronization provides greater accuracy in 802.11 PC networks, we selected RBS as our synchronization mechanism for our Linux PCs.

Mote-PC Connection: There are two ways the interface between the mote and PC networks can be modeled. In the first, the gateway mote acts as a network interface controller (NIC) for the PC it is connected to. This generally implies there is no separate critical path between the gateway mote and PC, but instead a unique critical path exists between the mote network and mote-PC pair. The second model places the critical path between the gateway mote and PC. We chose the latter because it gives us greater control over minimizing the nondeterministic message delay. We selected Elapsed Time on Arrival (ETA) [10], the underlying synchronization mechanism used in RITS. On the mote, a timestamp is taken upon transfer of a synchronization byte and inserted into the outgoing message. On the PC, a timestamp is taken immediately after the UART issues the interrupt, and the PC regards the difference between these two timestamps as the PC-mote offset, ϕ_{mote}^{pc}. The mote timestamp is taken at the latest time possible in the critical path, and the PC timestamp at the earliest. A timestamp is taken by the mote upon successful transmission over the serial connection of a synchronization byte and appended to the end of the message. Simultaneously, as the synchronization byte is received at the PC, a timestamp is taken in the serial driver at the kernel level. Serial communication bit rate between the mote and PC is 57600 baud, which approximately amounts to a transfer time of 139 microseconds per byte. However, the UART will not issue an interrupt to the CPU until its 16-byte buffer nears capacity or a timeout occurs. Because the synchronization message is six bytes, reception time in this case will consist of the transfer time of the entire message in addition to the timeout time and the time it takes to transfer the date from the UART buffer into main memory by the CPU. This time is compensated for by the receiver, and the clock offset between the two devices is determined as the difference between the PC receive time and the mote transmit time.

B. Clock Skew Compensation

Independent of synchronization protocol, there are several options for clock skew compensation. In some applications, event detection and synchronization always occur so close together in time that clock skew compensation is unnecessary. However, when it does become necessary, nodes must exchange timestamps periodically to ensure their clocks do not drift too far apart. In resource-constrained WSNs, this may be undesirable because it can result in high message overhead. To keep message overhead at a minimum, nodes can exchange synchronization messages less frequently and instead maintain a history of their neighbor's timestamps. Statistical techniques can then be used to produce an accurate estimate of clock offset at any time instant.

Linear Regression: A linear regression fits a line to a set of data points such that the square of the error between the line and each data point is minimized overall. By maintaining a history of *n* local-remote timestamp pairs $\{N_i(t_1), N_j(t_1)\}$, $\{N_i(t_2), N_j(t_2)\}$, ..., $\{N_i(t_n), N_j(t_n)\}$, node N_i can derive a linear relation $N_i(t) = \alpha + \beta N_j(t)$ and solve for the coefficients α and β. Here, β represents an estimation of $rr_j^i(t_k)$, and can be calculated by

$$\beta(t_k) = \frac{\sum_{k=1}^{n} (N_i(t_k) - \overline{N_i}(t_k))(N_j(t_k) - \overline{N_j}(t_k))}{\sum_{k=1}^{n} (N_j(t_k) - \overline{N_j}(t_k))^2}$$

where $\overline{N_i}(t_n)$ is the average of $\{N_i(t_1),...,N_i(t_n)\}$. One disadvantage of linear regression is that outliers have a strong impact on the result because data points are weighted by the square of their error to the fitted line. In some instances, however, these outliers can be removed beforehand. Another problem arises when attempting to improve the quality of the regression by increasing the number of data points. This can result in high memory overhead, especially in dense networks. However, it has been shown in [8] that sub-microsecond clock skew error can be achieved with as few as six timestamps in mote networks.

Exponential Averaging: Exponential averaging solves the problem of high memory overhead by keeping track of only the current relative rate and the most recent neighbor-local synchronization timestamp pair. When a new timestamp pair is available, the relative rate is updated by

$$\overline{rr_j^i}(t_k) = \alpha * \frac{N_i(t_k) - N_i(t_{k-1})}{N_j(t_k) - N_j(t_{k-1})} + (1 - \alpha) * \overline{rr_j^i}(t_{k-1}),$$

where α is a small weight constant (typically close to 0.1) that gives higher significance to the accumulated average. The value of α affects the convergence time. In addition, because each relative rate estimate is partially derived from its previous value, there will be a longer convergence time before an accurate estimate is reached. This can be reduced by

providing the algorithm with an initial relative rate, determined experimentally.

Phase-Locked Loop: Phase-locked loops (PLL) is a mechanism for clock skew compensation used in NTP [17]. The PLL compares the ratio of a current local-remote timestamp pair with the current estimate of relative rate. The PLL then adjusts the estimate by the sum of a value proportional to the difference and a value proportional to the integral of the difference. PLLs generally have a longer convergence time than linear regression and exponential averaging, but have low memory overhead. A diagram of our PLL implementation is illustrated in figure 2. There are three main components. The Phase Detector calculates the relative rate between two nodes and compares this with the output of the PLL, which is the previous estimate of the relative rate. The difference between these two values is the phase error, which is passed to the second-order Digital Loop Filter. Because we expect there to be some amount of phase error, we choose a filter with an integrator, which allows the PLL to compensate for it such that there is no remaining steady-state phase error. To implement this behavior in software, a digital accumulator is used, and is represented by $y(t) = (K_1 + K_2)u(t) - 10K_2K_1u(t-1) + 10K_2y(t-1)$. The resulting static phase error is passed to the Digitally Controlled Oscillator (DCO). The DCO sums the previous phase error with the previous output, which produces the current estimate of relative clock rate, and is fed back into the PLL. Techniques for selecting the gains are presented in [17].

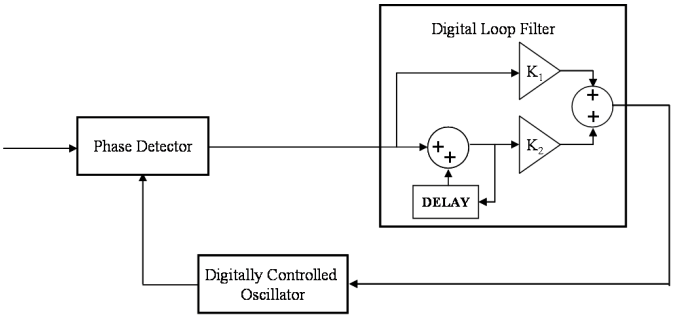


Fig. 2. Phase-locked loop

C. Experimental Setup

Our testbed consists of seven Crossbow Mica2 motes and four stationary ActiveMedia Pioneer robots with embedded Linux PCs, as illustrated in figure 3. In addition we employ an OpenBrick-E Linux server to transmit RBS beacons.

A PC-based reference broadcast node transmits a reference beacon containing a sequence number once every ten seconds. The arrival of these messages are timestamped in the kernel and stored in a reference table. Simultaneously, a designated mote broadcasts event beacons, once every four seconds. Six hundred event beacons are broadcast per experiment. The motes timestamp the arrival of the event beacon, and place the timestamp into a message. The message is routed over three hops in the mote network to the mote-PC gateways using RITS

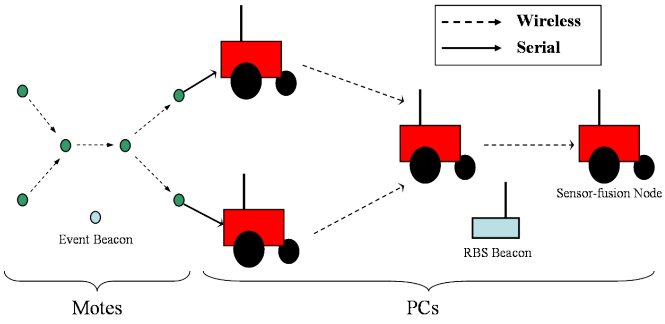


Fig. 3. Our sensor network testbed. Arrows indicate communication flow.

to convert the timestamp to the local timescale with each hop. The message is next transferred from the mote network to PC network over the mote-PC serial connection, and the event timestamp is again converted to the timescale of the gateway PC. The gateway PCs forward the message two additional hops to the sensor-fusion node.

In the PC network, the event message includes additional fields, specifically the most recent reference broadcast arrival timestamp and sequence number. These allowed us to reduce the message overhead of RBS. PCs receiving these messages are able to convert the event timestamp into their local timescale by comparing the sender PC reference broadcast timestamp with their own, adjusting the event timestamp by the offset, and compensating for clock skew. The experiment was repeated using the different clock skew compensation techniques described in Section II.

The implementation used in these experiments was bundled into a PC-based time synchronization service for reactive data fusion applications in HSNs. The service accepts timestamped event messages on a specific port, converts the embedded timestamp to the local timescale, and forwards the message toward the sensor-fusion node. The mote implementation uses the TimeStamping interface, provided with the TinyOS distribution [18].

D. Experimental Results

Mote-PC Synchronization: To evaluate synchronization accuracy between the mote and the PC, GPIO pins on the mote and PC were connected to an oscilloscope, and set high upon timestamping. The resulting output signals were captured and measured. The test was performed over 100 synchronizations, and resulting error was $7.32\mu s$ on average. The results are displayed in figure 4. The majority of the error is due to nondeterministic message delay resulting from jitter, both in the UART and the CPU.

HSN Synchronization: Figure 5 summarizes the synchronization error for each type of clock skew compensation technique. The results were obtained as follows: Two nodes, N_1 and N_2, simultaneously timestamp the occurrence of an event, such as a reference beacon. These timestamps are then transformed to the timescale of node N_3, and the absolute value of their difference, $|N_3(N_1(t_e)) - N_3(N_2(t_e))|$, represents the error in the timescale transformation. In the case

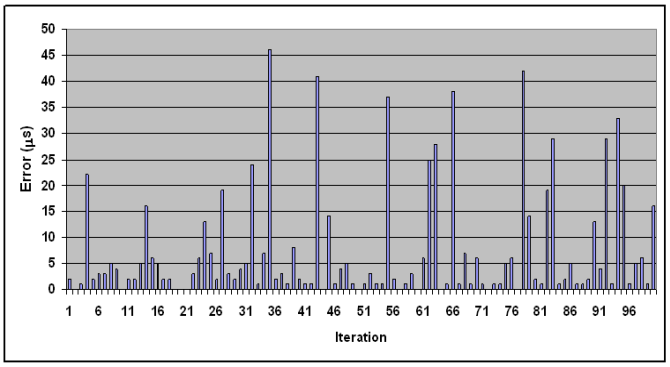


Fig. 4. Mote-PC error.

when no clock skew compensation was used, the average difference between the source timestamps was $85.66\mu s$, with a maximum of $689\mu s$. Linear regression was implemented with a history size of 8 local-remote timestamp pairs for each neighbor, and the relative rate was initialized to 1. As expected, there was a notable improvement in synchronization accuracy, with an average of $6.89\mu s$ error, with a maximum of $55\mu s$. Using exponential averaging gives error similar to linear regression. For exponential averaging, we chose a value of 0.10 for α, and initialized the average relative rate to 1. The average error recorded was $7.45\mu s$, with a maximum of $47\mu s$. The average synchronization error using phase-locked loops was $7.85\mu s$, with a maximum of $574\mu s$. For the digital loop filter, we used gains of $K_1 = 0.1$ and $K_2 = 0.09$.

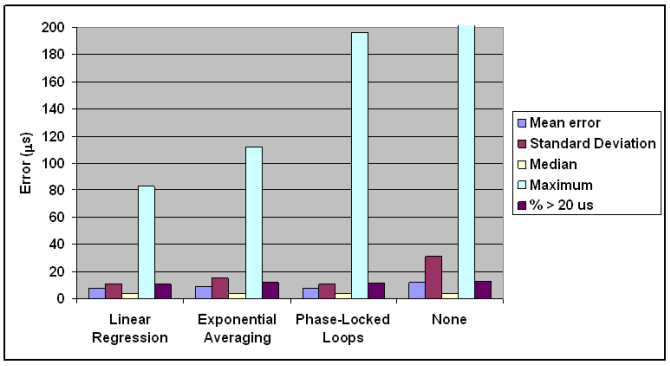


Fig. 5. HSN synchronization error.

Each experiment was run under both high and low network congestion and I/O load conditions. To simulate a high network load, an extra mote and PC (not pictured in figure 3) were introduced, each broadcasting messages of random size every 100 milliseconds. The results for each type of clock skew compensation in the presence of network congestion and I/O load show that synchronization accuracy is not overtly affected by these strenuous conditions in the network.

The majority of synchronization error in the HSN comes from RBS synchronization in the PC network. This was principally caused by a small number of synchronization attempts in which prolonged operating system interrupt operations occurred. Although these were difficult to avoid, they did not occur frequently, and the worst-case synchronization error for each type of clock skew compensation technique was sufficient for most kinds of HSN reactive data fusion applications.

IV. Time Synchronization for Multi-Modal Tracking

In this section, we briefly describe the our multi-modal target tracking system. We also present the software architecture which includes time-synchronization methods introduced in the paper. Finally, we present experimental results that demonstrate the improvement on the tracking performance.

Our multi-modal target tracking system is shown in Figure 6. We employ 5 audio sensors in a $36.5 \times 15 ft$ grid and 3 video sensors deployed on either side of a road. The objective of the system is to detect and track vehicles and people using both the audio and video data. The detection and tracking results are available at the sensor fusion center, which can then be made available to other applications.

A. Audio Sensing

Beamforming methods have been successfully applied to detect single or even multiple acoustic sources in noisy and reverberant areas. Beamforming takes advantage of the fact that the distance from the source to each microphone in a beamforming array is different, which means that the signals recorded by each microphone will be phase-shifted replicas of each other. The amount of phase-shift at each microphone in the array is dependent on the microphone arrangement and location of the source. A typical delay-and-sum beamformer divides the sensing region into beams. For each beam, assuming the source is located in that direction, the microphone signals are delayed according to the phase-shift and summed together to get a composite signal. The square-sum of the composite signal is the beam energy. Signal energies are computed for each of the beams, called the beamform and the maximum energy beam indicates the direction of the acoustic source.

The data-flow diagram of the beamformer used in our system is shown in Figure 7. The amplified microphone signal is sampled at 1MHz to provide high resolution for the delay lines, required by the closely placed microphones. The raw signal is filtered to remove unwanted noise components and provide a band limited signal for down sampling in a later stage. The signal is then fed to a tapped delay line (TDL), which has M different outputs to provide the required delays for each of the M beams. The signal is down sampled and M beams are formed by adding the four delayed signals together. From the data streams blocks are formed and an FFT is computed for each block. A frequency selection component is used to select the frequencies of interest. The selected signal is then summed to compute the block power value μ_i, which is then smoothed by exponential averaging into beam power λ_i:

$$\lambda_i(k) = \alpha\lambda_i(k-1) + (1-\alpha)\mu_i.$$

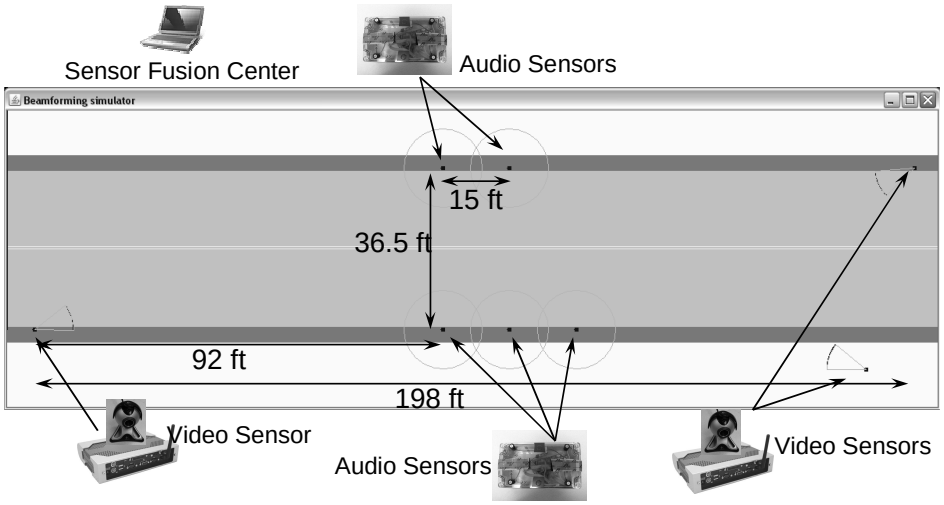


Fig. 6. Experimental setup

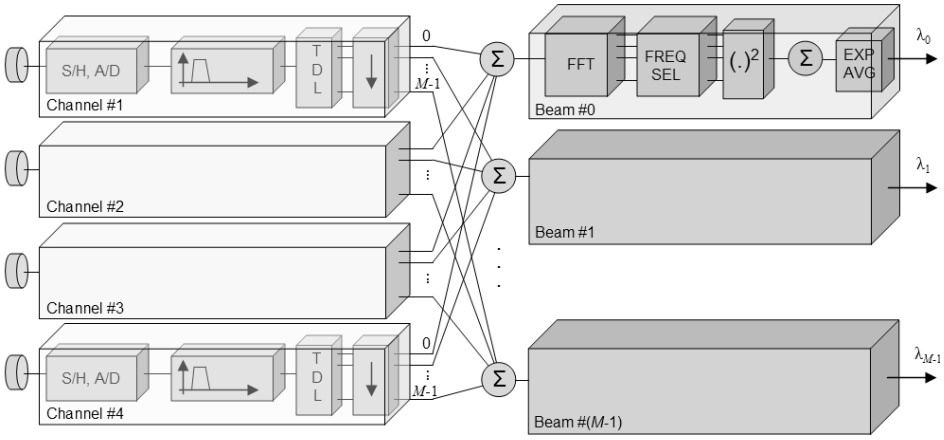


Fig. 7. Data-flow diagram of the audio beamforming algorithm

As an example, Figure 8 shows the normalized beam power of the audio signal generated by a person speaking a few meters from the sensor as measured on a street. The direction of the speaker is clearly detected, but it is also visible that the detection is not very sharp (due to the small number of microphones in the array).

The audio sensing is implemented on a sensor node with an array of four microphones shown in Figure 9. The sensor node is based on a MicaZ mote and on an onboard FPGA chip that is used to implement the beamformer [19].

B. Video Sensing

We use a moving object detection algorithm for video sensing. The dataflow in Figure 10 illustrates the algorithm and its components. The first step of moving object detection is background-foreground segmentation of the currently captured frame (I_t) from the camera. We use the algorithm described in [20] for background-foreground segmentation which is based on an adaptive background mixture model for real-time tracking. The mixture method models each background pixel by a mixture of K Gaussian distributions.

The foreground image (F_t) from the segmentation process is passed through a median filter which reduces speckle noise present in the foreground and smooths the foreground image. The foreground then goes through opening and closing morphological operations. Opening removes small features present in the foreground, while the closing operation fills small holes. Finally, the *enhanced* foreground image undergoes image segmentation to find connected components. Each of the connected components is enclosed by a rectangular bounding box which represent a moving object (O_t).

We have implemented the moving object detection algorithm in OpenCV by Intel [21]. In our application, OpenBrick-E Linux servers are running the video-based detection at 8 frames-per-second using video with 320×240 pixel resolution.

C. Sensor Fusion

The objective of our sensor fusion algorithm is to combine the detection values from both audio and video sensors. The acoustic sensors provide beam power values λ_j, $j = 0, 1, ..., M$, λ_j corresponding to directions $\phi_j = j\frac{2\pi}{M}$, where M is the resolution of the sensor. The video sensors provide

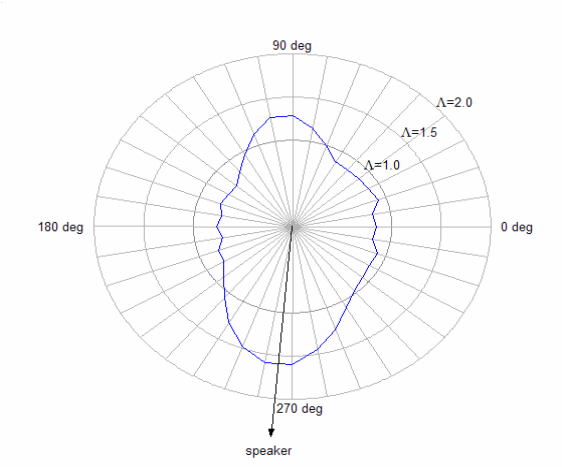


Fig. 8. The normalized power of an audio signal

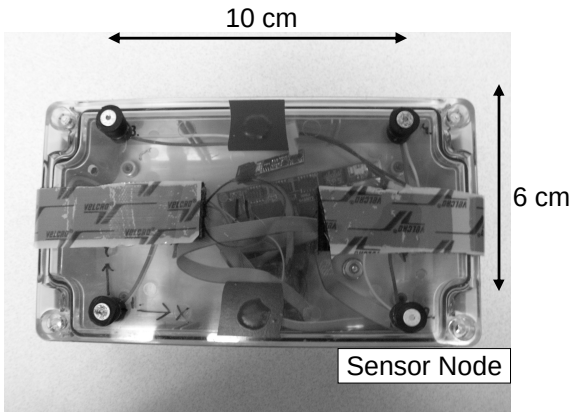


Fig. 9. Audio sensor node

detection angle ranges $[\alpha_l, \beta_l]$ for each detected object l, $l = 0, 1, ..., L$, where L is the number of perceived objects by the given sensor.

In order to perform sensor fusion, we define a detection function $\Xi(\phi)$ that formalizes the object detection for both types of sensors. For the acoustic sensors, we have

$$\Xi(\phi) = \lambda_j, j : \ |\phi_j - \phi| \bmod 2\pi < \frac{\pi}{M}$$

while for the video sensors

$$\Xi(\phi) = \begin{cases} 1 & \text{if } \alpha_l < \phi < \beta_l, \text{ for any } l, l = 0, 1, ..., L \\ 0 & \text{otherwise} \end{cases}.$$

Using the detection functions and the known locations and orientations of the sensors, the proposed algorithm computes a consistency function defined over the search space (2-dim

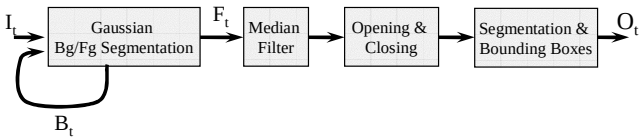


Fig. 10. Data-flow diagram of real-time moving object detection

plane in our case). Let K be the number of sensors placed on the plane at positions (x_k, y_k), $k = 1, .., K$. The search space is divided into N rectangular regions with center points (X_i, Y_i) and side lengths of Δx_i and Δy_i, $i = 1, ..., N$ as illustrated in Figure 11. The consistency function for each

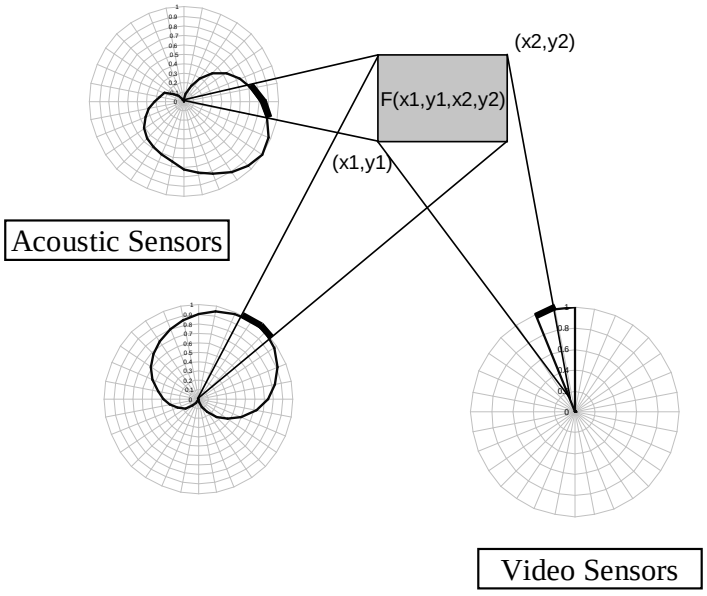


Fig. 11. Multi-modal sensor fusion. The sensing region is divided into rectangular regions and the consistency function is calculated for each region using acoustic and video data.

region i is defined as

$$C_i = \Sigma_{k=1}^{K} w^{(k,i)} \Xi^{(k,i)}$$

where $w^{(k,i)}$ is a weighting factor discussed later and the $\Xi^{(k,i)}$ value is the contribution of the sensor k in the i^{th} rectangular region. The value of the consistency function for a region is a measure of the likelihood that an object is present at that region. Therefore, our tracking algorithm estimates the position of the object (or multiple objects) by computing the peaks of the consistency function and returning the corresponding coordinates.

D. System Architecture

Figure 12 shows the audio-visual sensor fusion system architecture. The audio sensors are periodically sending their timestamped detection functions to the base station. The base station for each of the audio nodes and the video sensor nodes send their timestamped detection functions to the sensor fusion node through a time-synchronization daemon. The daemon uses the timescale transformation approach presented in the previous section for converting the timestamps to the local clock and stores the values of the detection function to appropriate buffers, one for each sensor in the system.

A sensor fusion scheduler triggers periodically and generates a timestamp. The trigger is used to retrieve the detection function values from the sensor buffers which are closest to the generated timestamp. The retrieved detection function

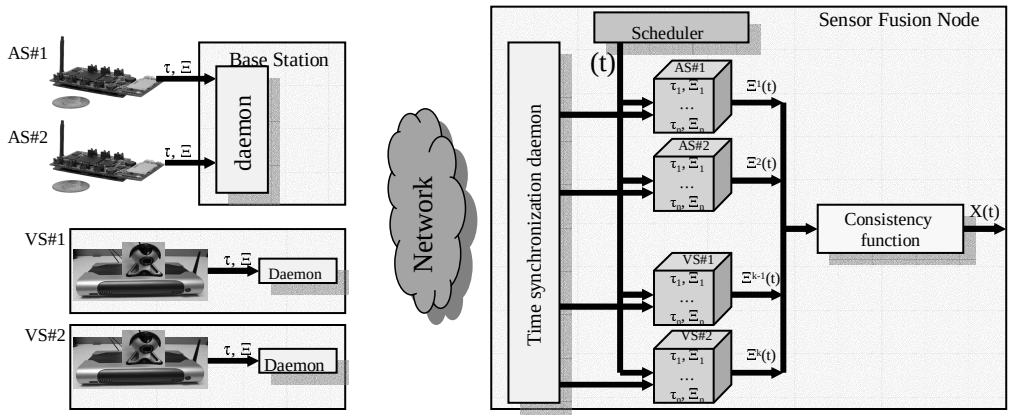


Fig. 12. Multi-modal tracking system architecture

values are then used to compute the consistency function for the sensing region. The maxima of the consistency function indicate the presence of a target. Note that the scheduler in this architecture decouples the tracking rate, at which the consistency function is computed, and the audio/video sensing rates.

E. Experimental Results

We have performed a series of experiments tracking people and/or vehicles and we present a few representative results. We show the consistency function in the sensing region (dark regions indicate higher value) for three scenarios. For each scenario, we plot the consistency function obtained only by the audio sensors and by both audio and video sensors to illustrate the advantages of multi-modal tracking. Figure 13 shows the results for experiment where a speaker was inside the network. Figure 14 shows results for an experiment with a speaker outside the network. Figure 15 shows the results for multiple speakers. The results demonstrate that using the video sensors results in a drastic decrease in the variance of the estimated position (the regions with high-consistency values are much smaller).

V. Conclusion

We have developed a methodology for accurate time synchronization in heterogeneous sensor networks. The synchronization methodology has been implemented as a network service and used in a multi-modal tracking application for performing fusion of audio and video data collected from heterogeneous sensor nodes. In our future work, we intend to explore more advanced tracking algorithms that use recursive and/or distributed methods as well as consider mobile sensor nodes in the network.

Acknowledgements This work was supported in part by ARO MURI grant W911NF-06-1-0076, NSF CAREER award CNS-0347440, and a Vanderbilt University Discovery Grant.

References

[1] G. Pingali, G. Tunali, and I. Carlbom, "Audio-visual tracking for natural interactivity," in *ACM Multimedia*, 1999.

[2] M. Yarvis, N. Kushalnagar, H. Singh, A. Rangarajan, Y. Liu, and S. Singh, "Exploiting heterogeneity in sensor networks," in *IEEE INFOCOM*, 2005.

[3] E. Duarte-Melo and M. Liu, "Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks," in *IEEE Globecom*, 2002.

[4] L. Lazos, R. Poovendran, and J. A. Ritcey, "Probabilistic detection of mobile targets in heterogeneous sensor networks," in *IPSN*, 2007.

[5] K. Romer, "Time synchronization in ad hoc networks," in *ACM Symposium on Mobile Ad-Hoc Networking and Computing*, 2001.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[7] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *ACM SenSys*, 2003.

[8] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *ACM SenSys*, 2004.

[9] J. Sallai, B. Kusy, A. Ledeczi, and P. Dutta, "On the scalability of routing integrated time synchronization," in *Third European Workshop on Wireless Sensor Networks (EWSN)*, 2006.

[10] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: A simple and versatile primitive for time synchronization services," *International Journal of Ad hoc and Ubiquitous Computing*, vol. 2, no. 1, January 2006.

[11] L. Girod, V. Bychkovsky, J. Elson, and D. Estrin, "Locating tiny sensors in time and space: A case study," in *ICCD*, 2002.

[12] B. H. Yoshimi and G. S. Pingali, "A multimodal speaker detection and tracking system for teleconferencing," in *ACM Multimedia*, 2002.

[13] D. N. Zotkin, R. Duraiswami, H. Nanda, and L. S. Davis, "Multimodal tracking for smart videoconferencing," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2001.

[14] J. Elson and K. Romer, "Wireless sensor networks: A new regime for time synchronization," in *Proceedings of the First Workshop on Hot Topics in Networks*, 2002.

[15] K. Romer, P. Blum, and L. Meier, "Time synchronization and calibration in wireless sensor networks," in *Wireless Sensor Networks*, I. Stojmenovic, Ed. Wiley and Sons, 2005.

[16] D. L. Mills, "Internet time synchronization: The network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, 1991.

[17] ——, "Modelling and analysis of computer network clocks," Electrical Engineering Department, University of Delaware, Tech. Rep. 92-5-2, 1992.

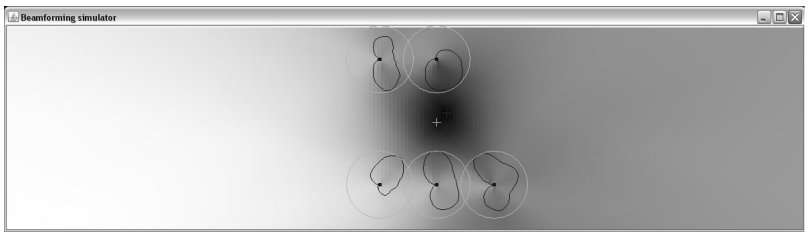
[18] P. Levis and et al., "The emergence of networking abstractions and techniques in tinyos," in *NSDI*, 2004.

[19] P. Volgyesi and et al., "Shooter localization and weapon classification with soldier-wearable networked sensors," in *Mobisys*, 2007.

[20] P. KaewTraKulPong and R. B. Jeremy, "An improved adaptive background mixture model for realtime tracking with shadow detection," in *2nd European Workshop on Advanced Video Based Surveillance Systems (AVBS)*, 2001.

[21] Open source computer vision library. Intel. [Online]. Available: http://www.intel.com/technology/computing/opencv/

without video sensors



with video sensors

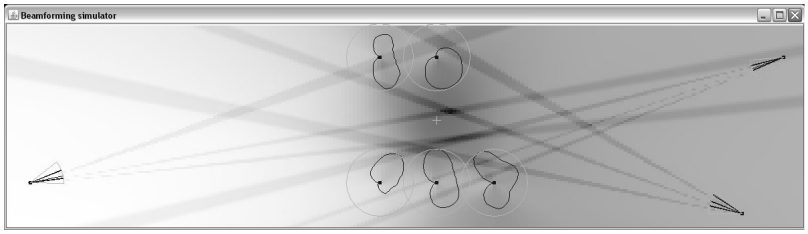
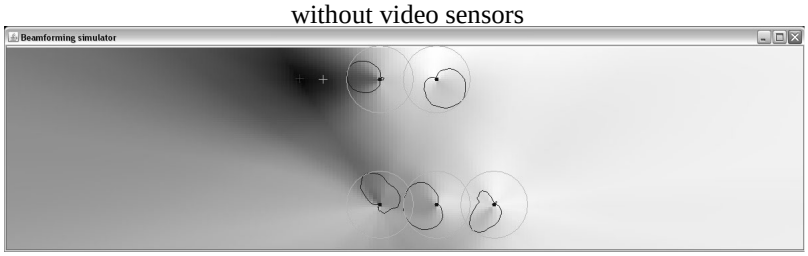


Fig. 13. Consistency function for a speaker inside the network

without video sensors



with video sensors

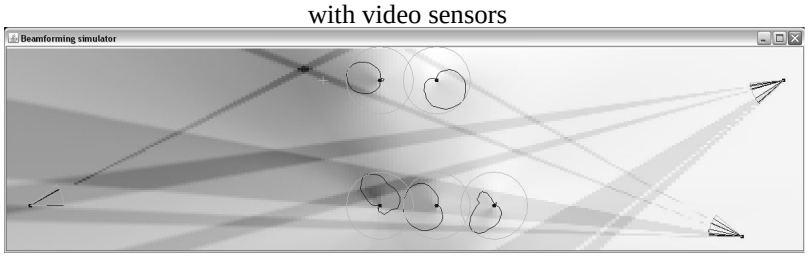
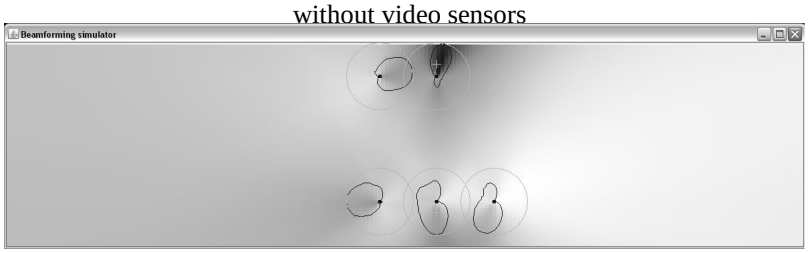


Fig. 14. Consistency function for a speaker outside the network

without video sensors



with video sensors

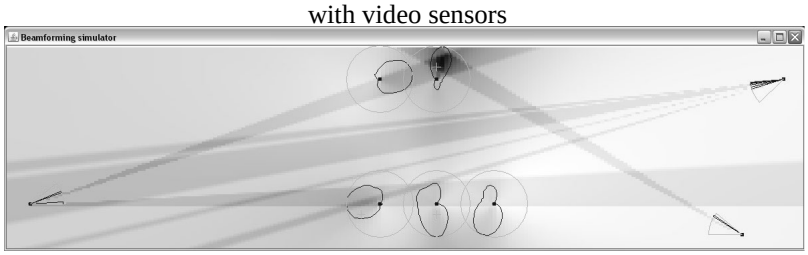


Fig. 15. Consistency function for multiple speakers