A Novel Approach to Multi-Sensor Data Synchronization Using Mobile Phones

Article *in* International Journal of Autonomous and Adaptive Communications Systems · September 2010

DOI: 10.1145/2221924.2221956

3 authors:

Jonas Wåhslén
KTH Royal Institute of Technology
7 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Thomas Lindh
KTH Royal Institute of Technology
22 PUBLICATIONS **91** CITATIONS
SEE PROFILE

Martin Eriksson
KTH Royal Institute of Technology
18 PUBLICATIONS **246** CITATIONS
SEE PROFILE

A Novel Approach to Multi-Sensor Data Synchronization Using Mobile Phones

Jonas Wåhslén
School of Technology and Health,
KTH, Stockholm, Sweden
Email: jwi@sth.kth.se

Thomas Lindh
School of Technology and Health,
KTH, Stockholm, Sweden
Email: thomas.lindh@sth.kth.se

Martin Eriksson
School of Technology and Health,
KTH, Stockholm, Sweden
Email: eriksson@sth.kth.se

ABSTRACT
This paper presents a new algorithm for synchronization of data from multiple sensors arriving to a mobile phone's Bluetooth interface with possibly unknown and different sampling frequencies. A system that provides feedback signals to an athlete is one example where it is crucial to synchronize data from several wireless sensors; especially sensor nodes use different and unknown sampling rates.

Keywords
Multi-sensor data synchronization, data fusion, wireless sensor networks, Bluetooth.

1. INTRODUCTION
Wireless body sensor networks have emerged as a feasible communication infrastructure in e.g. health monitoring and sports. Samples of data from a number of wireless sensors, on or near the body, can be used for proper feedback to an athlete during training or to patients for motor training. Decisions based on fusion of data from several sensors require that correct samples from different sensors are associated with each other. It can be achieved if the individual sensors are synchronized in time, which often is an unrealistic requirement. This paper addresses the problem to synchronize samples from several sensors arriving to a mobile phone without clock synchronization and without prior knowledge of sampling frequencies. We describe different approaches and present an algorithm with a maximum synchronization error of 1-2 samples. The synchronization problem is presented in Section 3, followed by analysis of different approaches and our algorithm in Section 4. Test results are analyzed in Section 5 and finally the conclusions in Section 6.

2. RELATED WORK
Advances in the field of body sensor networks have enabled monitoring of kinetic and kinematic performance unobtrusively outside the laboratory ([3], [4], [5], [9]). Today commercial systems that can measure physiological parameters, such as heart rate ([1], [15]) primarily for physical training, already exist.
This paper discusses methods to minimize errors in synchronizing data sent from several sensors to a mobile phone using Bluetooth. There are several protocols to synchronize wireless sensor nodes some described in ([6], [8], [10]). One approach to data synchronization is to timestamp packets when they arrive to the mobile phone [7]. A major drawback is the variable time error due to waiting time in buffers before the timestamp is inserted. These inaccurate timestamps mean that synchronization of data from several sensors fails. Another solution is to use a time synchronization protocol between the wireless sensor nodes and let the sending sensors insert timestamps. This requires access to the Bluetooth clock [12] or permission to modify the Bluetooth stack [2]. However, the algorithm proposed in this paper makes it possible to use any commercially available sensor without requiring access to the system clock or other system functions. Examples of sensors in sports applications are accelerometers, EMGs, EEGs, GPS devices, strain-gauges and pulse-oximeters. WiTilt v3.2 accelerometers from SparkFun [14] have been used in our tests.

Conference'10, Month 1–2, 2010, City, State, Country.

3. THE SYNCHRONIZATION PROBLEM
Figure 1 shows a mobile phone connected to multiple wireless sensor nodes. The embedded software uses socket APIs to receive data from the sensor nodes. Data samples from the sensors pass two buffers; the outgoing data buffer on the sensor and the incoming data buffer on the mobile phone. An application using sockets is not aware of how long time every sample spends in the buffers before read by the socket. The application can insert a timestamp when the packet is read from the incoming data buffer. This stochastic time error, $\Delta t_{bufferror}$, between the actual sample time and when the packet is read from the buffer, means that synchronization of packets from different sensors based on timestamps on the mobile phone is unreliable. In our analysis we treat the two buffers in Figure 1 as one virtual buffer, where $\Delta t_{bufferror}$ are caused by delays in one or both of the buffers.

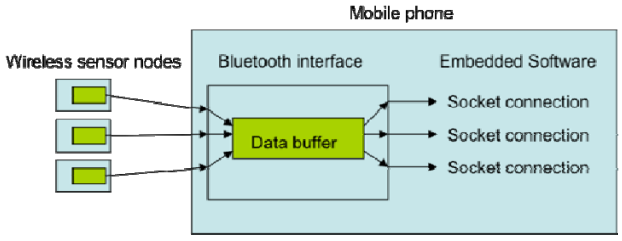

Figure 1. A schematic view of how a mobile phone handles data input from wireless sensor nodes. Data from sensors pass two buffers; the outgoing data buffer on the sensor and the incoming data buffer on the mobile phone.

One way of getting more accurate timestamps is to use libpcap [11] or similar low level APIs that enable access to packets sent from wireless sensors when they arrive, before they are buffered. However, this requires access to the internal operating system, normally not available on mobile platforms for users. Our algorithm for synchronizing data from multiple wireless sensors does not require access to the link or radio communication layers of the mobile phone. Access to the socket interface is sufficient.

4. MULTI-SENSOR DATA SYNCHRONIZATION

In this section we analyze single-thread and multi-thread solutions and present a new algorithm for minimizing errors in synchronization of data from several sensors received by a mobile phone.

4.1 Single-Thread Solutions

One approach is to let a single-thread handle all connections to the wireless sensor nodes and use a round-robin algorithm to collect data from the individual sensors. This works if every sensor transmits data with the same frequency and starts simultaneously. This approach has two drawbacks. Firstly, packet loss from a specific sensor means that the round-robin algorithm will be blocked until a new packet arrives from that sensor. A single packet drop will lead to an error in synchronizing data from the sensors. A sequence number in every packet can mitigate the problem. However, the round-robin algorithm needs to be recalibrated, and the algorithm is blocked every time packets are dropped. Secondly, this approach does not work for sensors with different sending frequencies. The same problem will occur if the wireless sensor nodes internal clocks are drifting.

4.2 Multi-Thread Solutions

Another approach to data synchronization of multiple wireless sensor nodes is to use separate threads for each socket connection. A round-robin algorithm cannot be used in this case. Instead every data packet needs to be timestamped upon arrival. However, log files of data received by a mobile phone from multiple sensors show varying time errors due to the variable waiting time in the incoming buffer. Data synchronization means that packet p_i from sensor S_A is associated with packet p_j from sensor S_B, if the timestamps are in the same time interval. The test results show that packets synchronized based on the receiving timestamps may have synchronization errors up to as much as 50 ms for 100Hz sampling frequency. This could mean that a packet p_i sent from sensor S_A is perceived by the mobile phone as being sent at approximately the same time as packet p_j from sensor S_B in fact should be associated with a packet sent 50ms earlier or later (sequence number p_{j-5} or p_{j+5}). The error increases as the number of sensors connected to the mobile phone increases.

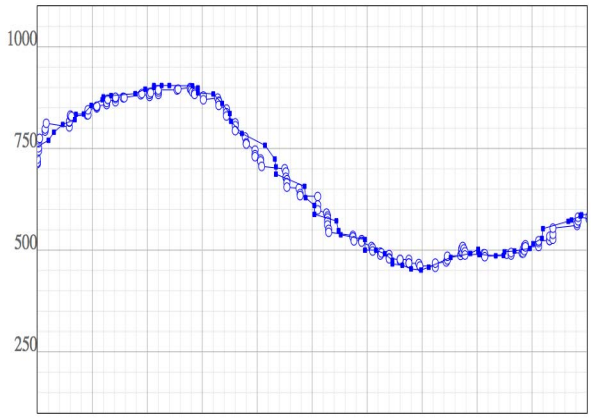

Figure 2. Accelerometer data received during 60 ms from two sensors sampled at 300Hz (circles) and 200 Hz (squares). The y-axis shows the raw uncalibrated accelerometer values. The x-axis shows time between 0ms and 60ms.

In our tests two FireFly accelerometer sensors are configured in the same way and tied together in order to sense the same acceleration. The sampling rates are 200Hz and 300Hz respectively. Figure 2 shows the accelerometer sample data during a 60 millisecond test when the two accelerometers (fixed together) are moved. The circles represent samples at 300Hz and the squares samples at 200 Hz. The samples are expected to have equal distances in time for the respective sensor. However, the samples are not evenly distributed but appear in lumps as seen in Figure 2. The main reason is the differences between the actual sampling time and the timestamp set when the packet is read from the buffer. Another minor contribution to the unevenly spread timestamps is the resolution using the function System.currentTimeMillis in JavaME. Our experiences indicate a precision in the order of 1ms.

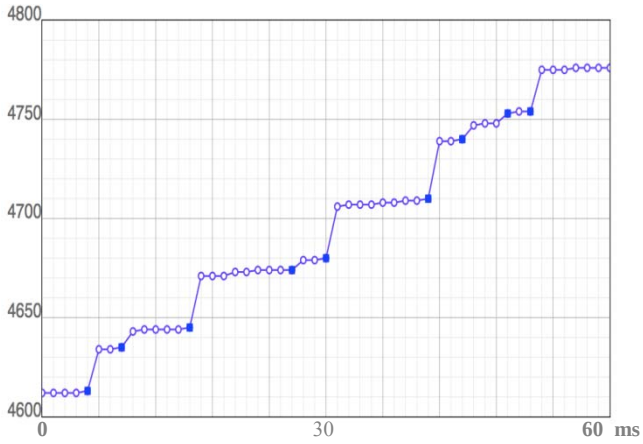

Figure 3. Accelerometer data and timestamps from a sensor connected to the mobile phone during 60ms. The last packets received before a context switch (to next thread) is marked with a filled square. The y-axis shows the raw uncalibrated accelerometer values. The x-axis shows time between 0ms and 60ms.

Figure 3 shows accelerometer data and timestamps for every received packet. The sensor node is sending 300 samples per second and we should expect a linear curve. The reason for the jumps in the curve is that multiple socket threads are running on the mobile phone. Some packets may be buffered for a longer time, before the thread starts processing them. Every timestamp will have a $\Delta t_{bufferror}$ added to the actual sample time that data packet arrived to the mobile phone. This $\Delta t_{bufferror}$ varies depending on the scheduling between the master and the slaves in the Bluetooth network and the thread status when data packet is sampled. The multi-thread approach is an improvement compared to the single-thread solution. It can handle sensor nodes with different sampling rates and dropped packets will not force the algorithm to wait. Yet, using timestamps on the mobile phone to synchronize data from several sensors will lead to errors due to the delay in the buffer ($\Delta t_{bufferror}$).

4.3 A New Approach

The error in data synchronization in the previous multi-thread approach is caused by the random waiting time in the buffers ($\Delta t_{bufferror}$). The problem is to eliminate or minimize this error to an upper limit. Our approach is to identify when the incoming buffer is empty. Timestamps that are inserted when a packet is picked from an empty incoming buffer has zero or a low upper limit for the waiting time (further explained below). These timestamps that are assumed to have the lowest $\Delta t_{bufferror}$ are used in

linear regression to estimate the timestamps for all received packets. To obtain this we define a new synchronization thread as seen in Figure 4. Incoming packets from sensor A, B and C are handled by three separate threads that send the sequence number, sensor ID and a timestamp to the new synchronization thread. A_i represents packet sequence number *i* from sensor A. The queue in the synchronization thread can be used to identify the last packet read by a thread before a context switch to serve a new thread occurs (bold text in Figure 4).

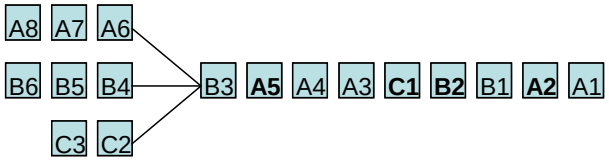

Figure 4. Three sensor nodes (A, B and C) send their sensor ID and sequence number for each received packet to the synchronization thread. Packets in bold text represent the last packet read by a thread before a context switch to serve another thread.

We assume that the mobile platform is capable to empty the buffer of incoming packets each time a thread is running. Our experiences show that this is a realistic and reasonable assumption for today's mobile platforms. The Sony Ericsson mobile phone K800i [13] has been used to implement, test and evaluate our algorithm (see Section 5).

A thread can be in three different modes; running, ready or suspended. Running mode means that a thread currently is being executed on the mobile phones CPU, ready mode means that it is waiting in a queue to be executed, and suspended mode means that the thread is waiting for an I/O device. A thread that is waiting for packets to read from a socket connection is in suspended mode. Once the I/O device has data available on the socket connection, the suspended thread will switch to ready mode. When the thread has finished reading available data, it will return to suspended mode until incoming data is available. A thread that continuously reads data and empty the buffer will return to suspended mode when there are no more data to read. This last packet, which is read before the thread is suspended and the operating system switches to a new thread, will have the smallest $\Delta t_{bufferror}$. These packets in each run of a thread can be identified in the synchronization thread (bold text in Figure 4). The packets marked with a square in Figure 3 have been located using this algorithm.

Figure 5 shows a packet stream from a wireless sensor node and two examples of a thread in running mode. A thread in running mode will read all available packets from the incoming buffer (the packets with the same colour as the thread in Figure 5). The timestamp for the packet is set when a thread reads the packet. The difference between the actual arrival time to the buffer and the timestamp put when the thread reads the packet is $\Delta t_{bufferror}$. The last packet's $\Delta t_{bufferror}$ will depend on when the context switch occurs.

Figure 5 illustrate how the time error occurs and its possible variation. The last packet read by the thread in running mode to the right in Figure 5 will have a small $\Delta t_{bufferror}$. The reason is that the last packet processed by the thread arrives just before the thread is suspended. The last packet read by the thread to the left in Figure 5 will have a larger error. In this case a packet is arriving to the incoming buffer but the entire packet has not been received, and is therefore not available. There are two alternatives. Firstly, the thread will be suspended since the previous packet already has been processed and there is no packet available in the buffer. Ones the entire packet has arrived to the buffer it will be read by the thread next time it is in running mode (to the right in Figure 5). Secondly, if the thread still is in running mode when the packet becomes available in the buffer, the thread will also read this packet. Consequently, the previous packet will no longer be the last packet processed by the thread before it is suspended. This also shows that the maximum $\Delta t_{bufferror}$ is lower than one sample time, $1/f_s$, where f_s is the sampling frequency.

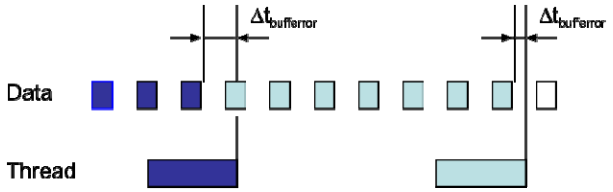

Figure 5. Two examples of a thread in running mode reading packets from the incoming buffer. The packets read by the thread have the same colour as the thread.

5. TEST RESULTS AND ANALYSIS

Our algorithm has been applied to the same data set from the two accelerometers shown in Figure 2. The last read packet in each run of the threads has been identified in the synchronization thread (as illustrated in Figure 3 and Figure 4). Linear regression is used to calculate the new corrected timestamps for all packets from both sensors. The results can be seen in Figure 6 below. Comparing Figure 2 and Figure 6 shows the difference. We have found that the synchronization error is 1-2 samples, which means maximum 10-20 ms at 100Hz sampling rate. This is close to the theoretical maximum of 1 sample (Section 4.3).

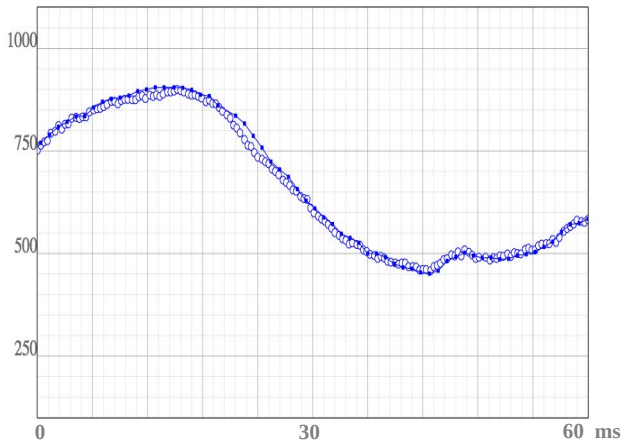

Figure 6. Accelerometer data received by a mobile phone during 60ms from two sensors after applying the synchronization algorithm. One sensor samples at 300Hz (circles) and the other samples at 200Hz (squares). The y-axis shows the raw uncalibrated accelerometer values. The x-axis shows time between 0ms and 60ms.

In order to compare the performance of our proposed algorithm (Section 4.3) with a traditional multi-thread algorithm (Section 4.2) linear regression was also applied to the original data set in Figure 2. The result can be seen in Figure 7. The offset and phase shift between the curves from the two sensors in Figure 7 is due to the error in synchronization of packets between the two sensors

caused by the $\Delta t_{bufferror}$ explained in the previous section. The proposed algorithm for minimizing $\Delta t_{bufferror}$ is valid under the assumption that the buffers do not overflow. For the platform we have used buffer overflow occurs at sampling frequencies above 500Hz with five connected sensor nodes.

Our analysis shows that a single-thread solution has severe disadvantages compared to a multi-threaded approach. A single thread cannot manage packet losses and unknown and different sampling frequencies. However, a traditional multithreaded solution has obvious problems in synchronizing packets from different sensors based on timestamps by a mobile phone. The difference between the actual sample time to the incoming buffer and the timestamp has the effect that a packet from one sensor is synchronized with a packet from another sensor that arrived several sample times earlier or later. The algorithm presented in this paper minimizes the error $\Delta t_{bufferror}$. The main technique is to detect when a switch between two threads occurs and identify the last read packet, which has the smallest $\Delta t_{bufferror}$. Using linear regression a maximum synchronization error of 1-2 samples is obtained.

In our analysis we have treated the two buffers, the outgoing buffer on the sensors and the incoming buffer on the mobile phone as one virtual buffer. Using a Bluetooth sniffer we have noticed that in the majority of the cases $\Delta t_{bufferror}$ is due to waiting time in the outgoing buffers. In the general case the mobile phone's buffer delay as well will contribute to the total waiting time.

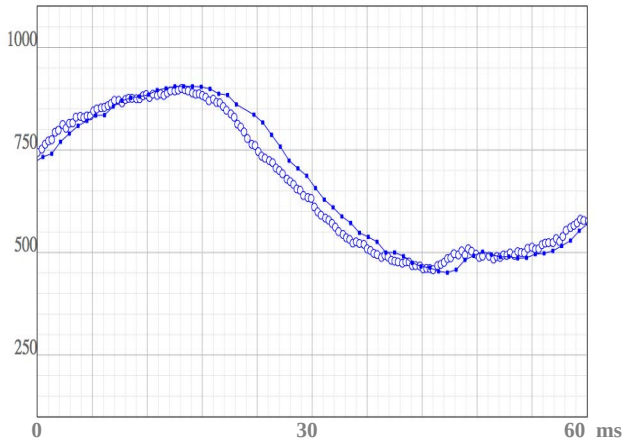

Figure 7. Accelerometer data received by a mobile phone during 60ms from two sensors before applying the synchronization algorithm. One sensor samples at 300Hz (circles) and the other samples at 200Hz (squares). The y-axis shows the raw uncalibrated accelerometer values. The x-axis shows time between 0ms and 60ms.

6. CONCLUSIONS

We have presented a new algorithm for synchronization of data from multiple sensors arriving to a mobile phone's Bluetooth interface with possibly unknown and different sampling frequencies. The new approach is based on analysis of the weakness of other single and multi-threaded solutions and does not require access to low layer system functions. An open programmable socket interface and a standard mobile phone are the only requirements. The algorithm is a multi-threaded approach that minimizes the error between the arrival time for a packet to the incoming buffer and the time when the packet is processed by the thread and timestamped. The main technique is to detect when a context switch between two threads occurs and identify the last read packet, which has the smallest error. The maximum synchronization error in our tests is 1-2 sample units.

7. ACKNOWLEDGEMENT

This work is supported by the Olympic Performance Centre (OPC) in Sweden. Thanks for presenting an interesting technical problem related to elite athlete training.

8. REFERENCES

[1] Activio, www.activio.com

[2] Ashraf, I., Gkelias, A., Dohler, M., and Aghvami, A. Time-synchronised multi-piconet Bluetooth environments. IEEE Communications 153(3) 445-452

[3] Baca A, Dabnichki P, Heller M, Kornfeind P. Ubiquitous computing in sports: A review and analysis. J Sports Sci. 2009 Oct; 27(12):1335-46.

[4] Bonato, P. Advances in wearable technology and applications in physical medicine and rehabilitation. Journal of NeuroEngineering and Rehabilitation, 2 (2).

[5] Chang, P., Chen, M., Canny, J. Tracking Free-Weight Exercises, UbiComp 2007: Ubiquitous Computing

[6] Elson, J and Römer, K. Wireless Sensor Networks: A New Regime for Time Synchronization. ACM SIGCOMM Computer

[7] Jadliwala, M.,Duan, Q.,Upadhaya, S., Xu and X, J. Towards a Theory for Securing Time Synchronization in Wirless Sensor Networks. WiSec '09

[8] N. Kaempchen and K. C. J. Dietmayer, "Data synchronization strategies for multi-sensor fusion," in Proceedings of ITS 2003, 10th World Congress on Intelligent Transportation Systems.

[9] Knight, J. F., Bristow, H. W., Anastopoulou, S., Baber, C., Schwirtz, A., and Arvanitis, T. N. 2007. Uses of accelerometer data collected from a wearable system. Personal Ubiquitous Comput. 11, 2 (Jan. 2007), 117-132.

[10] L. Diduch, A. Fillinger, I. Hamchi, M. Hoarau, V. Stanford, Synchronization of data streams in distributed realtime multimodal signal processing environments using commodity hardware, ICEM 08

[11] Libpcap, www.tcpdump.org

[12] Ringwald, M. and Römer, K. Practical Time Synchronization for Bluetooth Scatternets. BROADNETS ´07

[13] SonyEricsson, www.sonyericsson.com

[14] SparkFun, www.sparkfun.com

[15] Suunto, www.suunto.fi