Time Synchronization and Temporal Ordering of Asynchronous Sensor Measurements of a Multi-sensor Navigation System

John-Olof Nilsson and Peter Händel
Signal Processing Lab
ACCESS Linnaeus Centre
Royal Institute of Technology (KTH)
SE-100 44, Stochholm, Sweden
E-mail: jnil02@kth.se

***Abstract*—In this article we propose a filter based method to solve the time synchronization and minimum delay temporal ordering problem of asynchronous sensor measurements. A problem which inevitably arise in the sensor fusion of a multi-sensor navigation system implemented in realtime on a general purpose operation system (OS) without using functionality dedicated to realtime applications. The time synchronization is done up to a constant error by linear filtering of time stamps given to each measurement. The filtered time stamps together with predicted future time stamps are then used in a measurement temporal ordering algorithm to achieve a minimal delay temporal ordering subject to a user specified jitter tolerance. Finally, experimental time synchronization and temporal ordering results from the system implemented with a typical set of navigation sensors are presented.**

***Index Terms*—Synchronization, temporal ordering, asynchronous sensors, sensor fusion**

I. Introduction

For integrity, availability, and accuracy, navigation systems are advantageously designed as multi-sensor systems. This design approach is especially relevant with the increasing numbers of low-cost, light-weight, off-the-shelf sensors, such as MEMS-accelerometers, MEMS-gyroscopes, single chip GNSS-receivers, digital cameras, gas and contact pressure sensors, etc., available on the market as self contained devices or integrated into consumer products like smart phones and laptops. For flexibility, software portability, ease of integration with other systems, and even implementability on consumer products we would like to realize the sensor fusion of navigation systems on OSs not dedicated to realtime applications, that is standard scheduled OSs such as Windows and Linux, without using functionalities dedicated to realtime implementations. Unfortunately, in this setup the typically asynchronous sensors together with the scheduled OS give rise to integration problems. Sensor fusion methods, such as Kalman filters, generally assume synchronous measurements or perfect time information, and not often explicitly mentioned but of crucial importance, correct temporal order of measurements. However, the lack of a system-wide reference time (asynchronous sensors) together with a, from the navigation application point of view, non-deterministic scheduling, makes such assumptions invalid. Neglecting this fact, that is ignoring timing and temporal ordering errors, can lead to decreased navigation performance. The effects become especially noticeable with fast sampling sensors and high dynamic.

Time synchronization would typically be solved by either hardware or network synchronization protocols and the temporal ordering in some ad-hoc manner often by methods resembling constant jitter buffers e.g., pipeline forwarding [1], [2]. However, due to the extra hardware and interfaces or software synchronization capabilities needed at the sensors, hardware or communication based time synchronization are often undesirable, especially in low-cost flexible solutions. Further, with a simple analysis we can see that with time-driven sensors one can do better than a constant jitter buffer. Accordingly, in this study we propose how the time synchronization, apart from a non-observable constant delay (removable with calibration or correlation based techniques [3]–[5]), can be solved with recursive filters in a hardware independent manner. Further, given this time synchronization, the temporal ordering of measurements is then solved with a minimum latency given a user specified jitter tolerance and a user specified worst case delay. Experimental results from a multi-sensor navigation system, based on low-cost off-the-shelf sensors integrated on a standard industrial PC running Linux, are presented. Time stamping and clock drift measurement verify the correct behavior of, and the need for, time synchronization, while measurement of the temporal ordering show the performance characteristics of the method and quantify the effect of the jitter tolerance. From the experimental measurements and the derivation of the system we conclude that we can successfully handle the time synchronization and ordering of the measurements from asynchronous sensors of a multi-sensor navigation system implemented on a non-realtime OS. This without any extra hardware, with minimal delay, and with an adjustable wort case delay.

II. Constant error time synchronization

Without time synchronization the temporal ordering will make no sense since it most be done with respect to some time reference. Hence initially the time synchronization problem need to be solved before the temporal ordering problem can be addressed.

With time-driven sensors time synchronization in the current setting can be viewed as the problem of tracking the drifting relative phases of the sampling instances of different measurement streams in the presence of jitter. Without assumptions on the measurements themselves or additional measurements (hardware or software clock synchronization) on the initial processing, data links, scheduling procedures, etc. this phase is clearly not observable. That is an added constant delay to one stream cannot be detected. However, this only hold true for static components of the phase. By time stamping incoming data we are measuring the sampling instances plus some sensor dependent constant and stochastic delay. By defining a true time stamp as the sampling instance plus the mean delay we get an observable quantity which describes the sampling instance with a constant error.

A. Sampling instance dynamic

What we would like to estimate is the true sampling instances of a given set of sensors N (N also indicating the number of sensors). Denote the true sampling instances relative some reference time with t_{n,k_n} where $k_n \in \mathbb{N}$ is a measurement index of sensor $n \in N$. The sampling rates of the data sensors are assumed approximately known. That is, for each sensor we have a model for the sampling instances with m_n possibly time dependent (drifting) and/or uncertain parameters (periods)

$$\mathbf{r}_{n,k_n} = [r_{1,k_n} \quad r_{2,k_n} \quad \dots \quad r_{m_n,k_n}]^{\mathrm{T}}.$$

Adding the parameters to the sampling instance give the state vector

$$\mathbf{x}_{n,k_n} = [t_{n,k_n} \quad \mathbf{r}_{n,k_n}]^{\mathrm{T}}. \tag{1}$$

The assumption of the sensors being time-driven (invariant to different time references, i.e. invariant to time scale and time origin) together with (1) gives a first order state transition model

$$\mathbf{x}_{n,k_n+1} = \mathbf{F}_{n,k_n}\mathbf{x}_{n,k_n} + \mathbf{v}_{n,k_n} \tag{2}$$

where

$$\mathbf{F}_{n,k_n} = \begin{bmatrix} 1 & \gamma_{n,k_n} \\ \mathbf{0}_{m_n\times 1} & \mathbf{I}_{m_n\times m_n} \end{bmatrix} \tag{3}$$

and where γ_{n,k_n} is a k_n-dependent period selection row vector which picks out the k_n to k_n+1 sampling period. Further, the term $\mathbf{v}_{n,k_n}$ is a process noise term describing the model imperfection and the imperfection of the local sensor oscillator.

B. Time stamp observation

The time stamp $\tilde{t}'_{n,k_n}$ of measurement k_n from sensor n will be the sum of the sampling instance t_{n,k_n} and of a number of deterministic and stochastic delays. Starting from the sensor we first have an initial processing time at the sensor $\delta t_{n,k_n,proc}$, a transmission time $\delta t_{n,k_n,trans}$, a receiver buffer delay $\delta t_{n,k_n,buff}$, a scheduling delay $\delta t_{n,k_n,sched}$, etc,

$$\begin{aligned}\tilde{t}'_{n,k} = t_{n,k_n} + \delta t_{n,k_n,proc} + \delta t_{n,k_n,trans} +& \\ \delta t_{n,k_n,buff} + \delta t_{n,k_n,sched} + \dots .&\end{aligned} \tag{4}$$

By lumping all the delays together we can describe them by a constant mean delay $\bar{\delta t}_n$ and a zero-mean noise v_{n,k_n},

$$\begin{aligned}\bar{\delta t}_n + v_{n,k_n} = \delta t_{n,k_n,proc} + \delta t_{n,k_n,trans} +& \\ \delta t_{n,k_n,buff} + \delta t_{n,k_n,sched} + \dots .&\end{aligned}$$

This gives the relation

$$\tilde{t}'_{n,k_n} = t_{n,k_n} + \bar{\delta t}_n + v_{n,k_n}. \tag{5}$$

Define a "true" time stamp t'_{n,k_n} by

$$t'_{n,k_n} = t_{n,k_n} + \bar{\delta t}_n. \tag{6}$$

The time stamp can then be described as a noisy observation of the true time stamp. Defining a new state vector,

$$\mathbf{x}'_{n,k_n} = [t'_{n,k_n} \quad \mathbf{r}_{k_n}]^{\mathrm{T}}, \tag{7}$$

this gives

$$\begin{aligned}\tilde{t}'_{n,k_n} &= t_{n,k_n} + \bar{\delta t}_n + v_{n,k_n} \\ &= t'_{n,k_n} + v_{n,k_n} \\ &= \mathbf{H}_n\,\mathbf{x}'_{n,k_n} + v_{n,k_n}\end{aligned} \tag{8}$$

where the measurement matrix $\mathbf{H}_n$ has been defined as

$$\mathbf{H}_n = [1 \quad \mathbf{0}_{1\times m_n}]. \tag{9}$$

C. Time stamp dynamic

From (1), (2), (6), and (7) it is obvious that we can describe the dynamic of the true time stamp via the new state vector by

$$\mathbf{x}'_{n,k_n+1} = \mathbf{F}_{n,k_n}\mathbf{x}'_{n,k_n} + \mathbf{v}_{n,k_n}. \tag{10}$$

Together with (5) and (8) this gives the state space representation of the time stamping

$$\begin{aligned}\mathbf{x}'_{n,k_n+1} &= \mathbf{F}_{n,k_n}\mathbf{x}'_{n,k_n} + \mathbf{v}_{n,k_n} \\ \tilde{t}'_{n,k_n} &= \mathbf{H}_n\,\mathbf{x}'_{n,k_n} + v_{n,k_n}.\end{aligned} \tag{11}$$

D. Time stamp filtering

With the state space description (11) the true time stamps and the periods can be estimated with a Kalman filter. We here only give the Kalman filter estimation algorithm in Table I and the additionally involved quantities without further motivation. $\hat{(\cdot)}^{+}$ denotes estimates given time stamps up until the estimated instance. $\hat{(\cdot)}^{-}$ denotes estimates without some of the time stamps up until the estimated instance (predictions).

$$\mathbf{P}^{(+/-)}_{n,k_n} = \operatorname{cov}\big((\mathbf{x}_{n,k_n} - \hat{\mathbf{x}}^{(+/-)}_{n,k_n}), (\mathbf{x}_{n,k_n} - \hat{\mathbf{x}}^{(+/-)}_{n,k_n})\big),$$

TABLE I
SUMMARY OF THE TIME STAMP ESTIMATION ALGORITHM.

```
1:  for each $n$
2:      $\hat{\mathbf{x}'}^{-}_{n,0} := \mathbf{x}_{n,init}$, $\mathbf{P}_{n,0} := \mathbf{P}_{n,init}$
3:      for $k_n$
4:          % Update phase
5:          $\mathbf{K}_{n,k_n} = \mathbf{P}^{-}_{n,k_n}\mathbf{H}^{\mathrm{T}}_n(\mathbf{H}_n\mathbf{P}_{n,k_n}\mathbf{H}^{\mathrm{T}}_n + R_{n,k_n})^{-1}$
6:          $\hat{\mathbf{x}}^{+}_{n,k_n} := \hat{\mathbf{x}}^{-}_{n,k_n} + \mathbf{K}_{n,k_n}(\tilde{t'}_{n,k_n} - \mathbf{H}_n\hat{\mathbf{x}'}^{-}_{n,k_n})$
7:          $\mathbf{P}^{+}_{n,k_n} := (\mathbf{I}_{m_n+1\times m_n+1} - \mathbf{K}_{n,k_n}\mathbf{H}_n)\mathbf{P}^{-}_{n,k_n}$
8:          % Predict phase
9:          $\hat{\mathbf{x}'}^{-}_{n,k_n+1} = \mathbf{F}_{n,k_n}\hat{\mathbf{x}'}^{+}_{n,k_n}$
10:         $\mathbf{P}^{-}_{n,k_n+1} = \mathbf{F}_{n,k_n}\mathbf{P}^{+}_{n,k_n}\mathbf{F}^{\mathrm{T}}_{n,k_n} + \mathbf{Q}_{n,k_n}$
```

$$\mathbf{Q}_{n,k_n} = \mathrm{cov}(\mathbf{v}_{n,k_n}, \mathbf{v}_{n,k_n}),$$
$$R_{n,k_n} = \mathrm{cov}(v_{n,k_n}, v_{n,k_n}).$$

Throughout treatment and discussion on the Kalman filter techniques can be found in standard linear estimation literature e.g., [6].

What the Kalman filtering gives us is an unbiased estimate of the true time stamp. Now, the true time stamp only differs from the true sampling instance by a positive constant term. Hence what we have achieved is a constant error time synchronization. As noted before the constant term is not observable only from the time stamps but can be estimated from filter based methods or calibrations [3]–[5]. It should also be pointed out that the time synchronization is not with respect to the asynchrony of the sensor sampling between different sensors but rather with respect to the perceived clock rates of the different sensors which in turn determines the sampling. Hence, we have asynchronous sensors which are time synchronized, meaning that we have asynchronous measurements but (true) time stamps with respect to a common clock.

Worth noting about the filter is that typically the process noise, that is the oscillator drift, would be order of magnitude smaller than the measurement noise. This means that to save computational power the update phase of the Kalman filter could be run with a lower frequency than the predict phase. This could also be used to suppress short term temporal correlations in the time stamping noise v_{n,k_n}.

III. Temporal ordering

The asynchrony of the sensors is naturally handled by allocating separate threads to receive the data [7]. Hence the data is assumed to be delivered in buffers to a common integration thread. Then the integration thread must decide on how to order the data. This ordering should be done with a minimal latency, to minimize overall system delays, and such that it is robust to delayed or lost measurements. The ordering ought to be done based on the true sampling instances. Unfortunately these are not available and even worse, as argued earlier, not even observable without additional measurement or by taking the information in the measurements themselves into account. However, we will assume that the constant time synchronization errors in the estimation of (11) are estimated and fed back from a later main filtering stage described in [3]–[5] or found by calibration. This is illustrated in Fig. 1. *Accordingly, henceforth in this section we will refer to the true time stamp estimates as sampling instance estimates.*

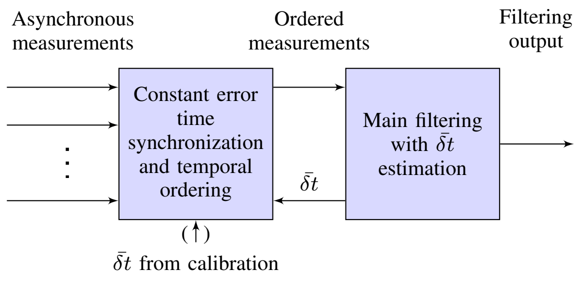
Asynchronous measurements
Ordered measurements
Filtering output
Constant error time synchronization and temporal ordering
Main filtering with $\bar{\delta t}$ estimation
$\bar{\delta t}$
(↑)
$\bar{\delta t}$ from calibration

Fig. 1. The block diagram illustrate the constant error time synchronization and the temporal in relation to the main filtering and the asynchronous measurements. The constant error in the time synchronization can be found either from estimation of the delay $\bar{\delta t}$ in the main filtering or from calibration as indicated in the diagram.

Then what is given are N asynchronous measurement streams accompanied with sampling instance estimates and models. From here the measurements could easily be ordered by a simple jitter buffer. Each measurement would have to wait for a fixed time window after which its sampling instance estimate is compared against all other available measurements. However, this would often make us wait longer than necessary and hence increase the overall delay of the system. This we can see from the fact that with typical clock drift values (noise values) in (11) we can quite accurately predict future sampling instances. Hence, if there are no smaller available estimated sampling instances or predicted sampling instances than that of an available measurement then we could filter (put next in order) the measurement immediately. Unfortunately two problems arises here. 1) The predicted sampling instance estimate can differ from the updated leading two incorrectly ordered data. Hence, a *jitter tolerance* δt_{jitter} has to be added to the comparison of the sampling instance estimate of the available data and the predicted sampling instance estimate of the not jet available data. However, no bound can be put on the difference and data incorrectly proceeded by other data will have to be dropped or handled by out-of-sequence-measurement (OOSM) routines. 2) If a measurement is lost or if there is a complete sensor failure there might be unacceptable delays or even deadlocks since the system will be waiting for a measurement that does not arrive. Hence for every measurement that arrives a check against all the predicted sampling instance estimates has to be done. If the sampling instance estimate is larger than the predicted sampling instance estimate plus a user defined *maximum delay* δt_{max} and under the condition that this predicted sampling instance estimate is blocking some available measurement then it should be discarded. This ensures that the worst case relative delay between streams cannot exceed δt_{max}.

Further to put this into an algorithm we need some ad-

TABLE II
PSEUDO CODE OF THE TEMPORAL ORDERING ALGORITHM.

```
1:  for each $i \in N$
2:    if $t_{i,limit} = \hat{t}^{+}_{i,k_i}$
3:      % Measurement available and previously blocked
4:      if not $\exists j : \{\hat{t}^{+}_{i,k_i} \leq t_{j,limit} : j \neq i, j \in N\}$
5:        % Nothing is blocking the measurement
6:        $t_{filter} = \hat{t}^{+}_{i,k_i}$
7:        $t_{i,limit} = \hat{t}^{-}_{i,k_i+1} - \delta t_{jitter}$
8:        filter measurement $k_i$ of stream $i$
9:      continue
10:   if $t_{i,limit} \neq \hat{t}^{+}_{i,k_i}$ and $\exists \hat{t}^{+}_{i,k_i}$
11:     % New measurement available
12:     if $\hat{t}^{+}_{i,k_i} < t_{filter}$
13:       % Incorrect ordering has occurred
14:       $t_{i,limit} = \hat{t}^{-}_{i,k_i+1} - \delta t_{jitter}$
15:       Dropp measurement $k_i$ of stream $i$
16:       continue
17:     else if $\exists j : \{\hat{t}^{+}_{i,k_i} \leq t_{j,limit} : j \neq i, j \in N\}$
18:       % Nothing is blocking the measurement
19:       $t_{last} = \hat{t}^{+}_{i,k_i}$
20:       $t_{i,limit} = \hat{t}^{-}_{i,k_i+1} - \delta t_{jitter}$
21:       filter measurement $k_i$ of stream $i$
22:       continue
23:     else
24:       % The measurement is blocked
25:       $t_{i,limit} = \hat{t}^{+}_{i,k_i}$
26:       continue
27:   if $\exists j : \{\hat{t}_{j,last} - t_{i,limit} > \delta t_{max} : j \neq i, j \in N\}$
28:     % Discard blocking limit
29:     $t_{i,limit} = \hat{t}^{-}_{i,k_i+1} - \delta t_{jitter}$
30:     continue
31:   continue
```

ditional variables. First, the algorithm need some variables to keep track of how predicted sampling instances and updated sampling instances are blocking other measurement streams. For this purpose we use a variable $t_{n,limit}$ for each measurement stream that describe up to which point one stream is blocking the other streams. Second, a memory of the sampling instance estimate of the last measurement sent forward, t_{filter} is needed to check for incorrect ordering. Third, the sampling instance estimate of the last measurements handled in the sensor threads $t_{i,last}$ are needed to check that the maximum delay is not exceeded. The measurement indices k_n are assumed to be kept track of as the next measurement to come or the defacto incoming measurement. Future predictions of sampling instances $\hat{t}^{-}_{n,k_n}$ are assumed available from the sampling instance model. The total algorithm is presented in pseudo-code in Table II.

Not covered in this publication but of great importance for an implementation of the algorithm in II is the pacing of the integration thread. The integration thread should not poll the buffers but only execute the algorithm in II when needed. This will require a number of extra book-keeping variables and will also be dependent on provided inter-thread signaling functionalities provided by the threading library used.

TABLE III
SUMMARY OF SENSORS, THEIR PHYSICAL INTERFACES, AND THEIR DATA PERIODS.

Sensor	*Interface*	*Data period(s)*
GPS receiver	USB	1s
IMU	USB	10ms
UWB node	RS232	5ms and 100ms
Camera	FireWire	≈100ms

IV. EXPERIMENTAL RESULTS

The constant error time synchronization together with the temporal ordering algorithm has been implemented in C with the Pthread library handling the threading. The sensors in the experiment are: a low-cost GPS receiver, a MicroStrain Inertia-Link IMU, an Allied Vision Techonology Guppy FireWire camera, and in-house built UWB ranging nodes. As summary of the sensors together with some technical specifications can be found in Table III. The sensor fusion unit was a standard Dell D600 laptop with Linux, Ubuntu 8.04. The measurement where done under medium and constant computational load of the computer. The data was collected with a data rate of approximately 210 measurements/s and 3.5 MB/s over 2 hours. The measurements show rather typical time stamp signal characteristics and performance. However, working with the system on different platforms, with different sensors, and under different computational loads, the experience has been that the characteristics of the time stamp signals has varied while the characteristics and the performance of the complete system has been rather constant. The presented data is not meant to give exact or statistical measures of the system but rather to show typical system characteristics.

A. Time stamp characteristics

The fundamental input to the constant error time synchronization (sampling instance estimation) is the time stamps. Accordingly, their statistical characteristics will ultimately determine the behavior of the system. For the time stamp estimation, from the theory of Kalman filtering we have that the filter is optimal (minimum variance) under the assumption of white gaussian noise (WGN). The process noise of (2) and hence also (10) is due to multiple independent sources. Hence, from (4), (5), (6), and the law of large numbers one might expect that the time stamp measurement noise is also approximately WGN. For all sensors but the IMU this has indeed been observed to be approximately the case. In Fig. 2 the residual of the camera image time stamps and the estimated autocorrelation are shown. As for the IMU two approximately gaussian distribution and a strong periodic correlation has been observed. This is believed to be due to time multiplexing of the USB-bus with a period close to a multiple of the IMU sampling rate. The residual and the estimated autocorrelation of the time stamps of the IMU measurements are shown in Fig. 3. Despite the far from WGN behavior of the IMU time stamps residuals the ordering system has been observed to work well.

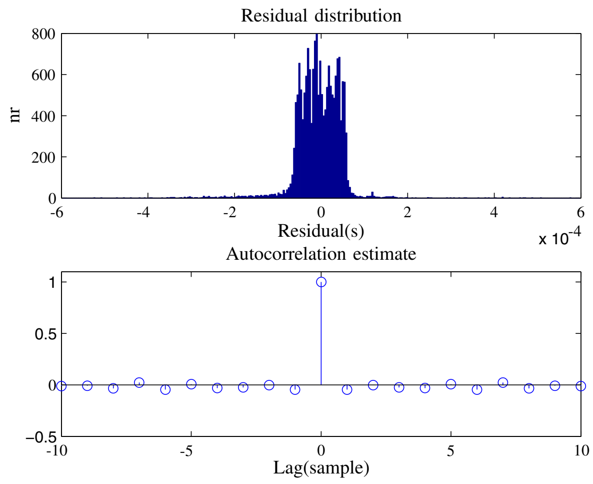

Fig. 2. Camera time stamp residual distribution and estimated autocorrelation.

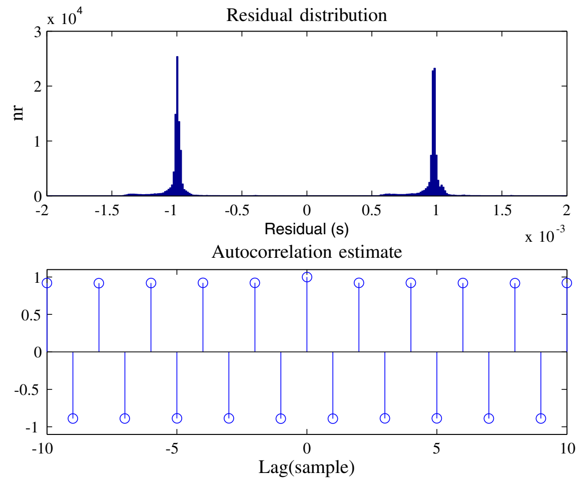

Fig. 3. IMU time stamp residual distribution and estimated autocorrelation.

B. Time synchronization

As noted before, time synchronization filtering can be seen as tracking the drifting phase of the sampling instances. We do so by estimating the true time stamps and the corresponding sampling periods. In Fig. 4 deviation of the estimated time stamps and the measured time stamps from their nominally values as given from data sheets are plotted for the first first 10 minutes of measurements.

Basically the effects of the time synchronization filtering are two. First the drift of the true time stamps are tracked. Second the measurement noise of the time stamps is suppressed. About the first effect Fig. 4 tells us that if we ignored the tracking and used nominal rates after 10 minutes the clock skew would have set us off by approximately 0.2s. A figure of merit of the second effect, assuming our sampling instance and measurement models are correct and that the measurement noise is significantly larger than the process noise, would be how many samples which are switched in order due to the

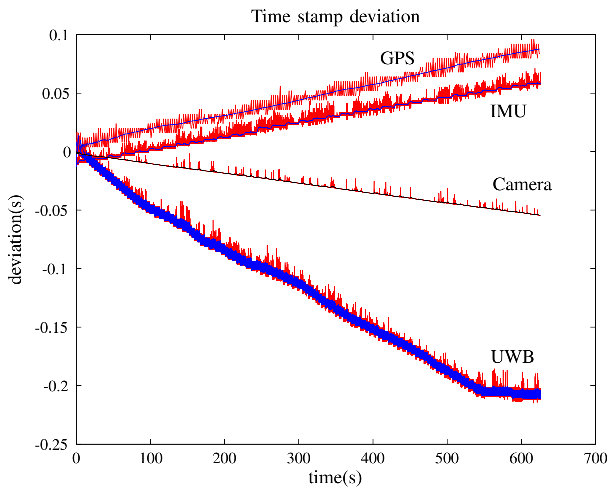

Fig. 4. Estimated and measured time stamp deviation from expected values based on nominal sampling periods. The deviation of the estimated time stamps in blue are plotted on top of that of the measured time stamps in red. The thick blue line of the UWB time stamps are due to the fact that measurements are acquired with two different periods with different deviations.

estimated time stamp as in comparison with the measured time stamps. For the measurements in Fig. 4 this was approximately 0.3%. However, this number is highly dependent of thread settings, the computational load, and the number of sampled per time unit.

C. Temporal ordering

The temporal ordering algorithm order the measurements according to the time synchronization. What the algorithm work with is the sampling instance predictions and the sampling instance estimates. Their inconsistency is given by the residual in Fig. 2 and 3. Now, the value of δt_{jitter} sets a limits for what inconsistency we tolerate. As explained in section III, a measurement with a time stamps estimate which deviate (smaller than) more than δt_{jitter} from the predicted value is in the risk of being ordered incorrectly (dropped) while two measurements with predicted time stamps within δt_{jitter} will have to wait for each other to determine their order. Hence, there is a trade-off relation in the setting of the δt_{jitter} parameter. We want the value of δt_{jitter} as small as possible not to add extra delay (blocking) meanwhile we have to set it large enough not the lose measurements. In Fig. 5 we see the portion of lost measurements and the portion of delayed measurements plotted against the value of δt_{jitter}. The trade-off relation between the two graphs, the constant relation to δt_{jitter} in the lower range of dropped measurements, and the abrupt drop in the number of dropped measurements around $\delta t_{jitter} = 0.1$ms tells us that we should either be in the lower end ($\delta t_{jitter} = 0$) or just above the drop (δt_{jitter}=0.1-0.5ms). That is either we should set the measurements in the order they come, and lose 0.2% of the measurements due to incorrect order of arrival, or we should delay 2-10% the measurements by 0.1-0.5ms and lose very few measurements.

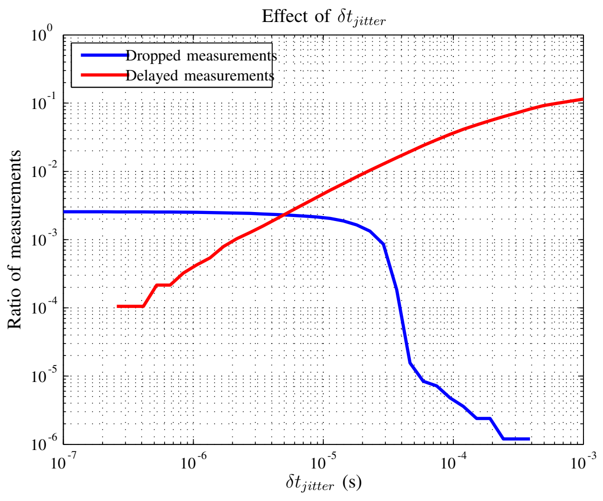

Fig. 5. Portion of dropped measurements and delayed (temporally blocked) measurements as a function of the delay tolerance due to the ordering process. The curves are in a trade-off relation to each other. We want to minimize both. Hence, the δt_{jitter} should either be zero or in the interval 0.1-0.5ms.

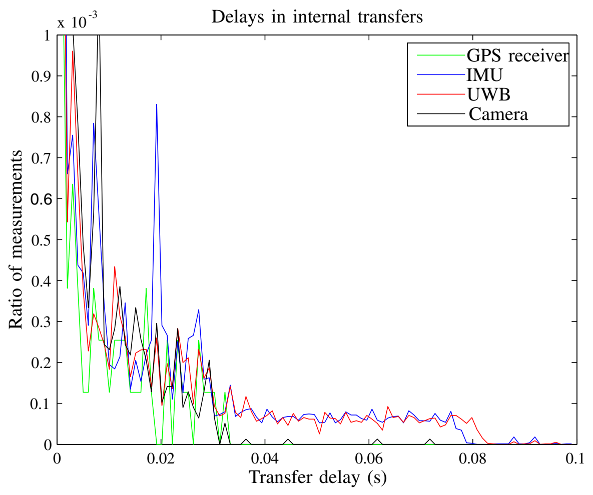

Fig. 6. Delays in internal measurement transfer between sensor threads and integration threads. The plot is done by placing the measurements in 100 bins in the range 0-0.1s and normalize the hight by the first bin. The area under the shown graph correspond to approximately 1% of the measurements.

This is in contrast to a constant jitter buffer in which we would have had to delay 100% of the measurements 0.1-0.5ms to achieve the same performance. Interestingly one may also note the slower decay of proportion of lost measurements for δt_{jitter} in the range 0.05ms-0.5ms. While the reminder of the curve can be explained by a Gaussian distribution of sampling instance predictions to sampling instance estimates residuals (found from simulations) this part is interpreted to be due to outliers. These outlier can also be observed in Fig. 4 as the spikes in the read curves. That is the spikes in the deviation from the slow drift. These spikes are most likely due to interfering scheduling from other processes running on the computer. However, these effect will typically be correlated between different measurement streams why the effect is better studied in Fig. 5.

Further, the ordering algorithm also have to take into account the delays introduced in the preprocessing and inter system transfer of data. The second adjustable value of the algorithm, δt_{max}, sets limits for the acceptable such delays. In Fig. 6 the inter-thread transfer delays for the different sensors are shown. The transfer delay values has been divided in 100 bins in the range 0-0.1s each and normalized to the hight by the first bin. The major part of the distribution close to zero is not shown but rather the tail that will be affected by the value of δt_{max}. The displayed part of the distribution correspond to approximately 1% of the total number of measurements. Hence, the value of the δt_{max} can safely be set between 5-100ms depending on preferences. However, giving a plot of the number of lost measurement against the value of δt_{max} is difficult since normally the inter-thread transfer delays will be correlated both with the time stamping and between threads.

V. Conclusions

A filter based method for time synchronization and temporal ordering of asynchronous measurement from multiple sensors has been suggested. A constant error time synchronization is achieved by filtering of time stamps. The suggested method then rely on measurement type specific filtering to estimate the constant time synchronization error not observable from time stamps. Based on the time synchronization the measurements are ordered with a minimum delay given a user specified jitter tolerance. The time synchronization and temporal ordering has been implemented with a typical set of sensors for a multi-sensor navigation system. Experimental data from the system show the need for the time synchronization and show that the jitter tolerance should either be set to zero or above a threshold over which very few measurements are lost.

References

[1] M. Baldi, G. Marchetto, and Y. Ofek, "A scalable solution for engineering streaming traffic in the future internet," *Computer Networks*, vol. 51, pp. 4092–4111, Oct 2007.

[2] N. Kaempchen and K. Dietmayer, "Data synchronization strategies for multi-sensor fusion," in *In Proc. ITS2003*, Madrid, Spain, Nov 2003.

[3] J.-O. Nilsson, I. Skog, and P. Händel, "Joint state and measurement time-delay estimation of nonlinear state space systems," in *Proc. ISSPA2010*, Kaula Lumpur, Malaysia, 10-13 May 2010. (Submitted).

[4] I. Skog and P. Händel, "Time synchronization errors in GPS-aided inertial navigation systems," *IEEE Trans. on Intell. Transp. Syst.*, 2010. (In revision).

[5] J. Fox, "Synchronization of input and output measurements using a Kalman filter," in *Proc. ICMIC2006*, pp. 429–434, Lanzarote, Spain, 6-8 February 2006.

[6] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. Prentice Hall, 2000.

[7] B. Agarwalla, P. W. Hutto, A. Paul, and U. Ramachandran, "Fusion channels: A multi-sensor data fusion architecture," Technical Report CC Technical Report; GIT-CC-02-53, Georgia Institute of Technology, 2002.