Temporal synchronization of radar and lidar streams

Aledo Ortega, D.; Manjunath, T.; Rajan, R.T.; Maksimiuk, Darek; van Leuken, T.G.R.M.

Publication date
2022

Document Version
Final published version

Published in
42nd WIC Symposium on Information Theory and Signal Processing in the Benelux (SITB 2022)

Temporal synchronization of radar and lidar streams

David Aledo
Circuits and Systems Group
Delft University of Technology
Delft, The Netherlands
d.aledoortega-1@tudelft.nl

Tanmay Manjunath
Circuits and Systems Group
Delft University of Technology
Delft, The Netherlands

Raj Thilak Rajan
Circuits and Systems Group
Delft University of Technology
Delft, The Netherlands

Darek Maksimiuk
Innatera Nanosystems
Rijswijk, The Netherlands

Rene van Leuken
Circuits and Systems Group
Delft University of Technology
Delft, The Netherlands

***Abstract*—In multi-sensor systems, several sensors produce data streams, commonly, at different frequencies. If they are let running wild without synchronization, after a period of time, they are likely to be disordered, presenting as simultaneous measures that have been recorded at different times. That can be disastrous in many data fusion applications. This paper is about their temporal synchronization and ordering, so they can be coherently fused. Some sensors do not have timestamps from which order the streams, and even if they have, they may be not trustable for different reasons. First, we define mathematically the problem of multi-sensor data stream synchronization. Then, we handle the problem of estimating the actual time of sensor measurement using mean or median filters. Next, we address the issue of reconstructing incoming sensor data streams according to the estimated sensor measurement times while maintaining minimal latency and synchronization error by employing an adaptive stream buffering technique utilized in distributed multimedia systems. In order to test our methods, we have recorded an easy-to-use dataset with a radar and a lidar sensors without timestamps. We define a synchronization event that is easily identifiable by a human annotator in both sensor streams. From this dataset, a suitable filter for timestamp estimation is selected, and an analysis of the effects of the stream synchronization algorithm's parameters on buffering latency and synchronization error is presented. Finally, the solution is efficiently implemented on a FPGA.**

***Index Terms*—multi-sensor, synchronization,**

I. Introduction

For precise fusion of different sensors, measurements need to be synchronized both temporally and spatially. This paper aims to design a solution of the temporal synchronization problem for multi-sensor data fusion applications.

Consider a system with multiple data streams provided by different sensors. Probably, some of them have different measurement frequencies. Even if they have the same data rates, there may be drifts between their particular sensor timelines. If they are let running wild without any synchronization, after a period of time, they are likely to be disordered, presenting as simultaneous measures that have been recorded at different times. That can be disastrous in many data fusion applications.

Particularly, this work has been motivated by the scenarios presented in two European projects: PRYSTINE and ADACORSA. The PRYSTINE project addresses challenges in automotive applications, while ADACORSA aims to enable beyond-visual line of sight (BVLOS) for drone navigation. In both these projects, the environment perception of the agent (e.g., automotive or drone) is crucial and is achieved by sensor fusion of on-board sensors e.g., cameras, radars and lidars. In both scenarios, the temporal synchronization must to be realized in real-time, with minimum latency, and low resources (specially power). While data fusion involving cameras has been more extensively studied, including its synchronization, radar and lidar data synchronization and fusion can be more challenging. The main reason is that camera images are more easily understandable by the human annotators, allowing for easier test and calibration mechanisms like chessboards. Our consider system includes a radar sensor and a lidar sensor, which provides two data streams without timestamps. The streams are collected in a centralized processing system. The module that collects the data streams withing the processing system is the acquisition system, which can add a timestamp on the data upon its arrival. The goal is to introduce a synchronization module, just after the acquisition system, that can correctly order the data streams, with a minim latency and resource utilization, before the main data fusion or processing.

Since some sensors do not have timestamps from which order the streams, and even if they have, they may be not trustable for different reasons, first we need to estimate the measurements timestaps. Therefore, the original problem is divided into two: measure timestamp estimation, and real-time synchronization of timestamped streams.

To better illustrate the problem we are trying to solve, Fig. 1 shows the synchronization problem for the two streams coming from the radar and lidar sensors, and the same two streams correctly synchronized by our solution.

The paper is structured as following: first, a related work section highlighting the available solutions in the state of the

This work was supported by the PRYSTINE Project, funded by Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU) in collaboration with the European Union's H2020 Framework Programme and National Authorities, under grant agreement no. 783190.
This work is partially funded by the European Leadership Joint Undertaking (ECSEL JU), under grant agreement No 876019, the ADACORSA project - "Airborne Data Collection on Resilient System Architectures."

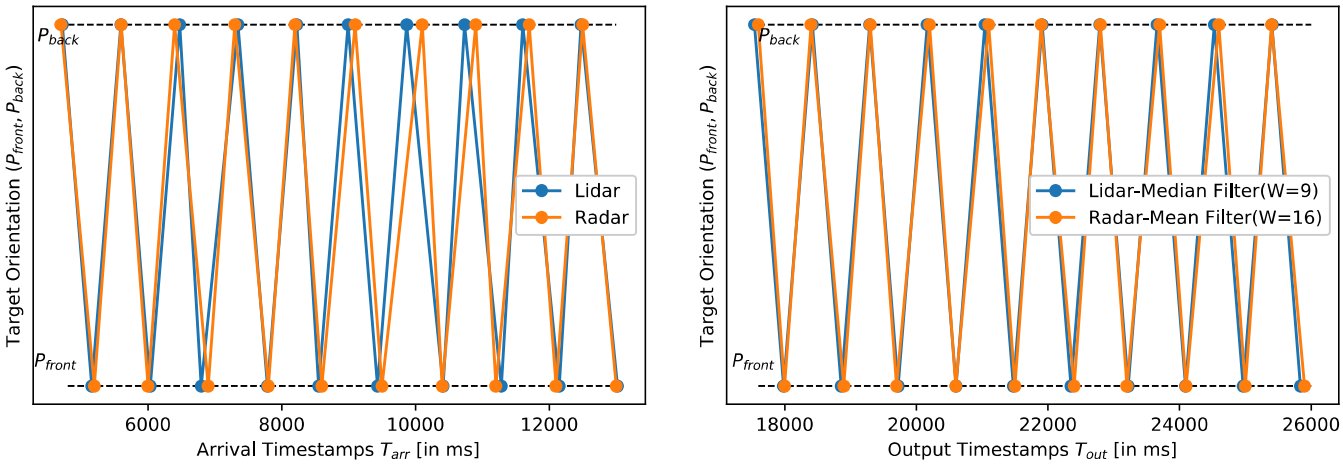

Fig. 1. Synchronization events at given timestamps before (left) and after (right) synchronization.

art for both the timestamp estimation problem and the synchronization problem. Then, in section III, we formulate these problems mathematically, and present the solution algorithm. Section IV describes the dataset collected for the experimental results presented in the following section or results. Finally, in VI, the hardware implementation is described, and in VII the conclusions.

II. Related work

To address the issue of timestamping sensor measurements in free-running sensor network systems, earlier, in simpler multi-sensor systems, the timestamp on arrival at the acquisition system was used as the true time of measurement for fusion applications [1]. This approach completely overlooks the possibility that the sensor data may be subjected to delays and jitters during transmission and acquisition, causing asynchrony between the measurements. In [2], a software timestamping approach which aims to improve the timestamps quality by reducing delays and jitters during acquisition, is proposed. However, the the sensor data is still timestamped after its arrival at the acquisition system and the data is still subjected to transmission delays and jitters. Hence, these solutions are not suitable for time critical applications where timing misalignment cannot be ignored.

A popular approach is to employ hardware based timestamping for sensor measurements. In [3], a device is used to attach a timestamp to each sensor data frame, before it is transmitted to the acquisition system through the communication link. It has an embedded GPS receiver to get precise UTC (Universal Time Coordinate) times for timestamping. However, this approach cannot be applied to all senors as it may require the sensors to be programmable and to also have special interfaces. Similar GPS receiver based hardware devices have been used for the synchronization of externally triggered sensors, where the device precisely triggers the sensors at the right instances [4], [5].

For free running asynchronous sensors systems without external synchronization support, the only timestamping that can be done, is at the acquisition system. Approaches presented in [6], [7] utilise these arrival times to estimate the true sensor measurement times. All these solutions are software timestamping based on linear Kalman filters to essentially filter out delay jitters from arrival times while preserving the effect of the internal sensor clock drift in the true measurement times. However, the estimation procedures in these solutions are software based and are not designed for real-time applications.

The problem of synchronizing streams of data has been extensively investigated in the area of distributed multimedia systems. A classical example of this scenario is the *lip-sync* problem [8], [9], where audio and video streams need to be accurately synchronized during play-out.

The problem of multimedia synchronization can be classified based on location, real-time requirement, type of synchronization (within or between streams), purpose of the synchronization protocol and availability of timing and network information. Based on the real-time requirement of the stream play-out, synchronization techniques are classified as live or synthetic. The former deals with synchronizing live data streams in real-time whereas the later deals with stored media frames [10]. Here, only live synchronization solutions are presented, as they are relevant to our problem.

Media synchronization is an end-to-end problem [11], hence, based on the application, it can be addressed either on the source [12], receiver [13], [14] side, or both [10], [15]. On the source side, a common solution is to attach timestamps and sequence numbers to the transmitting frames [13], [14]. Other source-side techniques mainly consists of changing the properties of the media streams. In some cases, the sources can interleave streams into a single stream before transmission as in [16], [17]. This solution succeeds in eliminating the need for inter-stream synchronization, however, intra-stream synchronization still needs to be addressed. In [17], [18], the source changes the transmission rate of the streams depending on the feedback received from the receiver on the network conditions to prevent asynchrony. On the receiver side, buffer-

ing techniques are commonly employed. The buffering time can either be static based on a maximum jitter value or can be made to vary depending on the network delays [19] and the available buffer size [14]. Other receiver-side techniques consist of dropping late arriving frames [10] or older frames during buffer full conditions [19]. The dropped frames are either left empty or interpolated [14].

For intra-stream synchronization, techniques that aim at reducing the effects of jitter are used. This includes receiver buffering techniques [19]–[21] to smooth the effects of delay variabilities. For inter-stream synchronization, normally, master/slave techniques are used, where one stream is set as a master or reference and the rest as slave streams [20], [21]. At a certain point in time, the bottleneck stream, i.e., the stream affected by the most delay is chosen as the master or reference. Then, the play-out rates of slave streams are adapted to maintain the correct temporal relations with the master stream. In [20], dynamically switch master and slave streams during run-time. However, it is necessary to first remove the effects of network jitter by establishing intra-stream synchronization between the frames before applying inter-stream techniques [11].

Moreover, the complexity of the synchronization solutions majorly depend on the nature of the timing of media frames (as in periodic or non-periodic) and network information such as delays and jitters. If the nature of frame generation is non-periodic then timestamps of frame generation from the source side is compared with the arrival timestamps at the receiver to estimate jitter and buffer the frames accordingly [22]. On the other hand, for periodic streams, arrival period at the receiver can be compared with the period of the stream to estimate jitters and also, inter-stream synchronization becomes less complex than the non-periodic case. In certain systems, assumptions can be made on the network delays and jitter based on the network characteristics. If exact bounds on network jitters are known then constant delay buffers at the receiver would suffice to ensure both inter- and intra-stream synchronization. Also, this eliminates the need for timestamps from the source side. However, if the maximum bound on jitters is too high, then the frames need to buffer for a longer time, leading to large buffering latencies. Besides, in most applications, an exact bound on jitters cannot be known. In such cases, a tolerable synchronization error value is set and the frames are buffered for lesser time. This leads to a trade off between the quality and latency of the synchronization algorithm. To overcome it, adaptive control based solutions are proposed in [20], [21]. They consists of a control algorithm which keeps the latency and quality at check by changing the buffering delays during run-time according to the current jitter conditions while maintaining a pre-set minimum latency and synchronization error. Ideally the control algorithm makes sure that the buffering delays are large enough to compensate for the effects of jitter and stay within tolerable synchronization error but not too large, to keep the latency minimum.

TABLE I
SYSTEM EVENTS AND THEIR TIMESTAMPS

Event	Description	**Timestamp**
e1	Radar measurement	T^R_{mea}
e2	Radar frame transmission starts	T^R_{tr}
e3	Arrival of radar frame at acquisition system	T^R_{arr}
e4	Lidar measurement	T^L_{mea}
e5	Lidar frame transmission starts	T^L_{tr}
e6	Arrival of lidar frame at acquisition system	T^L_{arr}
e7	Radar synchronization output	T^R_{out}
e8	Lidar synchronization output	T^L_{out}

III. MATHEMATICAL FORMULATION OF THE SYNCHRONIZATION PROBLEM

To formulate the problem formally, we first define all the events in the system, which includes the sensors radar and lidar, and the acquisition system that collects the data streams before processing. We introduce a synchronization module just after the acquisition system. For each event, a timestamp is assigned. Events and their corresponding timestamps are listed in Table I. The events ($e1$, $e2$, $e3$) and ($e4$, $e5$, $e6$) occur in a sequence. We define these sequences of events as: the radar acquisition process ($P^R : e1 \rightarrow e2 \rightarrow e3$), and the lidar acquisition process ($P^L : e4 \rightarrow e5 \rightarrow e6$). These processes occur asynchronously and are independent of each other. There are any relationships between the events of the two processes, since, depending on the sensor sampling rate and the initial start offset between the sensors, the order can change during the run time.

The considered radar and lidar sensors do not have clocks or any other mechanism to record the times of their events. However, at the acquisition system, the arrival times of radar and lidar frames (T^R_{arr} and T^L_{arr}) are assigned by a common clock. A solution to this issue may be to connect external clocks to record the event timestamps. However, this solution is both expensive and unreliable since the triggering of an event cannot be observed accurately from external clocks due to distortions introduced by the interconnecting cables.

From the moment the sensors capture the measurements to the point they arrive at the acquisition system, the sensor data stream is subjected to various delays. These delays are unpredictable and their magnitudes have no definite bounds, leading to vastly varying arrival latencies. In addition, delays are different for each sensor data stream resulting in measures captured at different time being presented as simultaneous to the fusion algorithm. A good understanding of the nature of the delays along with the source of delay variability factors is essential to formulate an efficient synchronization algorithm. Some of the significant delays in the considered system are:

- **Pre-rocessing delays:** are delays that are incurred on the sensors for modifying/processing the raw sensor measurements. It includes delays due to on-sensor chip pre-processing steps and packetization of sensor measurements into appropriate format transmission. With respect

to the model, these are the delays between events $e1$ and $e2$ in the radar, and between $e4$ and $e5$ events in the lidar.

- **Communication delays:** are the delays experienced by the sensor data during its transfer from the sensor to the acquisition system. Depending on the size of the measurements, a single sensor measurement can be broken down into several packets/frames[1]. The delays due to queuing of these frames, the physical transmission of the data, all constitute communication delays. In the model, these are the delays between events $e2$ and $e3$ in radar, and between $e5$ and $e6$ events in the lidar.

We now define the term *measurement latency*, which refers to the overall latency experienced by the k^th sensor measurement from its capture at the sensor to its arrival at the acquisition system. *Measurement latency* ($\Delta\tau^R[k]$, $\Delta\tau^L[k]$) is given by:

$$\Delta\tau[k] = T_{arr}[k] - T_{mea}[k]. \tag{1}$$

Measurement latency is essentially an abstraction of all the delays and their associated delay variability that a sensor measurement is subjected to, and hence, can also be modelled as:

$$\Delta\tau[k] = d - \delta_{jitter}[k] \tag{2}$$

where $\delta_{jitter}[k]$ denotes the jitters in delays for the k^{th} sensor measurement and, d represent the expected value (mean) of their respective sensor delays.

With events, timestamps, and delays of the system defined, we now mathematically formulate the problem. On the top level, we aim to maintain the same temporal relationships between the measurements in which they were originally recorded. We define two types of temporal relations in the model:

- **Intra-stream relations:** refers to the temporal relationship between the frames belonging to the same stream. More precisely, the intra-stream relation is the time difference between events of two consecutive sensor measurements captured by the same sensor. The intra-stream temporal relation between the $[k-1]^{th}$ and k^{th} events of the same stream is:

$$\Delta T[k-1, k] = T[k] - T[k-1]. \tag{3}$$

- **Inter-stream relations:** refers to the temporal relationship between corresponding frames belonging to different streams. It is the delay difference between events of two corresponding measurements captured by different sensors. The inter-stream temporal relation between between the k^{th} radar and lidar events is:

$$\Delta T^{R,L}[k] = T^R[k] - T^L[k]. \tag{4}$$

To meet our goal, we need to achieve:

1) **Intra-stream synchronization:** to maintain the intra-stream relationships in which the measurements were originally recorded, the following condition must hold:

$$\Delta T_{out}[k-1, k] = \Delta T_{mea}[k-1, k]. \tag{5}$$

2) **Inter-stream synchronization:** analogously,

$$\Delta T^{R,L}_{out}[k] = \Delta T^{R,L}_{mea}[k]. \tag{6}$$

However, due to the jitter component of the delay factors, disturbances are introduced into the temporal relationships of sensor measurements as they arrive at the acquisition system. Thus, we can expect

$$\Delta T_{arr}[k-1, k] \neq \Delta T_{mea}[k-1, k], \tag{7}$$

and

$$\Delta T^{R,L}_{arr}[k] \neq \Delta T^{R,L}_{mea}[k]. \tag{8}$$

Using (1), (2), and (3), we can view $\Delta T_{arr}[k-1, k]$ as a noisy version of $\Delta T_{mea}[k-1, k]$:

$$\Delta T_{arr}[k-1, k] = \Delta T_{mea}[k-1, k] + \delta_{jitter}[k] - \delta_{jitter}[k-1] \tag{9}$$

where δ_{jitter} is assumed to be zero-mean white noise.

With this, a straightforward solution will be to re-construct the sensor data streams at the acquisition system by buffering accordingly to compensate for the effect of jitters on each sensor frame. Unfortunately, in our concerned system, T^R_{arr} and T^L_{arr} are the only timing information available. Therefore, we formulate the solution into the following steps:

1) Estimate intra-temporal relations of measurements ($\Delta\hat{T}_{mea}[k-1, k]$) by filtering the observed $\Delta T_{arr}[k-1, k]$.
2) Extract estimated timestamps ($\hat{T}_{mea}$) from temporal relation estimates.
3) Buffer and re-construct the data streams. The main objective here is to ensure that the incoming sensor data is streamed out in real-time while maintaining time synchronization.

As we do not have any measure or estimate of the measurement latency, and since for us it is more important the stream ordering rather than knowing the exact time of a measurement, the arrival time of the first sensor measurement at the acquisition system is assigned as estimator of the first $e1$ and $e4$ timestamps

$$\hat{T}_{mea}[0] = T_{arr}[0], \tag{10}$$

although a better estimate can be obtained if d is known or estimated

$$\hat{T}_{mea}[0] = T_{arr}[0] - d. \tag{11}$$

The timestamps extracted from the above method can be inaccurate during the following two scenarios and have to be corrected accordingly:

1) When the calculated measurement timestamp is greater than its corresponding arrival timestamp ($\hat{T}_{mea}[k] >$

[1]For simplicity, we associate each measurement with a single frame.

$T_{arr}[k]$). It would mean that the measurement is captured in the sensor later than its arrival at the acquisition system. Hence, the estimate is reset with (10) or (11) using the current arrival timestamp.

2) When a frame is lost. In this case, the timestamp of the next measurement can be wrongly associated to the timestamp of the lost measurement. The observed cycle time after a lost measurement is larger than normal cycle times. Therefore, a threshold is set to detect lost measurements ($\Delta\hat{T}_{mea}[k-1,k] > Th_{lost}$), and $\hat{T}_{mea}[k]$ is reset like in the previous case.

After timestamp estimation, the implemented algorithm is based on solutions presented in [20], [21], [23]. These algorithms ensure intra- and inter-stream synchronization in real-time by employing control-based adaptive buffering techniques. With the estimates and arrival timestamps available, the algorithm accomplishes synchronization by comparing and equalising measurement latency of each sensor. The equalization is carried out by piece-wise adjustment of buffering times while meeting a set of Quality of Service (QoS) factors along with a minimal overall latency. The QoS factors are:

- **Maximum intra-stream phase distortion**. Intra-stream phase distortion (Intra-SPD), $\Delta\phi_{intra}$, is the difference between the measurement latencies of two consecutive frames of the same sensor stream. A maximum allowable threshold on intra-SPD is set for each sensor ($Th.\Delta\phi_{intra}$). If the arrival of an incoming sensor measurement does not fall within it, then, the frame is considered to have arrived too late and thus, discarded. Intra-SPD is calculated as:

$$\Delta\phi_{intra}[k-1,k] = \left|\Delta\tau[k] - \Delta\tau[k-1]\right|. \tag{12}$$

  Since we do not know $\Delta\tau[k]$, we use its estimator

$$\Delta\hat{\tau}[k] = T_{arr}[k] - \hat{T}_{mea}[k]. \tag{13}$$

- **Maximum inter-stream phase distortion**. Inter-stream phase distortion (inter-SPD) is the difference between the measurement latencies of two adjacent sensor measurements belonging to different sensor data streams. Here, *adjacent sensor measurements* refer to measurements that have been most closely captured by two different sensors. A maximum allowable threshold on inter-SPD ($Th.\Delta\phi^{R,L}_{inter}$) is set. Any frame arriving beyond it, is discarded. Inter-SPD is calculated as:

$$\Delta\phi^{R,L}_{inter}[k] = \left|\Delta\tau^{R}[k] - \Delta\tau^{L}[k]\right|. \tag{14}$$

Intra-SPD and inter-SPD quantifies the disruption in intra- and inter-stream relationships, respectively.

The synchronization algorithm can be divided into two schemes focusing on intra-stream and inter-stream synchronization. On the top level, intra-stream synchronization is first established by adaptive buffering and then inter-stream synchronization is ensured by maintaining the buffering alignment of different streams.

It occurs in two steps for every data stream independently: 1) *output time decision*, and 2) *adaptive control algorithm for buffering*.

Output Time Decision: in this step, the output time of the each sensor measurement (T_{out}) is decided. A virtual clock-timer is employed on the acquisition system and the sensor measurements are streamed out with respect to the its timeline. The virtual timer can be set-back or advanced, thereby controlling the buffering times. We define three output cases *wait*, *nowait* and *discard* where the sensor measurement is buffered, streamed out immediately and discarded, respectively. The output case is decided by comparing the time of arrival of the sensor measurement at the acquisition system, recorded by the virtual timer, and the estimated time of its capture. The virtual timer is initialised to the arrival time of the first sensor data frame. The virtual timer value at current time is denoted T_{vt}. The conditions for each of the output cases are:

- **Wait** case: if the arrival time of the incoming sensor measurement (recorded by the virtual timer) is lesser than the estimated measurement capture time, means that the current measurement arrived earlier, compared to the previous sensor measurement, and hence, it needs to be buffered. The *wait* case condition for sensor frame k is

$$T_{arr}[k] \equiv T_{vt} < \hat{T}_{mea}[k]. \tag{15}$$

  In this case, the frame is buffered until the virtual timer reaches $\hat{T}_{mea}[k]$. Therefore, the buffering time is

$$\Delta\tau_{buffer}[k] = \hat{T}_{mea}[k] - T_{arr}[k]. \tag{16}$$

- **Nowait** case: the incoming sensor measurement is considered to arrive late if its arrival time (recorded by the virtual timer) is larger than the estimated measurement capture time. In this case, if its arrival time falls within the intra-SPD threshold, the frame is streamed immediately. The *nowait* case condition for sensor frame k is

$$\hat{T}_{mea}[k] < T_{arr}[k] \equiv T_{vt} < \hat{T}_{mea}[k] + Th.\Delta\phi_{intra}. \tag{17}$$

- **Discard** case: if the incoming frame is too late that its arrival time falls out of the maximum intra-SPD allowed, it is discarded. The *discard* case condition for sensor frame k is

$$T_{arr}[k] \equiv T_{vt} \geq \hat{T}_{mea}[k] + Th.\Delta\phi_{intra}. \tag{18}$$

An adaptive control algorithm for buffering is employed to keep the synchronization error versus buffering time latency trade-off in check. The control algorithm keeps a count of each of the output cases. At any point in time, the count values of the *wait*, *nowait* and *discard* cases represent the arrival distributions of the sensor measurements. Hence, based on this values and certain pre-set thresholds, the algorithm determines whether the sensor measurements are being under- or over-buffered over the past window of time. Furthermore, the algorithm setbacks or advances the virtual timer depending on under-buffer and over-buffer conditions, accordingly.

- **Under-buffer** case: if the the count of *nowait* or *discard* cases is greater than its threshold (Th_{nowait} and $TH_{discard}$). *Nowait* and *discard* cases introduce synchronization errors because the frames are not streamed out at $\hat{T}_{mea}$. So, when under-buffering is detected, the virtual timer is set-back to ensure higher buffering times and reduce the synchronization error. The set-back displacement (ΔT_{vt}^{-}) is

$$\Delta T_{vt}^{-} = (1 - \#wait/Th_{wait})\, max.\Delta T_{vt} \tag{19}$$

  where $\#wait$ is the count of *wait* cases, and $max.\Delta T_{vt}$ is the maximum shift allowed. Further, the count values of *nowait* and *discard* are reset to 0.
- **Over-buffer** case: if the count of *wait* cases is greater than its upper threshold and *nowait* and *discard* counts are below a lower threshold (LTh_{nowait} and $LTh_{discard}$). The virtual timer is advanced to reduce the latency of future frames, and $\#wait$ is reset to 0.

$$\Delta T_{vt}^{+} = (1 - \#nowait/Th_{nowait})\, max.\Delta T_{vt} \tag{20}$$

Intra-stream synchronization is established by equalising measurement latencies, thereby, ensuring that the offset between the virtual arrival time and the measurement capture time of the sensor measurement is corrected. In a similar fashion, inter-stream synchronization is established by further buffering sensor measurements in order to align the virtual timers of the two streams. Firstly, a reference sensor stream is selected. The reference stream is the stream whose measurement frames experience larger delays among the considered sensor streams. It can be easily identified as the stream with the smallest virtual timer value. This is because, considering the larger measurement latency, the control algorithm would have initiated set-back calibrations to the stream. Initially, any arbitrary sensor stream can be set as the reference stream. We denote the virtual timer value of the reference stream T_{vt}^{ref}, and the follower one T_{vt}^{fol}. Overall, the idea is to set-back the virtual timer of the follower stream if the offset between the streams is more than a tolerable bound of inter-SPD. The inter-stream offset condition is

$$T_{vt}^{fol} - T_{vt}^{ref} \leq Th.\Delta\phi_{inter}^{ref,fol} - \max(Th.\Delta\phi_{intra}^{ref}, Th.\Delta\phi_{intra}^{fol}). \tag{21}$$

This condition is checked every time a set-back or advance calibration is performed on the reference stream. If the condition is not satisfied, then the follower stream is set-back by the difference between the two sides of the inequality (21). In addition, any advance displacement of the follower stream is cancelled to match with the slower buffering rate of the reference stream.

IV. Experimental Dataset

To verify the accuracy of the temporal synchronization solution, a moving Meccano contraption, as shown in Fig. 2 was set up to be used as a common target for both the radar and lidar sensors. The device consists of a clear and distinguishable target (the twin plates), which revolves around a fixed axis at a constant speed. This setup ensures that the position measurements obtained by both sensors are distinguishable for comparison and that the spatial error in the position measurement is negligible. Thus, the differences in the positions of the target observed from the sensors are solely due to the temporal errors.

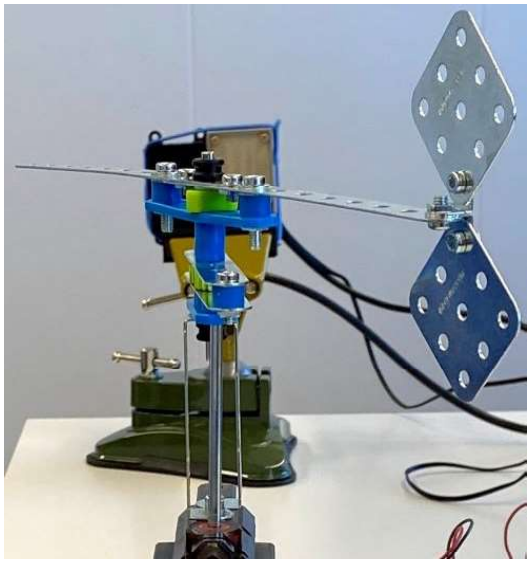

Fig. 2. Rotating device used as a common target of the sensors.

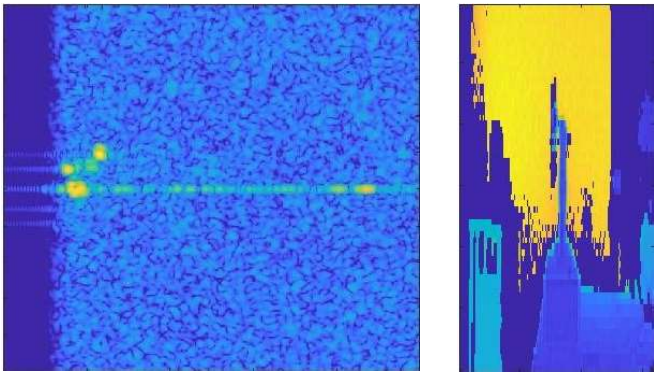

Fig. 3. Example frames of a synchronization event. Left: radar range-Doppler map. Right: lidar depth map.

The radar sensor is a 77GHz radar from RFBeam model MR3003_RD. And the lidar sensor is a solid state lidar from Hypersen model HPS-D160-U. Fig. 3 show the captured radar range-Doppler map and lidar depth map of the rotating device. The position of the revolving plates can be clearly observed from both the range-Doppler map as well as the depth map. To verify the synchronization, the observed positions of the revolving plates by both the sensors are compared at the output of the synchronization system. This setup, though simple, is effective and sufficient enough to evaluate the correctness of the temporal synchronization solution.

For our experiments, the sensor data acquisition and arrival timestamping was performed on a Windows 10 based laptop

TABLE II
MEAN AND VARIANCE OF THE OBSERVED CYCLE TIMES

	Mean (ms)	Variance (ms^2)
$\Delta T^R_{arr}[k-1,k]$	99.998	2.25
$\Delta T^L_{arr}[k-1,k]$	109.315	0.9616

TABLE III
INTER-STREAM RELATIONS FOR DIFFERENT ESTIMATORS

Radar filter	Lidar filter	Average $\Delta T^{R,L}_{out}$ (ms)
none	none	52.709
mean ($W = 16$)	mean ($W = 16$)	43.342
mean ($W = 16$)	mean ($W = 50$)	44.711
mean ($W = 16$)	median ($W = 9$)	27.390
mean ($W = 16$)	median ($W = 59$)	28.742
median ($W = 9$)	mean ($W = 16$)	47.042
median ($W = 9$)	mean ($W = 50$)	47.685
median ($W = 9$)	median ($W = 9$)	37.997
median ($W = 9$)	median ($W = 59$)	37.892

with a 1Gb network interface. The radar data stream comes from an Ethernet port and, the arrival timestamps for the measurements are generated after the measurement is read from the TCP/IP socket. Similarly, the lidar data stream comes from a serial port and, the arrival timestamps are generated after reading each data measurement from the serial port. The time resolution of the timestamps is 1ns and stored as a 64bit unsigned integers.

V. RESULTS

A population size of 5000 frames is used for calculation of the statistical measures. The mean and the variance values of the observed intra-stream relations are summarized in Table II. Fig. 4 shows the observed intra-stream relations at arrival for the radar and lidar sensors, and the estimators at measure time obtained by different filters. The radar shows a repeating pattern of alternating between two Gaussian distributions with variances of 0.00045ms^2 and 0.00046ms^2 centered at 100.59ms and 99.40ms, respectively. The median filter is not very effective in removing this high frequency details. However, on the lidar, that shows a lot of outliers, the median filter is more robust.

Fig. 5 shows the corrected inter-stream relation between synchronization events with different combinations of filters, and Table III their average. We need them to be preferably smaller than approximately half the average cycle time of the sensors ($\Delta T[k-1,k]$) to prevent wrong association with another sensor measurement.

The choice of appropriate $Th.\Delta\phi^R_{intra}$, $Th.\Delta\phi^L_{intra}$, and $Th.\Delta\phi^{R,L}_{inter}$ parameters is based on the Intra-SPD and Inter-SPD. They are set to 0.8ms, 1ms and 2ms, respectively, as shown in Fig. 6 and 7.

The effect of window size (of the adaptive control buffering algorithm) on the total number of setback and advance calibrations of the virtual timer is shown in Fig. 8. The ratio of the output events (wait:nowait:discard) was kept constant as (7:2:1). Next, we observed that the window size did not have significant impact on the number of *wait*, *nowait* and *discard* output cases (Fig. 9). No noticeable trend in buffering latency or synchronization error was observed, however, we cannot expect robust results from smaller window sizes as they may fluctuate the buffering process over small changes in the arrival delay.

TABLE IV
OUTPUT CASES COUNTS FOR DIFFERENT THRESHOLD RATIOS

Th ratio	wait	nowait	discard
1:7:2	739	4239	21
2:6:2	1807	3188	4
3:5:2	2196	2800	3
4:4:2	2424	2572	3
5:4:1	2098	2897	4
6:3:1	2647	2348	4
7:2:1	3548	1148	3
8:1:1	4296	700	3

To analyse the effect of *wait*, *nowait* and *discard* thresholds, results of buffering latency and synchronization error (inter-SPD on *nowait* cases) with varying buffer control configurations are shown in Fig. 10 and 11. The maximum calibrating factor $max.\Delta T_{vt}$ is set to the average delay variation observed in the stream (average intra-SPD), which is 0.6ms and 0.3ms for radar and lidar streams, respectively. This ensures a smooth offsetting of timer reference. Keeping the window size to 100, the total number of each output case for different threshold ratios are presented in Table IV. There is a direct correlation between the corresponding thresholds and the observed number of events. However, it is to be noted that the set threshold do not hard guarantee the same ratio of events at output. We can observe that with a relatively higher Th_{wait}, buffering latency increases and synchronization error decreases. This is expected because the buffer control algorithm will frequently hit the under-buffer condition due to lower Th_{nowait} and $TH_{discard}$ and hence, increases buffering latency by setting back the virtual timer. Overall, the increase in buffering time also ensures lower synchronisation errors. Conversely, we observe that with lower Th_{wait}, synchronisation error increases and buffering latency decreases due to setting of over-buffering condition leading to advance calibration of virtual timer.

VI. HARDWARE IMPLEMENTATION

The design consists of a buffer, virtual timer, filter, and control block per sensor stream; and a common inter-stream control. The buffers store incoming sensor data and eventually streamed them out as AXI4 streams, according to the decision of control block (based on the estimators provided by the filter). First, the arrival time is recorded by the virtual timer block. Next, the measure timestamp is estimated by the filter. The estimated measurement timestamp is available 2 clock cycle after the arrival of the sensor measurement. The control block also checks the over-buffer and under-buffer cases and sends appropriate setback or advance commands to the virtual timer.

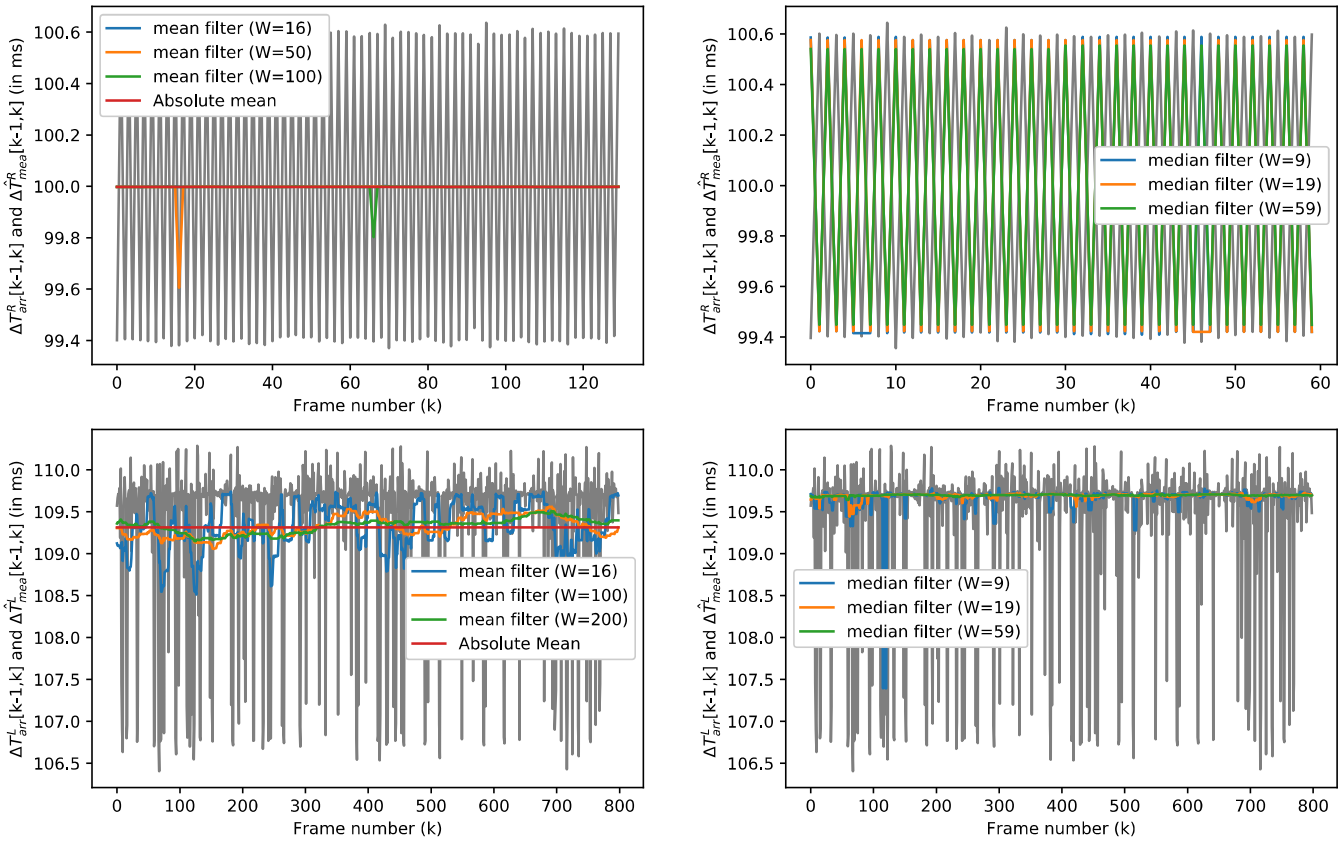

Fig. 4. Observed intra-stream relations at arrival (grey) and estimators obtained by different filters (see legends) for radar (top) and lidar (bottom).

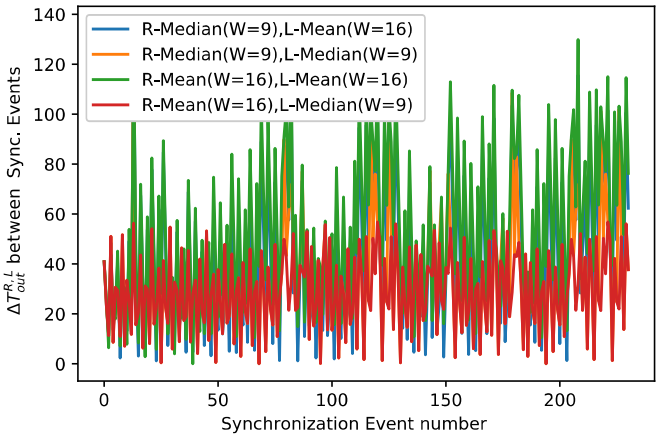

Fig. 5. Inter-stream relation (in ms) between synchronization events with different combinations of filters.

The common inter-stream control block takes care of communications between the streams and ensures inter-stream synchronization. The timestamps are recorded as 64-bit unsigned integers (which is standard for sensor measurement timestamps). A precision of 30ns is used so that drift due to sensor clock, if present, can be taken into account. The design is fully parameterized.

TABLE V
RESOURCE UTILIZATION OF SYNCHRONIZATION BLOCK COMPONENTS

Component	LUT	FF	DSP
Timer	233	136	0
Control	238	92	1
Inter-stream	295	133	0
Full design[a]	2105	1639	2

[a]Excluding buffers.

TABLE VI
RESOURCE UTILIZATION OF FILTERS

Filter	Window	LUT	FF	DSP
Mean	16	378	569	0
	20	296	642	1
	32	438	858	0
	50	567	1183	1
	100	1019	2084	1
	128	1401	2588	0
Median	9	462	443	0
	19	1072	624	0
	59	2240	1345	0

Resource utilization of some components of the implemented solution are presented in Table V obtained in Vivado 2018.3 for a Zynq ZC702 evaluation kit. Table VI shows the resource utilization of different filters.

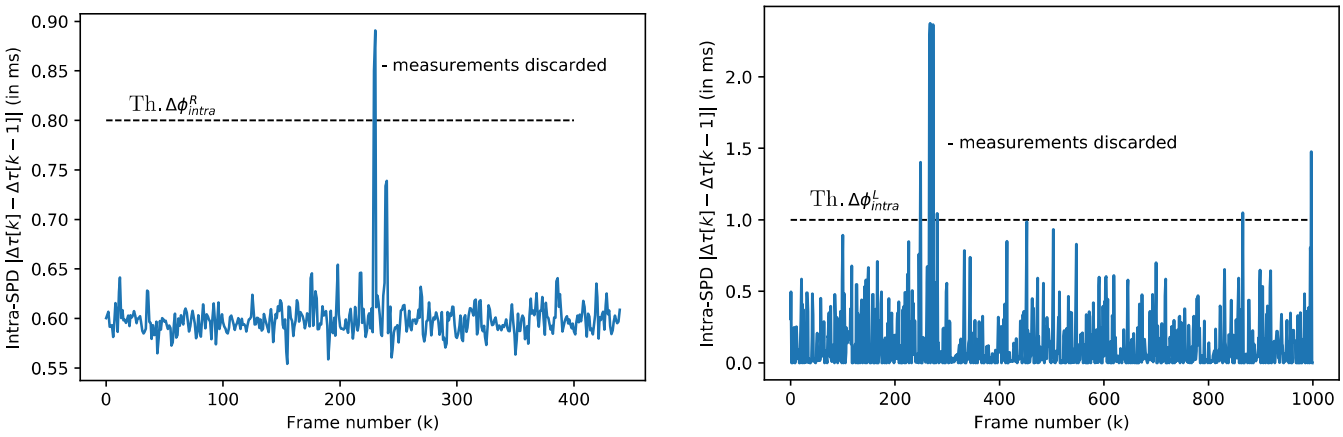

Fig. 6. Intra-SPD and its threshold for radar (left) and lidar (right).

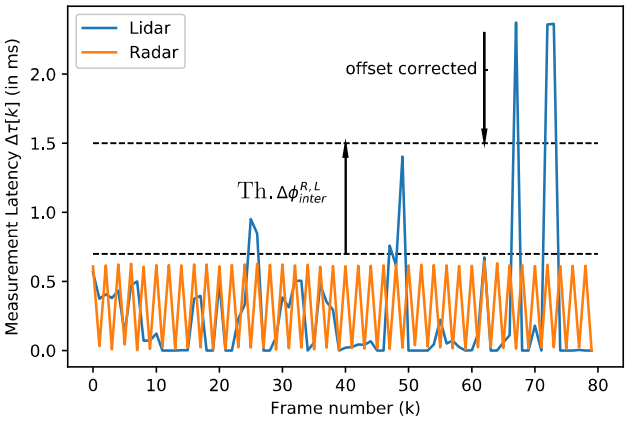

Fig. 7. Measurement latency of adjacent radar and lidar measurements and the maximum inter-SPD threshold.

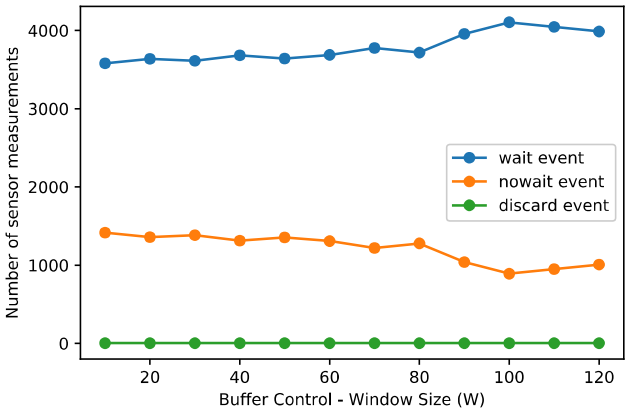

Fig. 9. Effect of window size on the buffering output cases.

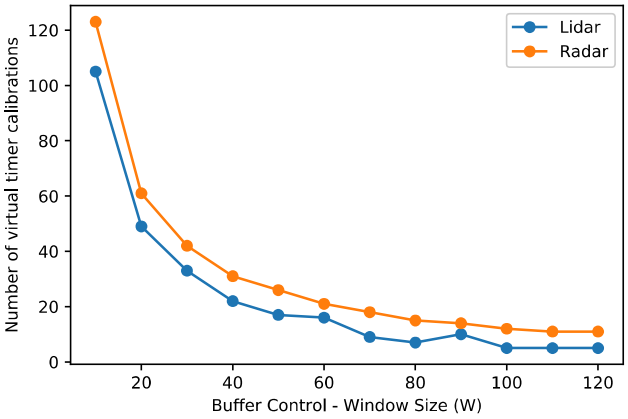

Fig. 8. Effect of window size on the number virtual timer calibrations.

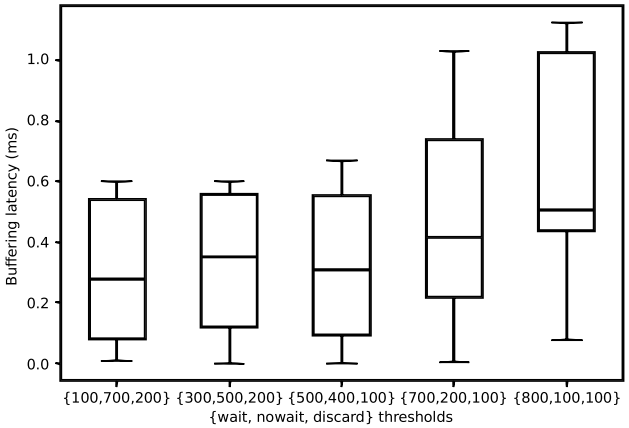

Fig. 10. Box plot of buffering latency for different output cases thresholds.

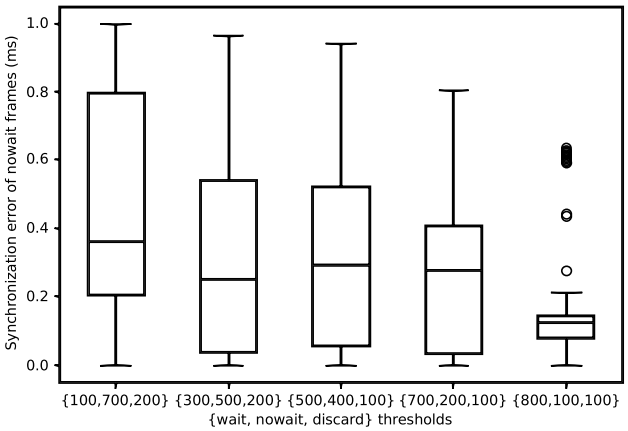

Fig. 11. Box plot of synchronization error caused by streaming-out immediately for different output cases thresholds.

VII. Conclusions

A temporal synchronization of sensor streams is proposed, which is flexible and can be tuned to the right level of latency versus synchronization error trade-off, according to the needs of the application. For the particular application that triggered this research, composed of 2 streams (radar and lidar), the most optimal output case thresholds are (500, 400, 100) with average buffering latency of 0.32ms and synchronization error of 0.316ms. With a mean filter with window size of 16 for the radar stream, and a median filter with window size of 9 for the lidar stream.

We have recorded an easy-to-use dataset with a radar and a lidar sensors without timestamps. The synchronization event is easily identifiable by a human in both sensor streams. This dataset is perfectly suited for other data fusion application tests. Thus, it is currently on the process of being published.

Finally, an efficient hardware implementation of the synchronization block have been developed, which has a low resource utilization.

It is to be noticed that the acquisition of sensor measurements was done on a machine with Windows10 OS, which does not guarantee real-time requirements and can introduce uncertain delays. Hence, the observed cycle times used in our analysis are not representative of the cycle times observed when sensor acquisition is done directly on hardware or on a real-time operating system.

References

[1] C. Kwok, D. Fox, and M. Meil, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 469–484, 2004.

[2] V. Di Lecce, A. Amato, and M. Calabrese, "Gps-aided lightweight architecture to support multi-sensor data synchronization," in *IEEE Instrumentation and Measurement Technology Conference*, 2008, pp. 149–154.

[3] M. Kais, D. Millescamps, D. Bétaille, B. Lusetti, and A. Chapelon, "A multi-sensor acquisition architecture and real-time reference for sensor and fusion methods benchmarking," in *IEEE Intelligent Vehicles Symposium*, 2006, pp. 418–423.

[4] D. T. Knight, "Achieving modularity with tightly-coupled gps/ins," in *IEEE PLANS 92 Position Location and Navigation Symposium Record*, 1992, pp. 426–432.

[5] B. Li, "A cost effective synchronization system for multisensor integration," in *Proceedings of the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS)*, 2004, pp. 1627–1635.

[6] A. Westenberger, T. Huck, M. Fritzsche, T. Schwarz, and K. Dietmayer, "Temporal synchronization in multi-sensor fusion for future driver assistance systems," in *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2011, pp. 93–98.

[7] J.-O. Nilsson and P. Händel, "Time synchronization and temporal ordering of asynchronous sensor measurements of a multi-sensor navigation system," in *IEEE/ION Position, Location and Navigation Symposium*, 2010, pp. 897–902.

[8] M. Chen, "A low-latency lip-synchronized videoconferencing system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '03, 2003, p. 465–471.

[9] I. Kouvelas, V. Hardman, and A. Watson, "Lip synchronisation for use over the internet: analysis and implementation," in *Proceedings of GLOBECOM'96 IEEE Global Telecommunications Conference*, vol. 2, 1996, pp. 893–898.

[10] F. Boronat, J. Lloret, and M. García, "Multimedia group and inter-stream synchronization techniques: A comparative study," *Information Systems*, vol. 34, no. 1, pp. 108–131, 2009.

[11] M. Montagud, P. Cesar, F. Boronat, and J. Jansen, *MediaSync: Handbook on multimedia synchronization*. Springer, 2018.

[12] S. Baqai, M. Farrukh Khan, M. Woo, S. Shinkai, A. A. Khokhar, and A. Ghafoor, "Quality-based evaluation of multimedia synchronization protocols for distributed multimedia information systems," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1388–1403, 1996.

[13] T. D. C. Little and A. Ghafoor, "Interval-based conceptual models for time-dependent multimedia data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 4, pp. 551–563, 1993.

[14] D. P. Anderson and G. Homsy, "A continuous media i/o server and its synchronization mechanism," *Computer*, vol. 24, no. 10, pp. 51–57, 1991.

[15] K. Ravindran and V. Bansal, "Delay compensation protocols for synchronization of multimedia data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 4, pp. 574–589, 1993.

[16] S. Tasaka and Y. Ishibashi, "Media synchronization in heterogeneous networks: stored media case," *IEICE Transactions on Communications*, vol. 81, no. 8, pp. 1624–1636, 1998.

[17] ——, "A performance comparison of single-stream and multi-stream approaches to live media synchronization," *IEICE Transactions on Communications*, vol. 81, no. 11, pp. 1988–1997, 1998.

[18] Y. Ishibashi, T. Kanbara, and S. Tasaka, "Inter-stream synchronization between haptic media and voice in collaborative virtual environments," in *Proceedings of the 12th Annual ACM International Conference on Multimedia*, ser. MULTIMEDIA '04. Association for Computing Machinery, 2004, pp. 604–611.

[19] K. Rothermel and T. Helbig, "An adaptive stream synchronization protocol," in *Network and Operating Systems Support for Digital Audio and Video*. Springer, 1995, pp. 176–189.

[20] Y. Xie, C. Liu, M. J. Lee, and T. N. Saadawi, "Adaptive multimedia synchronization in a teleconference system," *Multimedia Systems*, vol. 7, no. 4, pp. 326–337, 1999.

[21] Y. Ishibashi and S. Tasaka, "A synchronization mechanism for continuous media in multimedia communications," in *Proceedings of INFOCOM'95*, vol. 3, 1995, pp. 1010–1019.

[22] J. Escobar, C. Partridge, and D. Deutsch, "Flow synchronization protocol," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 111–121, 1994.

[23] H. Liu and M. El Zarki, "A synchronization control scheme for real-time streaming multimedia applications," in *Packet Video*, vol. 2003, 2003.