Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2021-5082501

M.Sc. Thesis

Temporal Synchronization of Sensors

Tanmay Manjunath

Abstract

Advanced automotive vehicles are based on the real-time fusion of an increasing number of automotive sensors. For precise fusion of different sensors, measurements need to be synchronized both temporally and spatially. This thesis aims to design a hardware temporal synchronization block as part of the PRISTINE[1] systolic array accelerator project for multi-sensor data fusion. In this process, we study and address several temporal sensor synchronization issues that are characteristic of the considered system as well as any other typical sensor fusion system. First and foremost, we handle the problem of estimating the actual time of sensor measurement by exploring well-known filtering techniques such as Kalman, mean and median filters. A suitable filter is selected for implementation based on the statistical characteristics of the observed sensor cycle times, the complexity of the filters and the quality of obtained estimates. 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. An analysis of the effects of the stream synchronization algorithm's parameters on buffering latency and synchronization error was presented. Finally, the above synchronization solution was efficiently implemented on hardware by making certain modifications and design decisions to the algorithm. A method to evaluate the whole temporal synchronization process is proposed and the obtained results on real sensor data are presented.

Temporal Synchronization of Sensors

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Embedded Systems

by

Tanmay Manjunath
born in Bangalore, India

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Temporal Synchronization of Sensors"** by **Tanmay Manjunath** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: October 26, 2021

Chairman: _____

Prof. Dr. Ir. Rene van Leuken

Advisor: _____

Dr. Ir. David Aledo Ortega

Committee Members: _____

Dr. Ir. Raj Thilak Rajan

_____

Abstract

Advanced automotive vehicles are based on the real-time fusion of an increasing number of automotive sensors. For precise fusion of different sensors, measurements need to be synchronized both temporally and spatially. This thesis aims to design a hardware temporal synchronization block as part of the PRISTINE[1] systolic array accelerator project for multi-sensor data fusion. In this process, we study and address several temporal sensor synchronization issues that are characteristic of the considered system as well as any other typical sensor fusion system. First and foremost, we handle the problem of estimating the actual time of sensor measurement by exploring well-known filtering techniques such as Kalman, mean and median filters. A suitable filter is selected for implementation based on the statistical characteristics of the observed sensor cycle times, the complexity of the filters and the quality of obtained estimates. 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. An analysis of the effects of the stream synchronization algorithm's parameters on buffering latency and synchronization error was presented. Finally, the overall synchronization solution was efficiently implemented on hardware by making certain modifications and design decisions to the algorithm. A method to evaluate the whole temporal synchronization process is proposed and the obtained results on real sensor data are presented.

Acknowledgments

Firstly, I would like to thank David Aledo Ortega for his able guidance throughout this thesis. Also, I genuinely appreciate him for checking in on my well-being and health during these uncertain times. I would also like to thank Rene van Leuken and Raj Thilak Rajan for their valuable support and feedback, which immensely helped me to improve and shape my thesis. Additionally, I would like to acknowledge Darek Maksimiuk for his assistance with the sensors during my thesis work.

I am extremely thankful to my friends for their love and support. Finally, I would like to express my gratitude towards my parents for their unconditional support, patience and encouragement throughout my life.

Tanmay Manjunath
Delft, The Netherlands
October 26, 2021

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Background

Over the last decade, research in the area of autonomous vehicle technologies has advanced at a significant pace. To invariably describe the complete range of autonomous driving features, the SAE has defined 6 levels of driving automation based on the amount of intervention and attentiveness required from the driver [3]. An overview of these levels is shown in Figure 1.1. An important milestone in these levels is the shift from levels SAE-2 to SAE-3 on-wards, where the vehicle is fully responsible for environmental perception and the driver no longer has to observe the surrounding environment. In addition, at higher levels of automation, the vehicle is required to be fail-operational, as it is responsible for handling safety-critical functions as well.

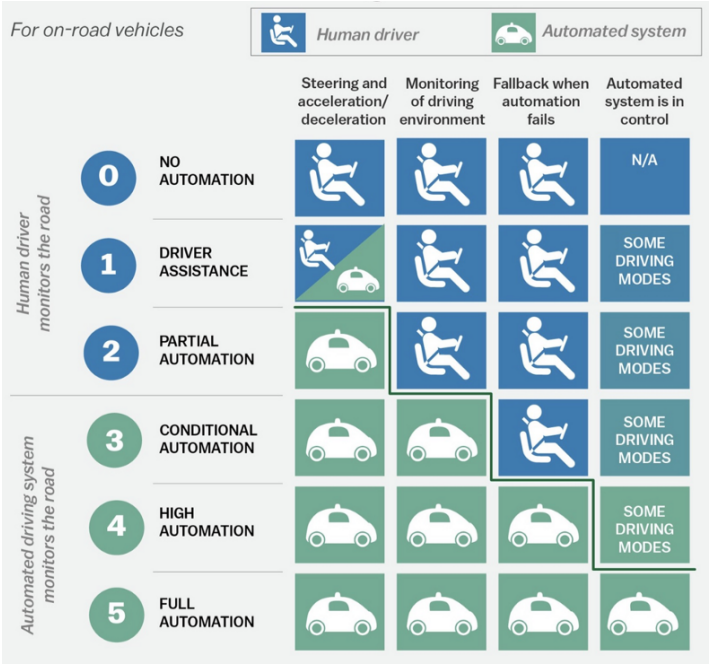

Figure 1.1: SAE Levels of Automation [2]

Environmental perception for automated driving requires a robust and reliable fusion of sensor measurements that may be supplementary or complementary. A great extent of research has been done in the field of sensor fusion to merge the coverage of different sensors in time and space and to exploit the redundancies in complementary sensor measurements.

Some of the commonly used sensor fusion modalities are namely, spatial, temporal,

and complementary sensor fusion. Spatial fusion combines the coverage areas of individual sensors to offer either a 360-degree coverage of the surrounding environment or substantial coverage around the vehicle, or both. Temporal fusion integrates the results of several sensor readings to create a history and make it easier to forecast the status of the environment in the future. Complementary sensor fusion combines measurements having complementary capabilities such as reliability, resolution, etc., to make sure the system is fail-operational.

A crucial step that needs to be done precisely before sensor data fusion is the sensor data association. Data association is a method to associate the right pair of the measurements from two or more sensor sources to achieve optimal sensor fusion result [4]. In general, data association includes temporal and spatial synchronization of sensor measurements. Temporal synchronization involves providing reliable relative timing information to maintain the same temporal relationships between the sensor measurements. With this, the sensor inputs can be provided coherently from the same instant or period. On the other hand, spatial synchronization involves ensuring that the fields of view of the different sensors coincide or match.

This thesis aims to provide a solution to the problem of temporal synchronization of Radar and Lidar measurements. Further, this solution is realized on FPGA hardware as a part of a systolic array accelerator for multi-sensor data fusion.

1.2 Motivation

Advanced automotive vehicles are based on the real-time fusion of an increasing number of automotive sensors. For precise aggregation or fusion of measurements from different sensors, all the measurements must point to the same ground reality. To achieve this, the sensor measurements need to be time-aligned/synchronized in the same manner as they were initially recorded by the sources. Figure 1.2 shows the disassociation between measurements from two sensors, both providing 10 frames per second, if they are aligned based on their frame numbers, i.e., first-in-first-out manner upon arrival. Incorrect alignment of different sensor measurements leads to wrong position or velocity calculations of target objects, thereby leading to poor data fusion performance. This can cause serious consequences in time-critical situations like pre-crash and emergency braking conditions [5, 6]. Thus, even the best sensor fusion algorithm will result in sub-par results without precise time synchronization of sensor measurements. Overall, the issue of time synchronization of sensor data is crucial in the implementation of multi-sensor fusion systems and has to be handled first and foremost in its design.

Present-day automotive sensors can be widely categorised based on their capability to be externally triggered to produce measurements at required times and their on-sensor measurement timestamping capabilities. Sensors without external trigger capabilities, commonly known as free-running sensors, continuously produce sensor data at varying intervals of time. These sensors can overall be regarded as independent subsystems with separate clocks and unreliable time periods between measurements. In case these sensors do not provide measurement timestamps, the arrival times of the sensor data is the only timing information that is available to the application. Arrival times do not best represent the original timing relationships between the sensor

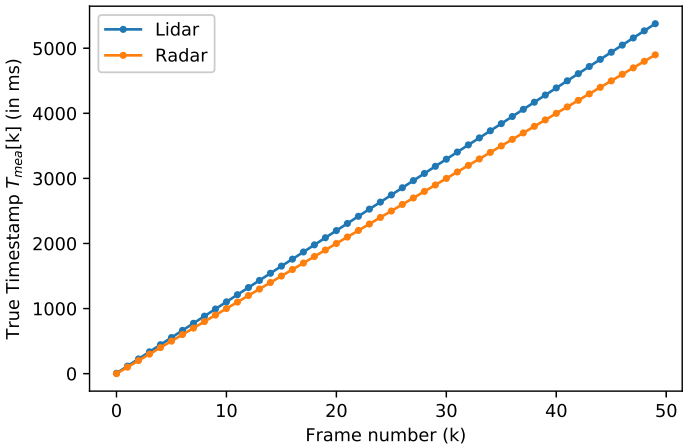

Figure 1.2: Sensor Data Disassociation.

measurements since, different sensors capture measurements at different rates, perform different pre-processing steps and transmit measurements through various hardware interfaces. All these factors add an unknown jittering latency to the arrival times of these measurements at the acquisition system. Also, sensors can have different initial startup times, which again impacts the timing alignment of the measurements. These unknown latencies and jitters can add up to considerable delay values, very soon, and hence can lead to the improper association of sensor measurements if it were to be aligned with respect to their arrival times.

Similarly, in the case of externally trigger-able sensors where no sensor timestamps are provided, the trigger time of the sensor may not be reliable enough to be used as actual measurement capture time since there is still a possibility of a variable delay between the trigger time and the actual capture time. Further, an additional mechanism needs to be employed by the application to generate the trigger signals for the sensors. Also, sensors with time reliable externally trigger capabilities can be very expensive and cannot be a viable option for many applications where multiple sensors are required. Lack of on-sensor timestamps and unreliable timing information of sensor measurements motivates the need to solve the problem of accurate estimation of the actual time of sensor measurement.

In the case of sensors with timestamping capability, it is to be noted that sensors have their own clocks and time references and hence, the time references of different sensors need to be matched before the timestamps can be used. These sensors are often expensive and there is still a need to synchronize arriving jittery sensor data streams according to the measurement timestamps at the acquisition end of the application system.

In addition to synchronizing sensor data streams according to the true sample time of measurements, real-time multi-sensor fusion systems are also required to keep in line with its real-time constraints and altogether ensure minimal latency. As these systems are time-critical, the overall system's worst-case latency is of significant importance.

Overall, the above-mentioned requirements and challenges motivate the need to design temporal synchronization solutions for advanced multi-sensor fusion systems with real-time requirements.

1.3 Aim and Scope

This thesis aims to design a hardware temporal synchronization block as part of the PRISTINE systolic array accelerator project for multi-sensor data fusion. This thesis addresses several temporal sensor synchronization issues that are characteristic of the considered accelerator system as well as any other typical sensor fusion system. They are as follows :

- Handle the problem of estimating the actual time of sensor measurement.
- Ensure synchronization of incoming sensor streams with minimal latency and synchronization error while tolerating uncertain arrivals of sensor data.
- Efficient implementation of the synchronization solution on hardware.
- Develop a methodology to evaluate the performance of the temporal synchronization solution on real world data.

1.4 Thesis Outline

This thesis report is structured as follows.

- Chapter 2 presents a brief description of PRYSTINE's [6] systolic array demonstrator platform. Further, the problem of temporal asynchrony between the sensor data is formally modeled.
- In Chapter 3, a few filtering techniques in the literature are reviewed for the estimation of the true sensor measurement cycle time. In addition, a method to extract timestamps of sensor measurements is explained.
- In Chapter 4, existing works in the field of data stream synchronization is presented. Further, a detailed explanation of the implemented stream synchronization is given.
- In Chapter 5, an experiment to verify the timestamp estimation and stream synchronization is developed. Results of timestamp estimation and data stream synchronization algorithm are presented and analyzed.
- Chapter 6 presents the details of the hardware implementation of the synchronization solution. Further, the implemented hardware is evaluated and the results are presented.
- Finally in Chapter 7, Conclusions, potential improvements and the continuation of the work is outlined.

System Description and Problem Modelling

2

2.1 Overview

In this chapter, a brief description of PRYSTINE's [1] systolic array demonstrator platform is presented. Further, the problem of temporal asynchrony between the sensor data is formally modeled. With the formal model defined, it is easier to understand the requirements of the problem and to precisely formulate algorithms for synchronization.

2.2 Systolic Array Accelerator : Architecture

The systolic array accelerator is a hardware accelerator architecture designed for environment perception based on multi-sensor data fusion application. The accelerator itself is realised using FPGA hardware. Figure 2.1 shows the FPGA architecture of the systolic array demonstrator. The main components of the demonstrator are described below:

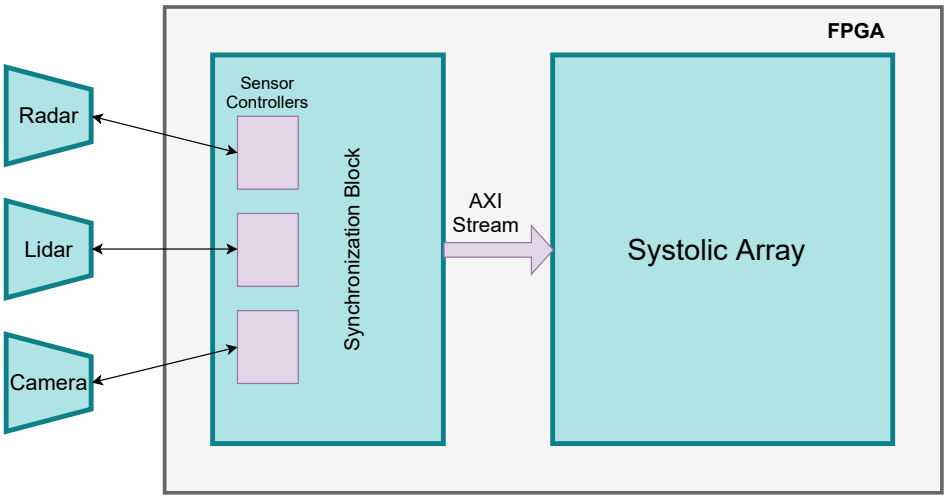

Figure 2.1: FPGA architecture of the Systolic Array demonstrator.

1. Sensors

The system employs three sensors namely, Radar, Lidar, and Camera. All the employed sensors are free-running sensors and hence, cannot be externally triggered and are required to be directly interfaced either with a computer system or a FPGA platform. An account of the properties of the sensors are presented below:

- **Radar**: Radar sensors measure the distance and velocity of an object by transmitting a high-frequency electromagnetic signal and observing the object's reflection of these waves. More precisely, the distance to the object is calculated using the round-about time of the signal and the radial velocity is estimated using the frequency shift of the reflected signal.

  The Radar sensor employed in this system is produced by RFBeam (Switzerland) and, is based on an NXP Radar chipset (MR3003) and a specialized MCU (S32R274) to perform onboard Radar signal processing. The sensor has an Ethernet interface and can produce and transmit measurements at a rate of 10 frames per second. The sensor provides no guarantees on a constant sampling period of the sensor measurements and also, no timestamps for the actual time of measurement capture are provided. Hence, the exact capture/sampling times of the measurements are unknown and cannot be directly derived from sensor's sampling period.

- **Lidar**: Similar to the Radar approach, Lidar sensors measure the distance of a target object by emitting light pulses and observing the reflected pulse. Further, Lidar sensors produce a 3D map of the surrounding environment by integrating many such precise distance measurements. These maps not only give information on the object's position but also aids in the detection and identification of the object.

  The Lidar sensor used in this system is a solid-state Lidar produced by Hypersen Technologies Ltd (China). The sensor can operate in two different settings: single shot and continuous modes. The sensor produces and transmits measurements through a serial USB interface at a rate of 10 frames per second and 30 frames per second in a single shot and continuous mode, respectively. Similar to Radar, the considered Lidar sensor does not ensure reliable sampling periods or provide any information on the measurement capture time.

- **Camera**: The camera used in this system is an industrial camera sensor (mvBlueCOUGAR) produced by Matrix Vision GmbH. This camera sensor can be programmed to be externally triggered or to operate in free-running mode. In free-running mode, it can produce high-resolution images at the rate of 10 frames per second. It has a gigabit ethernet interface and can also perform several on-chip operations such as Flat field correction, color correction, etc. on the image to reduce the load of the host system. In addition, the camera sensor provides timestamps of image capture which can be used synchronization purposes.

A summary of the sensor properties is presented in Table 2.1.

2. Sensor Controllers

The purpose of the sensor controllers is to configure and read the respective sensors' incoming data. The controllers are to be designed based on the individual sensor's bus interface and controllers for some interfaces such as ethernet, SPI, etc. are readily available as IP blocks.

Table 2.1: Summary of Sensor Properties

Sensor	Interface	Rate	Size (bytes/measurement)
Radar	Ethernet	10 fps	262144 (raw samples)
Lidar	USB	10 fps	153600 (Depth map + Point Could)
Camera	Ethernet	10 fps	-

3. Synchronization Block

The synchronization block has to work alongside the sensor controllers to maintain the original temporal relationships between the sensor measurements. Considering the non-availability of actual sensor measurement times in the case of our Radar and Lidar sensors, the synchronization block is responsible for both estimating the actual measurement times and synchronization of independent sensor data streams. The output of this block would be AMBA AXI4 streams of sensor measurements which are provided as fusion input.

4. The Systolic Array

In this system, Convolutional Neural Networks (CNN) is used for performing sensor data fusion. The synchronized input streams from different sensors are applied as input features in the CNN and are merged in different layers of the CNN to produce desired fusion outputs. A systolic array-based computational unit is implemented for computing CNNs since it is proven to be fast and energy-efficient.

2.3 Events

To formulate the problem formally, we first define all the events occurring in the system. An event is essentially an occurrence of something at a particular place and time. Let us consider a section of the Systolic array accelerator which includes the sensors (Radar and Lidar) and the sensor data acquisition system. Figure 2.2 shows a detailed block diagram along with all the events that are essential for our problem formation. The different events that are defined, are listed in Table 2.2.

With our knowledge of the working of the system, we can say that the events $(e1, e2, e3)$ and $(e4, e5, e6)$ occur in a sequence. With this, we can assume the existence of two processes with the process defined as a sequence of events, namely,

$$\text{Radar acquisition process } (P^R) \text{ - } (e1 => e2 => e3)$$
$$\text{Lidar acquisition process } (P^L) \text{ - } (e4 => e5 => e6)$$

The superscript R and L denote the Radar and Lidar sensors, respectively and this representation is used throughout this report. The acquisition processes P^R and P^L occur asynchronously and are independent of each other. It is to be noted that we cannot make any inference on the 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 run time.

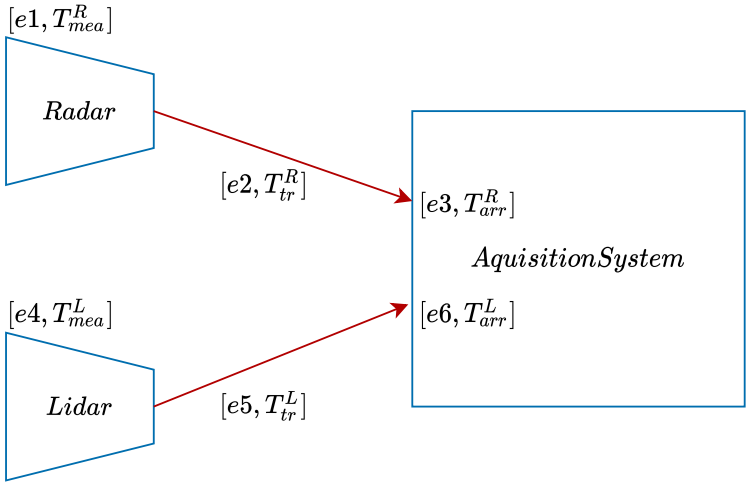

Figure 2.2: System Block diagram with Events and Timestamps.

Table 2.2: System Events and its corresponding Timestamps.

Events	**Timestamps**
e1 Radar measurement	T^R_{mea}
e2 Radar measurement transmitted onto the communication link	T^R_{tr}
e3 Arrival of Radar frame at acquisition system	T^R_{arr}
e4 Lidar measurement	T^L_{mea}
e5 Lidar measurement transmitted onto the communication link	T^L_{tr}
e6 Arrival of Lidar frame at acquisition system	T^L_{arr}

2.4 Timestamps

With the events defined, we now introduce the notion of time associated with them. For every event, a timestamp is assigned which represents the point in time at which the event occurred. The events and their corresponding timestamps are listed in Table 2.2.

It is to be noted that 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 and hence there exists a shared timeline for the arrivals making it easier to compare the arrival times of the two streams.

Overall, in the considered system, none of the above timestamps except for arrival times at the acquisition system (T^R_{arr} and T^L_{arr}) can be measured/observed. A pretty straightforward solution to this issue would 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.

2.5 Sources of Delay and Delay Variability factors

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. For automotive applications with safety and real-time requirements, these delays are detrimental to their reliability and dependability. In addition, delays are different for each sensor data stream resulting in data of different sensors belonging to completely different periods of time being given as input 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 as follows:

- **Processing delays:** Pre-processing 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 our model, the sensor stream is subjected to these delays between events $e1$ and $e2$ in Radar, and between $e4$ and $e5$ events in the Lidar sensor system.

- **Communication delays**: Communication delay is the delay 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. The delays due to queuing of these frames at the communication interface, the physical transmission of the frames onto the communication link, and the actual propagation of the sensor measurement to the destination, all constitute communication delays. Concerning our model, the sensor stream is subjected to these delays between events $e2$ and $e3$ in Radar, and between $e5$ and $e6$ events in the Lidar sensor system.

- **Data acquisition delays**: It includes delays at the receiving end of the communication link at the acquisition system such as queuing and de-packetization delays of the arriving frames. In our model, these delays occur during $e3$ and $e6$ events of the Radar and Lidar processes, respectively.

All the delay factors listed above depend on the properties of the sensors and the acquisition system such as, (i) the type of sensor pre-processing performed, performance specifications of the processing units; (ii) the communication protocol employed and the specifications of the communication link (e.g., bandwidth). In addition, these delays are associated with some form of variability or jitter. This may be caused due to several factors such as network jitters, jitters in processing times due to heating of processing hardware, etc.

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

$$\begin{aligned} \Delta\tau^R[k] &= T^R_{arr}[k] - T^R_{mea}[k] \\ \Delta\tau^L[k] &= T^L_{arr}[k] - T^L_{mea}[k] \end{aligned} \tag{2.1}$$

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

$$\begin{aligned} \Delta\tau^R[k] &= \sum_{i=1}^{N} d_i^R + \delta_{jitter}[k] \\ \Delta\tau^L[k] &= \sum_{i=1}^{M} d_i^L + \delta_{jitter}[k] \end{aligned} \tag{2.2}$$

where $\delta_{jitter}[k]$ denotes the jitters in delays for k^{th} sensor measurement and, d_i^R and d_i^L $(i = 1, 2, ..NorM)$ represent the mean of their respective sensor specific delays due to N or M delay factors.

2.6 Sensor Data Model

Before we model the synchronization problem, a description of the components of the sensor data stream is presented. The entire sensor data stream model can be described using the following terms:

- **Data Frame**: A Data frame refers to a single information unit that is transmitted by the sensor through the communication link. We denote the k^{th} frame of Radar and Lidar streams as f_k^R and f_k^L, respectively. A single measurement captured by the sensor at a point in time can be transmitted as several data frames. The number of data frames per measurement is based on the sensor measurement size and the communication protocol employed. We denote k^{th} measurement of Radar and Lidar streams as m_k^R and m_k^L, respectively.
- **Data Stream**: A Data Stream is essentially a sequence of data frames transmitted by the same sensor over time. We denote the data streams by Radar and Lidar as s^R and s^L, respectively.
- **Synchronization reference**: Synchronization reference is either a data frame or data stream that other data frames or data streams need to be synchronized against.

2.7 Problem Model

With events, timestamps, and delays of the system defined, we now mathematically formulate the problem. On the top level, we aim to provide the sensor data streams to the systolic array while maintaining the same temporal relationships between the measurements in which they were originally recorded. We define two types of temporal relations in our model:

1. **Intra-Stream relations**: Intra-Stream relation refers to the temporal relationship between the frames belonging to the same stream. More precisely, we consider

the Intra-Stream relation as the time difference between events of two consecutive sensor measurements captured by the same sensor. Equations 2.3, 2.4 and 2.5 gives the Intra-Stream temporal relation between $[k-1]^{th}$ and k^{th} sensor measurements during capture, on arrival at acquisition system and after synchronization, respectively.

$$\begin{aligned} \Delta T^R_{mea}[k-1,k] &= T^R_{mea}[k] - T^R_{mea}[k-1] \\ \Delta T^L_{mea}[k-1,k] &= T^L_{mea}[k] - T^L_{mea}[k-1] \end{aligned} \tag{2.3}$$

$$\begin{aligned} \Delta T^R_{arr}[k-1,k] &= T^R_{arr}[k] - T^R_{arr}[k-1] \\ \Delta T^L_{arr}[k-1,k] &= T^L_{arr}[k] - T^L_{arr}[k-1] \end{aligned} \tag{2.4}$$

$$\begin{aligned} \Delta T^R_{out}[k-1,k] &= T^R_{out}[k] - T^R_{out}[k-1] \\ \Delta T^L_{out}[k-1,k] &= T^L_{out}[k] - T^L_{out}[k-1] \end{aligned} \tag{2.5}$$

In the board sense, Intra-Stream temporal relations essentially expresses the sensor's cycle time. Hence, for simplicity, $T_{mea}[k-1,k]$ and $T_{arr}[k-1,k]$ are also referred to as true and observed measurement cycle times, respectively.

2. **Inter-Stream relations**: Inter-Stream relations refers to the temporal relationship between corresponding frames belonging to different streams. In our formulation, we consider the Inter-Stream relation as the delay difference between events of two corresponding sensor measurements captured by different sensors. Equations 2.6, 2.7 and 2.8 gives the Inter-Stream temporal relation between between k^{th} Radar and Lidar measurements during capture of said measurements, on arrival at acquisition system and after synchronization, respectively.

$$\Delta T^{R,L}_{mea}[k] = T^R_{mea}[k] - T^L_{mea}[k] \tag{2.6}$$

$$\Delta T^{R,L}_{arr}[k] = T^R_{arr}[k] - T^L_{arr}[k] \tag{2.7}$$

$$\Delta T^{R,L}_{out}[k] = T^R_{out}[k] - T^L_{out}[k] \tag{2.8}$$

The above defined temporal relations are illustrated in Figure 2.3.

Now, consider the Figure 2.4 and Figure 2.5 in which a shared timeline of Radar and Lidar acquisition processes (P^R and P^L) for two consecutive measurements is illustrated. In these timing models, T_{mea} and T_{arr} denote the sensor measurement time and the arrival time at the acquisition system, respectively. T_{out} represents the time at which the sensor measurement needs to be outputted to maintain temporal synchronization.

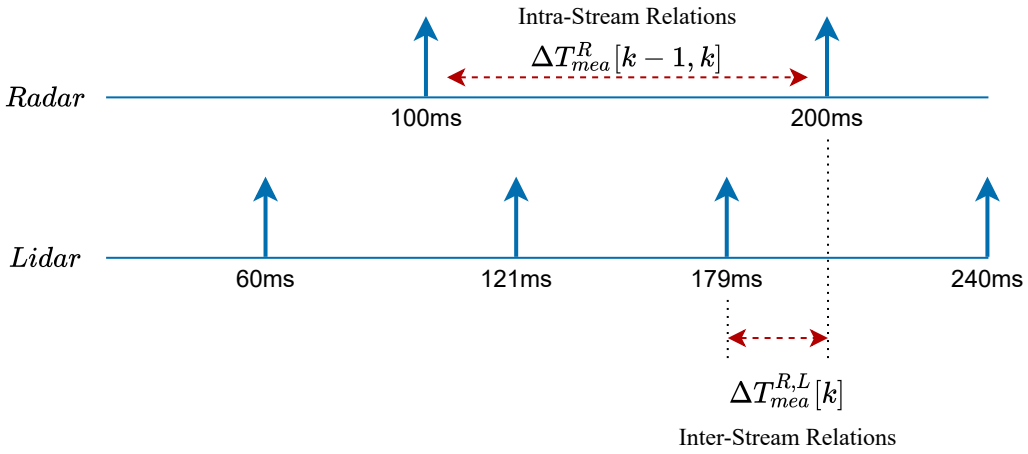

Figure 2.3: Temporal Relationships.

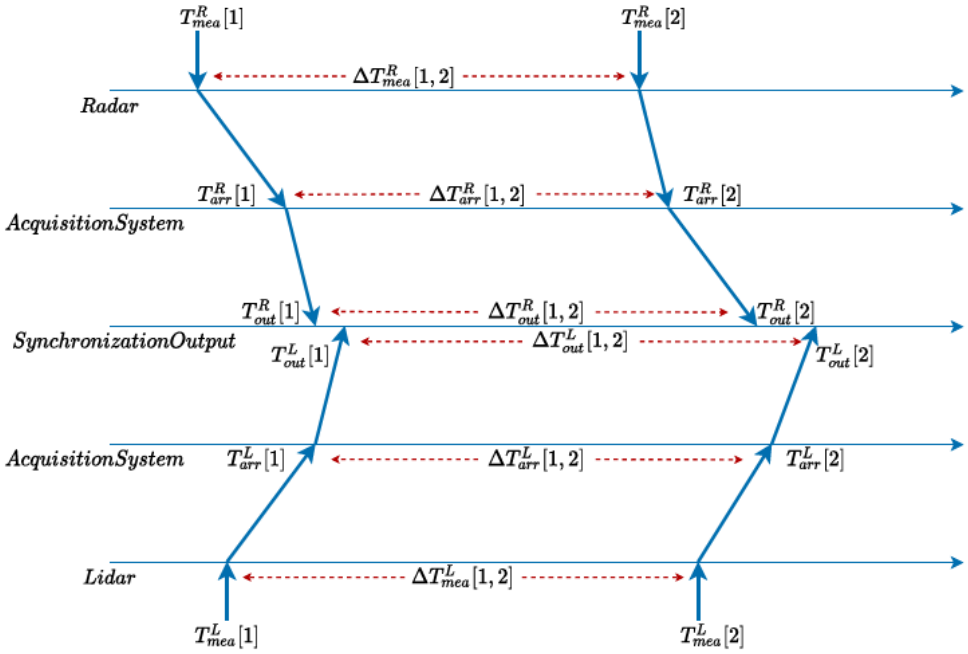

Figure 2.4: Intra-Stream timing model.

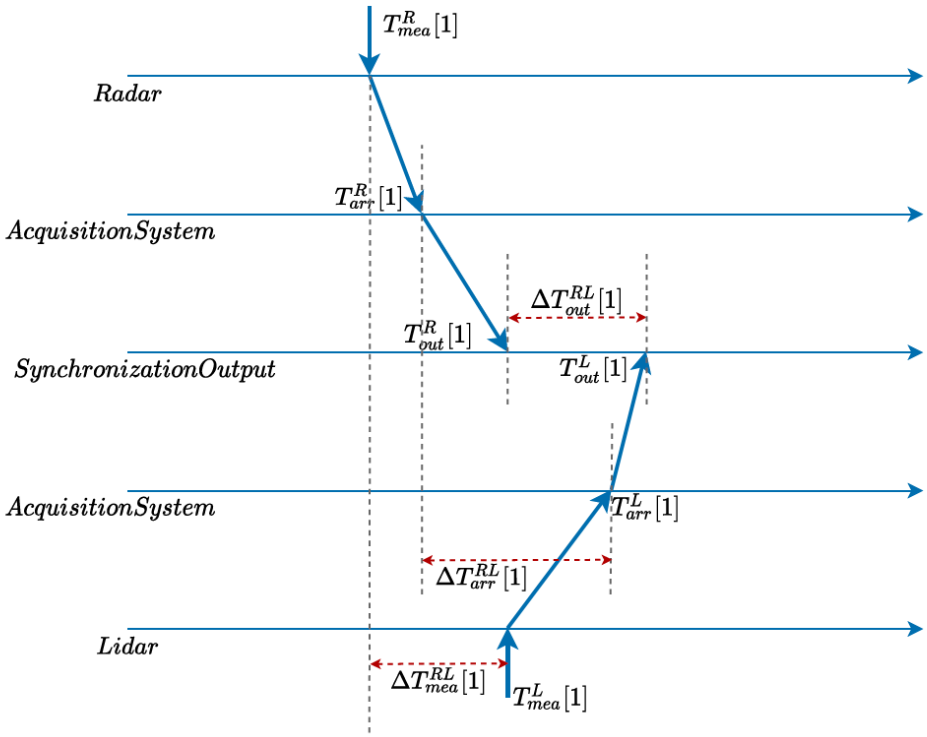

Figure 2.5: Inter-Stream timing model.

To meet our aim of temporal synchronization, we need to achieve the following:

1. **Intra-Stream Synchronization**: To output sensor measurements to the systolic array while maintaining the same Intra-Stream temporal relationships in which they were originally recorded. The following condition must hold for Intra-Stream Synchronization:

$$\begin{aligned} \Delta T^R_{out}[k-1,k] &= \Delta T^R_{mea}[k-1,k] \\ \Delta T^L_{out}[k-1,k] &= \Delta T^L_{mea}[k-1,k] \end{aligned} \tag{2.9}$$

2. **Inter-Stream Synchronization**: To output sensor measurements to the systolic array while maintaining the same Inter-Stream temporal relationships in which they were originally recorded. The following condition must hold for Inter-Stream Synchronization:

$$\Delta T^{R,L}_{out}[k] = \Delta T^{R,L}_{mea}[k] \tag{2.10}$$

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,

$$\begin{aligned} \Delta T^R_{arr}[k-1,k] &\neq \Delta T^R_{mea}[k-1,k] \\ \Delta T^L_{arr}[k-1,k] &\neq \Delta T^L_{mea}[k-1,k] \end{aligned} \tag{2.11}$$

and

$$\Delta T^{R,L}_{arr}[k] \neq \Delta T^{R,L}_{mea}[k] \tag{2.12}$$

From Equations 2.1, 2.2 and 2.4, we can also represent $\Delta T^{R}_{arr}[k-1,k]$ as,

$$\Delta T^{R}_{arr}[k-1,k] = (T^{R}_{mea}[k]+\sum_{i=1}^{N} d^{R}_{i}+\delta_{jitter}[k])-(T^{R}_{mea}[k-1]+\sum_{i=1}^{N} d^{R}_{i}+\delta_{jitter}[k-1]) \tag{2.13}$$

We can now view $\Delta T^{R}_{arr}[k-1,k]$ as a noisy/distributed version of $\Delta T^{R}_{mea}[k-1,k]$.

$$\Delta T^{R}_{arr}[k-1,k] = \Delta T^{R}_{mea}[k-1,k] + \delta_{jitter}[k] - \delta_{jitter}[k-1] \tag{2.14}$$

where $\delta_{jitter}[k]$ is assumed to be zero-mean white noise. Mean Sensor specific delay ($\sum_{i=1}^{N} d^{R}_{i}$) can be ignored in our model since it is a constant factor and does not affect the relative temporal relations.

With this, a straightforward solution to satisfy the Intra-Stream and Inter-Stream synchronization conditions of Equations 2.9 and 2.10 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 measurement.

Unfortunately, in our concerned system, the solution is quite challenging since T^{R}_{arr} and T^{L}_{arr} are the only timing information available. We formulate the solution to our synchronization problem into the following steps:

1. Estimation of Intra temporal relations (actual sensor cycle times) during capture of sensor measurements ($\Delta T^{R}_{mea}[k-1,k]$, $\Delta T^{L}_{mea}[k-1,k]$) from the observed temporal relations (observed cycle times) on arrival at acquisition system ($\Delta T^{R}_{arr}[k-1,k]$, $\Delta T^{L}_{arr}[k-1,k]$). The resulting estimates are denoted by $\Delta \hat{T}^{R}_{mea}[k-1,k]$ and $\Delta \hat{T}^{R}_{mea}[k-1,k]$.

2. Extract Timestamps of measurement capture at sensors ($\hat{T}^{R}_{mea}$, $\hat{T}^{L}_{mea}$) from temporal relation estimates ($\Delta \hat{T}^{R}_{mea}[k-1,k], \Delta \hat{T}^{L}_{mea}[k-1,k]$).

3. Re-constructing the sensor data streams according to the extracted Timestamps of measurement capture ($\hat{T}^{R}_{mea}$, $\hat{T}^{L}_{mea}$) by stream buffering, to meet the Intra and Inter-Stream synchronization conditions (Equations 2.9 and 2.10)

Steps 1 and 2 are further explained in the Chapter 3 and Step 3 is detailed in Chapter 4.

2.8 Summary

In this chapter, a description of the Systolic Array demonstrator and the involved sensors was given. We first define relevant events occurring in the system and associate timestamps with each of these events. With this, we look in to the possible sources of delay and delay variability factors affecting the incoming sensor streams. Further, we define the temporal relations between measurements, both within and across data streams and formulate the conditions for intra-stream and inter-stream synchronization. Finally, we come up with a three step solution to our synchronization problem:

1. Estimate sensors' measurement cycle times, 2. Extract measurement timestamps from cycle time estimates and 3. Re-construct the sensor streams according to original measurement timestamps by stream buffering techniques.

Timestamping Sensor Measurements

3

3.1 Overview

In this chapter, a few filtering techniques in the literature are reviewed for the estimation of sensor cycle times ($\Delta\hat{T}_{mea}[k-1,k]$) during measurement capture. Further, based on the characteristics of arrival sensor cycle times ($\Delta T_{arr}[k-1,k]$) and the complexity of the filtering technique, an appropriate technique is selected for implementation. With the obtained estimates ($\Delta\hat{T}_{mea}[k-1,k]$), a simple method to extract the timestamps of sensor measurement capture is also presented.

3.2 Related Works: Timestamping Sensor Measurements

Quite a few approaches have been proposed previously to address the issue of timestamping sensor measurements in free-running sensor network systems for synchronization applications. Earlier, in less advanced multi-sensor systems, the timestamp of the sensor data on arrival at the acquisition system was used as the true time of measurement for fusion applications[7]. 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 [8], 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 [9], a hardware device is used to attach a timestamp to each and every sensor data frame, before it is transmitted to the acquisition system through the communication link. The hardware device in [9] even 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 [10][11].

For free running asynchronous sensors systems without external synchronization support, the only timestamping that can be done, is at the acquisition system after the arrival of the sensor data frames. Approaches presented in [6], [5] and [12] 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.

3.3 Filters

The purpose of filtering is to extract the essential information from data while ignoring the noise. Our first problem of extracting/estimating the true temporal relations between sensor measurements ($\Delta\hat{T}_{mea}[k-1,k]$) from jittery arrival times can be solved by well-known filtering techniques such as Kalman filters, mean filters, median filters. The theory behind these filters are detailed below:

3.3.1 Kalman Filter

Kalman Filter [13] is one of the most common estimation methods to quickly estimate the true value of a state when the measured values of the state over time are uncertain with random errors and variations. Essentially, the Kalman filter is an iterative math process that keeps calculating new estimates of the state until the variations in the estimate become less i.e., as estimates become closer to the true value. Over the years, Kalman filters have been used in various applications for prediction and tracking tasks [14]. The standard Kalman Filter in state-space form is explained below:
Assume we want to estimate a variable within a process of the form,

$$x_{k+1} = \Phi x_k + w_k \tag{3.1}$$

where, x_k is the state vector of the process at time k;
Φ is the state transition matrix of the process;
w_k is process zero-mean white noise.

Observation of this variable can be modelled as,

$$z_k = Hx_k + v_k \tag{3.2}$$

where z_k is the measurement of x at time k;
H is the transition matrix from state space to measurement space.
v_K is the measurement noise which is zero-mean white noise.

The standard linear Kalman filter is overall a mean squared error minimizer. However, for the kalman filter to be optimal, the system errors should follow Gaussian distributions. Co-variance matrices of w_k and v_k are given by:

$$Q = E[w_k w_k^T] \tag{3.3}$$

$$R = E[v_k v_k^T] \tag{3.4}$$

Q makes up for the inherent error in the theoretical model of the system and R makes up for errors in the measurement. A good estimate of Q is needed.

Error Co-variance matrix is given by :

$$P_k = E[(x_k - \hat{x_k})(x_k - \hat{x_k})^T] \tag{3.5}$$

where $\hat{x_k}$ is the estimate of x_k.
Co-variances matrices give a feel of how fat the data is distributed from the average.

With this, the standard recursive Kalman filter algorithm is as follows:

1. **Kalman Gain Calculation**: Kalman Gain (K_G) is calculated by comparing on the errors in the estimate and of that in the measurements and is given by:

$$K_G = \frac{P_{kP}H}{HP_{kP}H^T + R} \tag{3.6}$$

   The value of K_G determines the weight put on estimates or the measurement. Larger values of K_G implies measurements are accurate and that the estimates are unstable whereas smaller K_G values imply stable estimates and inaccurate measurements.

   As the kalman algorithm proceeds, K_G becomes smaller and smaller indicating that the estimates are getting closer to the true values. Smaller K_G values also ensures that the estimates are not disturbed by the measured values as their error is higher than our estimates, at this point.

2. **Update new estimate**: The new estimate is updated by combining the measurement value and the old estimate while taking Kalman gain value into consideration.

$$\hat{x_k} = \hat{x_{kP}} + K_G[z_k - H\hat{x_{kP}}] \tag{3.7}$$

   where $\hat{x_{kP}}$ is the prior estimate of $\hat{x_k}$. In the first iteration, the value of $\hat{x_{kP}}$ is chosen accordingly based on the knowledge of the system.

3. **Update error co-variance**: Error co-variance is given by Equation 5.5. An update equation for error co-variance is formulated by substituting the new estimate from Equation 5.7 in Equation 5.5 . The resulting equation is as follows :

$$P_k = (I - K_G H)P_{kP} \tag{3.8}$$

4. **New state prediction**: With the estimate of the variable ($\hat{x_k}$) now known, the next state is predicted based on the physical state model as follows :

$$\hat{x_{kP}} = \Phi\hat{x_{k-1}} \tag{3.9}$$

   In addition, the error co-variance matrix needs to be projected into the next time interval to complete the recursion and is given below :

$$P_{kP} = \Phi P_{k-1}\Phi^T + Q \tag{3.10}$$

3.3.2 Mean Filter

Mean Filter or Average Filter is a windowed linear filter that essentially smoothens the given input signal [15]. The mean filter works by replacing each element of the signal with the mean of the N past neighbouring signal elements. This idea is illustrated in Figure 3.1. The value N, also known as window size, is an important parameter that needs to be decided according to the needs of the application.

Overall, the mean filter makes sure that the filtered value is representative of N signal values with reduced variation between them. The Mean Filter can successfully remove Gaussian noise from the signal and is also computationally simple. However, random big spikes in signal data can significantly affect the Mean filter's output values.

3.3.3 Median Filter

Median Filter is a non-linear windowed filter that replaces each element of the signal with the median value of the past N signal elements. The working of a Median Filter is illustrated in Figure 3.1.

The median filter is more effective in removing noise due to large spikes as the median value is not affected by these random spikes in the signal. However, Median Filters are computationally intensive.

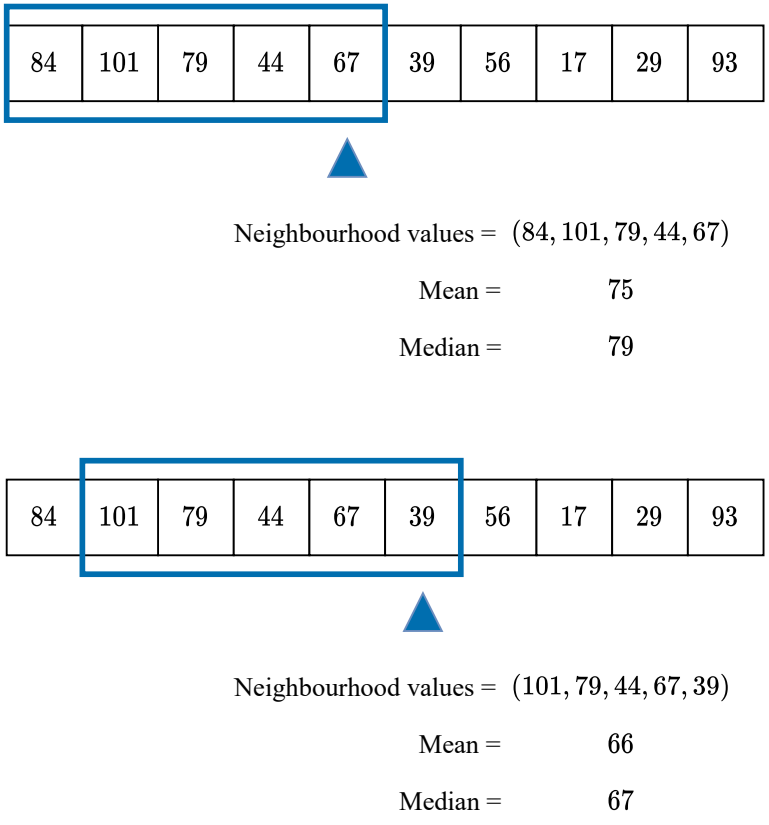

Figure 3.1: Working of Mean and Median Filters.

3.4 Estimation of true cycle times

In our considered system, sensor data frames are timestamped on arrival at the acquisition system. Based on the nature of jitter/noise in the cycle time obtained from the arrival timestamps($\Delta T_{arr}[k-1,k]$), appropriate estimation techniques to estimate the true measurement cycle times is chosen for analysis. Figure 3.2 and Figure 3.3 show the observed measurement cycle times from arrival timestamps of Radar and Lidar sensors, respectively.

We now characterize the observed measurement cycle times using statistical measures. A population size of 5000 sensor measurement observations is used for calculation of the following statistical measures. The mean and the variance values of the observed measurement cycle times of Radar and Lidar sensors are summarized in Table 3.1. Overall, we observe a higher variance in the case of the observed Radar cycle times compared to that of Lidar cycle times.

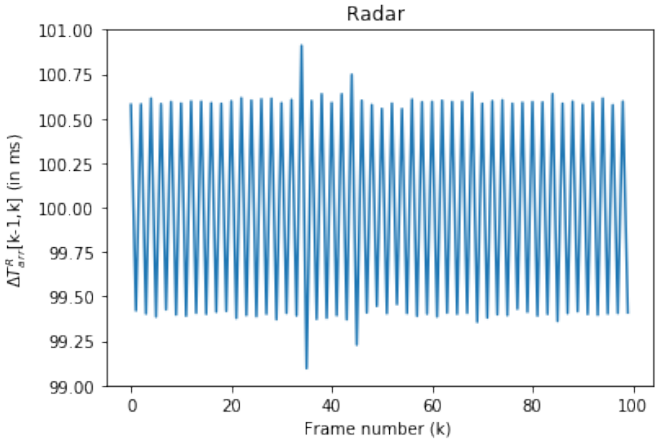

Figure 3.2: Observed Radar cycle times $\Delta T^R_{arr}[k-1,k]$ over 100 measurements.

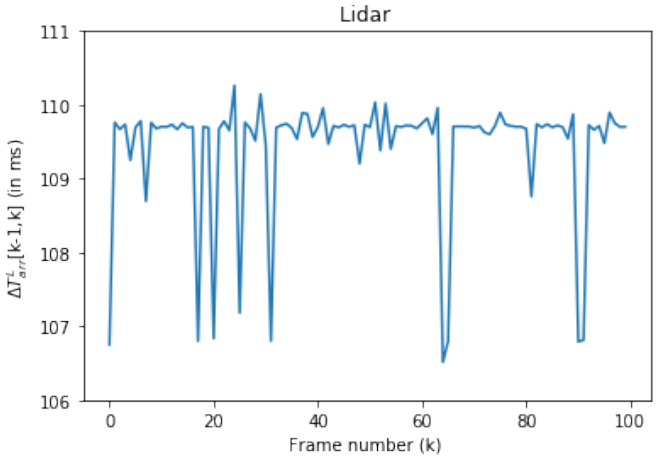

Figure 3.3: Observed Lidar cycle times $\Delta T^L_{arr}[k-1,k]$ over 100 measurements.

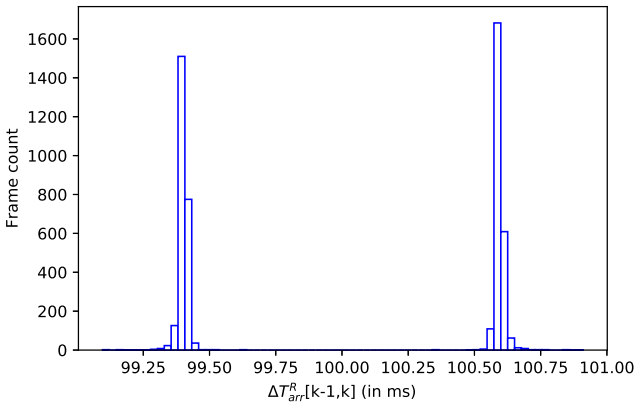

Figure 3.4: Density plot of $\Delta T^R_{arr}[k-1,k]$ for 5000 measurements.

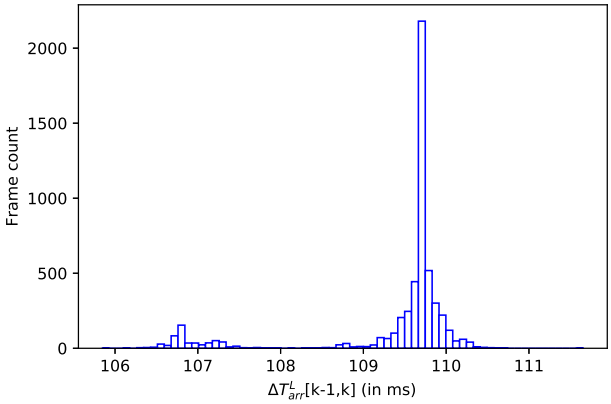

Figure 3.5: Density plot of $\Delta T^L_{arr}[k-1,k]$ for 5000 measurements.

Table 3.1: Mean and Variance of the observed Radar and Lidar measurement 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	0.96157

To have a good approximation of the probability density function of the observed measurement cycle times, a histogram of the observations is plotted (Figures 3.4 and 3.5). From the general trend of the observed Radar cycle times and its histogram plot, we can observe that the $\Delta T^R_{arr}[k-1,k]$ values alternate between an average high and low value of $100.59ms$ and $99.40ms$. In addition, we observe a Gaussian-like spread with variances of $0.00045ms^2$ and $0.00046ms^2$ about the points $100.59ms$ and $99.40ms$, respectively. With regards to the observed Lidar cycle times, we observe a Gaussian-like spread of $\Delta T^L_{arr}[k-1,k]$ values about the mean value of $109.315ms$ with a long left tail.

Since, the histogram/density plots do not show the order of the observations, we cannot make inferences on recurring or time-dependent patterns of jitter or noise in the cycle time observations. However, it is possible for certain sensors to have cycle times with drift due to temperature variation of the quartz in the sensors. Hence, it is essential to check and identify trends and periodic patterns in the jitter, if any, to obtain better estimates of the true cycle times. A plot of the cycle time observations over time would make periodic patterns obvious. Additionally, plots of rolling-mean and rolling-variance of the observations over time would show if there is presence of drift in the statistical measures of mean and variance over time.

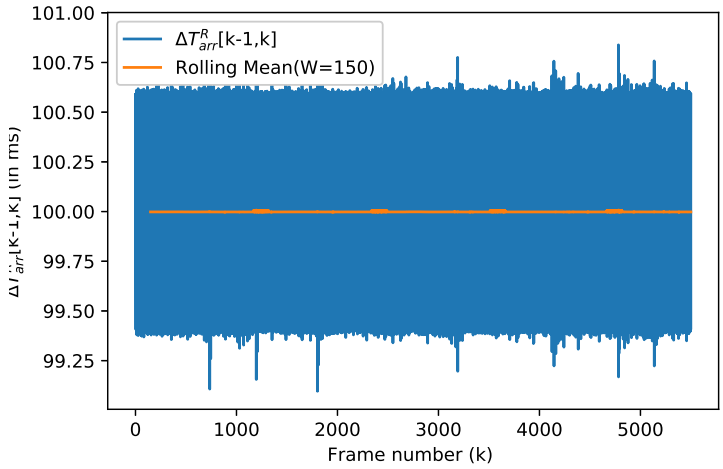

Figure 3.6: Observed Radar cycle times $\Delta T^R_{arr}[k-1,k]$ and corresponding Rolling-mean.

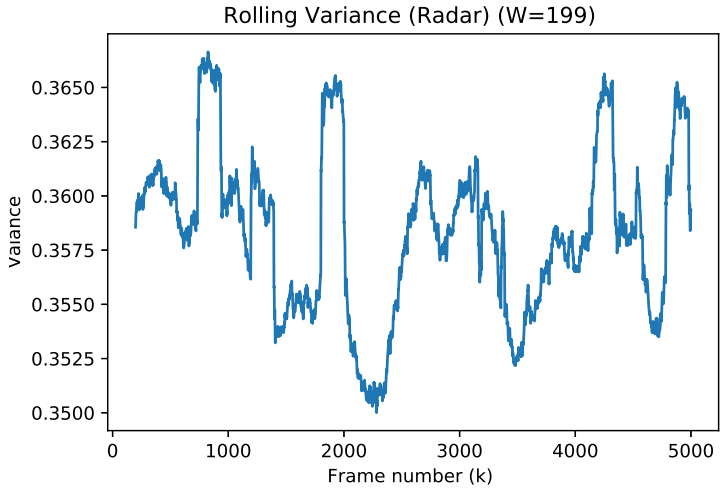

Figure 3.7: Rolling-Variance over observed Radar cycle times $\Delta T^R_{arr}[k-1,k]$.

From Figure 3.6 and Figure 3.7, we can see that there is no significant drift or trend in the observed Radar measurement cycle times as well as its rolling mean and

variance over time. Similar observations are seen in Figure 3.8 and Figure 3.9 for Lidar measurement cycle times. From the above plots, it can be visually proved that the time series of observed cycle times is time stationary and hence, drift need not be modelled for estimation of true cycle times. This eliminates the need to use a Kalman Filter in our system. Therefore, a simple estimator such as mean or median filter would suffice to obtain good cycle time estimates. The results of cycle time estimates obtained from mean and median filters are presented in Chapter 5.

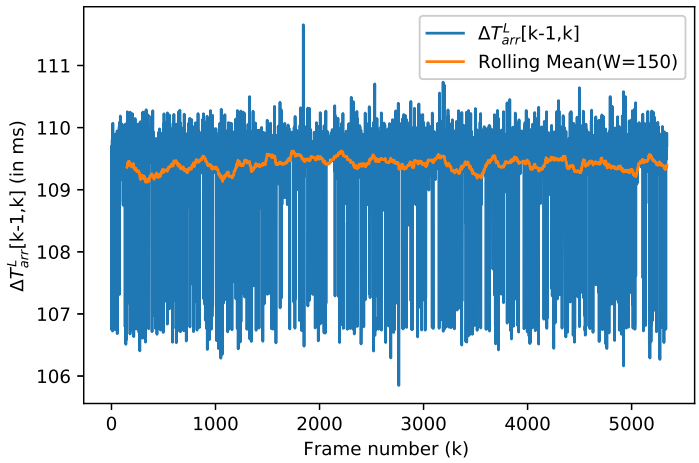

Figure 3.8: observed Lidar cycle times $\Delta T^L_{arr}[k-1,k]$ values and corresponding Rolling-mean.

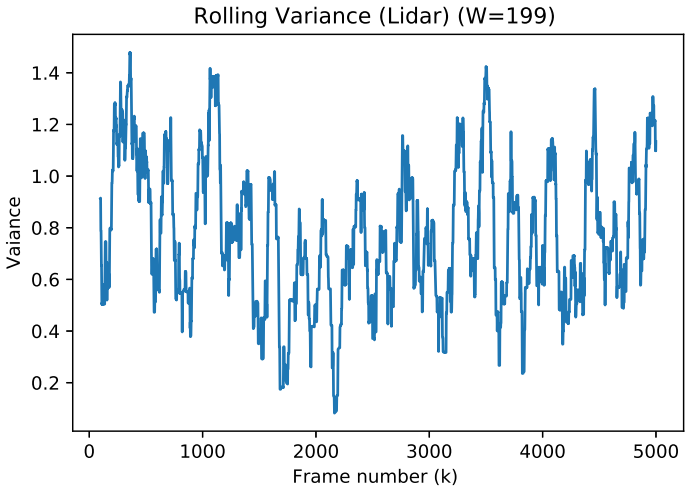

Figure 3.9: Rolling-Variance over observed Lidar cycle times $\Delta T^L_{arr}[k-1,k]$ values.

3.5 Timestamp Extraction from True cycle time estimates

The goal here is to extract a good approximation of the time of measurement capture $\hat{T}_{mea}[k]$ from the estimated true measurement cycle times $\Delta\hat{T}_{mea}[k-1,k]$.

Consider the $k-1^{th}$ sensor measurement with arrival time $T_{arr}[k-1]$. With $\Delta\hat{T}_{mea}[k-1,k]$ known, an estimate of $\hat{T_{mea}}[k]$ can simply be obtained as,

$$\hat{T}_{mea}[k] = T_{arr}[k-1] + \Delta\hat{T}_{mea}[k-1,k] \tag{3.11}$$

Subsequent measurement capture timestamps can be obtained by continuing to add cycle time estimated to previous measurement capture timestamps. For instance, the arrival time of the first sensor measurement at the acquisition system is considered as a baseline and any arbitrary measurement capture timestamp $\hat{T}_{mea}[k]$ can be calculated as follows,

$$\hat{T}_{mea}[1] = T_{arr}[1] \tag{3.12}$$

$$\begin{aligned} \hat{T}_{mea}[k] &= \hat{T}_{mea}[k-1] + \Delta\hat{T}_{mea}[k-1,k] \\ \hat{T}_{mea}[k] &= \hat{T}_{mea}[1] + \sum_{n=2}^{k} \Delta\hat{T}_{mea}[n-1,n] \end{aligned} \tag{3.13}$$

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

1. **Calculated measurement timestamp greater than its corresponding arrival timestamp, i.e., $\hat{T}_{mea}[k] > T_{arr}[k]$:** In this case, $\hat{T}_{mea}[k]$ estimate is considered inaccurate or invalid since it would mean that the measurement is captured at the sensor at a time, later than its arrival at the acquisition system. Hence, $\hat{T}_{mea}[k]$ estimate is corrected as

   $$\hat{T}_{mea}[k] = T_{arr}[k] \tag{3.14}$$

   since, in this scenario, $T_{arr}[k]$ is closer to the true measurement capture timestamp. An illustration of this case is shown in Figure 3.10.

2. **Lost Sensor Measurement**: In case of lost sensor measurement during transmission from sensor to the acquisition system, the timestamp of the next measurement capture can be wrongly associated to the lost measurement's timestamp. This is illustrated in Figure 3.11.

   To handle this situation, measurement losses need to be detected. We can expect the observed cycle time after a lost measurement to be larger than the usual cycle times. With this, an upper bound on the observed cycle time is set such that cycle

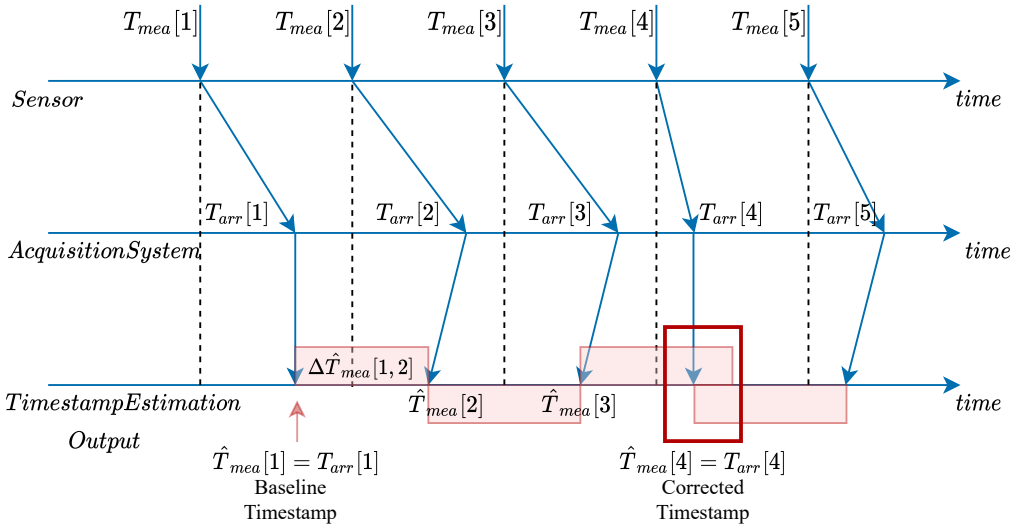

Figure 3.10: Timestamp Correction when $\hat{T}_{mea}[k] > T_{arr}[k]$.

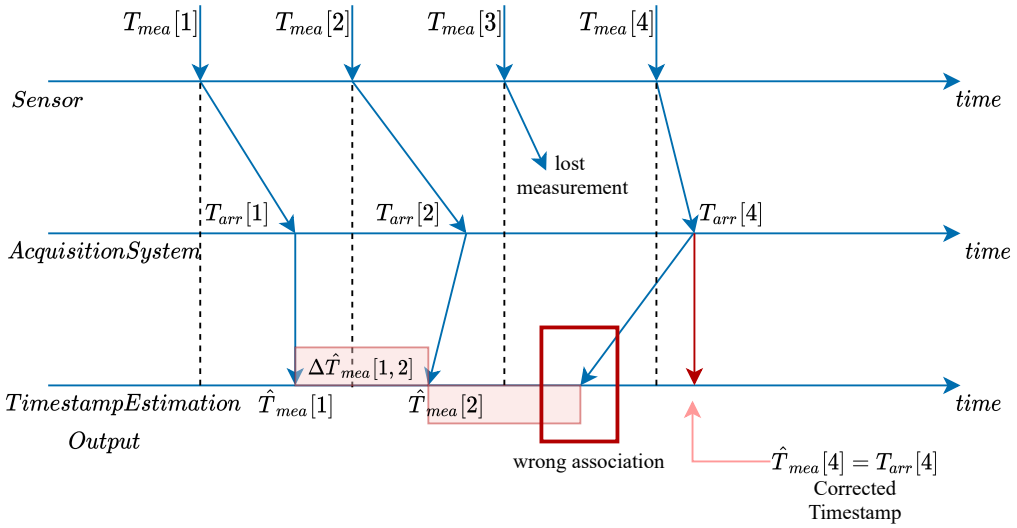

Figure 3.11: Timestamp Correction when a Measurement is lost.

times longer than the upper bound indicate a lost measurement and not included in the filtering process.

$$\Delta\hat{T}_{arr}[k-1,k] > upperBound \tag{3.15}$$

The $\hat{T}_{mea}[k]$ estimate for the measurement is then set as

$$\hat{T}_{mea}[k] = T_{arr}[k] \tag{3.16}$$

Overall, the timestamps extracted by the above method along with the corrections can be considered as the true timestamp of sensor measurement capture, as long as the cycle time estimates are sufficiently accurate.

3.6 Summary

In this chapter, we first present the existing timestamping techniques in sensor based systems. Since our sensors do not have any external trigger or other support for timestamping, we look into well known filtering techniques such as Kalman filters, Mean filters and Median filters for estimating true sensor measurement cycle times from observed cycle times. Next we characterized the observed sensor cycle times using statistical measures and looked for any time-dependent drifts in the data. Overall, we did not observe any prominent drifts, thereby eliminating the need to use a Kalman filter in our system. Finally, we present a simple method to extract the timestamps of measurement capture from cycle time estimates obtained by the filters.

4 Data Stream Synchronization

4.1 Overview

In this chapter, we look into the problem of re-constructing sensor data streams according to the estimated timestamps of measurement capture ($\hat{T}^R_{mea}, \hat{T}^L_{mea}$). The main objective here is to ensure that the incoming sensor data is streamed out in real-time while maintaining time synchronization. First, existing solutions to the problem of data stream synchronization are explored. Further, the implemented data stream synchronization algorithm for our system is explained in detail.

4.2 Related Works: Data Stream Synchronization

The problem of synchronizing streams of data has been extensively investigated in the area of distributed multimedia systems. These systems usually consist of a number of sources which transmit different media streams to a receiver, which presents or plays out the streams as an unit. In some cases, the streams can have well-defined temporal relationships between them, and hence, would require some form of synchronization solutions to ensure the streams to played out while maintaining the expected temporal relationships. A classical example of this scenario is the famous 'lip-sync' problem [16, 17], where audio and video streams need to be accurately synchronized during playout.

Numerous approaches to the execution of multimedia synchronization scenarios are available and the choice of the solution is usually based on the application and network characteristics. Broadly, the problem of multimedia synchronization can be classified based on several factors such as location, real-time requirement of streams, 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[18]. In this section, only live synchronization solutions are presented, as they are relevant to our problem.

Media synchronization is viewed as an end-to-end challenge [19] and hence based on the flexibility of application system, the issues can be addressed either on the source [20] or receiver [21, 22] side or even both [23, 24]. On the source side, a common solution employed is that of attaching timestamps and sequence numbers to the transmitting frames [21, 22]. This timing information can be very useful at the receiver to calculate the play-out times of the frames. 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 [25, 26]. This solution succeeds in eliminating the need for inter-stream synchronization, however, intra-stream synchronization still needs to be addressed. In [27] and [26], the source changes the transmission rate of the streams depending on the feedback received from the receiver on the network conditions to overall prevent asynchrony. On the receiver side, buffering techniques are commonly employed to temporarily store the frames before play out. The main purpose of buffering is to smooth-en out the effects of varying network jitter of frames within and between streams. The buffering time can either be static based on a maximum jitter value or can be made to vary depending on the network delays [28] and the available buffer size [22]. Other receiver-side techniques consist of dropping late arriving frames [23] or older frames during buffer full conditions [28]. The dropped frames are either left empty or is interpolated [22].

Different approaches are needed to solve to inter and intra stream synchronization problems. For intra-stream synchronization, techniques that aim at reducing the effects of jitter are needed. This includes receiver buffering techniques [28, 29, 30] to smooth out the effects of jitter or 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 [29, 30]. Based on the level of asynchorny of streams 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. Overall, the idea here is to adapt the play out rates of slave streams to maintain the correct temporal relations with the master stream. Some proposals that involve dynamic switching of master and slave streams during run-time are also available [29]. 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 [19].

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 bounds on network 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 [31]. 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 sync 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, there is no need for timing information such as 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 larger buffering latencies. In such cases, a tolerable synchronization error value is set and the frames are buffered accordingly, for lesser time. Overall, this leads to a trade off between the quality and latency of the synchronization algorithm. In most applications, an exact bound on system jitters cannot be assumed to be known, as it can vary significantly from time to time.

To overcome the trade off problem between latency and quality of synchronization, adaptive control based solutions are proposed in [29] and [30]. These solutions con-

sists 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.

4.3 Implemented Algorithm

The algorithm that we have chosen to implement are based on solutions presented in [29], [30] and [32]. These algorithms ensure live intra and inter stream synchronization in real-time by employing control-based adaptive buffering techniques. With the $\hat{T}_{mea}$ estimates and T_{arr} available, the algorithm accomplishes synchronization by comparing and equalising end-to-end delays of each sensor measurement from its capture at the sensor to its arrival at the acquisition system. Equalization of end-to-end delays is carried out by piecewise adjustment of stream buffering times at output while meeting a set Quality of Service (QoS) factors along with a minimal overall latency.

The considered QoS parameters are discussed below :

1. **Maximum Intra-stream phase distortion** ($\Delta\phi^{R}_{intra}, \Delta\phi^{L}_{intra}$): Intra-stream phase distortion (Intra-SPD) is the difference between the end-to-end delays of two consecutive sensor measurements of the same sensor stream. A maximum allowable bound on Intra SPD ($max.\Delta\phi^{R}_{intra}$, $max.\Delta\phi^{L}_{intra}$) is set for each sensor and if the arrival of an incoming sensor measurement does not fall within the $max.\Delta\phi_{intra}$ bound, then the frame is considered to have arrived too late to effectively buffer out a synchronized stream and thus, discarded. The equation for Intra-SPD is given by:

$$\Delta\phi^{R}_{intra}[k, k-1] = |D^{R}_{end-to-end}[k] - D^{R}_{end-to-end}[k-1]| \tag{4.1}$$

where $D_{end-to-end}[k]$ is the end-to-end delay of thr k^{th} sensor measurement and is given by,

$$D_{end-to-end}[k] = T_{arr}[k] - \hat{T}_{mea}[k] \tag{4.2}$$

2. **Maximum Inter-stream phase distortion** ($\Delta\phi^{R,L}_{inter}$): Inter-stream phase distortion (Inter-SPD) is the difference between the end-to-end delays 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 bound on Inter-SPD ($max.\Delta\phi^{R,L}_{inter}$) is set and any frame arriving beyond this bound is discarded. The equation for Inter-SPD is given by:

$$\Delta\phi^{R,L}_{intra}[k] = |D^{R}_{end-to-end}[k] - D^{L}_{end-to-end}[k]| \tag{4.3}$$

where $D_{end-to-end}[k]$ is the end-to-end delay of thr k^{th} sensor data frame and is given by,

$$D_{end-to-end}[k] = T_{arr}[k] - \hat{T}_{mea}[k] \tag{4.4}$$

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

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 maintained the buffering alignment of different streams. Each of these schemes are discussed in detail below

4.3.1 Intra-Stream Synchronization Scheme

Intra-Stream Synchronization is solved by adaptive buffering to equalise end-to-end delays of sensor measurements. It occurs in two steps for every data stream independently. The steps are explained below :

1. **Output Time Decision**: In this step, the output time of the each sensor measurement (T_{out}) is decided. A virtual clock-timer is employed at the receiver end on the acquisition system and the sensor measurements are streamed out with respect to the it's timeline. The virtual timer can be set-backed or advanced, thereby controlling the buffering times at the receiver end.

   We define three output events `wait`, `nowait` and `discard` where the sensor measurement is buffered, streamed out immediately and discarded, respectively. The output event is decided by comparing the time of arrival of the sensor measurement at the acquisition system as recorded by the virtual timer and the estimated time its capture ($\hat{T}_{mea}$). The virtual timer is initialised to the arrival time of the first sensor data frame and then it is made to start. With the virtual timer value at current time defined as T_{vt} , the conditions for each of the output events are as follows:

   (a) `wait` case: If the arrival time of the incoming sensor measurement as recorded by the virtual timer is lesser than the estimated measurement capture time, then it is implied that the current measurement arrived earlier compared to the previous sensor measurement and hence, the it needs to be buffered. The `wait` case condition for sensor measurement k that has currently arrived, is given by :

   $$T_{vt} < \hat{T}_{mea}[k] \tag{4.5}$$

   In this case, the sensor measurement is buffered until the virtual timer is reaches the value of $\hat{T}_{mea}[k]$. The buffering time is given by:

   $$bufferTime = \hat{T}_{mea}[k] - T_{vt} \tag{4.6}$$

   (b) `nowait` case: The incoming sensor measurement is considered to have arrived late if its arrival time as recorded by the virtual timer is larger than the estimated measurement capture time.

In the situation where the data frame arrives late, the sensor measurement is either streamed out immediately at T_{vt} or discarded, depending on whether or not its arrival time falls within the bound of maximum Intra-SPD. The `nowait` case condition for a currently arrived sensor measurement can be written as:

$$T_{vt} \geq \hat{T}_{mea}[k] \tag{4.7}$$

and

$$T_{vt} < \hat{T}_{mea}[k] + max.\Delta\phi_{intra} \tag{4.8}$$

(c) `discard` case: The incoming sensor measurement is discarded if its arrival is late and its arrival time does not fall within the bounds of the maximum Intra-SPD. The above condition is given by:

$$T_{vt} > \hat{T}_{mea} + max.\Delta\phi_{intra} \tag{4.9}$$

Figure 4.1 summarizes the three output event conditions on a timeline.

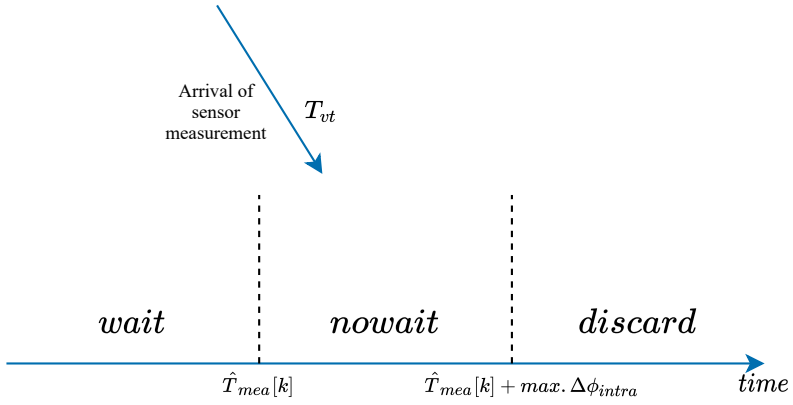

Figure 4.1: Output event conditions on a timeline.

2. **Adaptive Control Algorithm for Buffering**: An adaptive control algorithm is employed to keep the synchronization error versus buffering time latency trade-off in check.

   The control algorithm keeps a count of the occurrences of each of the output events (`wait`, `nowait` and `discard`). At any point in time, the count values of the `wait`, `nowait` and `discard` events represent the arrival distributions of the sensor measurements. Hence, based on the count values and certain pre-set count thresholds, the algorithm determines whether the sensor measurements are being under or over buffered in general, over the past window of time. Synchronization errors and buffer time latency results for different count thresholds values are presented in Chapter 5. Furthermore, the algorithm setbacks or advances the virtual timer depending on under-buffer and over-buffer conditions, accordingly.

- **Under-Buffer case**: The sensor stream is considered to be under-buffering if the counts of `nowait` and `discard` events are greater than their upper threshold values.

$$nowait\_cnt \geq \texttt{T\_NOWAIT} \tag{4.10}$$

or

$$discard\_cnt \geq \texttt{T\_DISCARD} \tag{4.11}$$

During `nowait` and `discard` output events synchronization errors are introduced since the measurements are not streamed out according to the $\hat{T}_{mea}$ values. If under-buffering conditions are met, it implies that the number of measurements with synchronization errors are more and hence, the virtual timer is set-back to ensure higher buffering times and to cut back on `nowait` and `discard` counts. The set-back displacement Δ^- is given by :

$$\Delta^- = (1 - \frac{wait\_cnt}{\texttt{T\_WAIT}})\Delta_{max} \tag{4.12}$$

where, Δ_{max} is the maximum shift value appropriate for the application. Further, the count values of `nowait` and `discard` events are set to 0.
Overall, if the control algorithm determines that the sensor measurements have been under-buffering, then the virtual timer value is setback by Δ^- value, thereby increasing the buffering times of future frames.

$$T_{vt} = T_{vt} - \Delta^- \tag{4.13}$$

- **Over-Buffer case**: The sensor stream is considered to be over-buffering if the count of `wait` output event is greater than its upper threshold and `nowait` and `discard` counts are well below a set lower threshold (value below half of upper threshold).

$$wait\_cnt \geq \texttt{T\_WAIT} \tag{4.14}$$

and

$$\begin{aligned} nowait\_cnt &< \texttt{LT\_NOWAIT} \\ discard\_cnt &< \texttt{LT\_DISCARD} \end{aligned} \tag{4.15}$$

If over-buffering conditions are met, it implies that the stream is buffering at a slow rate resulting in larger buffer time latency. In this case, the virtual timer is advanced by a factor Δ^+, given by:

$$\Delta^+ = (1 - \frac{nowait\_cnt}{\texttt{T\_NOWAIT}})\Delta_{max} \tag{4.16}$$

where, Δ_{max} is the maximum shift value appropriate for the application. Further, the count value of `wait` event is set to 0.
Overall, if the control algorithm determines that the sensor measurements have been over buffering, then the virtual timer value is advanced by Δ^+ value, thereby reducing the buffering times of the future frames.

$$T_{vt} = T_{vt} + \Delta^+ \tag{4.17}$$

4.3.2 Inter-Stream Synchronization Scheme

As explained earlier, intra-stream synchronization is established by equalising end-to-end delays, 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 and can be easily identified as the stream with the smallest virtual timer value. This is because, considering the larger end-to-end delays, the control algorithm would have initiated set-back calibrations to the stream. Initially, any arbitrary sensor stream can be set as the reference stream.

Let's denote the virtual timer value of the reference stream, at any point in time by T_{vc}^{ref} and the virtual timer value of the follower stream as T_{vc}^{foll}. Overall, the idea is to set-back the virtual timer of the follower stream if the offset between the streams is more than our set tolerable bounds of inter-SPD. In addition, we also have to account for the intra-SPD of individual streams as we still want to maintain the intra-stream synchronization. We can write the inter-stream offset condition as follows :

$$T_{vt}^{foll} - T_{vt}^{ref} \leq max.\Delta\phi_{inter}^{ref,foll} - max\_of(max.\Delta\phi_{intra}^{ref}, max.\Delta\phi_{intra}^{foll}) \tag{4.18}$$

The above inter-stream stream offset 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. In addition, any advance displacements of the follower stream is cancelled to match with the slower buffering rates of the reference stream. The setback displacement value is taken as the offset between the virtual timers of the two streams.

$$\Delta_{inter}^{-} = (T_{vt}^{foll} - T_{vt}^{ref}) - (max.\Delta\phi_{inter}^{ref,foll} - max\_of(max.\Delta\phi_{intra}^{ref}, max.\Delta\phi_{intra}^{foll})) \tag{4.19}$$

$$T_{vt}^{foll} = T_{vt}^{foll} - \Delta_{inter}^{-} \tag{4.20}$$

4.4 Summary

This chapter focuses on the problem of re-constructing incoming sensor streams according to the true measurement capture times. This problem has been investigated in the area of distributed multimedia systems and relevant existing solutions are presented. Finally, details of the implemented adaptive buffering based solution which keeps the latency and synchronization error under check even during considerable delay variation is given.

Experimental Results and Analysis

5

5.1 Overview

In this chapter, the proposed experimental setup for the evaluation of temporal synchronization of Radar and Lidar is detailed. The results of Timestamp extraction from cycle time estimates by Mean and Median filters are presented and analyzed. Further, the results and analysis of the implemented data stream synchronization algorithm are shown.

5.2 Evaluation setup

For our synchronization evaluation experiments, the sensor data acquisition and arrival timestamping was performed on a Windows10 based laptop 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.

To verify the accuracy of the temporal synchronization solution, a moving Meccano contraption, as shown in Figure 5.1 was set up to be used as a common target object for both Radar and Lidar sensors. The Meccano contraption consists of a clear and distinguishable common target (twin plates) which revolves around a fixed axis at a constant speed. This setup ensures that the position measurements obtained by both Radar and Lidar are distinguishable for comparison and that the spatial error in the position measurement is negligible. Thus, the differences in the positions of the common target observed from Radar and Lidar sensors are solely due to the temporal errors

Figure 5.2 and Figure 5.3 show the captured Radar image and Lidar point cloud of the Meccano setup. The position of the revolving plates can be clearly observed from both the Radar image as well as the Lidar point cloud. To verify the synchronization, the observed positions of the revolving plates by both the sensors are compared at the output of the synchronization unit. If temporal synchronization is fairly accurate, we should observe synchronized target positions after the synchronization algorithm. Overall, this setup though simple and inexpensive is effective and sufficient enough to evaluate the correctness of the temporal synchronization solution.

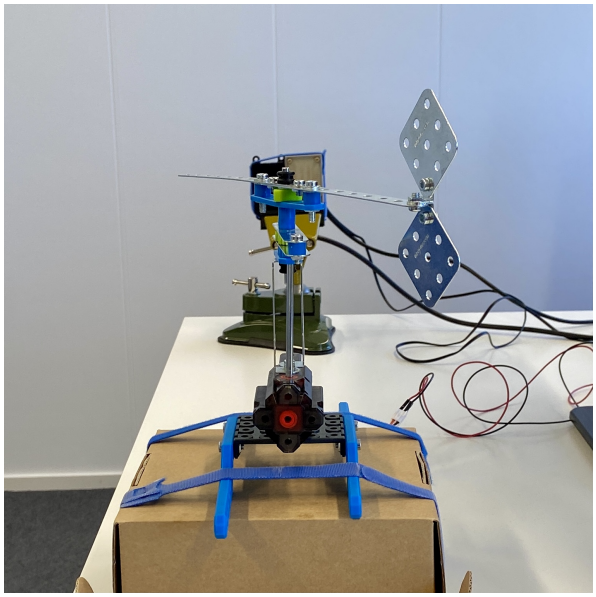

Figure 5.1: Meccano Contraption for Evaluation of Temporal Synchronization.

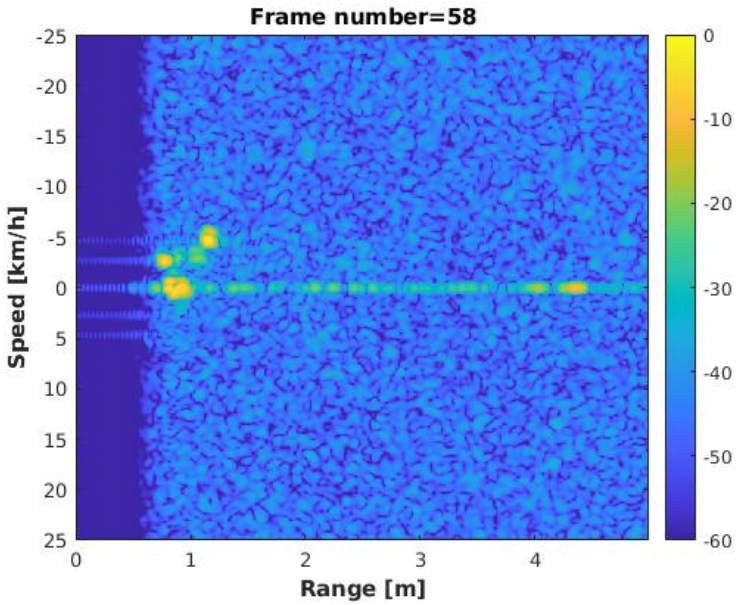

Figure 5.2: Radar Data of the Meccano contraption.

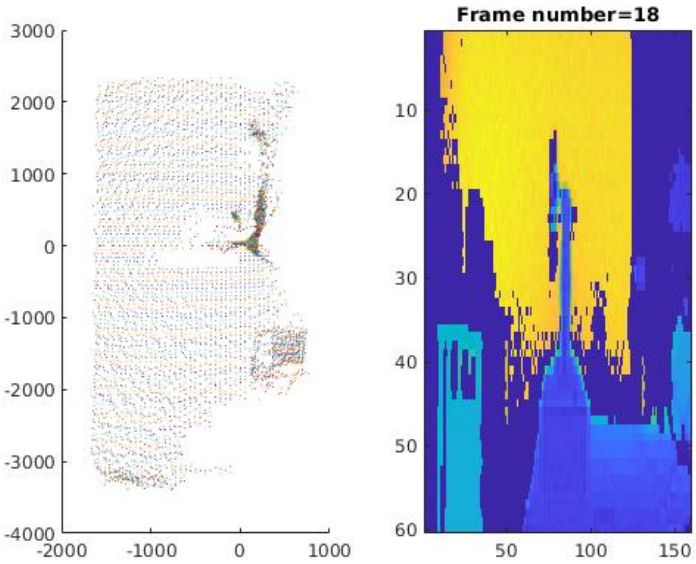

Figure 5.3: Lidar Data of the Meccano contraption.

5.3 Timestamp Estimation: Results and Analysis

In this section, the results of true cycle time estimates ($\Delta\hat{T}_{mea}$[k-1,k]) from observed cycle times ($\Delta T_{arr}[k-1,k]$) is presented and analysed. Figure 5.4 and Figure 5.6 shows the Radar's true cycle time estimates ($\Delta\hat{T}^{R}_{mea}$[k-1,k]) by Mean and Median filters of different Window sizes (W), respectively. Similarly, Figure 5.5 and Figure 5.7 shows the Lidar's true cycle time estimates ($\Delta\hat{T}^{L}_{mea}$[k-1,k]) by Mean and Median filters of different Window sizes (W), respectively. The observed cycle times ($\Delta T_{arr}[k-1,k]$) is also presented in the colour grey for comparison in the above graphs. We can notice from Figure 5.4 that the observed cycle time of the Radar sensor (grey plot-line) shows a repeating pattern of alternating between an average high and a low value of 100.59ms and 99.40ms, respectively. Regarding the observed Lidar cycle times, we can see from Figure 5.5 that cycle times varies considerably over nearly $2ms$, which is on account of several delay factors such as transmission, pre-processing etc.

In the Mean filter case, the variations in cycle times are considered extraneous disturbances in the data and minimised in both Radar and Lidar estimates (Figure 5.4 and Figure 5.5). This is characteristic of Mean filtering since if enough points are chosen, the noise is reduced by summing to its own (nearly) zero average value [33]. Further, we can observe that the estimates obtained from larger window sizes approximately follows the absolute mean value of the observed arrival cycle time of the sensor which, was calculated over different sets of timestamps collected over different periods of time. Hence, we can expect a Mean filter with a smaller window size to give better estimates in the case of localized time-dependent variations in the actual cycle times.

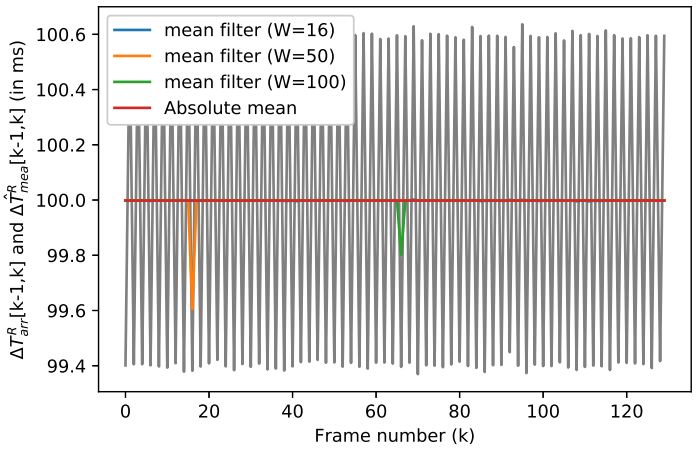

Figure 5.4: Estimated Radar ($\Delta\hat{T}^{R}_{mea}$[k-1,k]) cycle times obtained by Mean filter of different window sizes.

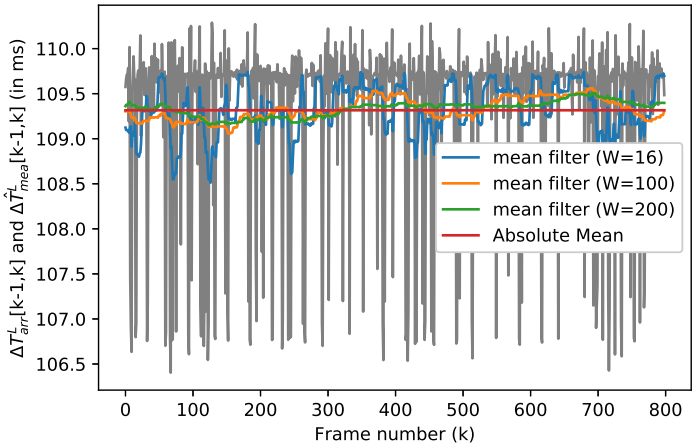

Figure 5.5: Estimated Lidar ($\Delta\hat{T}^{L}_{mea}$[k-1,k]) cycle times obtained by Mean filter of different window sizes.

In the Median filter case, we observe that the Radar's estimates obtained are not devoid of the repeating pattern of disturbance we see in it's observed cycle times (Figure 5.6). We can expect this behaviour because the Median filter is not very effective in removing high spatial frequency details, which correspond to finer details in the signal [34]. However, the estimated Lidar cycle times obtained by Median filter as in Figure 5.7, is more robust compared to that obtained by the Mean filter. This is because an atypical point in the data does not significantly affect the median value, resulting in stable cycle time estimates with minimal influence of unrepresentative data points.

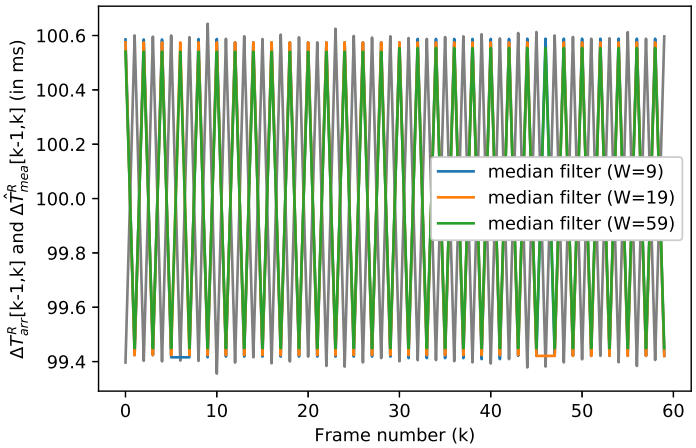

Figure 5.6: Estimated Radar ($\Delta\hat{T}^{R}_{mea}$[k-1,k]) cycle times obtained by Median filter of different window sizes.

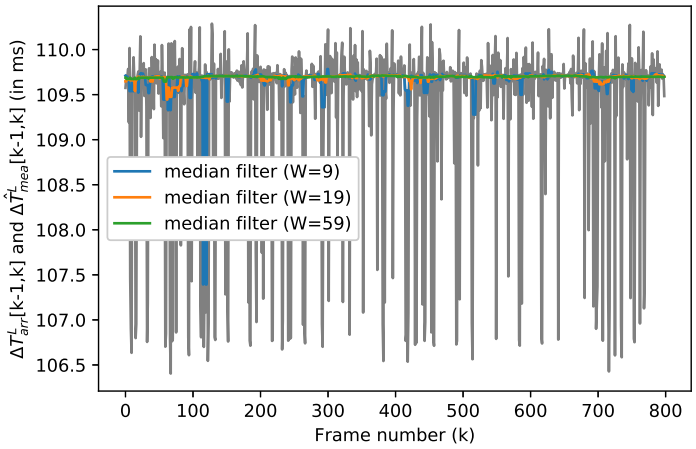

Figure 5.7: Estimated Lidar ($\Delta\hat{T}^{L}_{mea}$[k-1,k]) cycle times obtained by Median filter of different window sizes.

5.3.1 Estimator Selection

While selecting an appropriate estimator for a system, several factors need to be considered. These factors include the quality of the estimates, the complexity which corresponds to the latency of the estimator and the hardware resources required for the realization of the estimation solution.

With regards to hardware implementation, we can expect the Median filter to be more expensive and complex to realize since a sorting operation is required for input every data point. Table 5.1 shows the hardware utilization of the two filters with different Window sizes on ZYNQ ZC702 Platform. Mean Filters with window sizes which are powers of 2, were also considered to eliminate the need for a DSP block for

division. As expected, we can see that the resource requirement of Median filters is higher than Mean filters of comparable window size.

Table 5.1: Resource Utilization of Filters.

Component	Window Size	LUTs	Registers	DSP
Mean filter	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 filter	9	462	443	0
	19	1072	624	0
	59	2240	1345	0

The quality of the estimates can be formulated as the error in the estimates, given by:

$$e_{est} = |\hat{T}_{mea} - T_{mea}| \tag{5.1}$$

where $\hat{T}_{mea}$ and T_{mea} refer to the estimated and the true measurement cycle times, respectively. However, T_{mea} values remain unknown. Hence, in practice, we need different ways to check the quality and correctness of the estimates.

- In some systems, certain assumptions on the nature of the true measurement cycle times (T_{mea}) can be made based on our prior knowledge on the sensors and the acquisition systems. With the nature of T_{mea} known, one can check if these assumptions are valid for the estimates as well.

- To better evaluate the estimates, the final synchronisation results can be analysed for different estimators. Since the final data association between sensors depend on the quality of the estimated timestamps, one can comment on the correctness of the estimates by analyzing the association between sensor measurements.

For our estimates' quality evaluation, sensor measurements of the Meccano target at two orientations were selected by visually examining the Radar and Lidar images. The orientations correspond to the target at the closest and farthest point from the sensors during every rotation and are denoted by P_{front} and P_{back}, respectively. For the sake of simplicity, we refer to these target orientations as synchronization events. Figure 5.8 and Figure 5.9 shows a graph of Radar and Lidar measurements of the considered two target positions plotted against their arrival and synchronization output timestamps, respectively. Overall, the plots in Figure 5.8 and Figure 5.9 show the correctness of our synchronization solution.

It is to be noted that a Radar and a Lidar measurement corresponding to the same rotation and orientation (P_{front} or P_{back}) may not physically point to the same exact target position on the ground. This is because the sensors have different capture rates

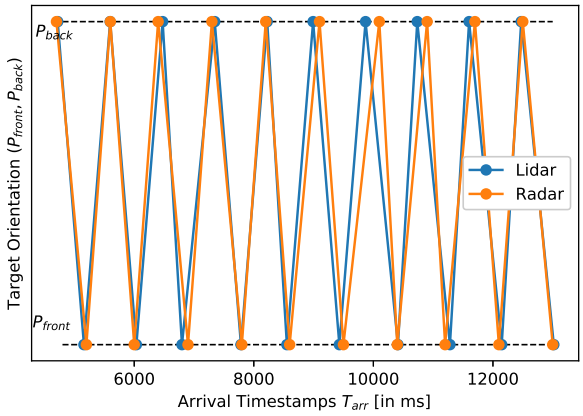

Figure 5.8: Synchronization Event measurements plotted against Arrival Timestamps before Synchronization.

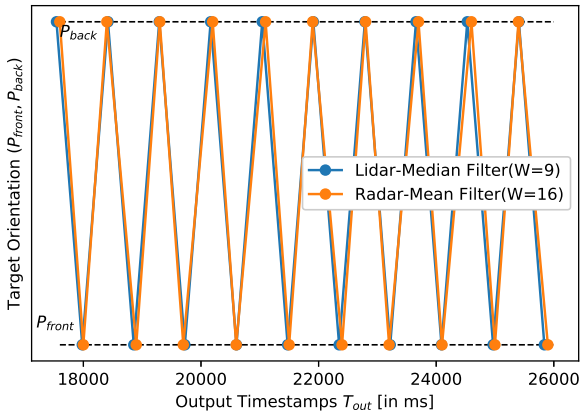

Figure 5.9: Synchronization Event measurements plotted against Output Timestamps after Synchronization.

and the target's closest and farthest position as captured by different sensors need not be the same position on the ground. Hence, we cannot expect $\Delta T_{out}^{R,L}[n]$ to be exactly zero. However, if the timestamp estimates are of sufficient accuracy, we should still expect Radar and Lidar measurements of the same orientation and rotation closely associated after synchronisation. More precisely, we need to expect their inter-stream temporal relations at the output ($\Delta T_{out}^{R,L}[n]$) to be smaller values and preferably less than approximately half the average cycle time of the sensors to prevent wrong association with another sensor measurement. Figure 5.10 shows the Inter-stream temporal relation between synchronization event measurements on arrival ($\Delta T_{arr}^{R,L}[n]$) at the acquisition system. $\Delta T_{arr}^{R,L}[n]$ values around $100ms$ and above indicate that the arriving measurements are wrongly associated by more than a sensor clock period.

Inter-stream temporal relation values ($\Delta T_{out}^{R,L}[n]$) between the Radar and Lidar mea-

surements corresponding to the synchronization event for different combinations of estimators are presented in Figure 5.11, Figure 5.12, Figure 5.13 and Figure 5.14 and the average $\Delta T_{out}^{R,L}[n]$ values are tabulated in Table 5.2.

We can conclude from the Figure 5.15 and the average $\Delta T_{out}^{R,L}[n]$ values that the estimates from the Mean filter for Radar and Median filter for Lidar result in better sensor data association. Also, we observe that with changes in window sizes, the $\Delta T_{out}^{R,L}[n]$ values vary only slightly. To further verify the accuracy of the estimates obtained from different window sizes, a more sophisticated evaluation setup with exact target position extraction from sensor measurements is required. With this, sensor measurements of exact same target positions can be compared and expect $\Delta T_{out}^{R,L}[n]$ value to be zero. Based on the trade-off between the hardware utilization and the accuracy of the estimates, a suitable window size is selected. Currently for our system, the employed evaluation method is considered adequate to verify the correctness of the synchronization algorithm. However, this evaluation method is not sufficient to obtain the exact errors in the estimates. Finally, for our hardware implementation, Mean and Median filter configurations with the least hardware resource requirement is selected.

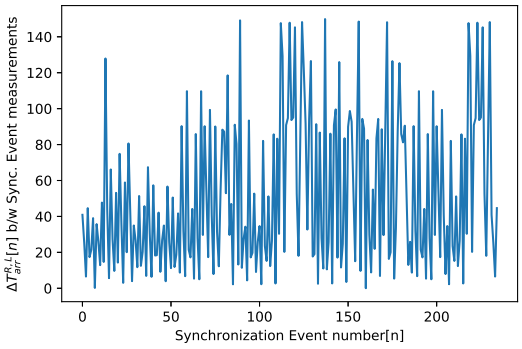

Figure 5.10: Inter-stream temporal relation between synchronization event measurements with respect to arrival times.

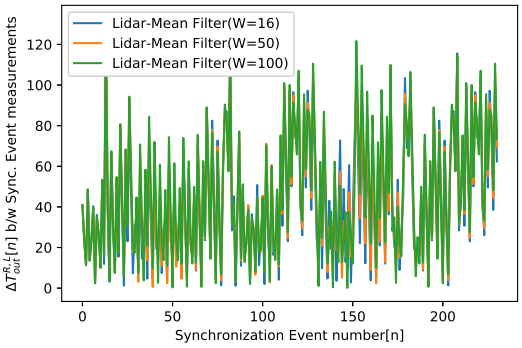

Figure 5.11: Inter-stream temporal relation between synchronization event measurements at output using Radar-Mean filter(W=16) timestamp estimates.

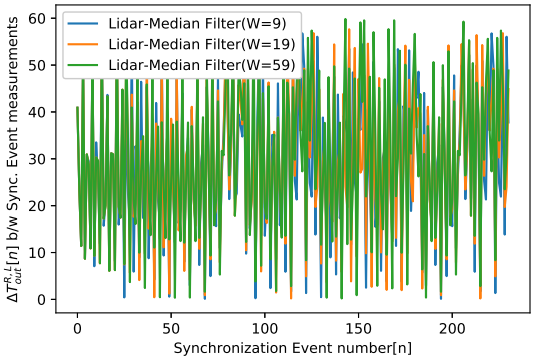

Figure 5.12: Inter-stream temporal relation between synchronization event measurements at output using Radar-Mean filter(W=16) timestamp estimates.

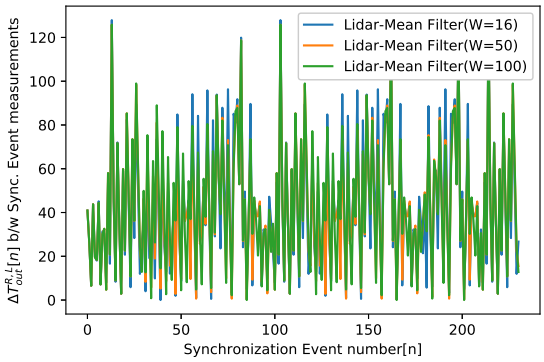

Figure 5.13: Inter-stream temporal relation between synchronization event measurements at output using Radar-Median filter(W=9) timestamp estimates.

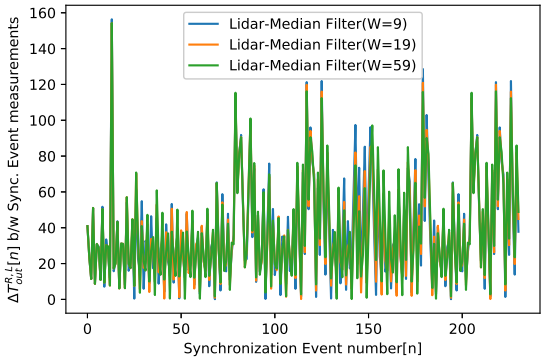

Figure 5.14: Inter-stream temporal relation between synchronization event measurements at output using Radar-Median filter(W=9) timestamp estimates.

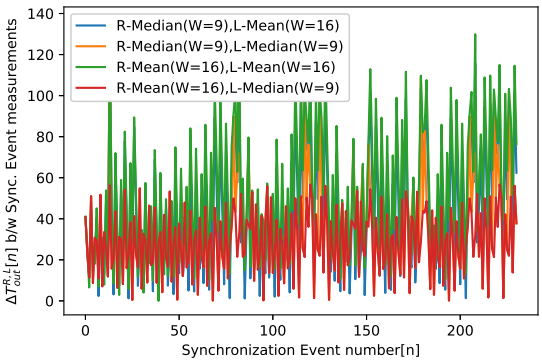

Figure 5.15: Inter-stream temporal relation between synchronization event measurements with different estimators.

Table 5.2: Average $\Delta T_{out}^{R,L}[n]$ values for different combinations of Estimators.

Estimator		**Average $\Delta T_{out}^{R,L}[n]$ values (in ms)**
Radar	**Lidar**	
none	none	$\Delta T_{arr}^{R,L}[n] = 52.709$
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

5.4 Data Stream Synchronization: Results and Analysis

The objective of data stream synchronization is to ensure that the arriving sensor streams are streamed out according to their true measurement capture rate. Assuming that there is no error e_{est} in the measurement timestamp estimates, we define a data stream synchronization error term for corresponding measurements belonging to two streams as,

$$e_{dss}[k] = |\Delta\hat{T}_{mea}^{R,L}[k] - \Delta T_{out}^{R,L}[k]| \tag{5.2}$$

In our algorithm, the error e_{dss}[k] for buffered sensor measurements is zero since buffering time is set according to the original relationships between the measurements. Therefore, the error in stream synchronization is solely due to the measurements which are streamed out immediately due to late arrival. Another evaluating factor to consider is the number of measurements with *nowait* and *discard* events at output.

First, consider the algorithm's QoS parameters $max.\Delta\phi_{intra}$ and $max.\Delta\phi_{inter}$. These parameters indicate the worst case delay difference between measurements beyond which the measurement can either be deemed unusable and discarded or the offset

between the streams is corrected. This is illustrated in Figure 5.16, Figure 5.17 and Figure 5.18 for Radar and Lidar measurements. A fixed threshold to discard measurements can result in severe measurement loss, if there were to be an increase in variation of measurement latency during run-time. On the other hand, if a large threshold to tolerate measurement latency variations is chosen, then the measurements either have to buffer for longer periods of time or have shorter buffer periods at an expense of higher synchronization error (e_{dss}). Though the implemented algorithm overcomes the drawback of fixed threshold by adaptive reference(virtual) timer offsetting, it is still necessary to choose appropriate QoS parameters based on typical delay variations of the incoming data stream. Based on the Intra-SPD and Inter-SPD plots of the Radar and Lidar sensors, values of $max.\Delta\phi^{R}_{intra}$, $max.\Delta\phi^{L}_{intra}$ and $max.\Delta\phi^{R,L}_{inter}$ are set to $0.8ms$, $1ms$ and $2ms$, respectively.

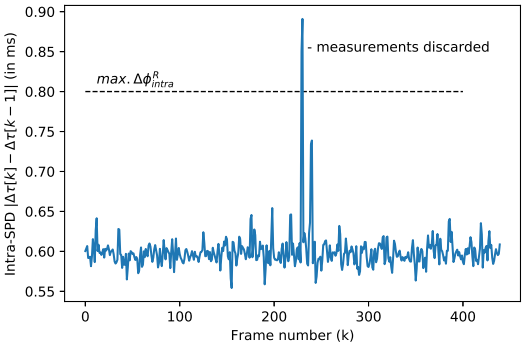

Figure 5.16: Intra-SPD values of Radar measurements.

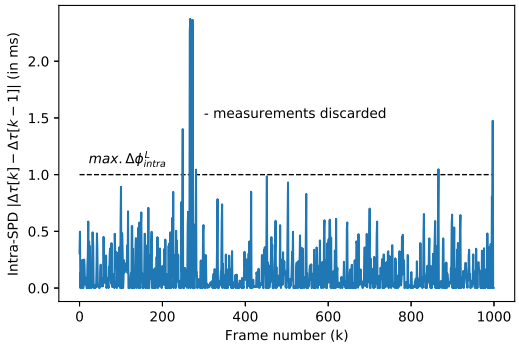

Figure 5.17: Intra-SPD values of Lidar measurements.

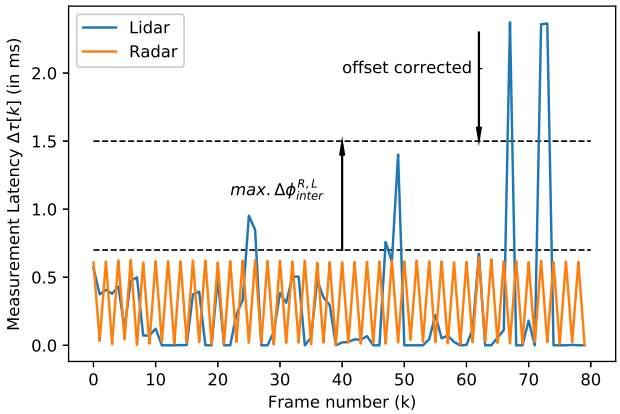

Figure 5.18: Measurement latency ($T_{arr}[k] - T_{mea}[k]$) of corresponding Radar and Lidar measurements.

The implemented data streaming algorithm is flexible and can be tuned to work within a tolerable buffering latency and synchronization error e_{dss} bound, even with

considerable measurement latency variations. For instance, a sophisticated sensor fusion system can tolerate higher levels of synchronization errors but may require the measurements at a faster rate. On the other hand, certain fusion algorithms could required an access to one or two previous measurements. The tuning of the algorithm according to the needs of the sensor fusion application can be done by setting the `T_WAIT`, `T_NOWAIT` and `T_DISCARD` thresholds of the adaptive buffer control algorithm. These three output event thresholds are determined as an allowable number of each event's occurrence over a maximum event counting window(W). Overall, ratios of the events ($R_{wait} : R_{nowait} : R_{discard}$) are selected as deemed fit for the application system and then multiplied by window size W to obtain the output event thresholds.

To analyse the effect of `T_WAIT`, `T_NOWAIT` and `T_DISCARD` thresholds, results of buffering latency and synchronization error (e_{dss}) values with varying buffer control configurations were recorded for 5000 arrivals of sensor measurements. We set the maximum calibrating factor Δ_{max} to the average delay variation observed in the stream (Avgerage Intra-SPD), which is $0.6ms$ and $0.3ms$ for Radar and Lidar streams, respectively. This ensures a smooth offsetting of timer reference. Some of the main inferences of the results are presented below :

1. **Effect of Window size (W):** Window Size (W) signifies the sensitivity of the buffer control algorithm, determining the frequency of advance or setback shift of the streams. This is proved by the trend in Figure 5.19 which shows the total number of setback and advance calibrations on the stream for varying window sizes. The ratio of the output events ($R_{wait} : R_{nowait} : R_{discard}$) was kept constant and set as (7:2:1). Further, we observed that the window size (W) did not have significant impact on the number of *wait*, *nowait* and *discard* events. (Figure 5.20). No noticeable trend in buffering latency or e_{dss} was observed, however, we cannot expect robust results from smaller window sizes as they may fluctuate the buffering process over even smaller changes in the arrival delay.

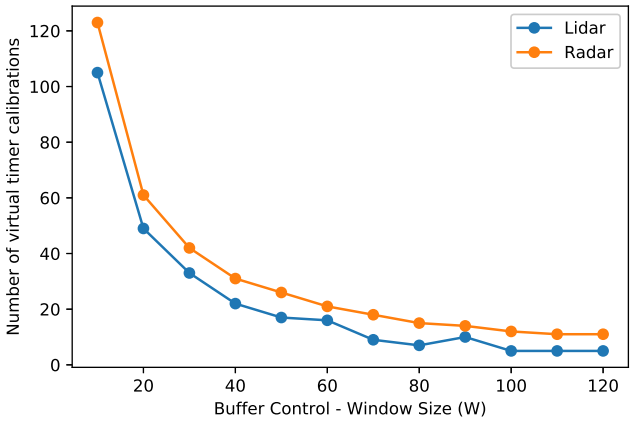

Figure 5.19: Effect of window size (W) on no. of calibrations.

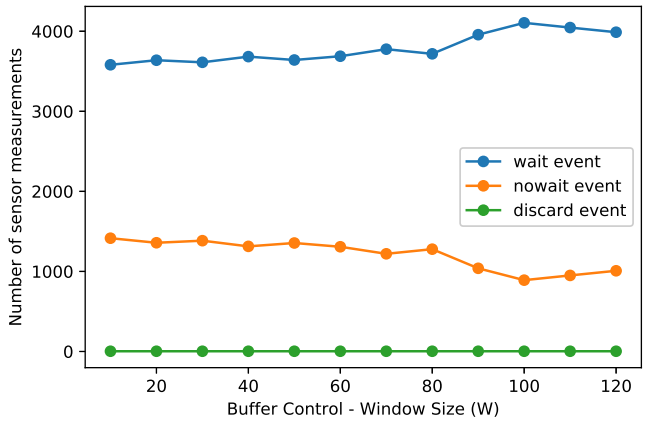

Figure 5.20: Effect of window size (W) on no. of output events.

2. **Effect of Output event thresholds:** Keeping window size W = 100, the total number of *wait*, *nowait* and *discard* events for different ($R_{wait} : R_{nowait} : R_{discard}$) values are presented in Table 5.3. We observe 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.

   Figure 5.21 and Figure 5.22 shows the box plots of buffering latency and synchronization error (e_{dss}) for different output event ratios, respectively. We can observe that with a relatively higher `T_WAIT` value, buffering latency increases and synchronization error decreases. We can expect this behaviour because the buffer control algorithm will frequently hit the under-buffer condition due to lower `T_NOWAIT` and `T_DISCARD` values 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 `T_WAIT` values synchronisation errors increase and buffering time decreases due to setting of over-buffering condition leading to advance calibration of virtual timer.

Table 5.3: Output event counts for different threshold ratios.

$R_{wait} : R_{nowait} : R_{discard}$	***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

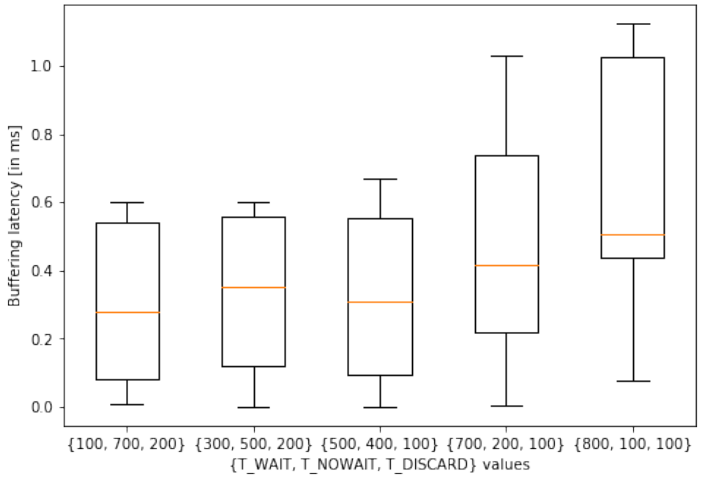

Figure 5.21: Effect of output event thresholds on buffering latency.

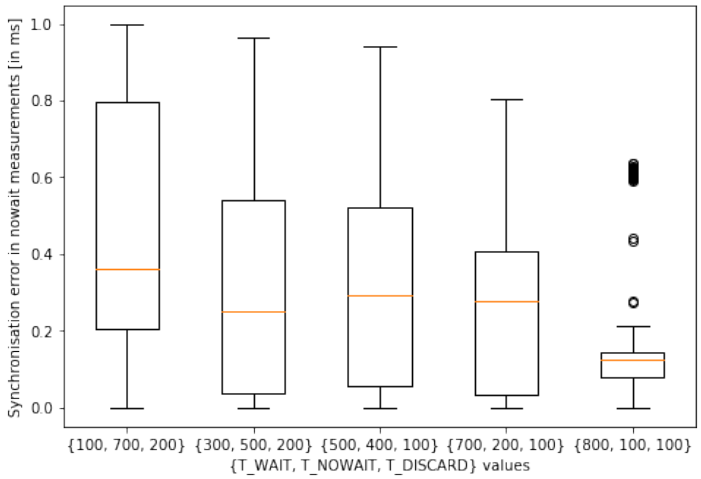

Figure 5.22: Effect of output event thresholds on synchronization error.

Based on the analysis presented above, the parameters can be selected based on the application's design goal. The most optimal (`T_WAIT,T_NOWAIT,T_DISCARD`) configuration in terms of latency and synchronization error is $(500, 400, 100)$ with average buffering latency of $0.32ms$ and synchronization error of $0.316ms$.

5.5 Summary

In this chapter, we develop an experimental methodology to evaluate the performance of our entire synchronization solution on real world sensor streams. A meccano contraption was setup and used as a common target object for both Radar and Lidar sensors. The position of the target as observed by the sensors are compared at output to verify the temporal synchronization process. An analysis of the final result with different filters for timestamp estimation is performed and a process to select the optimal

estimator based on resource requirement and quality of estimates is detailed. Overall, a Mean filter (W=16) for Radar and a Median filter for Lidar (W=9) were selected for our considered sensors.

Further, a detailed analysis on the implemented data stream synchronization algorithm is presented. The trends of synchronization error and buffering latency for different buffer control parameters is plotted and their effects on latency and synchronization error are summarized. With this, the right control parameters can be selected based on the application's latency and synchronization error requirement.

6 Hardware Implementation

6.1 Overview

In this chapter, the hardware implementation of the synchronisation solution is detailed. Further, the functionality and FPGA utilisation of each component and the full design is presented.

6.2 Full Design: Description

Figure 6.1 shows a simplified block diagram of the implemented synchronization solution. Overall, the design consists of a buffer in which the incoming sensor data is stored and eventually streamed out according to the decision of the synchronization block.

First, the arrival time of the incoming measurement is recorded by the `Timer` block. In addition, the `Timer` block is also responsible for the virtual clock-timer functionality. Next, the true timestamp ($\hat{T}_{mea}$) is estimated from the arrival timestamp in the `TS_Estimator` block. With true timestamp now known, the `Output_Decision` block decides on the output stream-out time of the measurement. Based on the decision, the sensor measurement is streamed out as AXI streams. The buffer control conditions i.e., over-buffer and under-buffer cases are checked and appropriate setback or advance commands on the virtual timer, if necessary, are given by the `Buffer_control` block. The `Inter_stream_Sync` block takes care of communications between the streams and overall ensures inter-stream synchronization.

The timestamps are recorded as 64-bit unsigned integers, which is a 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 and thus, the bit-widths of timestamps, if required, can be changed accordingly.

6.3 Components

1. `TS_Estimator`: The `TS_Estimator` unit estimates the true cycle time of the incoming sensor measurements by means of a filter based estimator. Two versions of the `TS_Estimator` block with Mean filter and Median filter, respectively was implemented. Further, this unit is responsible for generating the estimated measurement timestamps by adding up cycle time estimates and employing correction methods (Section 3.5) during exceptions like packet loss. Overall, the estimated measurement timestamp is available 2 clock cycle after the arrival of the sensor measurement, since it requires the sensor cycle time as observed by the `Timer` unit.

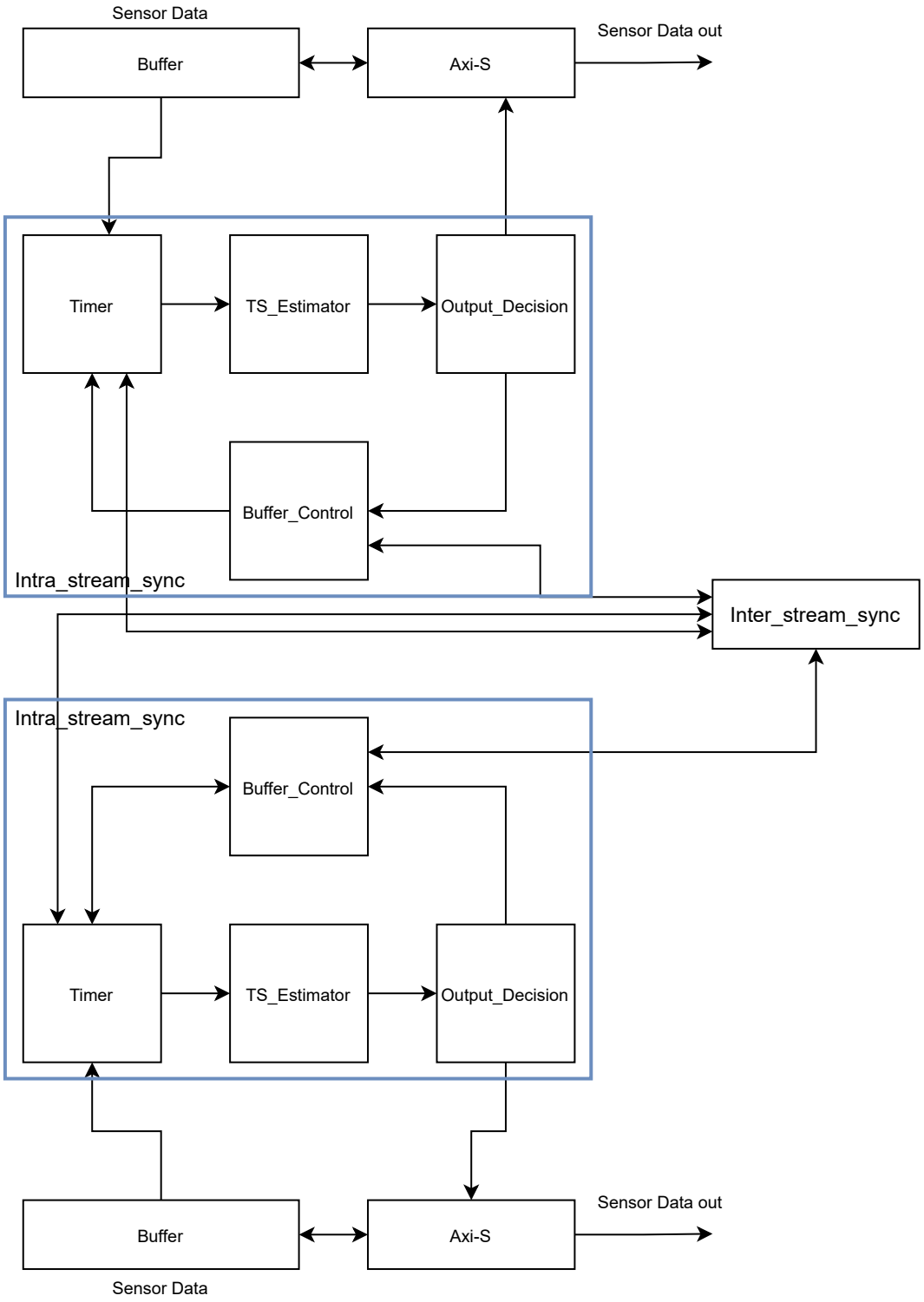

Figure 6.1: Block diagram of the implemented synchronization solution.

The architecture of the mean filter consists of a data array with length of the window size(W) of the filter and an accumulator which adds the incoming data and subtracts the outgoing data. The division by W operation is straightforward if its value is a power of 2. If not, it is implemented as a multiplication by a constant ($\frac{1}{W}$) operation.

Similarly, the median filter is implemented as a data array of length equal to the Window size (W) of the filter and a comparison network to insert the incoming data into a sorted array. The median value is then selected as the value at the middle of data array.

The filters operate on the arriving cycle times, whose values are in range of hundreds of milliseconds. Hence, smaller bit-widths can be used to represent the filter input data. In the implemented design, 18-bit filter input was used. This greatly reduces resource utilization as well as the filter latency as the complexity of the adder circuit in Mean filter and the comparison circuit in Median filter are reduced.

2. `Output_Decision`: The `Output_Decision` unit compares the arrival timestamps and the estimated true timestamp values to decide whether the measurements are to buffered, streamed or to be discarded, according to the Output decision equations in Section 4.3.1 . In addition, it calculates the buffering time and updates the `wait`, `nowait` and `discard` counters which is a basic addition operation.Its implementation is simple, consisting of adders and muxes.

   The decision of `wait`, `nowait` and `discard` and the buffering time is calculated by the next clock cycle after the availability of the estimated measurement timestamps from the `TS_Estimator` block. It is to be noted that the buffering time value is corrected for the latency of the `TS_Estimator` block and its own latency of 1 clock cycle. In addition, it also considers the latency of streaming out sensor measurements.

3. `Timer`: The `Timer` unit is basic timer with options of setback and advance calibrations. Essentially, it functions as the virtual clock-timer T_{vt}. It considers the appropriate flag signals set by the `Buffer_Control` and `Inter_stream_Sync` units and carries out setback and advance operations accordingly.

   The clock frequency for the timer depends on the timestamps' precision required for synchronization. In our system, $30ns$ precision is used.

4. `Buffer_Control`: The `Buffer_Control` unit is responsible for checking the under and over buffer conditions and accordingly calculate the setback and advance shift factors (Δ^- and Δ^+). These factors are given to the `Timer` unit in order to carry out setback or advance calibration the Virtual clock-timer. In addition, it sends signals to the `Output_Decision` block to reset the corresponding output event counters.

To ensure minimal latency an to reduce hardware utilization, certain optimizations were considered in the calculations of the shift factors (Δ^- and Δ^+). Consider the Equation for Δ^- calculation :

$$\Delta^- = (1 - \frac{wait\_cnt}{\texttt{T\_WAIT}})\Delta_{max} \tag{6.1}$$

(a) If `T_WAIT` and Δ_{max} values can be chosen as powers of 2, then the multiply-divide operations reduces to several shift operations. However, this may give very few choices to select `T_WAIT` and Δ_{max} values from.

(b) If `T_WAIT` and Δ_{max} values are NOT powers of 2, then the division operation can be replaced by multiplication of a constant $\frac{1}{(\frac{\texttt{T\_WAIT}}{\Delta_{max}})}$. This is performed in 2 steps:
Step 1: The divider is quantified as a fixed point number pre-computation. The accuracy of the result depends on the number of bits used for quantifying the divider.
Step 2: Only multiplication is performed, followed by right shift by number of bits used for quantifying the divider.

Overall, with this method, the shift factor value calculation requires 1 Multiplication operation followed by a right shift operation. Since the clock period is $30ns$, a single cycle 18-bit multiplication operation would suffice both timing and resource wise. Multiple cycle higher radix multipliers can be used in case of higher clock frequency since the `Buffer_Control` unit runs in the background i.e., in between the arrival of sensor measurements and hence, does not add to the latency of synchronization decision.

5. `Inter_stream_Sync`: The `Inter_stream_Sync` unit is responsible setting calibration signals for the virtual timer based on inter-stream synchronization conditions (Section 4.3.2). Further, it also set the reference stream by comparing the virtual timers of different streams. Its implementation is straightforward consisting mainly of adders and multiplexers.

6. `Axi_S`: The `Axi_S` unit is responsible for reading sensor data from a buffer and streaming it out as AXI streams based on the decision given by `Output_Decision` unit. It also includes a timer to stream out the measurements at the right output time.

   The latency of streaming depends on the size of the sensors' data block:

   - Radar produces 262144 bytes of raw samples from a single receiver and is represented as 16-bit integers. Hence, 16,384 clock cycles is required per raw sample to stream out one radar measurement.
   - Lidar produces 153600 bytes of data for (depth map + point cloud) and is represented as 32-bit integers. Overall, 4800 clock cycles is needed to stream out the Lidar measurement.

The above streaming latency is taken into account in the calculation of measurement buffering time. However, for the measurements that are streamed out immediately (*nowait* case), this latency adds to the synchronization error e_{dss}.

6.4 Evaluation

In the current system, the sensor interfaces to read the sensors directly by the FPGA are not yet developed. Therefore, for the purpose of functional verification of the implementation, the sensor measurements and the corresponding arrival times are stored in a block memory. The measurements and its timestamps are taken as input according to the arrival times, thus, emulating the arriving sensor measurements. Overall, the functionality of the implemented hardware solution was verified by simulation.

Hardware Utilization

The synthesis results of the hardware implementation are based on the ZYNQ-7 ZC702 Evaluation Board. Resource utilization of various components of the implemented solution are presented in Table 6.1. The last row shows the total utilization of the FPGA's resources for the full design of our synchronization solution for two streams. Further, in order to directly interface any sensor with the systolic array system, additional memory to temporarily store incoming sensor measurements is required. Overall, to store/buffer one set of Radar and Lidar measurement 415744 bytes of memory is necessary. Exact memory requirements can only be determined based on the latency and past sensor measurement requirements of the sensor fusion algorithm.

Table 6.1: Resource Utilization of various components in the considered platform.

Component	**LUTs**	**Registers**	**DSP**
TS_Estimator (Mean filter (W = 16))	378	569	0
TS_Estimator (Median filter (W = 9))	462	443	0
Output_Decision	186	88	0
Timer	233	136	0
Buffer_Control	52	4	1
Inter_stream_Sync	295	133	0
Axi_s	18	19	0
Full Design	2105 (3.9%)	1639 (1.5%)	2 (0.9%)

Latency

For an incoming sensor measurement, the latency of obtaining its output decision is due to the `Timer`, `TS_Estimator` and `Output_Decision` units, which adds up to 3 clock cycles. Further, the latency of streaming out data depends on the data type of the sensor measurements. Latency of 16,384 clock cycles and 4800 clock cycles are needed to stream out one Radar and Lidar measurement, respectively. With clock period of

$30ns$, we arrive at a total latency of $491610ns$ and $144090ns$ for each Radar and Lidar measurement, respectively. The above delay is corrected for, in the calculation of the buffering time and hence, has no impact on the buffered measurements. However, for sensor measurements that are to be streamed out earlier than the above delay, it adds to synchronization error (e_{dss}).

6.5 Summary

In this chapter, the details of the hardware implementation of our synchronization solution is presented. Certain modifications were made to the algorithm for better hardware implementation in terms of resources and latency. These include choosing the right bit-width for cycle times, simplifying the calculations of timer calibration shift values and the overall design of buffer control running in the background to ensure least latency on the incoming sensor measurement. The entire design was verified by simulation and synthesized on a FPGA board. Resource utilization and latency of the solution on the considered platform was tabulated. Further, we correct the algorithm to include this latency in the buffer time calculation of the measurement. However, this latency adds to the synchronization error (e_{dss}) when the buffering time is less than this latency.

7 Conclusion

7.1 Conclusions

After the analysis of the results, we now revisit our initial key objectives of the thesis, mentioned in Section 1.3 and conclude our findings.

1. **Handle the problem of estimating the actual time of sensor measurement:** A Filter-based technique was employed to estimate the timestamp of sensor measurement capture. First, true measurement cycle times of the sensors are estimated from observed sensor cycle times (from arrival timestamps) by filtering out the jitter. Well-known filtering techniques such as Kalman filters, mean filters and median filters were considered. Based on the characteristics of observed cycle times from arrival timestamps, the complexity of techniques and the quality of obtained estimates, the appropriate filter was selected for implementation. Kalman filter was ruled out since it is a complex filter for hardware implementation and with no observed drift in the cycle times, a simple filter would be sufficient for the task. Further, Mean and Median filters were evaluated for our considered sensors in terms of resource utilization and final synchronization results (Section 5.3.1). A Mean Filter(W=16) and a Median filter(W=9) were selected for estimating Radar and Lidar cycle times, respectively. Finally, the actual time of sensor measurement was obtained by simply adding up cycle time estimates and employing corrections during exceptional cases like packet loss and a possible estimation error. Overall, the above approach can be used on all types of sensors as it does not require any extra support like interfaces, timestamps or frame numbers from the sensor side.

2. **Ensure synchronization of incoming sensor streams with minimal latency and synchronization error while tolerating uncertain arrivals of sensor data:** Several existing stream synchronization algorithms which address the same problem in the area of distributed multimedia systems were considered. Since there is a trade-off between between buffering latency and synchronization error, an adaptive buffering technique is employed which consists of a control algorithm which keeps latency and synchronization at check by changing the buffering delays during run-time according to the delay jitter conditions. The control algorithm can be tuned with the help of parameters to choose the right levels of latency versus synchronization error, according to the needs of the application. A detailed analysis on the effects of these control parameters on buffering latency and synchronization error is presented.

3. **Efficient implementation of the synchronization solution on hardware:** Several modifications were made to the proposed synchronization solution for efficient hardware implementation in terms of resource use and latency. Mean filters with 2^n window size values were considered to remove the division operation without considerable change in the estimation error. Calculation of timer calibration shift values were simplified to involve less number of operations. Bit-widths for timestamps and sensor cycle times were selected with careful consideration of required precision for minimum synchronization error and resource utilization. Also, the entire solution was designed to keep the latency on the incoming sensor measurement minimum. This was done by running the adaptive buffer control algorithm in the background between sensor measurements arrivals. The entire implementation for two streams requires 2105 LUTs, 1639 Registers and 2 DSPs. Overall, the latency of obtaining the output decision (*wait, nowait, discard*) of a sensor measurement is 3 clock cycles. Additionally, there is latency due to streaming out of sensor data. With clock period of $30ns$, total latency of $491610ns$ and $144090ns$ is experienced by Radar and Lidar measurements, respectively. This latency is corrected for, in the calculation of the buffering time and hence as no impact on the buffer measurements. However, for sensor that are to be streamed out earlier than the above latency, it adds to the synchronization error (e_{dss}).

4. **Develop a methodology to evaluate the performance of the temporal synchronization solution on real world data:** To verify the accuracy of the temporal synchronization solution, a moving Meccano contraption (Figure 5.1) was set up to be used as a common target object for both Radar and Lidar sensors. This setup ensures that the position measurements obtained by both Radar and Lidar are distinguishable for comparison and that the spatial error in the position measurement is negligible. Thus, the differences in the positions of the common target observed from Radar and Lidar sensors are solely due to the temporal errors. However, for this thesis, exact position measurements were not extracted. Certain orientations of the common target were selected visually and compared at the output of synchronization. If the synchronization is sufficiently accurate, we can expect Radar and Lidar measurements pointing to the same orientation to have closer output times. Estimator selection for the sensor streams was also done by observing the association of sensor measurements at output. However, since the exact target positions were not extracted from the sensor measurements, the exact error in the estimates could not be determined. Overall, this setup though simple and inexpensive is effective and sufficient enough to evaluate the correctness of the temporal synchronization solution.

7.2 Limitations

A few limitations of the thesis are listed below:

- The acquisition of sensor measurements, including arrival timestamping was done on a machine with Windows10 OS, which does not guarantee real-time requirements and can introduce lot of 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. Additionally, there is a possibility that due to large operating system delays, we do not observe a drift in the cycle times over time.

- The implemented synchronization solution was verified for correctness by only observing close association between measurements. Since target position extraction was not done, we could not verify if the exact original intra and inter temporal relations were maintained at the output.
- The synchronization solution wasn't tested for long periods of sensor data, since the association between measurements were manually confirmed by visual observation.

7.3 Future Works

Some interesting ideas to follow up as an extension to this thesis are listed below:

- The implemented synchronization solution can be integrated to read sensor data streams directly from the FPGA sensor controllers.
- Precise error estimation can be done by exact position extraction of the target from synchronization event measurements. Also, the solution needs to be evaluated for longer periods of reliable sensor data, preferably acquired on a real-time operating system or directly on hardware.
- Further, performance analysis can be done on different sensors in the market. A linear Kalman filter can be used to estimate true cycle times in sensors with jitter and additive drift in cycle times.
- The proposed synchronization solution can be evaluated based on the final sensor fusion results. Additionally, an exploration can be performed to find tolerable levels of temporal asynchrony for different sensor fusion algorithms. Specifically, CNN data fusion algorithms for the PRYSTINE project.
- The synchronization solution can be scaled to include more sensors. With respect to the PRYSTINE project, Camera streams can be added.
- The synchronization solution can be extended to be used for synchronizing several sensors connected over a network and test the limits of the data stream synchronization solution for extreme cases of network delays.

Bibliography

[1] "Programmable systems for intelligence in automobiles," https://prystine.eu/.

[2] "Levels of driving automation," https://www.vox.com/2016/9/19/12966680/department-of-transportation-automated-vehicles.

[3] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2018.

[4] B. Duraisamy, T. Schwarz, and C. Wöhler, "On track-to-track data association for automotive sensor fusion," in *2015 18th International Conference on Information Fusion (Fusion)*, 2015, pp. 1213–1222.

[5] T. Huck, A. Westenberger, M. Fritzsche, T. Schwarz, and K. Dietmayer, "Precise timestamping and temporal synchronization in multi-sensor fusion," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 242–247.

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

[7] C. KWOK, D. FOX, and M. MEILA, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 469–484, 2004.

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

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

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

[11] 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)*, 2004, pp. 1627–1635.

[12] J. 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.

[13] R. Kálmán, "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic," 1960.

[14] "Tutorial: The kalman filter," http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf.

[15] R. Boyle and R. Thomas, "Computer vision: A first course," 1988.

[16] M. Chen, "A low-latency lip-synchronized videoconferencing system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2003, pp. 465–471.

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

[18] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437908000525

[19] M. Montagud, P. Cesar, F. Boronat, and J. Jansen, *MediaSync: Handbook on Multimedia Synchronization*, 03 2018.

[20] S. Baqai, M. F. 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] T. Shuji and Y. Ishibashi, "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.

[27] 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*, 2004, pp. 604–611.

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

[29] Y. Xie, C. Liu, M. J. Lee, and T. N. Saadawi, "Adaptive multimedia synchronization in a teleconference system," in *Proceedings of ICC/SUPERCOMM '96 - International Conference on Communications*, vol. 3, 1996, pp. 1355–1359 vol.3.

[30] 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 vol.3.

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

[32] H. Liu and M. Zarki, "A synchronization control scheme for real-time streaming multimedia applications," 2003.

[33] "Trading systems," http://traders.com/Documentation/FEEDbk_docs/2005/03/Abstracts_new/Ehler/ehler.html.

[34] "Median filter," https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm.