Technische Universität Berlin

TU Berlin Industrial Automation Technology Department
Fraunhofer Institute for Production Systems and Design Technology
Pascalstraße 8-9, 10587 Berlin, Germany

add IPK logo.

Bachelor Thesis

Development and Evaluation of a Manufacturer-Independent Synchronization Framework for GenICam Industrial Cameras

Yessmine Chabchoub

Matriculation Number: 465977

Berlin, January 29, 2025

Supervised by
Prof. Dr.-Ing. Jörg Krüger
Prof. Dr. Sabine Glesner

Assistant Supervisor
M.Sc. Oliver Krumpek

Abstract

Update this with your abstract content.

Declaration of Authorship

I hereby declare that I have prepared this Bachelor's thesis independently, without external assistance, and have used no sources or aids other than those indicated. All passages taken literally or in substance from other sources are clearly identified as such.

Sign this.

Berlin, January 29, 2025

Signature

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

With the growing need for optimization in manufacturing—particularly in labor-intensive processes like sorting and classification—machine vision emerged in the late 20th century as a technology that revolutionized industrial automation.

In its early stages, machine vision was primarily hardware-based. High-resolution cameras capture images under carefully predefined conditions, including lighting, object positioning, and camera angles. Deterministic algorithms then processed these frames to determine outcomes, such as approving or rejecting a product during quality inspection. However, with recent advancements in machine learning—machine vision being no exception—the field has shifted increasingly toward software-driven solutions. Deep learning models, in particular, are now used to perform process control and quality assurance tasks. Consequently, machine vision has evolved into a key enabler of computer vision, offering capabilities including image recognition, object detection, and semantic segmentation.

The rapid expansion of machine vision applications has also triggered significant shifts in the camera manufacturing industry. The global market for industrial cameras continues to grow [3], prompting an increase in industrial camera manufacturers. To address this diversification, standards such as GenICam [4], developed by the European Machine Vision Association (EMVA [5]), have been introduced. These standards aim to regulate and streamline the operation of cross-manufacturer cameras, thereby reducing integration costs and simplifying implementation for both users and developers.

1.2 Problem Statement

As machine vision evolves into computer vision, unlocking capabilities for more dynamic settings introduces several challenges.

One primary challenge is the reliance on machine learning models. While traditional machine vision systems employ deterministic algorithms to process predefined inputs, the integration of artificial intelligence requires large, diverse datasets to handle real-world variability effectively. For example, in fault detection, 3D data provides significantly greater accuracy than grayscale 2D data. However, acquiring 3D images—for instance, through contact 3D scanners—can be prohibitively expensive [6].

As a passive, non-contact 3D scanning method, stereoscopy offers a viable alternative for capturing different perspectives, but a multi-camera setup adds more complexity. One issue is integrating multiple cameras from various manufacturers into a single system. Although standardization efforts like GenICam strive to improve cross-manufacturer compatibility, many vendors still rely on proprietary software tied to their hardware, hindering seamless integration.

Another concern involves synchronizing frames from more than one camera. In many machine vision applications—such as industrial inspection and robotic guidance—perfect timing is crucial to avoid artifacts caused by motion, lighting inconsistencies, or other environmental factors. Even slight delays can lead to errors or missed defects, which can be particularly costly in high-speed manufacturing contexts. These scenarios introduce strict timing constraints, requiring precise synchronization across all sensors and cameras.

1.3 Goal and Scope

This thesis proposes a manufacturer-independent solution for configuring and controlling a synchronized multi-camera setup. The approach involves evaluating hardware and software options for camera synchronization, implementing the chosen method(s) in a unified framework and evaluating the resulting outcomes. This framework will enable users to easily configure and trigger synchronized recordings across up to six cameras.

The primary goal is to prevent hardware from becoming a bottleneck in various industrial applications. To this end, the project will leverage the GenICam standard alongside existing timestamps synchronization technologies. It will develop a scalable synchronization method and create an intuitive graphical user interface (GUI). This GUI will simplify the configuration and operation of the synchronization system, making it more accessible and straightforward for end users.

Beyond the scope of this thesis are tasks for synchronizing additional hardware (e.g., lighting or external sensors) and supporting non-GenICam-compliant devices. This work does not address data processing tasks, machine learning integration, advanced object detection algorithms, or real-time data analysis. While these areas are critical for specific applications, they fall outside the immediate focus of this project, which is primarily concerned with camera synchronization and user interface development. These expected outcomes can be summarized as follows:

- **Evaluation of existing camera synchronization methods:** Includes an in-depth review of both hardware-based and software-based synchronization mechanisms. The evaluation will also focus on the key advantages and drawbacks of the different approaches, considering factors like integration, scalability, and costs.
- **Requirements specification and concept definition:** Details the hardware and software specifications, including network dependencies. It also identifies performance metrics for timing accuracy, sets latency thresholds, and defines constraints related to network bandwidth. These specifications ensure the system's scalability and adherence to industrial standards. From these requirements, implementation concepts are derived and illustrated, for instance, through diagrams.
- **Implementation of a multi-camera synchronization solution:** Builds a software layer for synchronous acquisition-triggering and frame-timestamping of a multi-camera setup.
- **Development of a GenICam-based graphical user interface (GUI):** Streamline a GUI for the configuration and management of diverse cameras. This GUI features options for camera recognition, features control, and synchronous acquisition triggering.
- **Testing and evaluation:** Validates the proposed solution's functionality, reliability, and performance. The tests ensure that the system meets specified requirements and can be integrated effectively into industrial environments.

1.4 Outline

This thesis follows the outlined structure:

Chapter 2: Defines fundamental terms and technologies (GenICam, Synchronization, PTP..) and compares current implementations.

Chapter 3: **Chapter 3**: Outlines the design of the proposed solution, explaining and justifying design choices. Introduces the software, hardware, and network requirements.

Chapter 4: Describes the implementation results and final product.

Chapter 5: Evaluates the results, detailing testing methodologies and validation criteria.

Chapter 6: Concludes the thesis by summarizing primary findings and highlighting encountered challenges.

2 Literature review / State of the technology

This chapter covers related work in both industrial and academic contexts for synchronizing and controlling multi-camera systems. We discuss common methods and protocols for synchronization, followed by an overview of popular camera streaming and control libraries used in machine vision. The latter part evaluates the existing approaches and discusses the open challenges.

2.1 Synchronization

In industrial high-performance cameras, synchronization refers to the alignment of frames in time. In the industrial computer vision context, this typically means synchronizing frames with precision ranging from microseconds to milliseconds [7]. While achieving such accuracy can be challenging, the issue of synchronization is not a new one. In the following sections, we will review the most commonly used approaches to address this problem. These methods are categorized into two main types: post-capture and pre-capture synchronization techniques.

2.1.1 Post-Capture Synchronization

Post-capture synchronization methods estimate the exact timing or timestamps of frames from multiple cameras **after** recording. These approaches typically rely on matching temporal markers or events to compute offsets and align frames.

A widely used post-capture strategy is *feature extraction*, in which the system identifies and correlates common "fingerprints" or events. These can be visual (such as a sudden flash of light), audio (such as a sharp noise), or audio-visual (a combined cue). By applying an adaptive threshold to detect luminance variations across frames, [8] demonstrates how one can identify a flash event and match it across all videos, thereby determining the offset between the recordings. For instance, Figure 2.1 presents four sequential frames alongside their corresponding luminance histograms. The second frame exhibits notably higher brightness, which can serve as a detectable event. Similar approaches rely on audio signals or audio-visual events are discussed in [9] and [10].

An alternative approach extends feature extraction by injecting an independent event into the common field of view of all cameras, then extracting it to enable more precise sub-frame timestamping [11]. Sub-frame timestamping means determining not only that an event occurred at, for example, "Frame 10" but also whether it happened "12 ms into Frame 10." This allows synchronization accuracy at the sub-frame level.

In addition to feature-based approaches, other specialized algorithms have been proposed to enhance multi-camera alignment. One such method, known as *bit-rate-based synchronization*, leverages fluctuations in the compressed video bit rate as an alignment signal, using a statistical measure called correntropy [12]. Another approach employs *deep learning* to automatically extract high-level features and estimate temporal alignment across multiple streams [13]. Finally, for Time-of-Flight cameras, *optical synchronization* techniques based on Time-Division Multiple Access (TDMA) help mitigate cross-talk and ensure precise alignment of optical signals [14].

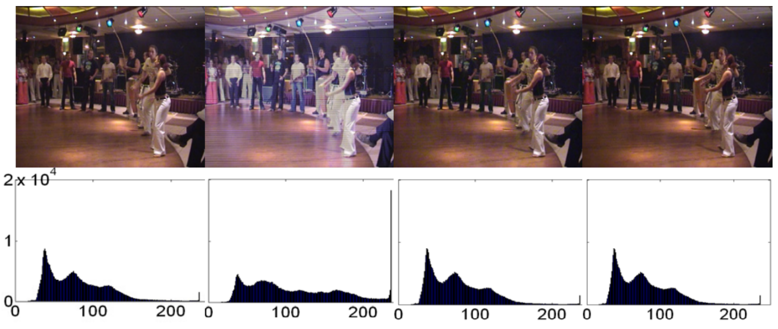

Figure 2.1: Example frames and associated brightness distributions. The second frame's increased luminance, as evidenced by its histogram, can be extracted as a feature for multi-camera synchronization.

Although post-capture methods are relatively easy to deploy—since they do not require specialized hardware—they depend on clearly identifiable events (e.g., flashes or audio cues) that must be visible in all camera feeds. In some environments, such events may be infeasible or difficult to reproduce, and poor signal quality or environmental noise can further impair feature detection. Moreover, synchronization is exact only for frames containing a detected feature, leaving other frames dependent on interpolation, which introduces additional uncertainty.

2.1.2 Pre-Capture Synchronization

Pre-capture synchronization methods ensure that cameras are synchronized **during** acquisition, rather than relying solely on post-processing. This is particularly useful when real-time synchronization is required, or to minimize the complexity of post-processing algorithms.

Trigger-Based Synchronization

One of the most common pre-capture techniques is *trigger-based synchronization*, in which a trigger signal initiates coordinated image acquisition. Triggers can originate from either hardware or software [15]. In *software triggering*, a command is sent from the manufacturer API and travels from the host operating system to interface protocol (e.g., GigE or USB) before reaching the camera firmware, leading to higher and less predictable network overhead.

By contrast, *hardware triggering* occurs when an electrical signal—either from an external device (e.g., a microcontroller or sensor) or an internal source (e.g., a timer or counter)—directly toggles registers in the camera. This generally provides lower latency and more deterministic timing. For example, [16] describes an Arduino-based circuit used to synchronize a camera with a structured-light projector. Figure 2.2 provides an example of a timing diagram for a camera configured to begin acquisition whenever it detects a rising edge on its I/O line.

Triggering offers reliable, low-latency synchronization and can be tied to other system components (e.g., light sources or motorized stages) to precisely control exposure timing. This is particularly beneficial in scenarios involving scanning or strobing illumination, or when working with samples that rapidly bleach or decay. However, triggers alone do not correct for *clock drift* over extended periods, which can become problematic in large-scale deployments where even millisecond-level offsets may be insufficient for high-precision applications.

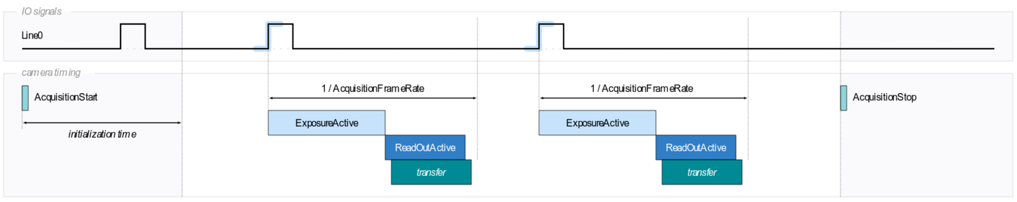

Figure 2.2: Example timing diagram illustrating hardware-triggered image acquisition [1]: Each pulse on Line0 initiates a new frame at the specified acquisition rate, and acquisition terminates upon receiving an *AcquisitionStop* command.

Network-Based Synchronization

To mitigate the long-term clock drift that can accumulate even with hardware triggers, many distributed camera systems incorporate network-based synchronization. By operating over Internet, these cameras can leverage standard clock protocols such as the Network Time Protocol (NTP)[17] or the Precision Time Protocol (PTP) [18]. These protocols focus on keeping the internal clocks of networked devices aligned. In some configurations, the cameras synchronize only their clocks, relying on separate triggers for actual frame capture; in others, the network synchronization governs the exact shutter release moments. NTP uses an hierarchical configuration, as showed in 2.3 where each layer is identified by a Stratum level (0-15), that represents the number of hops between nodes necessary to reach the root (level 0), meaning that as the stratum level increases so do the clock drifts and time inaccuracies. Therefore, sub-networks are organized in a way where various devices connect to higher stratum level servers in order to maintain an equal time reference.

NTP, https://tubcloud.tu-berlin.de

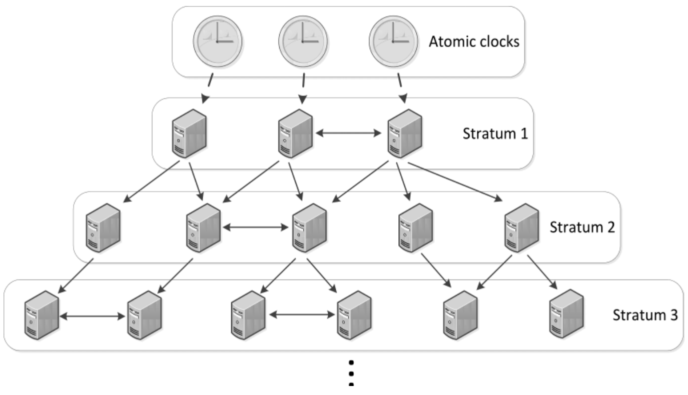

Figure 2.3: Architecture of the NTP with the different layers or strata [2].

2.1.3 Synchronization and Interfaces

Although clock synchronization protocols, such as PTP, are highly effective for GigE-based (Ethernet) cameras, they are not natively supported by other popular interfaces, such as USB3, CSI, or CoaXPress. Efforts are underway to extend the capabilities of these interfaces to achieve more precise timing. For instance, USB3 cameras can utilize voltage control as a timing mechanism, effectively synchronizing their internal clocks by adjusting power supply levels [19]. In the case of CoaXPress, synchronization is relatively straightforward, as a single frame grabber can handle precise timing for multiple cameras, using long and cost-effective coaxial cables [source]. For CSI cameras, software-based synchronization methods are often preferred due to their simplicity and ease of integration into embedded systems [source].

gige vs gige2? or in requirements

reference table and complete it

Synchronization Method	**Interface**			
	GigE	**CSI**	**USB3**	**CoaXPress**
Trigger				
Clock				
Software				
Accuracy	2 ms [?]	2 ms [?]	2 ms [?]	2 ms [?]

Table 2.1: Comparison of Synchronization Methods Across Different Camera Interfaces

2.2 Control and Configuration of Cameras

2.2.1 GenICam Standard

GenICam (Generic Interface for Cameras), developed by the European Machine Vision Association (EMVA), standardizes camera control and data exchange across different manufacturers:

- **SNFC (Standard Features Naming Convention)**: Defines how camera features are named and structured using an XML file.
- **GenTL (Transport Layer)**: Specifies data streaming between hardware producers (cameras) and software consumers (host applications). Vendors provide a `.cti` file implementing these functions.
- **GenApi**: A higher-level API that reads the XML description and translates user commands (e.g., changing frame rate) into low-level operations for each camera.

GenICam aims to simplify multi-vendor setups but does not inherently solve synchronization or cross-manufacturer interoperability challenges.

2.2.2 Aravis

Aravis is an open-source library designed to handle camera discovery, configuration, and streaming for GigE Vision and USB3 Vision cameras. It partially implements GenICam but does not rely on vendor-specific `.cti` files. Tools include:

- **aravis-tool**: Command-line utility for camera discovery and basic configuration.
- **aravis-viewer**: A GStreamer-based GUI for live image preview.

While Aravis simplifies multi-vendor setups, it focuses on image acquisition rather than camera synchronization.

2.2.3 Harvesters

Harvesters is a Python-based framework built atop GenICam Transport Layer Producer libraries. It facilitates multi-camera acquisition and control through an intuitive interface. Like Aravis, Harvesters does not provide robust synchronization mechanisms; users typically employ triggers or external protocols to coordinate cameras.

2.2.4 eBus Player

eBus Player, developed by Pleora, is a proprietary application commonly used for GigE Vision camera discovery, configuration, and real-time viewing. It offers useful diagnostic and performance measurements but lacks built-in solutions for precise multi-camera synchronization.

2.2.5 Vendor APIs

Vendor-specific APIs often provide advanced features tailored to their cameras, including synchronization options, but these solutions are typically limited to specific hardware ecosystems.

2.3 Conclusion

mention other ptp works

Synchronization of multi-camera systems has been explored extensively in both industry and academia. Early industrial setups relied heavily on hardware triggers in star or daisy-chain configurations. Although reliable, these solutions did not address clock drift without additional correction mechanisms. As cameras gained network connectivity (e.g., GigE Vision), software triggers became more common, offering simpler wiring at the cost of added latency and jitter.

To address higher accuracy needs, many industrial systems now integrate clock synchronization protocols like PTP. This approach allows cameras to maintain closely aligned clocks, reducing drift even in distributed networks. Some vendors offer proprietary blends of hardware triggers and custom synchronization logic, but these are typically manufacturer-specific and limit interoperability.

In research contexts, alternative methods—such as machine learning-based synchronization using shared features or event cues—have been investigated. While promising, they are less prevalent in time-critical manufacturing environments where deterministic behavior is paramount.

Overall, no single universally accepted standard comprehensively covers multi-camera synchronization. GenICam simplifies camera control but does not solve the complexities of sub-millisecond or microsecond-level synchronization in heterogeneous systems. Hence, there is a clear need for a vendor-agnostic, flexible synchronization framework that leverages established protocols (e.g., PTP) and remains compatible with GenICam-based devices.

3 Concept

This chapter introduces the architectural design of Component X. The component consists of subcomponent A, B and C.

In the end of this chapter you should write a specification for your solution, including interfaces, protocols and parameters.

3.1 Requirements

This section determines the requirements necessary for X. This includes the functional aspects, namely Y and Z, and the non functional aspects such as A and B.

3.2

3.3 Overview

In this chapter you will describe the requirements for your component. Try to group the requirements into subsections such as 'technical requirements', 'functional requirements', 'social requirements' or something like this. If your component consist of different partial components you can also group the requirements for the corresponding parts.

Explain the source of the requirements.

Example: The requirements for an X have been widely investigated by Organization Y.

In his paper about Z, Mister X outlines the following requirements for a Component X.

3.4 Technical Requirements

The following subsection outlines the technical requirements to Component X.

3.4.1 Sub-component A

Interoperability
Lorem Ipsum...

Scalability
Lorem Ipsum...

3.4.2 Sub-component B

Lorem Ipsum...

3.5 Sub-component A

The concept chapter provides a high-level explanation of your solution. Try to explain the overall structure with a picture. You can also use UML sequence diagrams for explanation.

Figure 3.1 illustrates the situation between Alice and Bob. (sequence diagram from www.websequencediagrams.com)

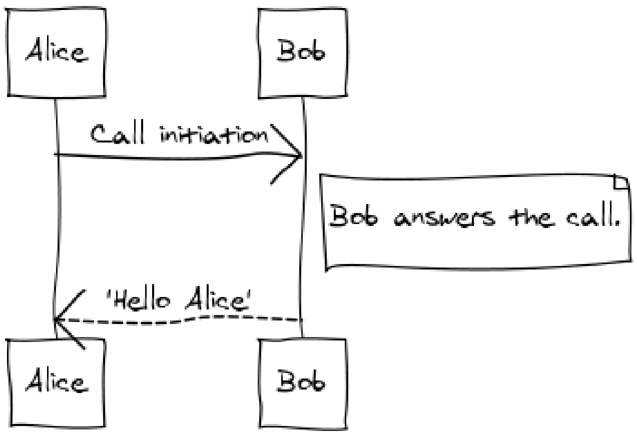

Figure 3.1: Alice and Bob

3.6 Sub-component B

Lorem Ipsum...

3.7 Proposed API

Lorem Ipsum...

3.8 Layer X

Lorem Ipsum...

3.9 Interworking of X and Y

Lorem Ipsum...

3.10 Interface Specification

Lorem Ipsum...

4 Implementation

This chapter describes the implementation of component X. Three systems were chosen as reference implementations: a desktop version for Windows and Linux PCs, a Windows Mobile version for Pocket PCs and a mobile version based on Android.

4.1 Environment

The following software, respectively operating systems, were used for the implementation:

- Windows XP and Ubuntu 6
- Java Development Kit (JDK) 6 Update 10
- Eclipse Ganymede 3.4
- Standard Widget Toolkit 3.4

4.2 Project Structure

The implementation is separated into 2 distinguished eclipse projects as depicted in figure 4.1.

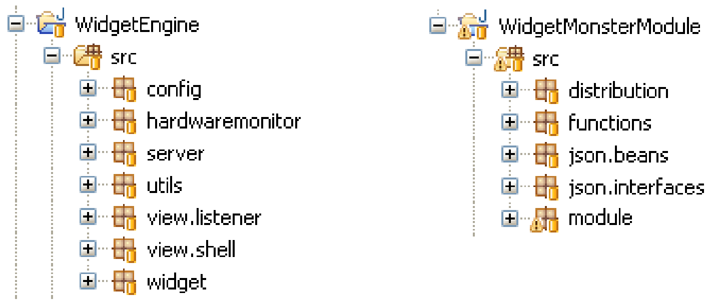

Figure 4.1: Project Structure

The following listing briefly describes the single packages of both projects in alphabetical order to give an overview of the implementation:

config
Lorem Ipsum...

server
Lorem Ipsum...

utils
Lorem Ipsum...

4.3 Important Implementation Aspects

Do not explain every class in detail. Give a short introduction about the modules or the eclipse projects. If you want to explain relevant code snippets use the 'lstlisting' tag of LaTeX. Put only short snippets into your thesis. Long listing should be part of the annex.

Listing 4.1: JSON String Code Snippet

```
{
        id: 1,
        method: "myInstance.getGroup",
        params: ["Teammates", 2, true]
}

{
        id: 2,
        result: [
                  "groupDesc":"These are my teammates",
                  {
                        "javaClass":"src.package.MemberClass",
                        "memberName": "Bob",
                  }
                ]
}
```

You can also compare different approaches. Example: Since the implementation based on X failed I choosed to implement the same aspect based on Y. The new approach resulted in a much faster ...

4.4 Graphical User Interface

Lorem Ipsum...

4.5 Documentation

Lorem Ipsum...

5 Validation

Evaluation add results chapter before this The reduction of frame rate due to the additionally required synchronization procedure was only 3 during the experiments In this chapter the implementation of Component X is evaluated. An example instance was created for every service. The following chapter validates the component implemented in the previous chapter against the requirements.

Put some screenshots in this section! Map the requirements with your proposed solution. Compare it with related work. Why is your solution better than a concurrent approach from another organization?

5.1 Test Environment

Fraunhofer Institute FOKUS' Open IMS Playground was used as a test environment for the telecommunication services. The IMS Playground ...

5.2 Scalability

Lorem Ipsum

5.3 Usability

Lorem Ipsum

5.4 Performance Measurements

Lorem Ipsum

6 Conclusion

Outlook

The final chapter summarizes the thesis. The first subsection outlines the main ideas behind Component X and recapitulates the work steps. Issues that remained unsolved are then described. Finally the potential of the proposed solution and future work is surveyed in an outlook.

6.1 Summary

Explain what you did during the last 6 month on 1 or 2 pages!

The work done can be summarized into the following work steps

- Analysis of available technologies
- Selection of 3 relevant services for implementation
- Design and implementation of X on Windows
- Design and implementation of X on mobile devices
- Documentation based on X
- Evaluation of the proposed solution

6.2 Dissemination

Who uses your component or who will use it? Industry projects, EU projects, open source...? Is it integrated into a larger environment? Did you publish any papers?

6.3 Problems Encountered

Summarize the main problems. How did you solve them? Why didn't you solve them?

6.4 Outlook

Future work will enhance Component X with new services and features that can be used ...

List of Acronyms

Update.

3GPP	3rd Generation Partnership Project
AJAX	Asynchronous JavaScript and XML
AP	Access Point
API	Application Programming Interface
AS	Application Server
CSCF	Call Session Control Function
CSS	Cascading Stylesheets
DHTML	Dynamic HTML
DOM	Document Object Model
EMVA	European Machine Vision Association
FOKUS	Fraunhofer Institut fuer offene Kommunikationssysteme
GenICam	Generic Interface for Cameras
GPS	Global Positioning System
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
I-CSCF	Interrogating-Call Session Control Function
IETF	Internet Engineering Task Force
IM	Instant Messaging
IMS	IP Multimedia Subsystem
IP	Internet Protocol
J2ME	Java Micro Edition
JDK	Java Developer Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JSR	Java Specification Request
JVM	Java Virtual Machine
NGN	Next Generation Network
OMA	Open Mobile Alliance
P-CSCF	Proxy-Call Session Control Function
PDA	Personal Digital Assistant
PEEM	Policy Evaluation, Enforcement and Management
PTP	Precision Time Protocol
QoS	Quality of Service
S-CSCF	Serving-Call Session Control Function
SDK	Software Developer Kit
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMS	Short Message Service
SMSC	Short Message Service Center

SOAP	Simple Object Access Protocol
SWF	Shockwave Flash
SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
Telco API	Telecommunication API
TLS	Transport Layer Security
UMTS	Universal Mobile Telecommunication System
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XCAP	XML Configuration Access Protocol
XDMS	XML Document Management Server
XML	Extensible Markup Language

Bibliography

[1] IDS Imaging Development Systems. Hardware trigger - ids cameras manual. `https://www.1stvision.com/cameras/IDS/IDS-manuals/en/operate-hardware-trigger.html`. Accessed: Month Day, Year.

[2] Marcin Bajer. Synchronization of current and voltage measurements in a modular motor diagnostic system. *Pomiary Automatyka Kontrola*, page 1080, 10 2013.

[3] Fortune Business Insights. Computer vision market analysis. Online. URL `https://www.fortunebusinessinsights.com/computer-vision-market-108827`. Accessed: January 21, 2025.

[4] European Machine Vision Association. Genicam standard. Online, . URL `https://www.emva.org/standards-technology/genicam/`. Accessed: January 21, 2025.

[5] European Machine Vision Association. Emva website. Online, . URL `https://www.emva.org/`. Accessed: January 21, 2025.

[6] Johannes Vater, Matthias Pollach, Claus Lenz, Daniel Winkle, and Alois Knoll. Quality control and fault classification of laser welded hairpins in electrical motors. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1377–1381, 2021. doi: 10.23919/Eusipco47968.2020.9287701.

[7] Vasanth Subramanyam, Jayendra Kumar, and Shiva Nand Singh. Temporal synchronization framework of machine-vision cameras for high-speed steel surface inspection systems. *Journal of Real-Time Image Processing*, 19(2):445–461, 2022. ISSN 1861-8219. doi: 10.1007/s11554-022-01198-z. URL `https://doi.org/10.1007/s11554-022-01198-z`.

[8] Prarthana Shrestha, Hans Weda, Mauro Barbieri, and Dragan Sekulovski. Synchronization of multiple video recordings based on still camera flashes. In *Proceedings of the 14th ACM International Conference on Multimedia*, MM '06, page 137–140, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595934472. doi: 10.1145/1180639.1180679. URL `https://doi.org/10.1145/1180639.1180679`.

[9] Mario Guggenberger, Mathias Lux, and Laszlo Böszörmenyi. Audioalign - synchronization of a/v-streams based on audio data. In *2012 IEEE International Symposium on Multimedia*, pages 382–383, 2012. doi: 10.1109/ISM.2012.79.

[10] Anna Casanovas and Andrea Cavallaro. Audio-visual events for multi-camera synchronization. *Multimedia Tools and Applications*, 74, 02 2014. doi: 10.1007/s11042-014-1872-y.

[11] Yunhyeok Han, Stefania Lo Feudo, Gwendal Cumunel, and Franck Renaud. Sub-frame timestamping of a camera network using a coded light signal. *Measurement*, 236:115046, 2024. ISSN 0263-2241. doi: https://doi.org/10.1016/j.measurement.2024.115046. URL `https://www.sciencedirect.com/science/article/pii/S026322412400931X`.

[12] Igor Pereira, Luiz F. Silveira, and Luiz Gonçalves. Video synchronization with bit-rate signals and correntropy function. *Sensors*, 17(9), 2017. ISSN 1424-8220. doi: 10.3390/s17092021. URL `https://www.mdpi.com/1424-8220/17/9/2021`.

[13] Nicolas Boizard, Kevin El Haddad, Thierry Ravet, François Cresson, and Thierry Dutoit. Deep learning-based stereo camera multi-video synchronization, 2023. URL `https://arxiv.org/abs/2303.12916`.

[14] Felix Wermke, Thorben Wübbenhorst, and Beate Meffert. Optical synchronization of multiple time-of-flight cameras implementing tdma. In *2020 IEEE SENSORS*, pages 1–4, 2020. doi: 10.1109/SENSORS47125.2020.9278667.

[15] Basler AG. Triggered image acquisition. `https://docs.baslerweb.com/triggered-image-acquisition`. Accessed on Month Day, Year.

[16] Hieu Nguyen, Dung Nguyen, Zhaoyang Wang, Hien Kieu, and Minh Le. Real-time, high-accuracy 3d imaging and shape measurement. *Appl. Opt.*, 54(1):A9–A17, Jan 2015. doi: 10.1364/AO.54.0000A9. URL `https://opg.optica.org/ao/abstract.cfm?URI=ao-54-1-A9`.

[17] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991. doi: 10.1109/26.103043.

[18] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pages 1–499, 2020. doi: 10.1109/IEEESTD.2020.9120376.

[19] Ricardo Sousa, Martin Wäny, and Pedro Santos. Multi-camera synchronization core implemented on usb3 based fpga platform. *Proceedings of SPIE - The International Society for Optical Engineering*, 9403, 02 2015. doi: 10.1117/12.2082928.

[20] Alan B. Johnston. *SIP, understanding the Session Initiation Protocol, Second Edition.* Artech House Publishers, 2003. ISBN: 1580536557.

Annex

```
<?xml version="1.0" encoding="UTF-8"?>
<widget>
         <debug>off</debug>
         <window name="myWindow" title="Hello Widget" visible="true">
                 <height>120</height>
                 <width>320</width>
                 <image src="Resources/orangebg.png">
                        <name>orangebg</name>
                        <hOffset>0</hOffset>
                        <vOffset>0</vOffset>
                </image>
                 <text>
                         <name>myText</name>
                         <data>Hello Widget</data>
                         <color>#000000</color>
                         <size>20</size>
                         <vOffset>50</vOffset>
                         <hOffset>120</hOffset>
                 </text>
        </window>
</widget>
```

Listing 1: Sourcecode Listing

```
INVITE sip:bob@network.org SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
Max-Forwards: 70
To: Bob <sip:bob@network.org>
From: Alice <sip:alice@ims-network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Subject: How are you?
Contact: <sip:xyz@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=alice 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
t=0 0
c=IN IP4 100.101.102.103
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.network.org:5060;branch=z9hG4bK83842.1
;received=100.101.102.105
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Bob <sip:bob@network.org>;tag=314159
From: Alice <sip:alice@network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Contact: <sip:foo@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=bob 2890844526 2890844526 IN IP4 200.201.202.203
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Listing 2: SIP request and response packet[20]