Event camera for acoustic levitation

Real-time tracking and motion control

M. El Shenawy

Report no : 2024.041
Coach : Vincent Bos
Professor : Dr. G.J. Verbiest
Specialisation : Mechatronic System Design
Type of report : Master Thesis
Date : 10-07-2024

Master of Science Thesis



TU Delft Delft University of Technology

High-Tech Engineering

Event camera for acoustic levitation

Real-time tracking and motion control

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

M. El Shenawy

June 25, 2024

Faculty of Mechanical, Maritime and Materials Engineering (ME)
Delft University of Technology



Copyright © Precision and microsystems engineering (PME)
All rights reserved.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Dr. G.J. Verbiest, for his invaluable guidance, support, and expertise throughout the entire process of my Master's Thesis. His insights and feedback have played a crucial role in shaping this work.

I also want to thank my daily supervisor, Vincent Bos, for his continuous assistance, valuable feedback, and for helping me when I encountered problems. The knowledge I gained from his guidance has not only been beneficial for this project but will also serve as a strong foundation for my future engineering career.

Furthermore, I want to thank my family and friends for their patience, kindness, and encouragement. Without them, it would have been much more difficult. Moreover, I want to thank the other professors, PhD, and Master's students from the KET4 group, as they welcomed me and shared their valuable perspectives. Lastly, I also want to thank Prof. Dr. P.G. Steeneken for being willing to become a member of the thesis committee.

During this thesis, I had the opportunity to experience cutting-edge technology, learn many new things, and face numerous challenges, which helped me grow significantly. I enjoyed creating something that will be useful in the future and am grateful to have contributed to this work.

Delft, University of Technology — M. El Shenawy
June 25, 2024

Abstract

The trend towards miniaturization drives the demand for efficient, cost-effective, and compact systems. Miniaturization creates challenges in handling delicate, small components, rendering traditional methods inadequate, necessitating an alternative. Acoustic levitation, which uses ultrasound waves to suspend objects without physical contact, offers a promising solution for containerless transportation. This technology can accommodate various materials regardless of their electromagnetic or optical properties and eliminates particle generation due to friction. Applications span across chemistry, biology, and high-tech industries, enabling manipulation and assembly of small components. However, acoustic levitation is highly sensitive to environmental factors such as temperature and air currents, which can compromise stability. Most studies have primarily focused on understanding the dynamics of acoustic levitation and developing models to describe the underlying physical phenomena. This study shifts towards actively controlling them by adding artificial damping.

To achieve this, this thesis introduces the use of event cameras, which capture high-speed movements without redundant frame data, making them highly efficient. The goal is to create a control feedback loop to correct for disturbances and improve the step response of acoustically levitated objects.

The methodology involved integrating an event camera into the acoustic levitation setup and implementing a tracking algorithm to track the position of a levitated object. System identification was performed, revealing higher-order harmonics and allowing the development of a second-order model of the levitated object. This model was used to design a feedback controller integrated with the event camera.

Experimental results demonstrated a substantial improvement in system performance. The bandwidth of the system improved to 40.8 Hz, the settling time decreased from 10 seconds to 0.8 seconds, an improvement by a factor of 12.5. Additionally, the steady-state error was significantly reduced, and the introduction of artificial damping successfully eliminated the first resonance peak, though a second higher-order resonance peak remains. This work highlights the potential of using event cameras in control loops for acoustic levitation, representing a significant step towards more precise control of levitated objects.

Table of Contents

Acknowledgements **i**

1 Introduction **1**

1-1 Research Goal and Objectives 2

1-2 Thesis outline 3

2 Background Theory **4**

2-1 Acoustic levitation 4

2-1-1 Acoustic Levitation Configurations 5

2-1-2 Gor'kov's Potential Theory 5

2-1-3 Vibrations and Stiffness 6

2-1-4 Dynamics of Acoustic Levitation 7

2-2 System identification 8

2-3 Control 10

2-3-1 Motivation behind control 10

2-3-2 Stability Margins 11

2-3-3 Disturbances 14

2-3-4 PID control 15

2-3-5 Loop-Shaping 17

3 Event-Based Camera **19**

3-1 Event camera working principles 19

3-2 Event camera types 21

3-3 Sensor Characteristics 21

3-3-1 Contrast sensitivity 22

3-3-2 Background rate 23

3-3-3 Pixel Latency and Jitter 24

3-4 Event Data Processing . . . 24
3-4-1 Total latency . . . 26
3-5 Event Visualization and Representation . . . 26
3-5-1 Image and Video rendering . . . 27
3-5-2 XYT view . . . 29
3-6 Event Signal Processing . . . 29
3-6-1 Hardware Filters . . . 29
3-7 Region of interest . . . 30
3-8 Optimal Lighting Conditions for Event Cameras . . . 31
3-9 Biases . . . 32
3-9-1 Contrast Sensitivity Bias . . . 32
3-9-2 Low-pass and High-pass filters . . . 33
3-9-3 The Refractory Period bias . . . 36
3-10 Bias selection . . . 37
3-11 Advantages and disadvantages of Event-Based Cameras . . . 39

4 PHROPHESEE SDK 40
4-1 SDK . . . 40
4-2 Pipelines . . . 41
4-3 Tracking Algortihm . . . 42
4-3-1 Spatter Tracking . . . 42
4-3-2 Algorithm settings . . . 44

5 Experimental Setup 48
5-1 Schematic of setup . . . 48
5-2 Processing real-time data . . . 50
5-3 Optics and Sensor setup . . . 51

6 Methodology 52
6-1 Offline processing . . . 53
6-1-1 Light settings . . . 53
6-1-2 Bias settings . . . 54
6-1-3 Tracking settings . . . 54
6-1-4 Tracking accuracy . . . 59
6-1-5 Tracking speed . . . 59
6-2 Real-time loop . . . 59
6-2-1 Reference signal . . . 60
6-2-2 Delay comparison . . . 60
6-3 System identification . . . 60
6-4 Control . . . 63
6-4-1 Loop Shaping . . . 64

7 Results and Discussion **68**
7-1 Results . . . 68
7-1-1 Event camera . . . 68
7-1-2 Spectogram . . . 70
7-1-3 Power Spectrum Density . . . 71
7-1-4 Bode Diagrams . . . 72
7-1-5 Step Response . . . 74
7-2 Discussion . . . 75
7-2-1 Event camera . . . 75
7-2-2 Spectogram . . . 77
7-2-3 Power Spectrum Density . . . 77
7-2-4 Bode Diagrams . . . 78
7-2-5 Step Response . . . 78
7-3 Sub-questions . . . 81

8 Conclusions and Recommendation **82**
8-1 Future Work . . . 83
8-1-1 Event Camera and Setup Improvements . . . 83
8-1-2 Controller improvements . . . 84

9 All PROPHESEE Algorithms **85**
9-1 Modules . . . 85
9-2 Processors . . . 86
9-3 Filters . . . 88
9-4 Metavision Studio . . . 89
9-4-1 Statistics . . . 89
9-4-2 Display . . . 89
9-4-3 Monitoring . . . 90
9-4-4 Biases . . . 90
9-4-5 Region of Interest . . . 91
9-4-6 Trigger In and Out . . . 91
9-4-7 Anti Flicker . . . 91
9-4-8 Event Trail Filter . . . 91
9-4-9 Event Rate Control . . . 91

10 Supplementary Algorithms **92**
10-1 3D object Edge Tracking . . . 92
10-2 Vibration Estimation Algorithm . . . 94

11 Network and Simulink implementation **95**
11-1 Boost.Aiso . . . 95
11-2 Simulink . . . 98

12 Project Software Setup **101**
12-1 Locating Code 101
12-2 Installing Integrated Development Environment (IDE) 102
12-3 Installing C++ compiler 103
12-4 Installing Cmake 106
12-5 compiling code 107
12-6 Bios settings 107

13 Methodology Settings **108**
13-1 Bias settings 108
13-2 Multisine settings 108
13-3 Controller settings 109

List of Figures

2-1 Pressure and force in the z-direction 7

2-2 A spring-mass-damper system consisting of a mass m , a damping coefficient c, and a spring stiffness k. 7

2-3 Figure depicting the oscillation amplitude for different excitation frequencies showing bifurcations [11]. 8

2-4 Bodeplot with the phase and gain margins [31, p. 264]. 12

2-5 Feedback loop with input, output and sensor noise. The movement of the object is coupled. When actuated in the z direction, the object also moves in the x direction, thereby affecting the z direction as well. The actual z position is a combination of the actuation signal and a movement due to the coupled dynamics. The sensor measures this combined movement. 13

2-6 Waterbed effect [31, p. 290]. 13

2-7 Analogy of the waterbed effect [32]. 13

2-8 Waterbed effect. 13

2-9 Nyquist plot of the sensitivity and complementary sensitivity [31, p. 261]. 14

2-10 Components of a PID controller 15

2-11 Bode plot applying the rules of thumb [31, p. 286]. 16

2-12 Loop-shaping [31, p. 313]. 18

3-1 Visualization of event detection. 19

3-2 Sensor pixel Readout [33]. 20

3-3 Schematic representation of how a pixel registers events [34]. 21

3-4 This table contains various event-based sensors from different manufacturers, as documented in [15].The event camera used is the GEN 4 CD from Prophesee. . . 22

3-5 Response-probability [35]. 23

3-6 Background rate [35]. 23

3-7 Pixel latency and jitter [36]. 24

3-8 EVT Decoding. 25
3-9 Illustration of the total Latency [35]. 26
3-10 Illustration of the components of event visualization and representation. 26
3-11 Illustration of accumulated events over time. 27
3-12 This figure provides examples of different accumulation times for various scenarios [39]. 28
3-13 XYT view of a dot on a rotating disk [39]. 29
3-14 Hardware filter options. 30
3-15 Multiple examples with ROI [40]. The single ROI mode seems useful for tracking the levitated sphere due to its low amplitude oscillations. 31
3-16 Subsampling where certain rows are used to register/count falling objects [41]. . 31
3-17 Comparison of Contrast Sensitivity Thresholds [41]. 32
3-18 This illustration showcases several checkerboards with varying levels of contrast, ranging from high contrast to no contrast at all. A more sensitive pixel makes distinguishing intensity variations in low-contrast scenes easier and requires a smaller contrast sensitivity threshold compared to high-contrast scenes. 33
3-19 Comparison of Low-pass Filter Effects [41]. 33
3-20 This figure shows the behavior the number of events generated for different `Bias_fo` and `Bias_Hpf` values for both the ON and Off pixels [42]. 34
3-21 Comparison of the delay between laser trigger and pixel registration observed in a blinking LED, as documented by Prophesee [41]. The delay decreases as the low-pass filter is increased. 34
3-22 Example of falling particles [41]. 35
3-23 Comparison of High-pass Filter Effects [41]. 35
3-24 Pedestrians walking as an example of low velocity [41]. 35
3-25 Comparison of Pixel Refractory Period effects [41]. 36
3-26 Visual representation of increasing the Refractory Period Bias [41]. 36
3-27 Example of a high-speed laser where the path is visible [41]. 36
3-28 A scene to showcase the various types of considerations to make when applying bias settings [41]. 37
3-29 Flowchart for selecting Bias settings for different situations. 38

4-1 Prophesee SDK modules [45]. 41
4-2 Comparison of simple and complex pipelines [46]. 41
4-3 Spatter tracking highlighting multiple moving objects [48]. 42
4-4 Cluster making. 43
4-5 Cluster matching. 44

5-1 Schematic of the setup currently employed. 49
5-2 This figure depicts how the data from the camera is processed and compiled into code. 50
5-3 Photo of the optics used in the setup. 51

6-1 Methodology steps for the event camera and tracking: On the left, offline processing; on the right, the real-time loop. . . . 52
6-2 Comparison of an object illuminated with a flash at 1000 Hz compared to using constant illumination while levitating in a node. . . . 53
6-3 The effect of different Accumulation times on a fast moving object. . . . 55
6-4 Comparison of different Grid-sizes. . . . 56
6-5 Sinusoidal movement pattern of object. . . . 56
6-6 Examples where several smaller clusters are being tracked as objects, while the blue tracker is only the tracker of interest. . . . 57
6-7 This set of images illustrates a sphere moving in a sinusoidal pattern, where occasionally the tracker detects two objects when the sphere reaches zero velocity and changes direction. . . . 58
6-8 Examples of a Spectogram. . . . 62
6-9 Examples of a PSD. . . . 62
6-10 Example of an idealized second-order system. . . . 63
6-11 Plant of an ideal system compared to a system with a 3.5 millisecond delay. . . . 64
6-12 Plant vs Open-loop. . . . 65
6-13 Plant vs closed-loop. . . . 65
6-14 Sensitivity and complimentary sensivity. . . . 66
6-15 Nyquist diagram zoomed-out. . . . 66
6-16 Nyquist diragram zoomed-in around -1 point . . . 66
6-17 Nyquist diagrams. . . . 66
6-18 Simulated step response of system with and without control. . . . 67

7-2 The mean values of the object's X and Z coordinates are plotted against the amplitude for three grid sizes. . . . 69
7-3 Spectrograms in X and Z directions . . . 70
7-4 Power Spectrum Diagrams in X and Z directions under different conditions . . . 71
7-5 A collection of PSD diagrams was made for the object while it was levitating in a node without any input displacement. . . . 72
7-6 Bode diagram in Z and X direction. . . . 73
7-7 Step response of the system with and without control. . . . 74
7-8 Step repsonse without control zoomed in. . . . 78
7-9 Measured step response zoom-in. . . . 79
7-10 Segment of the measured steep response showing sudden noisy peaks. . . . 79
7-11 PSD of a 30 second segment. . . . 80
7-12 The step response was measured using a multi-sine ranging from 1 to 60Hz, with each frequency maintaining the same amplitude. A minor secondary resonance peak can be observed around the 38Hz mark. . . . 80

9-1 A screenshot of Metavision Studio with the available options on the right side. . . . 89
9-2 Different color themes. . . . 90
9-3 Example of a Bias file. . . . 90

10-1 On the left: Predetermined shape (red) searching for the object to be tracked. On the right: The edges of the tracked object highlighted in green. 92
10-2 Support points along edges [49]. 93
10-3 Example of vibration estimation on a motor, showing the frequency of each pixel [65]. 94

11-1 Network communication visualized [67]. 95
11-2 C++ libraries. 96
11-3 UDP socket creation. 96
11-4 Modified `Pipeline::tracker_callback` function. 97
11-5 Connecting to IP of other computer. 97
11-6 Simulink setup for network connection. 98
11-7 UDP receive and parameter selector blocks. 98
11-8 Code used in the Simulink block. 99
11-9 On the left is the standard coordinate system, and on the right is the converted coordinate system. 99
11-10Modified values for coordinate conversion. 100

12-1 SDK folder file path. 101
12-2 Spatter tracking file path 102
12-3 Visual studio and studio code installation button screenshot from the website of microsoft [69]. 102
12-4 Screenshot from installation menu of Visual Studio. 102
12-5 Visual studio extensions used. 103
12-6 UCRT64 terminal environment. 103
12-7 UCRT64 terminal command 1. 104
12-8 UCRT64 terminal output of command 1. 104
12-9 Terminal command 1.5 104
12-10UCRT64 terminal command 2. 105
12-11Command to confirm gcc version. 105
12-12UCRT64 terminal output from command 2 106
12-13Msys64 file location. 106
12-14Cmake file location. 106
12-15CMD commands for compiling exe 107
12-16CMD command to get help menu. 107

List of Tables

2-1 Sensitivity and Complementary Sensitivity Functions [31, p. 256]. 12
2-2 Robustness Criteria. 14
2-3 Rules of thumb. 16
2-4 PID Control Components. 17
2-5 Pros and Cons of PID Control Actions. 17

3-1 EVT Compression Ratios [37]. 24
3-2 EVT 3.0 encoding format. 25
3-3 Advantages and Disadvantages of Event-Based Cameras 39

4-1 Processing parameters for the tracking algorithm. These Options are accessible via `metavision_spatter_tracking.exe -h` command in the terminal. 45
4-2 Region of Interest options. 47

6-1 The tracker parameters are defined where T is the period of the sinusoidal input signal in milliseconds. 57
6-2 Final tracking values used for the multi-sine for the Z and X directions. 61
6-3 Multi-sine signal values used for the X and Z directions. 61
6-4 Rules of Thumb: The values of f_d, f_t, and f_i are related to the value of the cut-off frequency chosen. 65

7-1 Delay of individual components of the complete setup [59]. 68

9-1 Comparison between METAVISION SDK and METAVISION SDK PRO [63]. . . . 85
9-2 Advanced modules [63]. 85
9-3 Open modules [63]. 86
9-4 Algorithms processing events as input and outputting display-related data (e.g. frames): 86

9-5 Algorithms processing events as input and outputting some data from the event (counting, tracking etc.) [64]. 87
9-6 Algorithms taking events as input and outputting the same number of events but with a transformation applied [64]. 87
9-7 Filters are algorithms taking events as input and outputting only the events that satisfy certain conditions [64]. 88
9-8 Bias values 90
9-9 Anti Flicker values 91
9-10 Event Trail Filter values 91
9-11 Event Rate Control values: It is not entirely clear whether the event rate is capped randomly or if only the newest events are filtered out. 91

13-1 Final tracking values used for the multi-sine for the Z and X directions. 108
13-2 Multi-sine signal values used for the X and Z directions. 109
13-3 PID Controller Parameters 109

Chapter 1

Introduction

In recent years, there has been an increasing trend towards miniaturization, driven by the need for more efficient, cost-effective, and compact systems. At the forefront of this research is the Department of Precision and Microsystems Engineering (PME) [1], which merges mechanical engineering with micro and nanoscience to advance precision systems and devices. Miniaturization enables the development of sophisticated micro and nanoscale systems that can perform complex tasks with high precision and efficiency.

A prominent example of miniaturization is in the field of micro-assembly, where small components are assembled to create complex systems. As objects become smaller, they become more difficult to physically handle, pick up, and move. The physical manipulation of micro and nano-sized objects can lead to issues such as damage or breakage due to their delicate nature. Traditional methods of handling and assembly, such as pick-and-place machines [2], are becoming increasingly inadequate for these scales, necessitating the development of innovative techniques to address these challenges. Additionally, the small physical scale and limited handling speed make it difficult to scale up the production process [3, 4].

A promising solution to these challenges is acoustic levitation [5], where ultrasound waves are used to suspend small objects in the air without physical contact. By creating a standing wave with high intensity, objects can be trapped at the nodes of the wave, resulting in a situation where the acoustic pressure counteracts the force of gravity. Acoustic levitation offers distinct advantages compared to other levitation techniques, such as electromagnetic, optical, and electrostatic levitation. Primarily, it does not rely on the electromagnetic or optical properties of the material, enabling the manipulation of a broader range of materials. Furthermore, acoustic levitation is a container-free method, eliminating particle generation due to friction.

This technique has various applications in fields such as chemistry [6], where it facilitates the manipulation of small droplets [7] without contamination, and biology [8], where it enables the study of small organisms in a contact-free environment. The high-tech industry also benefits, as it allows the precise transportation and assembly of tiny components, which will be the main focus of this thesis. Additionally, acoustic levitation can be used to create holographic

image [9] and in-space processing [10]. This broadens the impact of this technology, making it an innovative tool across multiple industries.

In the past decade, there has been a significant increase in research related to acoustic levitation. The majority of these studies have primarily focused their work on understanding the dynamics of acoustic levitation and developing models to describe the underlying physical phenomena. These models are often nonlinear [11, 12, 13] and lack analytical solutions, making it challenging to study the global behavior and presenting obstacles when attempting to levitate objects with different geometries [14]. Much of the existing research has revolved around the observed oscillatory behavior in acoustic levitation, resulting in unwanted vibrations. While many aspects remain unexplored, there is a general consensus that the system tends to exhibit similar behavior to that of a linear mass-spring-damper system with small displacements [12].

1-1 Research Goal and Objectives

The objective of this research is to utilize acoustic levitation for object transportation. While it is currently possible to move levitated objects from one location to another, the goal is to enhance the system's step response. By increasing the velocity of the object and decreasing the settling time, we can improve the system's performance. Increasing the system bandwidth allows it to respond faster to input changes. However, the maximum frequency is limited by the resonance frequency peaks. The proposed solution is adding artificial damping. An increase in damping reduces unwanted vibrations, leading to more stable levitation. Ultimately, this would improve both precision and accuracy.

Unlike previous research, which primarily focuses on understanding these dynamics, this study shifts towards actively controlling them. To realize this goal, an accurate model describing the system needs to be obtained. The proposed research aims to bridge this gap by leveraging system identification to develop a more robust model. For this purpose, real-time tracking of the object with minimal delay is crucial. To address this challenge, this research pioneers the use of event-based cameras for control in acoustic levitation, marking a departure from the conventional practice of using lasers or high-speed cameras. Event cameras capture high-speed movement without producing redundant data [15], leading to low data rates. This allows for near real-time tracking of objects, unlocking the potential to revolutionize acoustic levitation systems.

Based on this research gap, the following research question is set up:

"To what extent can control in conjunction with event-based cameras be employed to improve the step response of an acoustically levitated object?"

To address the main research question, the following sub-questions are formulated:

- Are there hardware limitations or constraints that need to be addressed in the integration of control and event-based cameras?
- How effectively and rapidly can event-based cameras measure displacement and oscillations?
- What control strategies are best suited to address unwanted vibrations in acoustic levitation?
- To what extent can event-based cameras enhance the accuracy of position measurements in acoustic levitation systems?

1-2 Thesis outline

This Master thesis is structured as follows: Chapter 2 provides background information from literature, including fundamental principles of acoustic levitation, the dynamics involved, and various configurations. It also covers essential system identification methodologies necessary for modeling acoustic levitation systems and explores various control strategies. Chapter 3 introduces event-based camera technology, explaining their distinctive working principles and event processing. Chapter 4 presents the software development kit (SDK) supplied with the event camera, with a focus on the tracking algorithm. Chapter 5 details the experimental setup, including the tools, and procedures utilized to conduct the experiments. In Chapter 6, the research methodology is described, elaborating on the methods and techniques that will be employed to achieve the research objectives. Chapter 7 presents the results obtained from the experiments and provides a comprehensive discussion on their implications. Lastly, Chapter 8 concludes this study by summarizing key findings and providing suggestions for future work and recommendations.

Chapter 2

Background Theory

This chapter provides essential background information from the literature review, which is crucial for understanding the research goal and objectives. The chapter is organized into three main sections: the first focuses on the physics and dynamics of acoustic levitation; the second explains system identification; and the third discusses feedback control. Readers who are well versed in these topics may directly proceed to Chapter 3, where the event-camera will be discussed.

2-1 Acoustic levitation

The concept of using sound waves to manipulate matter traces its origin back to the 19th century. It began with pioneering contributions from scientists such as Kundt in 1866, who conducted experiments involving sound waves in a resonant tube [16]. Following Kundt, in the late 19th century Lord Rayleigh [17] explored the scattering of sound on small objects specifically small spheres. His insights contributed to our understanding on how sound waves could be manipulated to affect the motion of objects, setting the basis for acoustic theory. In 1934, Louis V. King [18] introduced the acoustic radiation force. King's pioneering work revolutionized our understanding of how sound waves could exert forces and torques on objects and particles. Approaching the mid-20th century, Lev P. Gor'kov [19] developed a method to calculate the acoustic radiation force through the Gor'kov potential gradient. His theory held up well and is relevant in today's research. The concept of acoustic radiation force derived from the Gor'kov potential forms the foundation for the creation of what we refer to as 'acoustic traps'. These traps represent low-pressure potentials in the acoustic field and play an important role in the suspension of objects in the air. Acoustic levitation, often referred to as 'Aclev', harnesses the mechanical properties of soundwaves to levitate objects. While acoustic levitation systems come in various configurations, the most prevalent setup employs two opposing piezoelectric ultrasonic arrays [5].

2-1-1 Acoustic Levitation Configurations

Acoustic levitators use phased array transducers that produce high-frequency sound waves at resonant frequencies for efficiency purposes [20]. The complex acoustic pressure P at a point $p(m)$ that is emitted by a single-frequency piston source can be formulated as follows [21, 22]:

$$p(m) = P_w V_a \frac{D_f(\theta)}{2d} e^{j(\omega t + \phi - kd)} \tag{2-1}$$

$$D_f(\theta) = 2J_1 \frac{ka\sin(\theta)}{ka\sin(\theta)} \tag{2-2}$$

P_w represents a constant amplitude defining the output of the transducer. V_a stands for the peak-to-peak value of the excitation voltage, while d indicates the Euclidean distance between the piston source and point x in space. ω represents the angular frequency of the piston and ϕ denotes the initial phase of the piston.

The function D_f as described in Equation 2-2 corresponds to the Fairfield directivity function and is dependent on the angle between the piston normal and x. Additionally, J_1 stands for the first-order Bessel function, a represents the piston radius, and θ indicates the angle between the piston normal and r, which serves as the position vector for the point m in space relative to the piston.

By summing up the acoustic pressure of each transducer, the total acoustic pressure can be obtained. The acoustic radiation force acting on a body can be controlled by manipulating the phase of these transducers. For additional details, readers are encouraged to consult the following papers from Remco van Hoorn [23] and Ghenna et al. [24].

Standing wave acoustic uses two opposing ultrasonic phased arrays with a phase difference of half the wavelength, creating nodes and anti-nodes in the acoustic field. Objects are trapped at the nodes, slightly below due to gravity. The position of nodes can be manipulated by changing the phase, allowing for object translation.

2-1-2 Gor'kov's Potential Theory

Gor'kov's theory [19] calculates the acoustic radiation force on a small sphere within a sound field provided that the sphere's wavelength is much smaller than the wavelength of the sound field and does not exceed 0.6 λ :

$$F = -\nabla U \tag{2-3}$$

Where U is Gor'kov's potential energy:

$$U = 2\pi R^3 \left(\frac{\langle p^2 \rangle}{3\rho c^2} - \frac{\rho \langle u^2 \rangle}{2} \right) \tag{2-4}$$

Where R $[m]$ represents the sphere's radius, p $[Pa]$ is the sound pressure, ρ $[\frac{kg}{m^3}]$ denotes air density, c $[\frac{m}{s}]$ represents sound velocity in the air, and u $[\frac{m}{s}]$ signifies airflow velocity. The brackets (<>) typify the mean-averaged acoustic pressure of the pressure and velocity respectively.

The acoustic radiation force in the vertical (z) direction, Equation 2-5 [25]

$$F_{rad} = \frac{5\pi R^3 k p^2}{6\rho c^2} \sin(2kz) \tag{2-5}$$

Where k is the wave number given by $k = \frac{\omega}{c}$, and ω represents the angular frequency.

The stiffness in the z-direction can be calculated using Equation 2-6 [25]. Additionally, the oscillation frequency f_a can be calculated using Equation 2-7 where k_d is the linear stiffness $\frac{N}{m^2}$ along direction d $[m]$ and m $[kg]$ the mass of the levitated sphere.

$$k_z = \frac{5\pi R^3 k p^2}{3\rho c^2} \cos(2kz) \tag{2-6}$$

$$f_a = \frac{1}{2\pi}\sqrt{\frac{k_d}{m}} \qquad d \in z, x \tag{2-7}$$

The drag force acting on a levitated sphere can be calculated using Equation 2-8 [26], with δ denoting the penetrating depth in Equation 2-9

$$F_d = -6\pi\mu R\left(1 + \frac{R}{\delta}\right)\dot{z} = -c_d\dot{z} \tag{2-8}$$

$$\delta = \sqrt{\frac{2\mu}{\rho\omega}} \tag{2-9}$$

It is worth noting that in other literature, researchers such as Marzo et al. [22] have demonstrated that spheres with a size equal to or larger than λ can levitate in the mie regime using virtual vortices and boundary holograms. Furthermore, a lot of research has been conducted on the dynamic behavior of levitating droplets made from different materials and densities. However, this Thesis will only consider small spheres to limit the scope of the study.

2-1-3 Vibrations and Stiffness

The main challenge facing acoustic levitation, are the vibrations which stem from various sources. The presence of a levitated object in a levitator affects its resonance frequency was shown by Perez et al. [25]. Factors such as the size of the object, its position within the levitator, and the ambient temperature of the surrounding air all contribute to these resonance frequency variations. Furthermore, the damping coefficient is dependent on the driving frequency. Another possible cause for instability is acoustic streaming [27].

The quality factor (Q), which measures the resonator's ability to sustain vibration, is inversely proportional to the damping ratio (ζ) Equation 2-10. Soichiro et al. found that for a small sphere, Q ranges from 100 to 1000. Higher damping leads to a lower Q factor, and vice versa. High amplitudes lower the stiffness and eventually eject the levitated object if too large.

$$Q = \frac{1}{\zeta} \tag{2-10}$$

The current system's Q factor is between 300 and 400, attributed to the low damping provided by air. To reduce oscillations, damping needs to be increased. This can be done with feedback control introducing artificial (active) damping.

2-1-4 Dynamics of Acoustic Levitation

Previous work conducted by Pérez et al. [12] demonstrated that a sphere levitating around an acoustic trap can also be described as a spring-mass-damper model when the displacement is about $\frac{1}{16}\ \lambda$. This is visualized in Figure 2-1.

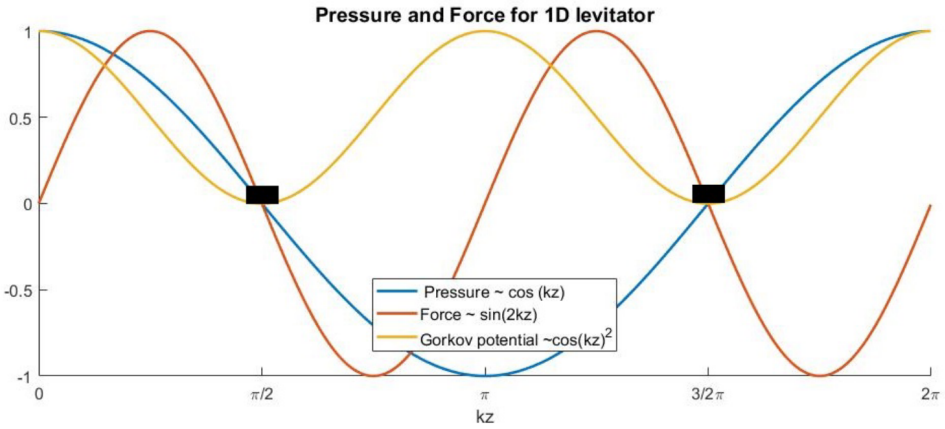


Figure 2-1: Pressure and force in the z-direction [28].

The spring mass-damper system is a second-order system and is depicted in Figure 2-2.

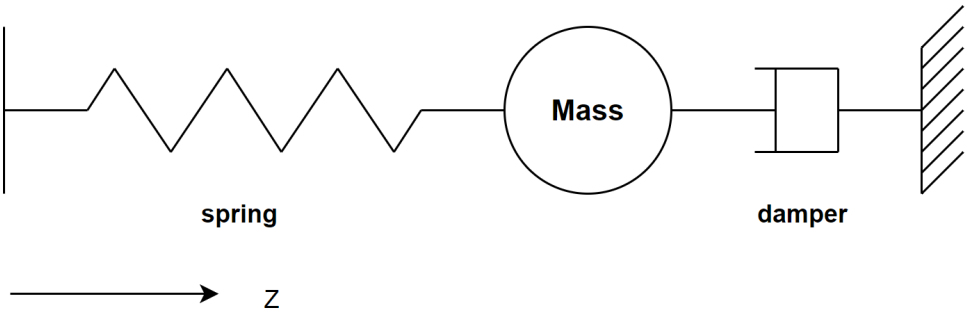


Figure 2-2: A spring-mass-damper system consisting of a mass m , a damping coefficient c, and a spring stiffness k.

The equation of motion for this system is given by:

$$m\ddot{z} = F_{rad} - F_{grav} - F_{drag} \tag{2-11}$$

Where F_{rad} is the acoustic radiation force, F_{grav} is gravity, and F_{drag} is the drag force:

$$F_{drag} = -6\pi\mu R\left(1 + \frac{R}{\delta}\right)\dot{z} = -c_d\dot{z} \tag{2-12}$$

However, in reality, these systems exhibit nonlinear behavior. Instead of adhering to the behavior of a simple harmonic damped mass-spring system, the levitated particles often behave like Duffing oscillators [13]. Unlike harmonic oscillators where the restoring force is directly proportional to displacement, Duffing oscillators feature a nonlinear relationship between displacement and the restoring force. This nonlinear behavior can give rise to chaotic oscillations alongside bifurcations within the system. Due to this behavior, the system demonstrates multiple solutions at specific frequencies. This phenomenon is depicted in Figure 2-3.

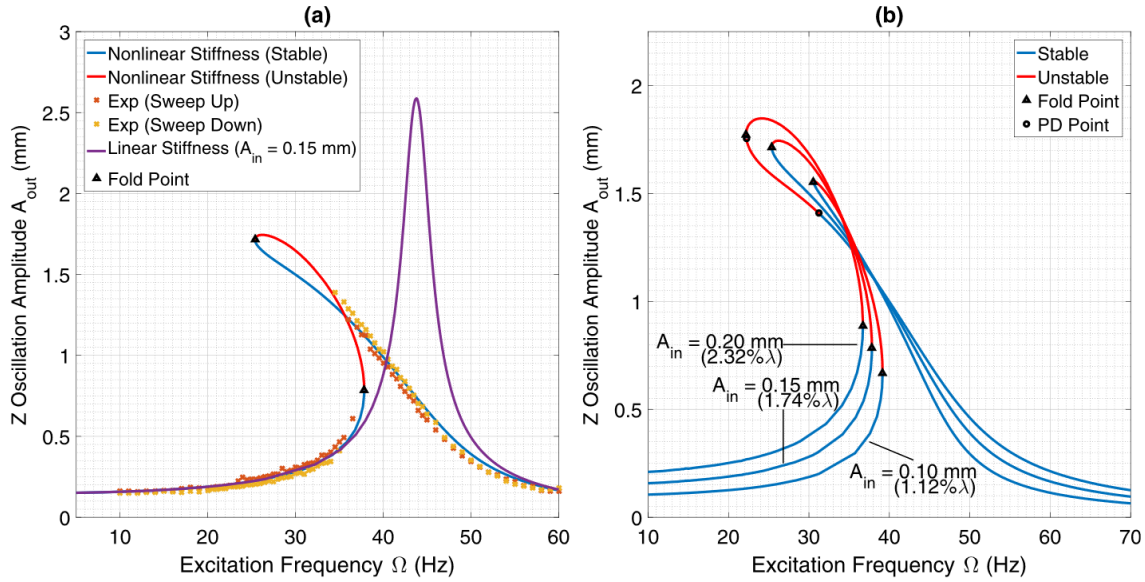


Figure 2-3: Figure depicting the oscillation amplitude for different excitation frequencies showing bifurcations [11].

Analytical models are hard to solve, and parameter adjustments complicate system behavior analysis. This leaves only computationally expensive numerical solvers that can't operate in real-time. Others have attempted using an objective function with optimization strategies to stably levitate objects [29, 30]. To address these challenges, Section 3 will use system identification to derive a model without needing extensive knowledge of the system's underlying physics.

2-2 System identification

System identification involves creating mathematical models of dynamic systems based on observed data. There are three primary approaches to this:

1. **White-box approach:** This method aims to create a model that aligns with a known physical representation of the system, typically using differential equations.

2. **Black-box approach:** This method models the input-output relationship without knowledge of the internal dynamics of the system. It treats the system as a "black box" and focuses on predicting system behavior based solely on observed data.

3. **Grey-box approach:** This combines elements of both white-box and black-box methods by incorporating partial knowledge about the system into the modeling process. Known parameters (such as mass and shape) and general behaviors are used, while some specifics (like resonance frequency) remain unknown.

Several methods exist for obtaining a grey-box model, and the choice depends on the system's characteristics. For instance, nonlinear behavior can limit the accuracy of linear approximations, introducing higher-order errors.

The most straightforward method to identify a system involves using excitation signals. By measuring and processing the output of these signals, a Bode plot can be generated, showing how the system responds over a range of frequencies. A Bode plot consists of two separate plots: the magnitude plot and the phase plot. The magnitude plot shows the system's gain response in decibels (dB) versus frequency, which is a measure of the amplitude of the output signal compared to the input signal. The phase plot illustrates the system's phase shift in degrees versus frequency, indicating the delay between the input and output signals at each frequency.

There are many types of excitation signals, each with its advantages and disadvantages, influencing the choice based on power, post-processing requirements, and clarity of the frequency band.

Types of Excitation Signals:

- **Impulse:** Computationally efficient but less accurate due to low power distribution and low signal-to-noise ratio.
- **White Noise:** Continuously applied signal that can quantify system noise but requires extensive post-processing.
- **Multisine:** High signal-to-noise ratio with power concentrated on frequencies of interest, but requires synchronization.
- **Frequency Sweep:** Systematic frequency characterization with higher power, capable of revealing nonlinear effects, but may miss resonant frequencies.

For this thesis report, a multisine will be used as an excitation signal due to its high signal-to-noise ratio and ability to concentrate power on frequencies of interest. A multisine consists of multiple sinusoids with varying frequencies, amplitudes, and phases and can be mathematically defined by Equation 2-13 [23]

$$u_{MS}(t) = Re\left(\sum_{k=-N/2+1}^{N/2-1} |U_k| e^{j\phi_k} e^{j2\pi f_\Delta kt/N}\right). \tag{2-13}$$

Where N is the number of time samples in one measurement interval, F is the number of frequency components, calculated as $F = \frac{N}{2} - 1$, ϕ_k is the distribution of phases between $[0, 2\pi)$, and $f\Delta$ is the frequency stepsize. The frequency stepsize $f\Delta$ depends on the number of time samples N and sampling frequency f_s, described as $f\Delta = \frac{f_s}{N}$. The frequency range can then be defined by selecting $|U_k| = 1$, or any other constant, for the frequency components of interest and selecting $|U_k| = 0$ for the rest.

When measuring the system response to an excitation signal, mechanical delays are always present. These delays are inherent in all dynamic systems. Other components contributing to delay include the sensor measuring the system, the computer used to analyze and perform tracking algorithms, and the data transmission between these individual components. Identifying and understanding these delays is crucial, as they need to be minimized to capture the best representation of the system dynamics. Delays reduce the phase margin of the system and as a consequence limit the controller bandwidth affecting its performance.

The frequency response of the system can be determined by calculating the the maximum likelihood solution G_{ML} depicted in Equation 2-14 [23].

$$G_{ML(j\omega_k)} = \frac{\frac{1}{M}\sum_{l=1}^{M} Y(k)}{\frac{1}{M}\sum_{l=1}^{M} U(k)} \tag{2-14}$$

Where, M is the number of measurements, $Y(k)$ is the input signal, and $U(k)$ is the output signal.

As it is already known that the system has charactertics of a second-order mass-spring-damper system, a generalized transfer function is given by:

$$H(j\omega) = \frac{1}{-m\omega^2 + jc\omega + k} \cdot e^{-s\tau} = \frac{Y(j\omega)}{U(j\omega)} \cdot e^{-s\tau} \tag{2-15}$$

where k is stiffness, c is damping, m is mass, and $e^{-s\tau}$ represents the delays in seconds s.

2-3 Control

This section introduces control methods for acoustic levitation, emphasizing feedback control's importance in ensuring stability. It details the benefits and trade-offs of using feedback control and introduces the concept of stability margins. This chapter introduces PID control and discusses the advantages and disadvantages of its components. Furthermore, practical rules of thumb are introduced. The chapter concludes by discussing performance criteria and loop shaping for the creation and tuning of a controller.

2-3-1 Motivation behind control

The utilization of feedback control in acoustic levitation is crucial to ensure stable levitation. Feedback control operates by continuously measuring the systems state and makes real-time adjustments to maintain the desired reference signal. This approach offers several advantages

over feed forward control. The biggest advantage of feedback control is the ability to address system uncertainties. This is in contrast to feed forward control, which relies solely on predicting and compensating for a known disturbance in advance.

One of the primary reasons behind the use of feedback in acoustic levitation is the need to incorporate artificial damping though control. The goal is to reduce the oscillations of the levitated object. Real-time control is essential to ensure that the system can promptly adapt to disturbances and variations in operating conditions, thereby maintaining performance within acceptable bounds. Using control also enhances the performance of the system. The primary goal is to increase the accuracy and precision of the levitated sphere when it moves from one point to another.

However, this does not come without drawbacks. Due to the waterbed effect, improving one aspect of the system could lead to unintended consequences elsewhere in the system. To ensure a robust system, some performance may need to be sacrificed. Additionally, the performance of the controller is also linked to the hardware used which introduce several known and unknown disturbances. Furthermore, hardware limitations, e.g. an actuators finite gain, impose constraints on the controller. For these reasons, the controller needs to be designed to be robust and capable of accounting for these uncertainties.

2-3-2 Stability Margins

The concept of bandwidth in controller design is fundamental to understanding the controller's ability to manage and influence a system's behavior. Bandwidth determines the frequency range over which the controller can effectively operate and exert control actions on the system. However, the value of bandwidth is not a parameter that can be chosen arbitrarily. In many cases, the maximum achievable bandwidth is constrained by the resonance frequency. The maximum allowable bandwidth is at the crossover frequency. A higher bandwidth allows the system to respond faster to input changes.

In conjunction with bandwidth, the phase margin and gain margin play important roles in determining the controller's behavior within this frequency range. The phase margin measures the angular difference between the phase of the system's open-loop transfer function and -180°at the frequency where the magnitude of the transfer function is 1 or 0 dB. The gain margin quantifies the maximum amount of gain that can be added to the system before it becomes unstable. It is calculated as the reciprocal of the magnitude of the open-loop transfer function at the frequency where the phase shift reaches -180 °. This can be seen in Figure 2-4.

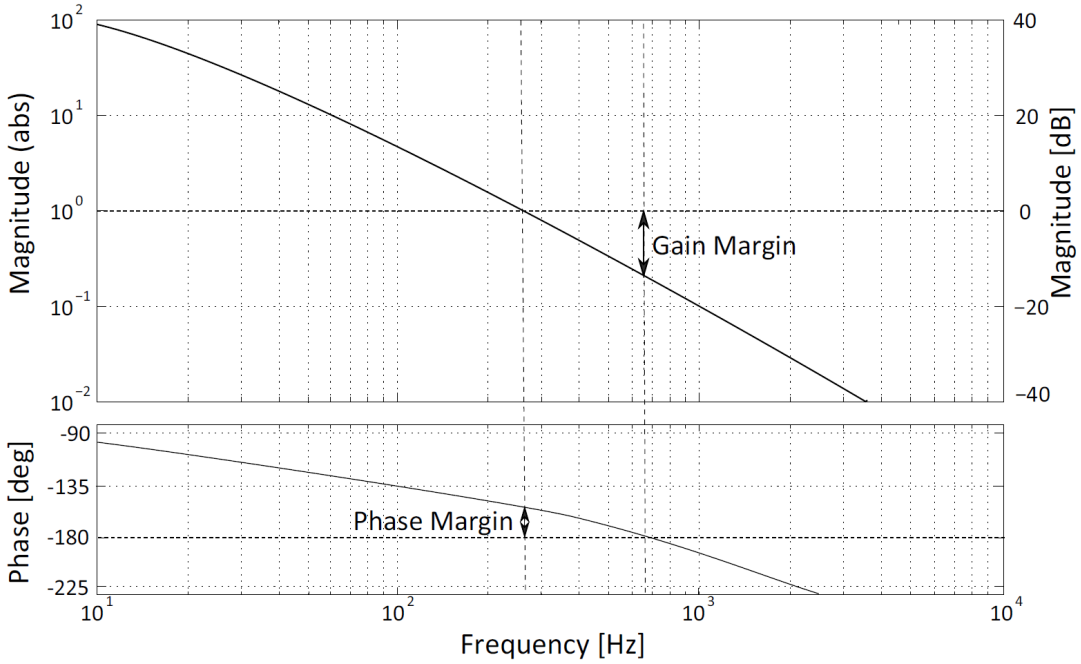


Figure 2-4: Bodeplot with the phase and gain margins [31, p. 264].

Determining the phase and gain margins is critical since they serve as indicators of the system's proximity to instability. These margins are important as there are always differences between the modelled plant and the real physical system that are not accounted for. These variations can change the gain and phase of the system, affecting its stability and performance and are called the process variations. The value of these margins depends on the specific system. Furthermore, filters and disturbances can significantly affect these margins. A larger phase and gain margin indicate greater stability, suggesting that the system is distant from the threshold at which it would start oscillating uncontrollably. This margin acts as a safeguard, ensuring that the system does not suddenly become unstable. However, there's often a trade-off involved. Increasing phase and gain margins for higher robustness can lead to a decrease in the system's bandwidth and overall performance.

In reality, systems often experience a combination of additional phase and gain. This can result in scenarios where a system that initially seems stable may ultimately become unstable. In addition to considering the phase and gain margins, it is essential to also assess the system's sensitivity. The sensitivity functions are expressed in Table 2-1. They are derived from the control loop depicted in Figure 2-5. There are multiple forms of disturbances that affect the system, namely the input disturbance, the output disturbance, and sensor noise. These will be further discussed in Section 2-3-3.

Sensitivity S(s)	**Complementary Sensitivity T(s)**
$\frac{y}{n} = \frac{1}{1+G(s)C(s)}$	$\frac{x}{r} = \frac{y}{r} = \frac{-x}{n} = \frac{G(s)GCs)}{1+G(s)C(s)}$

Table 2-1: Sensitivity and Complementary Sensitivity Functions [31, p. 256].

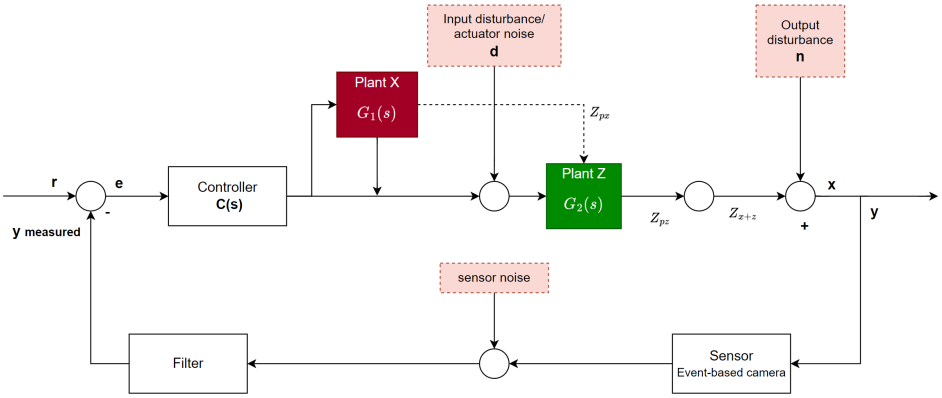


Figure 2-5: Feedback loop with input, output and sensor noise. The movement of the object is coupled. When actuated in the z direction, the object also moves in the x direction, thereby affecting the z direction as well. The actual z position is a combination of the actuation signal and a movement due to the coupled dynamics. The sensor measures this combined movement.

These functions reveal how the system responds to changes and disturbances. Sensitivity **(S)** quantifies the system's responsiveness to variations in inputs and internal dynamics. When sensitivity is high, even minor changes can significantly impact the system's behavior. Complementary sensitivity **(T)** illustrates the system's effectiveness in rejecting disturbances and minimizing errors between the desired reference input and the actual input. Achieving optimal performance involves a trade-off between the two. Sensitivity and complementary sensitivity are reciprocal, as indicated by the equation $\mathbf{S} + \mathbf{T} = \mathbf{1}$. For achieving good tracking and disturbance rejection, the goal is to achieve $\mathbf{T} = \mathbf{1}$ and $\mathbf{S} = \mathbf{0}$. Conversely, for effective noise rejection, this would be $\mathbf{T} = \mathbf{0}$ and $\mathbf{S} = \mathbf{1}$. However, attempting to maximize both simultaneously is impossible. This trade-off is commonly referred to as the waterbed effect, which can be visualized in Figure Figure 2-6.

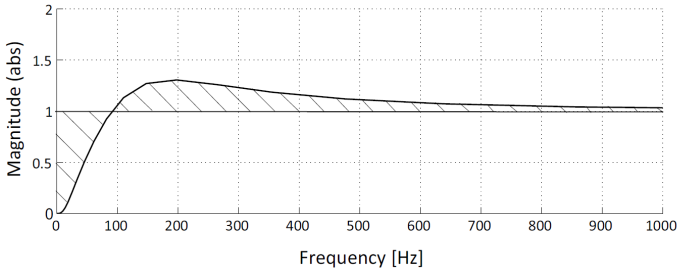


Figure 2-6: Waterbed effect [31, p. 290].

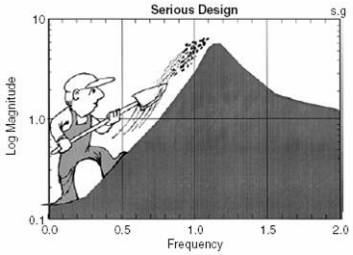


Figure 2-7: Analogy of the waterbed effect [32].

Figure 2-8: Waterbed effect.

Another graphical representation to assess stability is by using the Nyquist Stability Criterion (as shown in Figure 6-17). The Nyquist stability criterion states that a closed-loop control system is stable if and only if the Nyquist plot of its open-loop transfer function does not encircle the critical point of -1 in the complex plane when varying the frequency from zero to infinity. Additionally, the transfer function must not have poles in the right-half plane.

Increasing the gain margin indicates how much the plot can be scaled before crossing the -1 point, while increasing the phase margin indicates how much the plot can be rotated before the curve crosses the -1 point. There are some Nyquist plots that have great gain and phase margins, but certain directions of frequencies lead the plot toward the -1 point. The modulus margin finds the point that is closest in distance to the -1 point.

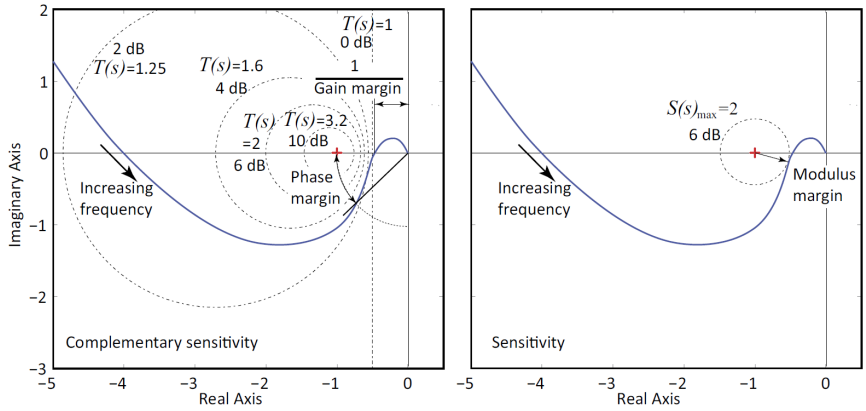


Figure 2-9: Nyquist plot of the sensitivity and complementary sensitivity [31, p. 261].

Specific rules of thumb with associated values for ensuring a robust system (including stability margins) can be found in Table 2-2.

Table 2-2: Robustness Criteria.

Criterion	Value
Gain Margin (GM)	$> 6\,\text{dB}$
Phase Margin (PM)	$> 30^\circ$
Modulus Margin (1/MM)	$< 6\,\text{dB}$

2-3-3 Disturbances

Disturbances can affect the gain and phase margin and consequently the system stability. When disturbances are introduced, they may alter the system's transfer function, potentially reducing the gain. Control strategies like filters can be introduced to reduce the effects of disturbances. Filters reduce or suppress unwanted frequencies. However, the main drawback with filters is that they can introduce delay and reduce the phase margin of the system.

Noise

Noise arises from various sources, but the primary source of noise is originated from the sensor. The sensor used to track the position of the levitated sphere is an event-based camera. This sensor noise is inherent and cannot be removed, resulting in less accurate measurements. The inherent noise floor can be reduced using event-based camera algorithms. This reduction is

achieved by decreasing the camera's sensitivity to changing events, resulting in slower reaction times. The event-based camera will be discussed in detail in Chapter 3.

Disturbances

Unlike noise, disturbances are not random and can often be predicted. The levitated sphere mainly experiences two forms of disturbance: external airflow and temperature fluctuations. These forms of disturbances do not play a significant role and can be accounted for by creating a controlled environment with a constant temperature and by blocking external sounds/breezes. The noise is higher at lower frequencies and decreases as the frequency is increased, exhibiting behavior comparable to pink noise (1/f).

Delays

Multiple components in the control loop introduce delays. These delays are added to the transfer function altering it, lowering the phase margin, which is accounted for by creating a robust controller using the stability margins. The amount of phase added can be calculated using Equation 2-16.

$$\phi = \sum_{i=1}^{n} 2\pi f_i t_i \tag{2-16}$$

In this equation, f_i represents the frequency in Hz and t_i represents the time delay in s for each individual component. Summing these individual delays estimates the total phase lost due to delays.

2-3-4 PID control

PID control is one of the most prevalent forms of control used in industries. A PID controller consists of three components that work together to minimize the error of a system. These components are the Proportional, Integral, and Derivative actions, often denoted as **P**, **I**, and **D** respectively. It is not necessary to use all three components, but their selection depends on the specific system as seen in Figure 2-10.

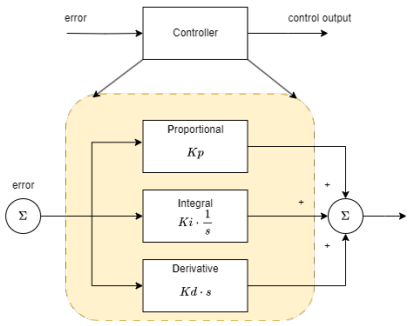


Figure 2-10: Components of a PID controller

The Proportional action is represented by a constant that is multiplied by the error. This primarily affects the magnitude plot by increasing the gain, whilst its impact on the phase is typically minimal and often negligible. The Integral action keeps track of the error by integrating it over time, effectively reducing steady-state errors. It increases low-frequency gain in the Bode magnitude plot and introduces phase lag, particularly at low frequencies. The Derivative action mainly affects the magnitude plot at high frequencies. It reduces gain at frequencies where rapid changes occur in the magnitude plot and introduces phase lead. The derivative action is often used to provide a damping effect. Figure 2-11 provides a visual representation, where f_i is the frequency where the integrator actions starts. f_d and f_t denote the frequencies at which the derivative action starts and ends. f_c is the crossover frequency that is chosen. The exact values can be determined using the rules of thumb outlined in Table 2-3.

Table 2-3: Rules of thumb.

P Control	PD Control	PID Control
$k_p = \left\lvert \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$	$f_d = \frac{f_c}{3},\ f_t = 3f_c$ $k_p = \left\lvert 0.33 \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$	$f_d = \frac{f_c}{3},\ f_t = 3f_c,\ f_i = \frac{f_c}{10}$ $k_p = \left\lvert 0.33 \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$

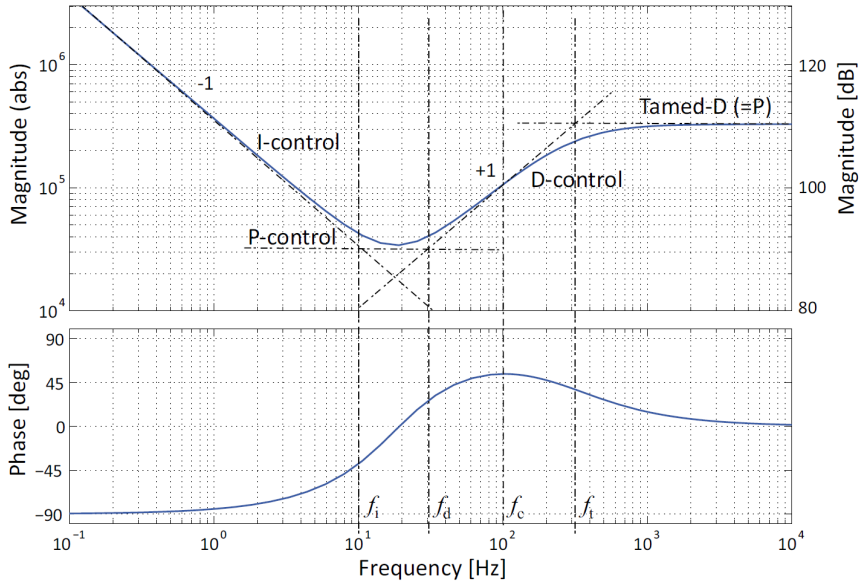


Figure 2-11: Bode plot applying the rules of thumb [31, p. 286].

The integral action does not come without disadvantages as it is susceptible to integral windup. When the integral action is increased beyond the actuator limit, it becomes saturated, and any additional increase in the integral term cannot be applied to the system. This error accumulates over time and can become excessive. To prevent this issue controllers are often designed with an integral limiter and saturation blocks. These components ensure that the integral action does not exceed the hardware limitations, thus mitigating the risk of overshoot and oscillations.

The derivative action is susceptible to high-frequency noise. This noise often consists of small fluctuations in the measured signal. This noise is mainly caused by sensor noise but could also be contributed to electrical interference or environmental disturbances. When the derivative gain k_d is set too high, the controller becomes sensitive to high-frequency noise. Increasing the gain amplifies the slope of the derivative, magnifying the fluctuations. This can be counterproductive, making the system overly responsive to noise. To mitigate this, filters are often used to remove high-frequency noise before it reaches the derivative action. The use of filters should be considered carefully as they can introduce delays reducing the phase and gain margins of the system, affecting the robustness. These finding are summarized in Table 2-4 and Table 2-5

Table 2-4: PID Control Components.

Component	**Equation**	**Purpose**
Proportional (P)	$u(t) = K_p \cdot e(t)$	Reduce steady-state errors
Integral (I)	$u(t) = K_i \cdot \int e(t)\,dt$	Eliminate steady-state errors
Derivative (D)	$u(t) = K_d \cdot \frac{de(t)}{dt}$	Reduce overshoot and oscillations

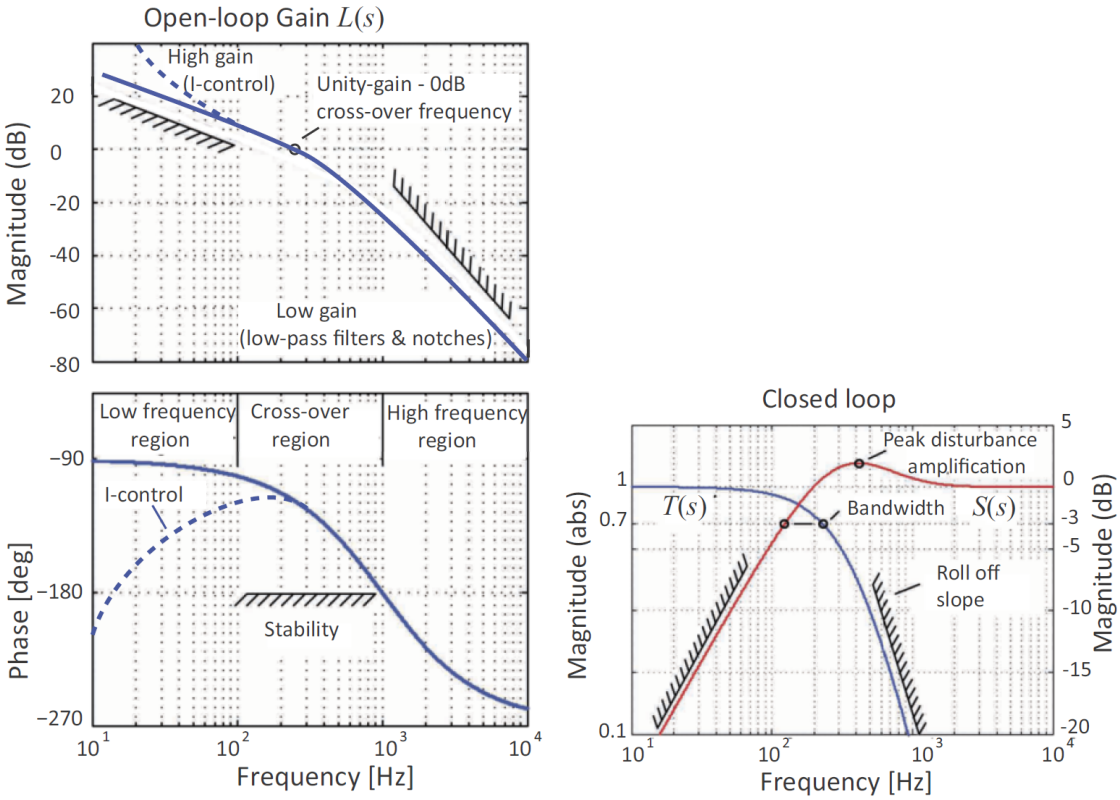
Table 2-5: Pros and Cons of PID Control Actions.

Control Action	**Pros**	**Cons**
Proportional (P)	Immediate response to current error.	Can lead to oscillations if gain is too high. May not eliminate constant errors.
Integral (I)	Eliminates steady-state errors. Corrects for long-term biases.	Slower response to sudden changes. Susceptible to integral windup when saturation occurs.
Derivative (D)	Prevents overshooting and reduces oscillations.	Amplifies high-frequency noise if gain is too high. Sensitive to measurement noise.

2-3-5 Loop-Shaping

The process of tuning a controller to achieve the desired system performance characteristics using previously established concepts such as gain/phase margin and sensitivity functions is called loop-shaping. Using loop-shaping, the total frequency of interest is divided into three bands as depicted in Figure 2-12. For an open-loop function, at low frequencies the slope is set to -2 to achieve good tracking and disturbance rejection. In the crossover region the slope

is adjusted to -1 to ensure optimal stability and robustness. At high frequencies the slope is set to -2 to reduce high-frequency noise and high-frequency resonances.



(a) Loop-shaping open-loop.

(b) Loopshaping closed-loop.

Figure 2-12: Loop-shaping [31, p. 313].

Chapter 3

Event-Based Camera

This chapter introduces event-based cameras, their distinctive working principles, and event processing. It begins by outlining the limitations of traditional cameras and introduces event-based cameras. The chapter also introduces several camera characteristics that can often be adjusted using camera settings called Biases. These Biases will be explained, and how changing them affects the sensor. Lastly, a summary of the advantages and disadvantages of event-based cameras is provided.

3-1 Event camera working principles

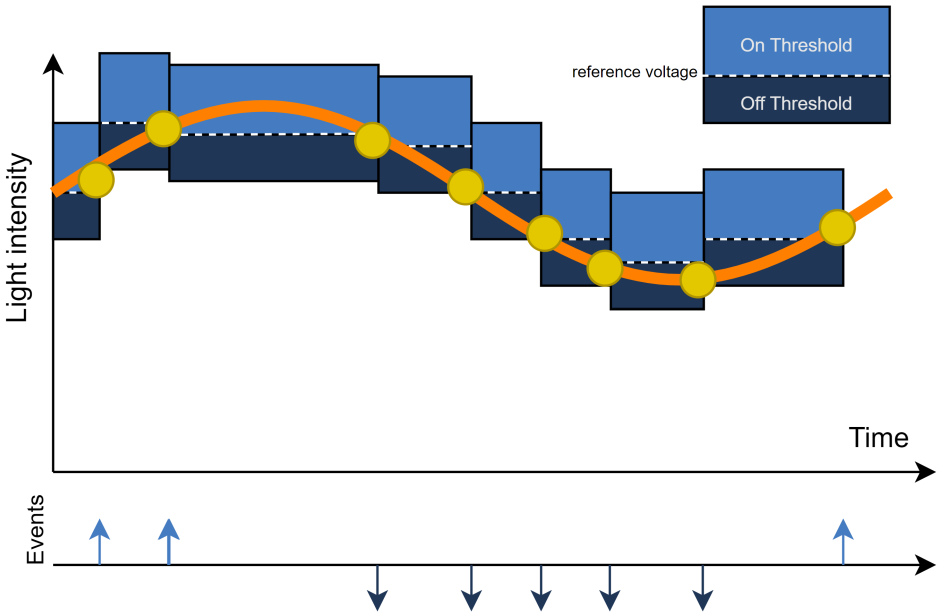


Figure 3-1: Visualization of event detection.

Conventional cameras generate video by capturing and stitching together frames at fixed intervals known as refresh rates, typically measured in Hertz $[Hz]$ or frames per second $[fps]$. Information occurring between two consecutive frames is lost and remains uncaptured. To address this limitation, conventional cameras can increase their frame rate to capture more details. However, achieving higher frame rates necessitates more expensive high-speed sensors, along with improved image signal processors (ISP). The rapid influx of data also demands faster memory and larger storage capacity. Another shortcoming of conventional cameras is their capture of unchanging frames, which often contain unnecessary information, further exacerbating the need for larger storage capacity. Event-based cameras address these issues by mimicking the operation of the human eye. The camera's individual pixels can independently detect and process changes, known as 'events' depicted Figure 3-1.

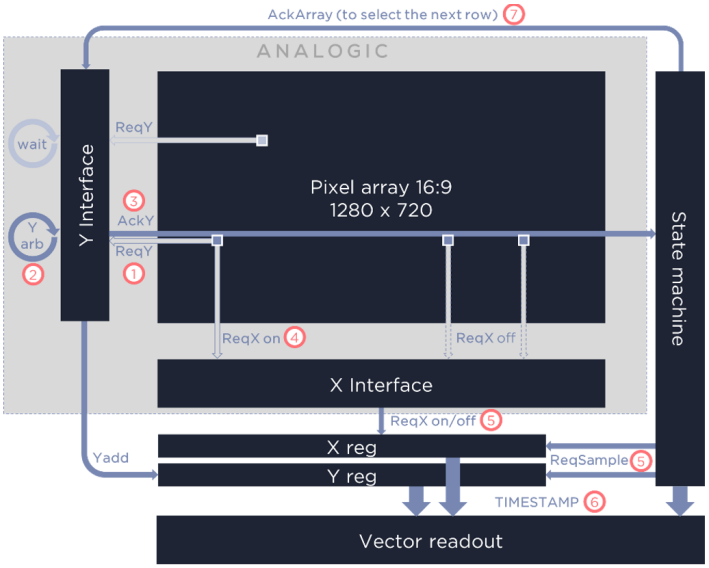


Figure 3-2: Sensor pixel Readout [33].

Only when a pixel's threshold is exceeded will the camera process a new event; an event is triggered for a readout, as shown in Figure 3-2. This readout operates on demand, meaning it does not involve periodic requests from the pixels; instead, the pixels independently initiate the readout process . The selection process follows a first-come-first-serve principle. Once chosen, the 'Y' address and a vector of active pixels within a row are read out. All events within the same row share a common timestamp. After the arbiter completes the readout, it cycles back to the row, adhering to the first-come-first-serve approach. This approach enables nearly continuous image capture in contrast to fixed, discrete intervals, and can achieve speeds of over 10,000 fps.

The sensor output includes the following information: a **timestamp**, the **pixel's address** in the sensor array, and the **contrast detection polarity**. A polarity value of 1 signifies a positive contrast, indicating increased light, and is represented as an 'ON' event. On the other hand, a polarity value of 0 denotes a negative contrast, representing decreased light and is manifested as an 'OFF' event. Additionally, an 'ON' event is represented by a white color, while an 'OFF' event is denoted by a dark blue color. The event data can be processed and visualized in different ways and will be discussed in Section 3-4 and Section 3-5 respectively.

3-2 Event camera types

Event cameras come in three distinct variants. The first type is the Dynamic Vision Sensor (DVS), capable of detecting changes in contrast (CD) and was the first commercially available event camera, depicted in Figure 3-3. The second variant is the Asynchronous Time-based Image Sensor (ATIS), which not only identifies changes (CD) but also provides exposure measurements (EM), which allows for greyscale output. The third, known as Dynamic Pixel and Active Vision Sensor (Davis), combines a standard frame-based camera with a DVS event camera in the same pixel array. Consequently, it delivers an output of DVS events and standard frames. The frame output is not in High Dynamic Range (HDR) but in standard dynamic range (SDR) and has a maximum dynamic range of 55 dB with a low spatial resolution. Notably, this camera type is not capable of outputting event data and frame data simultaneously, accommodating only one type at any given time.

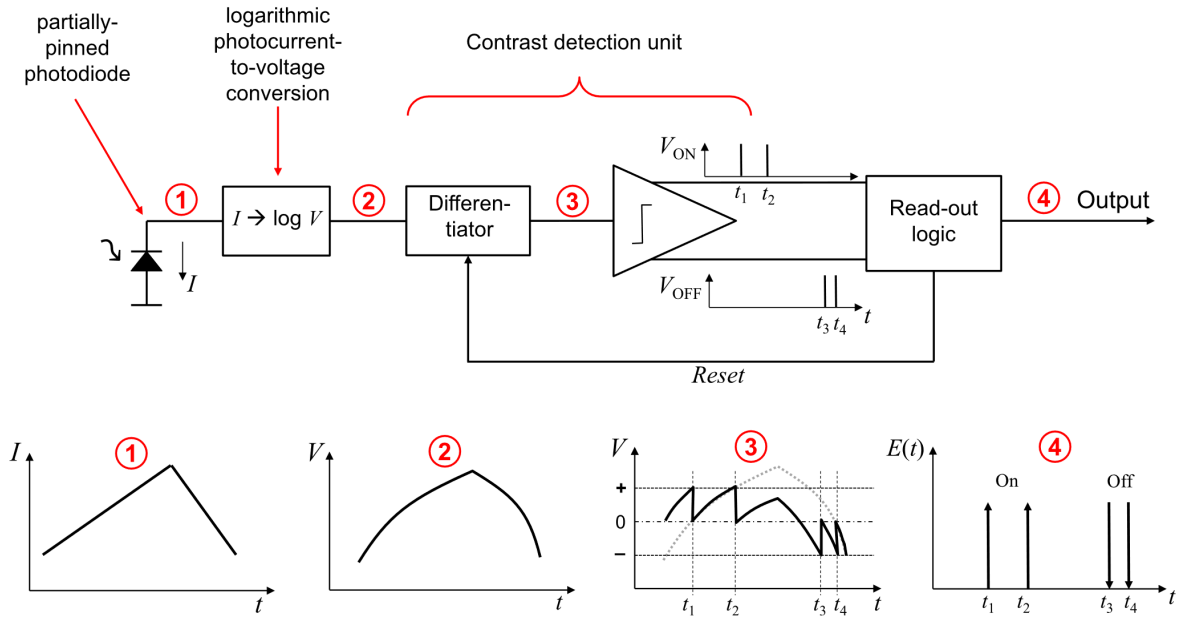


Figure 3-3: Schematic representation of how a pixel registers events [34].

The different event camera types exhibit varying pixel sizes, which stem from the diverse logic embedded within the pixels for visual processing. Specifically, for this master thesis, the EVK4HD model from Prophesee was chosen among the array of available event-based cameras. While various brands manufacture event-based cameras, this particular model was selected due to a combination of its specifications and well-documented software (Figure 3-4).

3-3 Sensor Characteristics

There are several important camera characteristics that will be explained in detail to better understand how an ATIS event camera processes events. This understanding is vital because these characteristics can often be individually adjusted and may influence event processing.

	Supplier	iniVation			Prophesee				Samsung			CelePixel		Insightness
	Camera model	DVS128	DAVIS240	DAVIS346	ATIS	Gen3 CD	Gen3 ATIS	Gen 4 CD	DVS-Gen2	DVS-Gen3	DVS-Gen4	CeleX-IV	CeleX-V	Rino 3
Sensor specifications	Year, Reference	2008 [2]	2014 [4]	2017	2011 [3]	2017 [67]	2017 [67]	2020 [68]	2017 [5]	2018 [69]	2020 [39]	2017 [70]	2019 [71]	2018 [72]
	Resolution (pixels)	128 × 128	240 × 180	346 × 260	304 × 240	640 × 480	480 × 360	1280 × 720	640 × 480	640 × 480	1280 × 960	768 × 640	1280 × 800	320 × 262
	Latency (µs)	12μs @ 1klux	12μs @ 1klux	20	3	40 - 200	40 - 200	20 - 150	65 - 410	50	150	10	8	125μs @ 10lux
	Dynamic range (dB)	120	120	120	143	> 120	> 120	> 124	90	90	100	90	120	> 100
	Min. contrast sensitivity (%)	17	11	14.3 - 22.5	13	12	12	11	9	15	20	30	10	15
	Power consumption (mW)	23	5 - 14	10 - 170	50 - 175	36 - 95	25 - 87	32 - 84	27 - 50	40	130	-	400	20-70
	Chip size (mm^2)	6.3 × 6	5 × 5	8 × 6	9.9 × 8.2	9.6 × 7.2	9.6 × 7.2	6.22 × 3.5	8 × 5.8	8 × 5.8	8.4 × 7.6	15.5 × 15.8	14.3 × 11.6	5.3 × 5.3
	Pixel size (µm^2)	40 × 40	18.5 × 18.5	18.5 × 18.5	30 × 30	15 × 15	20 × 20	4.86 × 4.86	9 × 9	9 × 9	4.95 × 4.95	18 × 18	9.8 × 9.8	13 × 13
	Fill factor (%)	8.1	22	22	20	25	20	> 77	11	12	22	8.5	8	22
	Supply voltage (V)	3.3	1.8 & 3.3	1.8 & 3.3	1.8 & 3.3	1.8	1.8	1.1 & 2.5	1.2 & 2.8	1.2 & 2.8		1.8 & 3.3	1.2 & 2.5	1.8 & 3.3
	Stationary noise (ev/pix/s) at 25C	0.05	0.1	0.1	-	0.1	0.1	0.1	0.03	0.03		0.15	0.2	0.1
	CMOS technology (nm)	350	180	180	180	180	180	90	90	90	65/28	180	65	180
		2P4M	1P6M MIM	1P6M MIM	1P6M	1P6M CIS	1P6M CIS	BI CIS	1P5M BSI			1P6M CIS	CIS	1P6M CIS
Camera	Grayscale output	no	yes	yes	yes	no	yes	no	no	no	no	yes	yes	yes
	Grayscale dynamic range (dB)	NA	55	56.7	130	NA	> 100	NA	NA	NA	NA	90	120	50
	Max. frame rate (fps)	NA	35	40	NA	NA	NA	NA	NA	NA	NA	50	100	30
	Max. Bandwidth (Meps)	1	12	12	-	66	66	1066	300	600	1200	200	140	20
	Interface	USB 2	USB 2	USB 3		USB 3	USB 3	USB 3	USB 2	USB 3	USB 3			USB 2
	IMU output	no	1 kHz	1 kHz	no	1 kHz	1 kHz	no	no	1 kHz	no	no	no	1 kHz

Figure 3-4: This table contains various event-based sensors from different manufacturers, as documented in [15].The event camera used is the GEN 4 CD from Prophesee.

Each subsection will focus on a specific characteristic. Some characteristics can be altered by tuning the Bias settings, a concept that will be addressed later in 3-9.

3-3-1 Contrast sensitivity

The main function of event cameras is the detection of contrast changes. This is temporal contrast, which is a contrast change over time for a single pixel and is defined as a ratio of the higher to lower luminance. This as depicted in Equation 3-1.

$$c = \frac{I_{High} - I_{Low}}{\log(1 + I_{Low})} \tag{3-1}$$

where (c) represents contrast based on the difference between the new measured intensity (I_{High}) and the old measured intensity (I_{Low}).

The contrast sensitivity depicts the temporal contrast response probability of pixels over a range of temporal contrasts for a given illumination, as shown in Figure 3-5. Response probability represents the percentage of pixels that generate events when exposed to a certain temporal contrast step. Contrast sensitivity depends on the light level and can be adjusted by controlling the environmental lighting or changing the type of lighting, allowing for the detection of small or large contrast changes. Furthermore, tuning the settings of **`bias_diff_on`** and **`bias_diff_off`** has the potential to modify the contrast sensitivity.

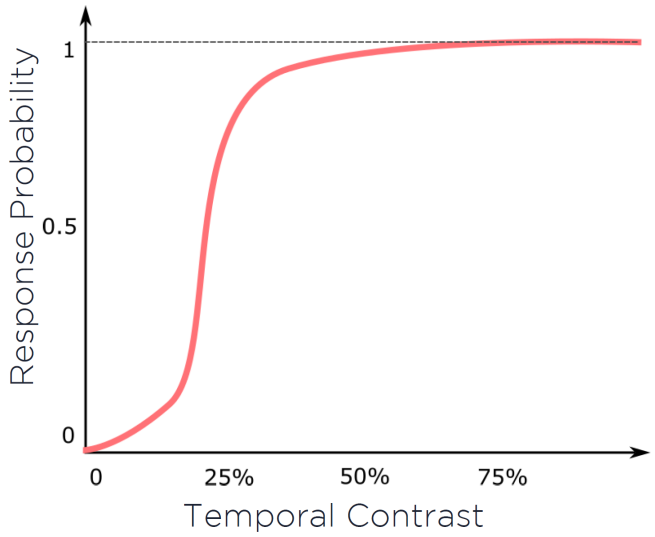


Figure 3-5: Response-probability [35].

3-3-2 Background rate

The background rate characterizes the number of events generated over time under constant and static illumination, signifying the absence of changes in the scene and essentially representing the background noise. It is expressed in Hz/pixel. As shown in Figure 3-6, the background rate is influenced by both photonic noise and electronic noise from each pixel and varies with the light level, as illustrated in the figure. Adjustments to the `bias_HPF` setting and sensitivity biases can effectively reduce the background rate.

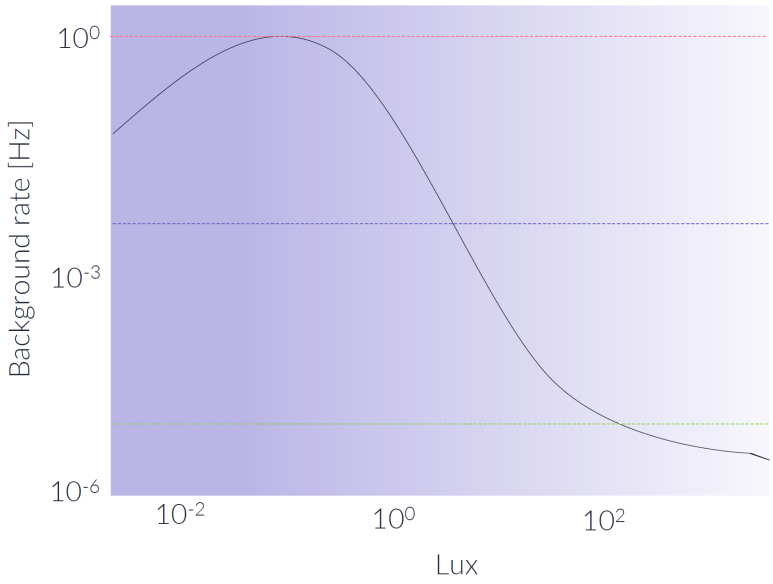


Figure 3-6: Background rate [35].

3-3-3 Pixel Latency and Jitter

The pixel latency is the duration between the change in the scene and the detection of the pixel, expressed in microseconds. A lower pixel latency is indicative of better performance. Pixel jitter represents the temporal precision at the pixel level. Both pixel latency and jitter, depicted in Figure 3-7 tend to decrease with higher light intensity and can be further enhanced by fine-tuning the `bias_fo` setting.

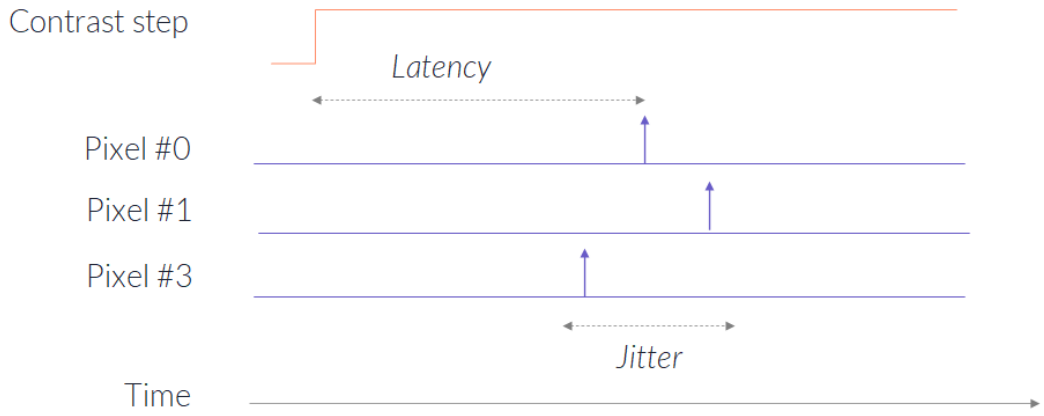


Figure 3-7: Pixel latency and jitter [36].

3-4 Event Data Processing

Event-based cameras store the logarithmic intensity of each event at the level of individual pixels. This change in intensity is referred to as a Change Detection (CD) event. This CD event includes a timestamp, pixel X and Y addresses in the sensor array, and an 'On'/'Off' polarity.
The Prophesee EVK4HD can also generate other types of events, such as External Trigger events. These events include specific timestamps, ChannelId, and Raise/Fall events. Additionally, there are monitoring events that provide periodic information or alerts about the system, such as temperature, voltage, and event counts. These events are encoded in one of the three proprietary EVT formats, depicted in Table 3-1.

Table 3-1: EVT Compression Ratios [37].

EVT format	**Compression Ratio (Bits/Event)**
EVT 2.1	2 bits/event
EVT 3.0	1.5 bits/event

The older EVT 2.1 format stores information in an 2 bit/event data format, primarily intended for use with older-generation sensors (Gen 3.x type). In contrast, the newer EVT 3.0 format

employs a compressed 16-bit format, utilizing the 4 most significant bits to define the format and word type. This format captures an event stream of a row of pixels, including a timestamp, the Y address of the pixel, and the X address of all the pixels within the row. Designed for compact data storage, it takes advantage of vectorized event data to avoid transmitting redundant information. To fully leverage this, it is advisable to position the object to generate events in rows rather than columns. EVT 3.0 exhibits the lowest compression ratio at 1.5 bits per event, showcasing its efficiency in data storage. However, it's important to note that the decoding process for EVT 3.0 incurs higher computational costs due to the state machine. Decoded events can be accessed using a call back function in the `Metavision::EventCD` class. A table detailing the stored event types and a visual representation of the encoding are presented in Table 3-2 and Figure 3-8, respectively.

Table 3-2: EVT 3.0 encoding format.

Event Type	Description
Evt_Addr_Y	Y coordinate, and system type (master/slave camera)
Evt_Addr_X	Single valid event, X coordinate and polarity
Vect_Base_X	Base X coordinate and polarity for subsequent vector events
Vect12	Vector event with 12 consecutive elements
Vect8	Vector event with 8 consecutive elements
Time_Low	Updates the lower 12-bit portion of the 24-bit time base
Time_High	Updates the higher 12-bit portion of the 24-bit time base
Ext_Trigger	External trigger output

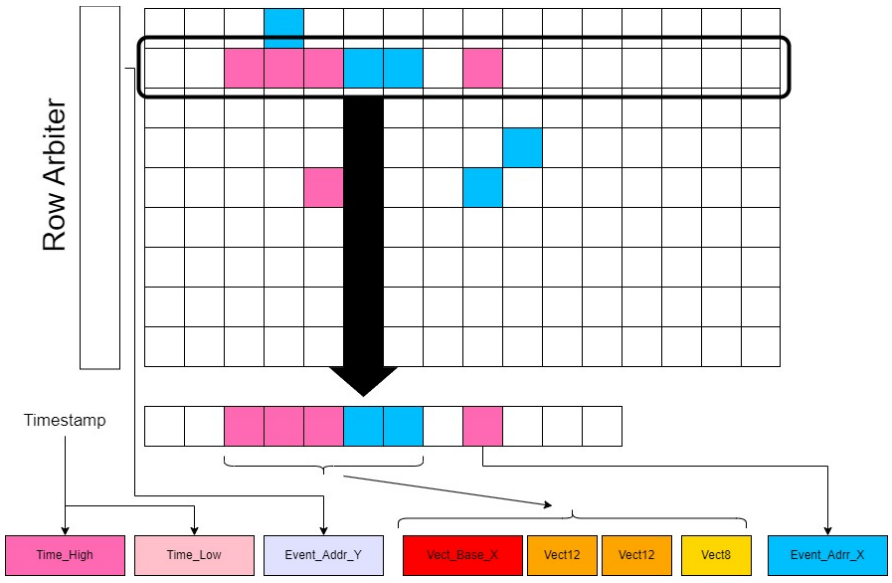


Figure 3-8: EVT Decoding.

3-4-1 Total latency

The delay between the scene and event processing in software is called the total latency and differs from pixel latency. Total latency is the sum of pixel latency, readout latency, transfer latency and software latency, illustrated in Figure 3-9. Readout, transfer, and software latency are influenced by the event rate, with higher rates contributing to increased latency. Optimization of the setup, configuring the ROI (explained in Section 3-7), Bias tuning and employing digital filters can collectively reduce total latency. Notably, the use of digital filters does not impact readout latency. Additionally, timestamping is implemented on the sensor chip, ensuring that the accuracy of event detection remains unaffected by data transfer and software latencies.

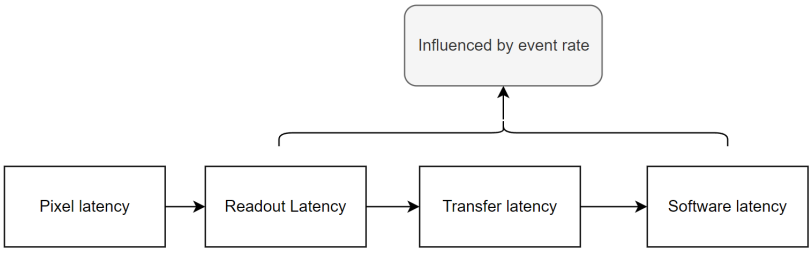


Figure 3-9: Illustration of the total Latency [35].

3-5 Event Visualization and Representation

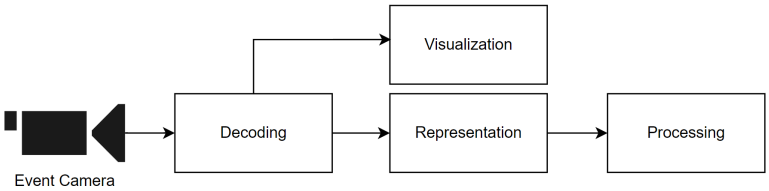


Figure 3-10: Illustration of the components of event visualization and representation.

The process of decoding transforms raw data into a format that can be conveniently visualized and represented, offering flexibility for various applications, depicted in Figure 3-10. Visualization serves as a means of enhancing human comprehension by providing a way to present the decoded data in a manner akin to a frame-based video, making it more accessible and user-friendly. However, this convenience comes with its own set of drawbacks. Observing accumulated events is only possible in slow motion during post-processing, where each 'frame can be examined'. This process takes a lot of time when viewing long recordings. It is possible to view a live recording and change camera settings in real-time but at 30 frames per second. However, live viewing often misses a lot of details that can only be seen in post-processing. Additionally, it is not feasible to apply algorithms or other actions while viewing a live feed from the camera without incurring heavy performance penalties. On the other hand, representation focuses on facilitating advanced processing and the application of algorithms. There are three primary types of event representations: event vectors, time-surfaces [38], and his-

tograms. Each type caters to specific requirements, providing a versatile toolkit for extracting meaningful insights from the decoded data.

3-5-1 Image and Video rendering

When events are represented as an image, multiple events accumulate over time, and this is referred to as accumulation time. This concept can be likened to the "Exposure Time" used in frame-based cameras. An image is generated by accumulating pixels over a specific period, and each pixel is colored based on the polarity of the last event. Figure 3-11

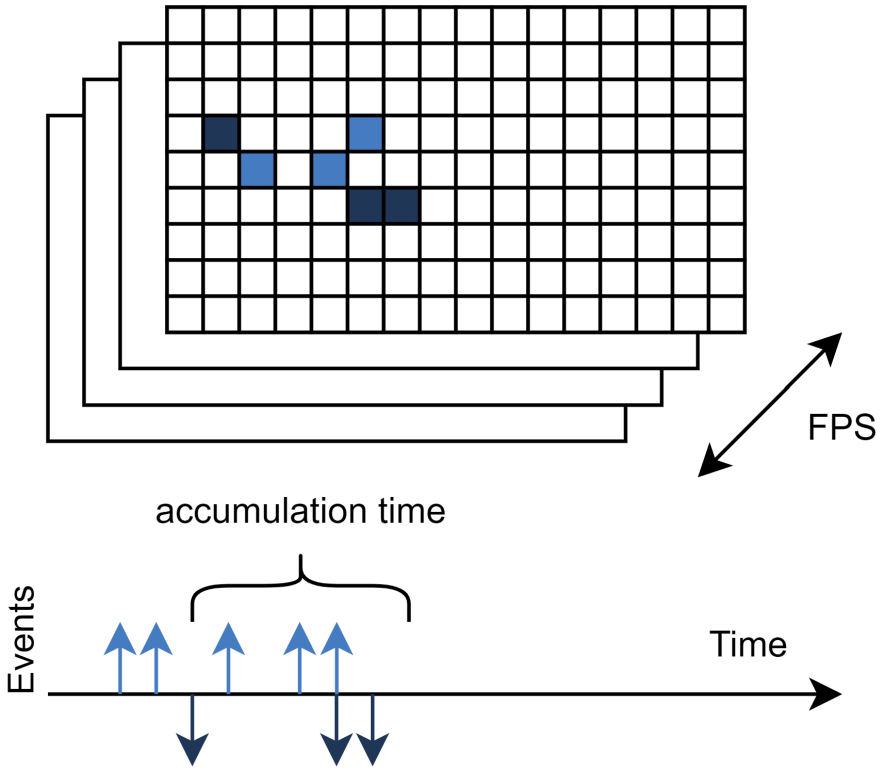


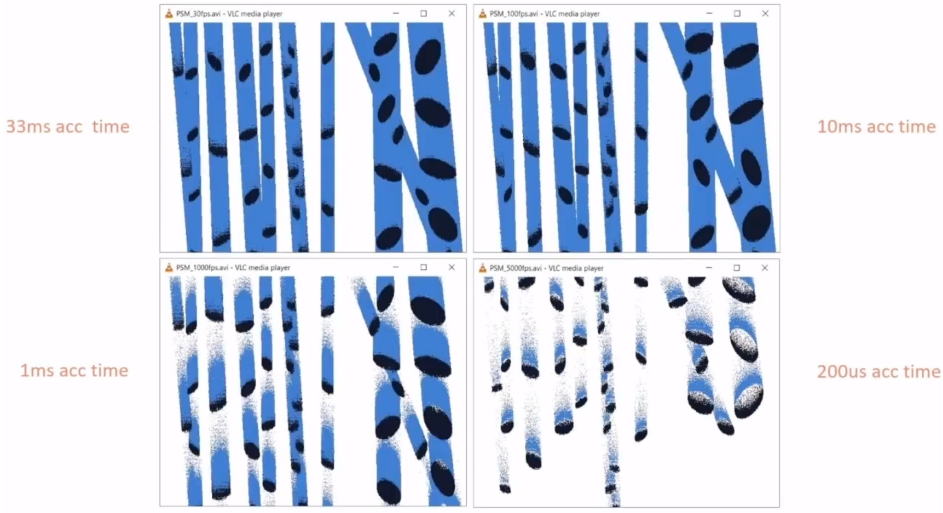
Figure 3-11: Illustration of accumulated events over time.

However, an image may not be the most effective way to represent events, especially in the case of fast movements within a scene. The shorter the accumulation time, the less data is depicted in an image. Using a large accumulation time for fast-moving objects can make it challenging to comprehend the unfolding events and the same difficulty arises when employing a small accumulation time for slow-moving objects. Various examples illustrating these differences for different accumulation times are presented in Figure 3-12

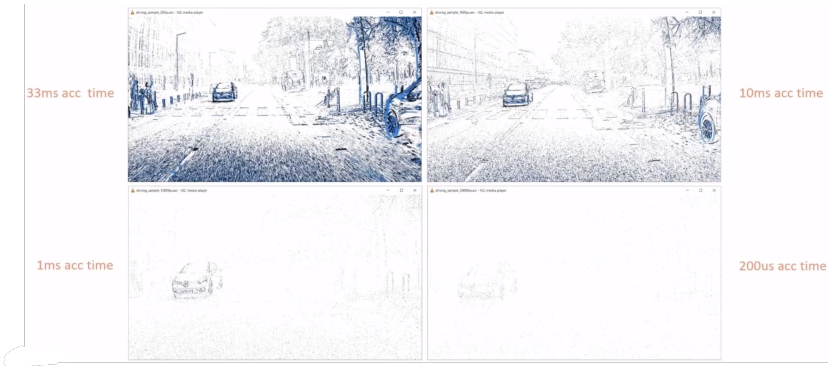
Utilizing the principle of generating images and sequencing them, a video can be created at a specific frame rate. This frame rate, used for rendering, may not be equal to the rate of acquiring new events. The faster an object moves, the higher the frame rate needed to effectively capture and follow the movement, resembling a slow-motion view. Conversely, with a slowly moving object, a lower frame rate suffices to comprehend the scene. The frame rate in $[Hz]$ is calculated as $\frac{1}{accumulationtime}$ in [µs].



(a) This figure depicts a rotating dot where lowering the accumulation time reveals the movement.



(b) The figure depicts falling particles with great velocity. Decreasing the accumulation time provides a clearer picture of the individual objects falling.



(c) This figure shows pedestrians crossing a street and a car. Due to the slow movement, increasing the accumulation time provides too little information.

Figure 3-12: This figure provides examples of different accumulation times for various scenarios [39].

3-5-2 XYT view

Another approach to visualizing events is the XYT view, wherein each event is represented as a 3D point with x and y coordinates on the spatial axes and the time axis. This representation effectively illustrates the continuous asynchronous properties of event-based vision. The frame-based equivalent is also portrayed as snapshots, resulting in a loss of substantial data between frames. Unfortunately, this visualization method is primarily useful for simpler scenes, and it may become challenging to comprehend in more complex scenarios.

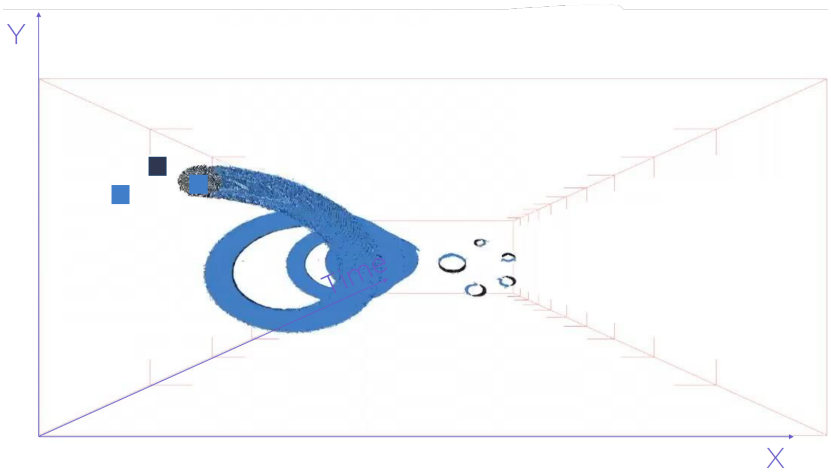


Figure 3-13: XYT view of a dot on a rotating disk [39].

3-6 Event Signal Processing

The Prophesee EVK4HD contains an Event Signal Processor (ESP). This hardware allows for direct filtering before any processing. There are three different types of filters with various settings, which will be explained. Additionally, some of the available hardware filters have a software alternative that can be implemented.

3-6-1 Hardware Filters

Anti-Flicker (AFK): The anti-flicker filter removes flickers using a 4x4 pixel grid on the sensor. This enables the filter to eliminate events that are generated by external light sources, such as fluorescent light. The flicker range that can be removed is between a period of 2 ms and 20 ms. This filter can be enabled in the Metavision Studio GUI depicted in Figure 3-14a or directly implemented in the code using `Metavision::I_AntiFlickerModule` and `Metavision::AntiFlickerModule`. Additionally, an equivalent software filter can be used: `Metavision::AntiFlickerAlgorithm`.

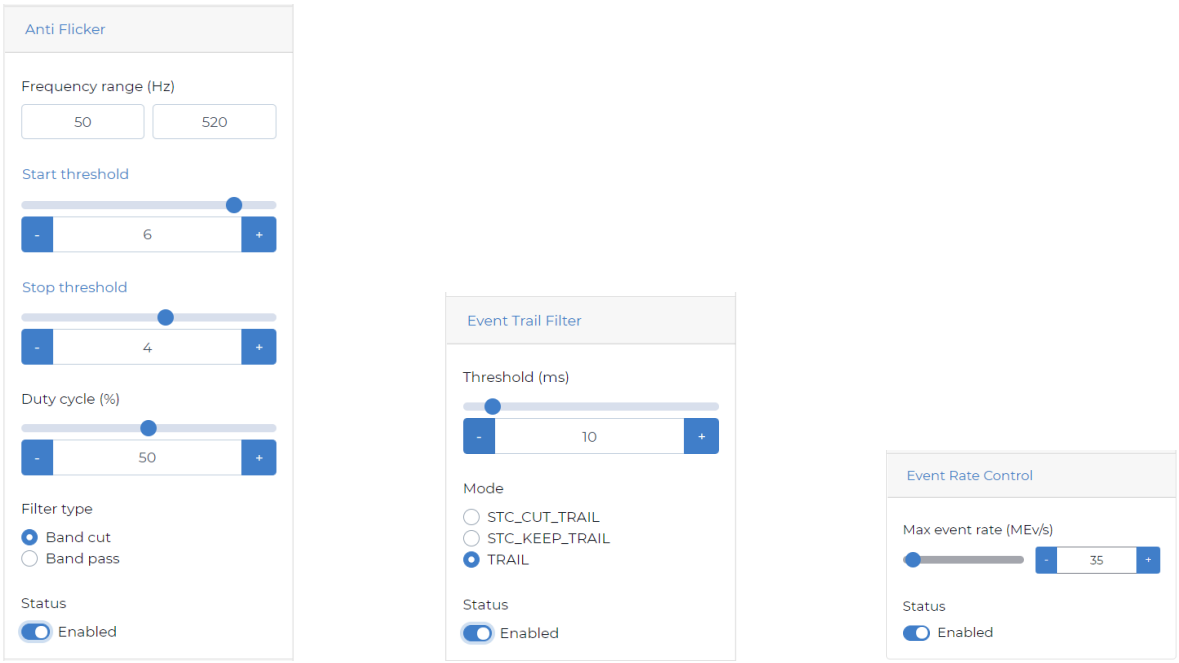
Event Trail Filter: This filter eliminates redundant events, often forming a trail following a moving object. Events of the same polarity that occur within a short time can be selectively removed based on the chosen option. While this filter effectively reduces the event rate, it comes at the cost of removing texture information. The applicable time

range for both options is between 1 ms and 100 ms. Additionally, the software filters `Metavision::TrailFilterAlgorithm` and `Metavision::SpatioTemporalContrastAlgorithm` can be utilized.

Spatio-Temporal-Contrast (STC): It selectively retains only the second event of a burst and is designed to filter out isolated events. This filter offers two variants: `STC_CUT_TRAIL` and `STC_KEEP_TRAIL`, allowing the user to decide whether to cut or keep the trail.

TRAIL: This function preserves the initial event within a burst following a polarity transition while discarding events of the same polarity generated within the Trail threshold period.

Event Rate Control (ERC): Influences the maximum event rate, ensuring it does not exceed the maximum processing capacity. This filter is applied after the Anti-Flicker (AFK) and Spatio-Temporal-Contrast (STC)/Trail filters to effectively reduce the event rate. The ERC can also be enabled in software using `Metavision::I_ErcModule` and `Metavision::ErcModule`.



(a) AFK. **(b)** ETF. **(c)** ERC.

Figure 3-14: Hardware filter options.

3-7 Region of interest

The pixels in the IMX 636 sensor can be considered active, producing events, or can be disabled, thus not generating events. Disabling or enabling certain parts of the sensor can be achieved using a Region Of Interest (ROI) or Region Of Non-Interest (RONI). Multiple ROIs/RONIs can be selected, reducing the event rate, depicted in Figure 3-15 .

This is useful for applications where the object occupies only a portion of the field of view. Additionally, focusing on a specific region helps reduce power consumption. Row/column

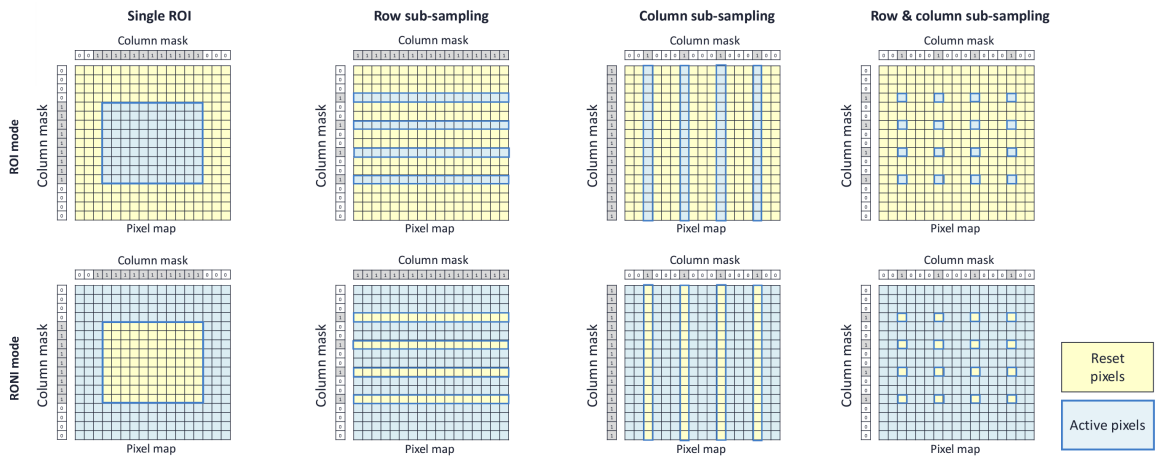


Figure 3-15: Multiple examples with ROI [40]. The single ROI mode seems useful for tracking the levitated sphere due to its low amplitude oscillations.

subsampling is useful for applications where multiple high-speed objects are passing, such as on an assembly line or in counting applications, depicted in Figure 3-16.

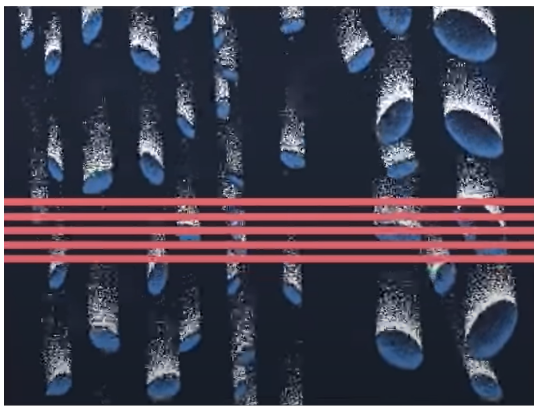


Figure 3-16: Subsampling where certain rows are used to register/count falling objects [41].

3-8 Optimal Lighting Conditions for Event Cameras

To effectively capture images using event cameras, proper illumination of the object is paramount. Event cameras rely on changes in contrast to detect and record visual information, making sufficient lighting crucial for obtaining clear images. The level of contrast detected is influenced by various factors, including the velocity and displacement of the object being recorded. Objects with minimal movement or stationary may not generate new events if lighting conditions remain controlled and constant. Several methods can be employed to illuminate objects effectively. One approach involves utilizing ambient environmental lighting, leveraging existing light sources without additional illumination. Alternatively, a constant light source emitting light continuously offers flexibility in positioning to achieve optimal illumination from different angles relative to the camera's position. Another technique involves using a flash or laser

[34] to illuminate the object with a certain exposure, triggering events based on variations in brightness. However, this method presents challenges, such as potential information loss between the flash frequency and generation of extra unnecessary events and noise. To address this issue, adjusting exposure settings, applying bias settings, and selectively deactivating sensor areas (applying ROI) can help mitigate unnecessary events and noise caused.

3-9 Biases

The purpose of tuning sensor settings is to optimize performance in diverse environmental conditions and specific applications, resulting in enhanced data quality. This tuning improves pixel sensitivity, pixel speed, and reduces noise. The use of various sensor settings facilitates the acquisition of valuable data, contributing to an optimized readout. This optimization is crucial to prevent processed events from becoming excessively large for efficient processing or storage. In the case of the IMX636, the maximal readout is 3 GeV/s. However, when all 921,600 pixels are saturated, the theoretical event rate surpasses 3 GeV/s. Given that most hardware can practically process only a few hundred MeV/s, it is advisable to maintain the event rate within manageable limits, particularly when higher rates are unnecessary. The sensor settings are referred to as Biases and are analog. They can be separated into three categories: contrast, filters, and pixel on/off time. The low-pass and high-pass filters are first-order filters. Each bias setting will be discussed separately in different subsections in detail.

3-9-1 Contrast Sensitivity Bias

The contrast sensitivity Bias setting alters the threshold voltage of the pixels. The `Bias_diff` settings controls the reference voltage used for comparison and are reset when an event is detected. However, it is recommended not to change them as it could damage the sensor. `Bias_Diff_on` governs pixel sensitivity to positive light changes, and `Bias_Diff_off` controls pixel sensitivity to negative light changes. Nevertheless, there are trade-offs that need to be considered when changing the `Bias_diff` settings, as they affect sensitivity and background rate depicted in Figure 3-17.

Decreasing contrast sensitivity thresholds
↓ Bias_diff_on ↓ Bias_diff_off*
Improves pixel sensitivity
Increases background rate
Can generate “stuck-at” (crazy) pixels
Improves pixel latency

(a) Effects of Decreasing Contrast Sensitivity Thresholds

Increasing contrast sensitivity thresholds
↑ Bias_diff_on ↑ Bias_diff_off*
Reduces pixel sensitivity
Decreases background rate
Reduces the number of generated events
Deteriorates pixel latency

(b) Effects of Increasing Contrast Sensitivity Thresholds

Figure 3-17: Comparison of Contrast Sensitivity Thresholds [41].

Decreasing the contrast sensitivity threshold is useful for applications with low-contrast and semi-transparent objects. This adjustment enhances the system’s ability to detect subtle

variations in intensity, making it well-suited for scenarios where the visual cues are not pronounced, Figure 3-18.

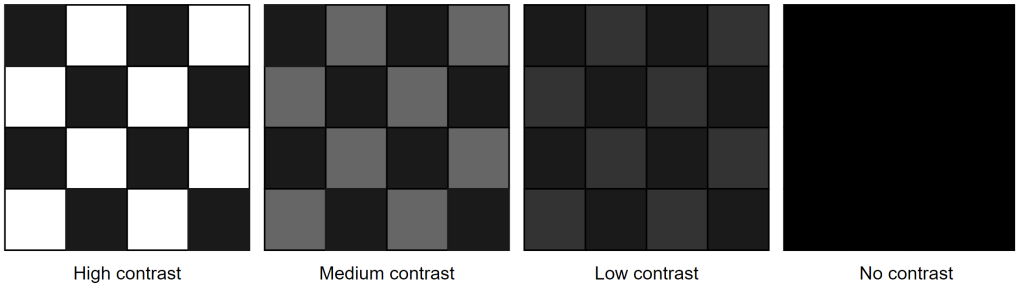


Figure 3-18: This illustration showcases several checkerboards with varying levels of contrast, ranging from high contrast to no contrast at all. A more sensitive pixel makes distinguishing intensity variations in low-contrast scenes easier and requires a smaller contrast sensitivity threshold compared to high-contrast scenes.

Conversely, increasing the contrast sensitivity threshold proves advantageous for applications characterized by high noise levels or scenes with frequent changes. For instance, in crowded street scenes or scenarios involving high-contrast objects, elevating the threshold helps filter out less significant changes, focusing on more pronounced alterations in the visual environment.

3-9-2 Low-pass and High-pass filters

The low-pass and high-pass filters control the low-pass and high-pass cut-off frequencies, respectively. The low-pass and high-pass filters are first-order filter and are illustrated in Figure 3-20. This allows for filtering specific frequencies or focusing on a particular frequency band. The low-pass filter is controlled using the `Bias_fo` setting, while the high-pass filter is adjusted using the `Bias_Hpf` setting. Adjusting the low-pass filter involves a trade-off between latency and noise, as depicted in Figure 3-19.

Decreasing low-pass filter
↓ Bias_fo*
Smooths the signal
Increases pixel latency
Latency also depends on light level
Bad for fast-moving objects
Removes high-frequencies
Reduces background rate, noise and event rate

(a) Effects of Decreasing low-pass filter

Increasing low-pass filter
↑ Bias_fo*
improves pixel latency Figure 3-21
• Latency also depends on light level
• Good for fast-moving objects
Expands range of observed high-frequencies
Increases background rate, noise and event rate

(b) Effects of Increasing low-pass filter

Figure 3-19: Comparison of Low-pass Filter Effects [41].

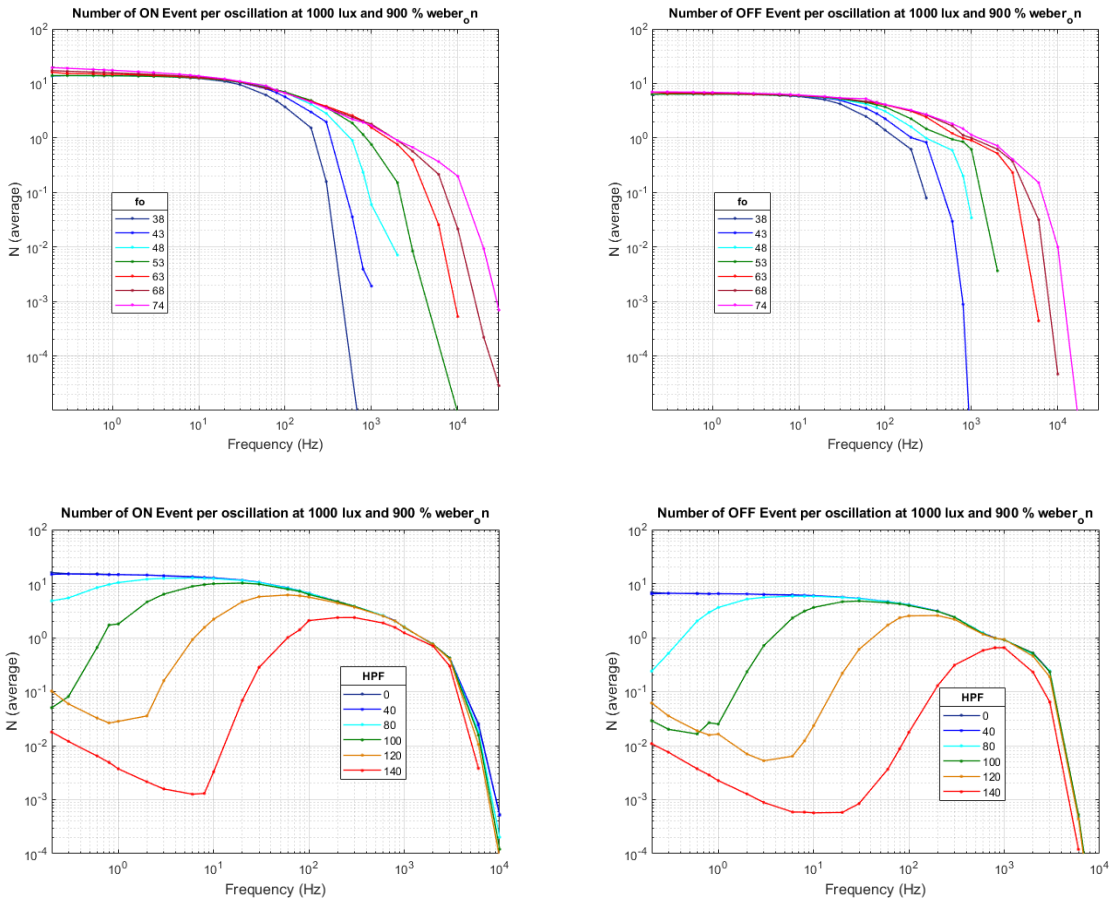


Figure 3-20: This figure shows the behavior the number of events generated for different `Bias_fo` and `Bias_Hpf` values for both the ON and Off pixels [42].

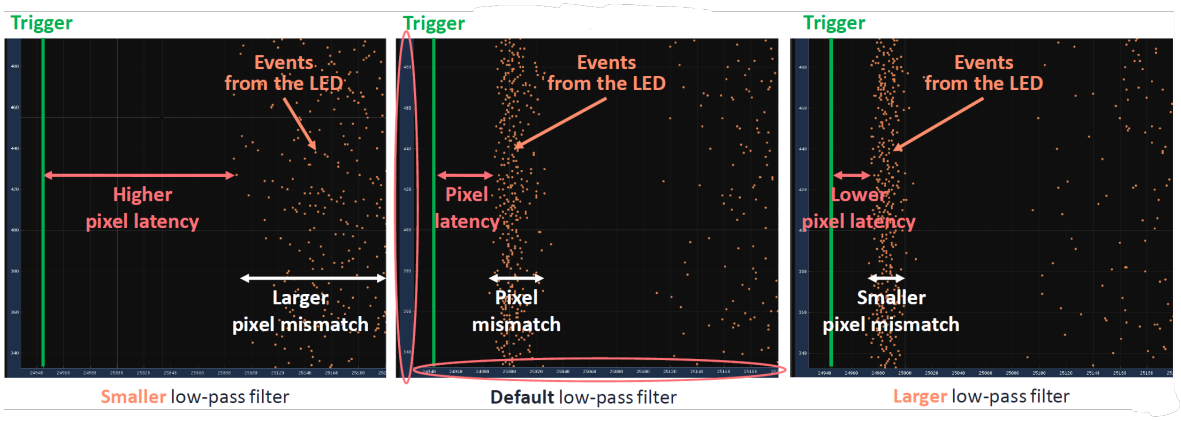


Figure 3-21: Comparison of the delay between laser trigger and pixel registration observed in a blinking LED, as documented by Prophesee [41]. The delay decreases as the low-pass filter is increased.

Decreasing the low-pass filter is useful for applications with light flickering or when there

is an interest in slow motion, where high-speed motion is unnecessary. On the other hand, increasing the low-pass filter is useful for tracking fast-moving objects or focusing on small objects with high magnification.

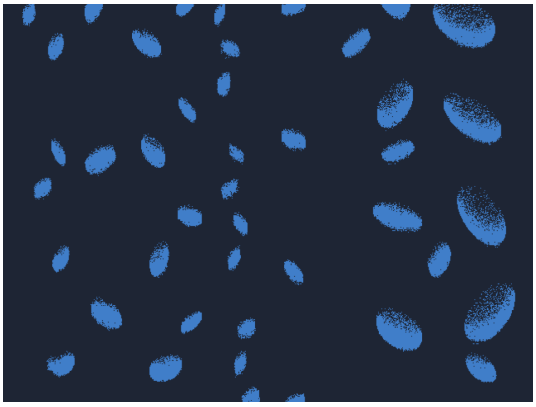


Figure 3-22: Example of falling particles [41].

Additionally, adjusting the high-pass filter also comes with drawbacks, involving the observation of low frequencies and the generation of events, as depicted in Figure 3-23.

Decreasing high-pass filter
Default Bias_hpf*
Preserves low-frequency signal
Preserves slow motion
Increases event rate (noise)

(a) Effects of Decreasing high-pass filter

Increasing high-pass filter
↑ Bias_hpf*
Removes low-frequency signal
Removes slow motion
Decreases event rate (noise)
Decreases sensibility at low light

(b) Effects of Increasing high-pass filter

Figure 3-23: Comparison of High-pass Filter Effects [41].

Decreasing the high-pass filter is used in applications where one is interested in slow motion or low-frequency signals, or in applications with low noise. On the other hand, increasing the high-pass filter is useful for applications where one is interested in only high frequencies or where low-frequency noise needs to be filtered out.



Figure 3-24: Pedestrians walking as an example of low velocity [41].

3-9-3 The Refractory Period bias

The refractory period is the duration during which a pixel does not detect a contrast change. This period can be considered as 'dead' time and is controlled using the `bias_refr` setting. Changing the refractory period affects pixel availability time and the generated events. It should be considered carefully not to exceed the maximum event rate that the sensor can handle. The trade-off between pixel availability and event rate is depicted in Figure 3-25.

Decreasing pixel refractory period
Bias_refr*
Improves pixel availability
Increases the number of events triggered by a large contrast change
Simplifies signal tracking

(a) Effects of Decreasing pixel refractory period

Increasing pixel refractory period
Bias_refr*
Reduces the number of events triggered by a large contrast change (and event rate)
Can be used to remove one polarity for periodic high-frequency signal

(b) Effects of Increasing pixel refractory period

Figure 3-25: Comparison of Pixel Refractory Period effects [41].

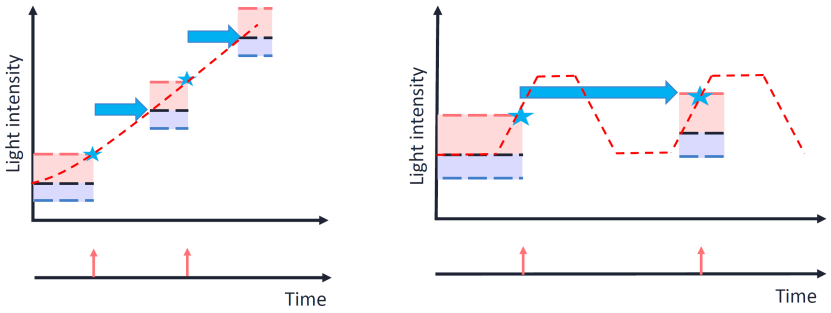


Figure 3-26: Visual representation of increasing the Refractory Period Bias [41].

Decreasing the refractory period is useful for applications where objects move at high speed, or high-speed tracking is required. Increasing the refractory period is useful for applications with a high event rate or where there is a need for a single polarity. For example, if a high object is being tracked, only the positive events are needed to obtain the position, and the negative polarity producing a trail is not required.

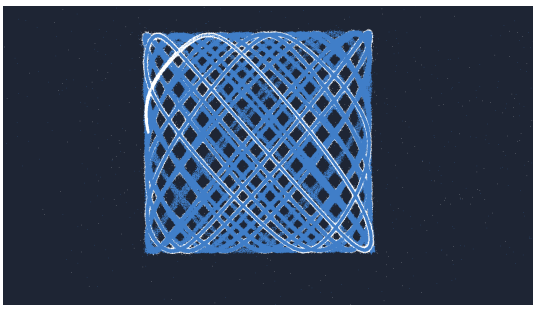


Figure 3-27: Example of a high-speed laser where the path is visible [41].

3-10 Bias selection

The following guide offers a concise step-by-step process for choosing the appropriate Biases. Often, there are trade-offs to consider when selecting bias settings. There is not a single correct approach, but systematically adjusting the settings enables users to better comprehend the situation and then refine the results through fine-tuning. This process often involves a significant amount of trial and error, going back and forth trying settings, analyzing a recording offline, and then applying the changes.

The first step is analyzing the environment that is being recorded, an example is depicted in Figure 3-28. It is clearly shown that much can be recorded but it is important to only focus on the most important subject and optimise for it to yield the best results as Bias tuning often involves making trade-offs.



Figure 3-28: A scene to showcase the various types of considerations to make when applying bias settings [41].

It is advisable to begin by adjusting the `bias_fo` or `bias_hpf` based on the velocity of the objects in the scene. Altering these filters to the required frequency band facilitates capturing the necessary information. Subsequently, adjust the `bias_diff_on` and `bias_diff_off` settings according to the contrast of the objects.

In specific scenarios, such as focusing on only one polarity, the refractory period needs to be changed, followed by the `bias_diff_on` and `bias_diff_off` settings. Each scene necessitates a distinct approach. The following flowchart outlines a systematic method for selecting Biases, with the aim of simplifying the process depicted in Figure 3-29. In Chapter 6, we will detail the application of these settings for our use-case.

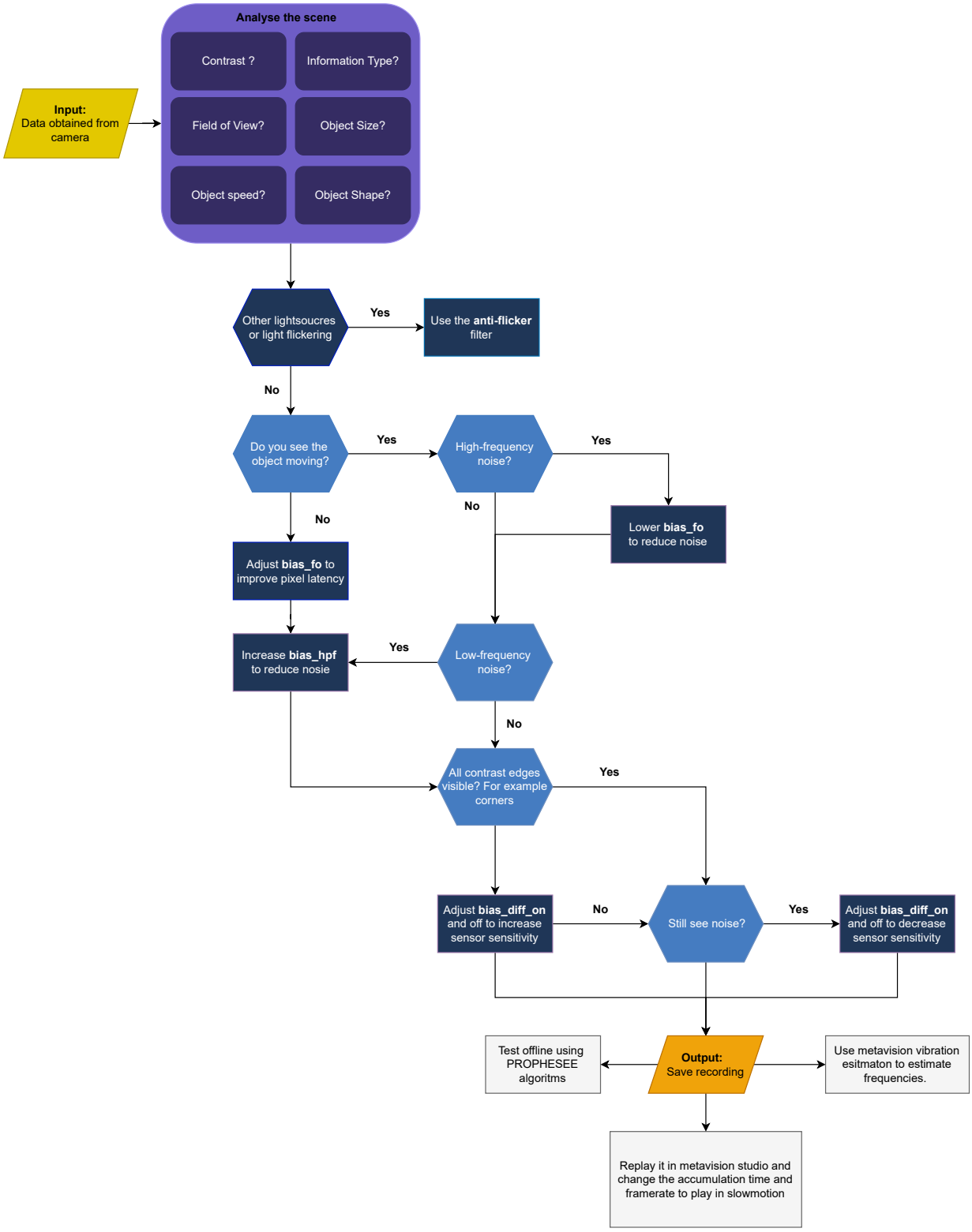


Figure 3-29: Flowchart for selecting Bias settings for different situations.

3-11 Advantages and disadvantages of Event-Based Cameras

Event-based cameras have several advantages and disadvantages compared to conventional cameras, which are listed Table 3-3 :

Table 3-3: Advantages and Disadvantages of Event-Based Cameras

Advantages	Disadvantages
• Low latency of ≤ 0.2 ms [41] • Minimal data bandwidth usage, enabling simultaneous tracking of multiple objects. • High temporal resolution. • Low power consumption (4.5 W). • Compact Size (108mm x 76mm x 45mm). • High dynamic range (140 dB compared to 60 dB in standard cameras). • Potential for integration with conventional vision systems [43, 44].	• Specialized data processing is required to stitch events together. • Few algorithms have been developed for event-based cameras. • Dependency on Prophesee's proprietary SDK (Software Development Kit). • Lack of color information. • To achieve the high-frame rates, certain parts of the sensor needs to be disabled.

Chapter 4

PHROPHESEE SDK

This chapter serves as an in-depth introduction to the Prophesee Software Development Kit (SDK), providing a comprehensive overview of its various modules, along with insights into their functionalities and use cases. The object tracking algorithm implementation will be discussed, while other beneficial algorithms will be covered in Chapter 10. Additionally, Prophesee offers Metavision Studio, a program featuring a graphical user interface (GUI), which will be further elaborated on in Appendix 9-4.

4-1 SDK

The PROPHESEE SDK consists of three modules. The first two modules are the Open and Advanced modules from PROPHESEE itself and the third comprises partner plugins developed by third-party partners. The open source Open module contains everything needed to make the camera hardware work. The Advanced module includes proprietary algorithms from PROPHESEE for use. The source code can only be accessed when the MetaVision SDK Pro license is purchased, which includes additional premium support. The algorithms from PROPHESEE are independent of the Open module and do not rely on PROPHESEE sensors and file formats. Therefore, as long as buffers and events are coded according to the API documentation, they should work with third-party event sensor hardware.

The Open and Advanced modules each encompass several submodules, each dedicated to fulfilling specific tasks required to execute certain algorithms, depicted in Table 9-3 and Table 9-2 respectively. Two main types of algorithms are available: Processor algorithms and Filter algorithms. Processor algorithms are assigned different tasks depending on the submodule. Conversely, filters take events as input and output only those events that meet certain predetermined conditions. While there are many algorithms available for use not all algorithms are useful. Therefore, only the algorithm that will be utilized will be discussed, and the remaining ones will be included in the Chapter 9. The most important algorithms will be presented in Section 4-3.

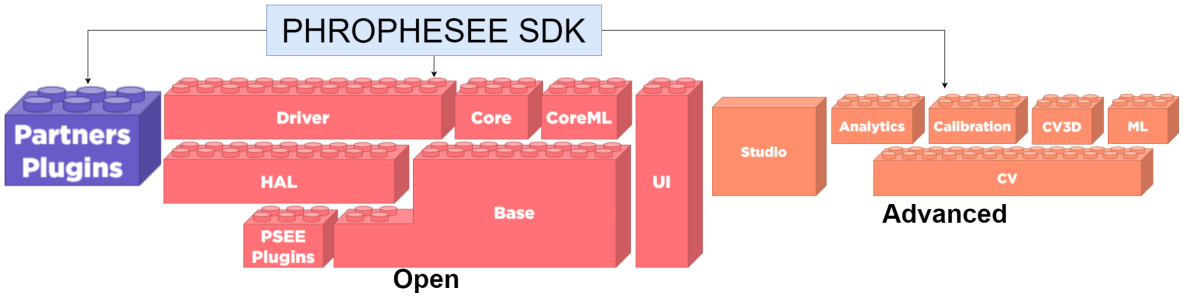
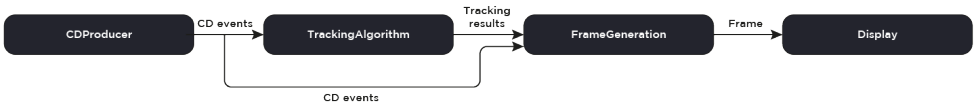


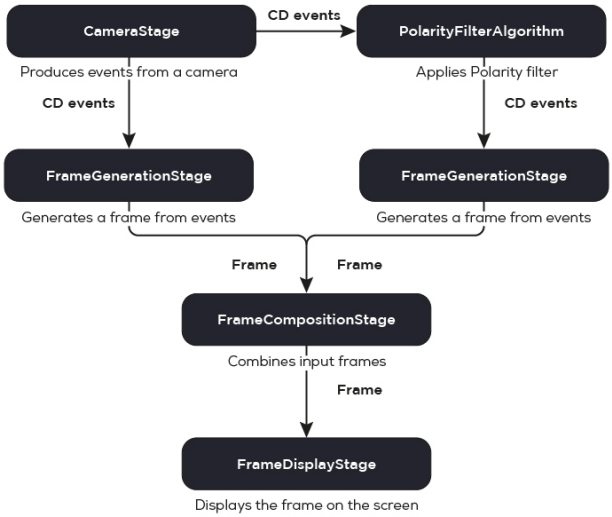
Figure 4-1: Prophesee SDK modules [45].

4-2 Pipelines

Algorithms are constructed using pipelines containing several blocks called stages. Each stage processes information from the previous stage and forwards it to the next stage. The benefit of using pipelines is easier interpretation of data transfer without unnecessary copies and allowing for multi-threading, improving performance. Stages of pipelines could transfer information to multiple other stages, allowing for parallel processing. Examples of a simple pipeline and a complex pipeline are depicted in Figure 4-2a and Figure 4-2b respectively.



(a) Example of a simple pipeline.



(b) Example of a complex pipeline.

Figure 4-2: Comparison of simple and complex pipelines [46].

4-3 Tracking Algortihm

The Prophesee SDK offers several algorithms tailored to different use cases. For tracking purposes, we focus on three algorithms: Generic Tracking [47], Spatter tracking [48], and 3D Edgelet Object Tracking [49]. Each algorithm comes with specific conditions and restrictions, as well as its own set of advantages and disadvantages. Ultimately, Spatter tracking has been chosen as the tracking algorithm for this study and will be discussed in detail in this chapter. Additionally, 3D Edgelet Object Tracking alongside the Vibration Estimation Algorithm will be detailed in Chapter 10.

4-3-1 Spatter Tracking

The Metavision Analytics module provides two tracking algorithms: the Generic Tracking Algorithm and the Spatter Tracking Algorithm. The Spatter Tracking Algorithm is a lighter implementation designed specifically for tracking non-colliding objects and is based on the Generic Tracking Algorithm. Both algorithms work by visualizing events and creating a bounding box around each tracked object, depicted in Figure 4-3. An ID is assigned to each tracked object, allowing for multiple objects to be tracked simultaneously. Spatter Tracking is ultimately chosen due to better performance and more available options for the current application.

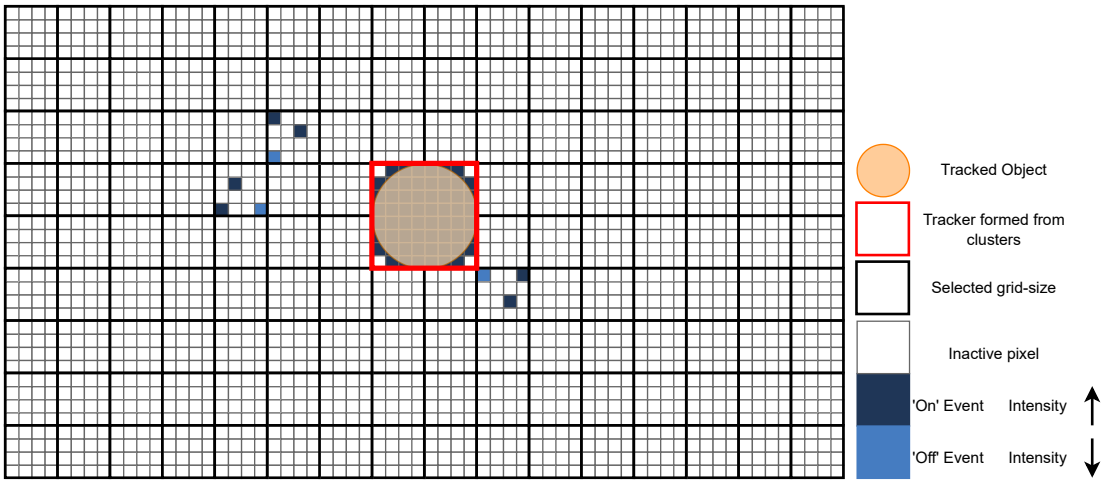


Figure 4-3: Spatter tracking highlighting multiple moving objects [48].

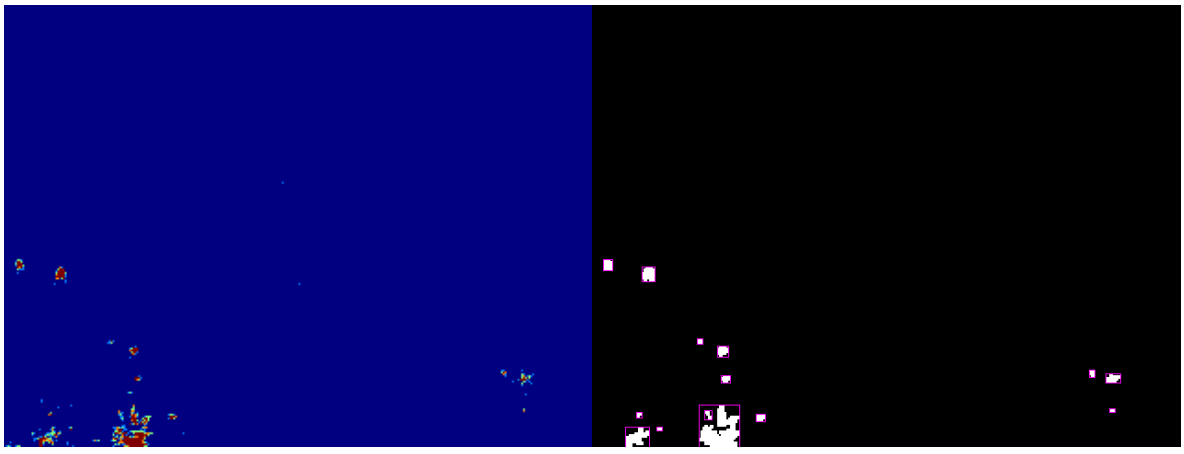
The tracking process consists of four main parts:

1. **Cluster Making**: The sensor is divided into grids defined by the user (in pixels), where input events that occur close together in both space and time are accumulated and grouped into clusters. Depending on the proximity of these grouped events, clusters are formed to outline an object. Figure 4-4a visualizes this process schematically. The light

blue pixels represent 'Off' events, the dark blue pixels represent 'On' events, and white pixels are inactive pixels that do not register events. The sensor grid size is determined by the user and is, in this case, 4x4 pixels large. The orange circle denotes the object captured by the event camera triggering 'On' pixels. When a sufficient number of 'On' pixels are close in both space and time, they form a cluster, and these clusters collectively define the object being tracked, as visualized with a big red outline. Active pixels failing to meet the cluster threshold are disregarded and not tracked. Figure 4-4b shows a heatmap of events, and Figure 4-4c shows bounding boxes; both images are provided by Prophesee [48].



(a) Visualisation of how clusters form an object.



(b) Eventcount heatmap. **(c)** Bounding box around detected clusters.

Figure 4-4: Cluster making.

2. **Data Association**: This step involves associating the correct data. The algorithm tries to match the current clusters with the existing trackers. If a match is found, the tracker is updated with the new data. If no match is found, the cluster is used to create a new tracker. Figure 4-5 visualizes the cluster matching process; the red clusters are

the previous clusters, and the green clusters represent the current clusters.

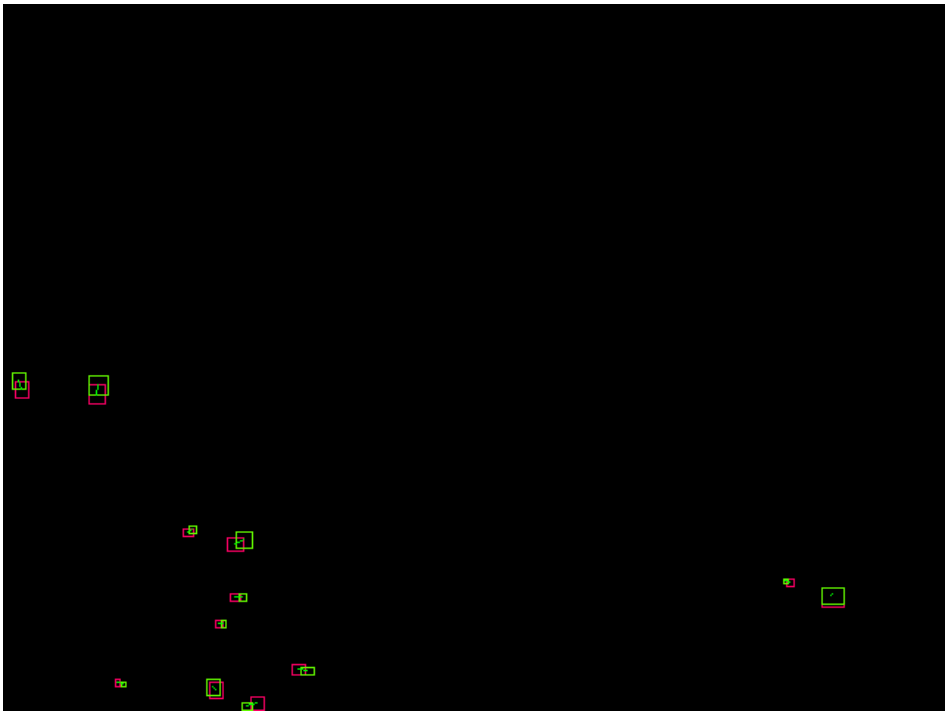


Figure 4-5: Cluster matching.

3. **Tracker Initialization**: The tracker is initialized. Each tracker maintains a state that includes the position, size, speed, and acceleration of the object. The state is updated at each step based on the new events that are associated with the tracker.

4. **Tracking**: The actual tracking of the objects occurs. The state of each tracker is used to predict the position of the object in the next step. The prediction is then used to associate new events with the tracker. The X and Y position of the object is determined by calculating the centroid of its bounding box.

4-3-2 Algorithm settings

Requirements for the algorithm to function effectively include:

- Object size between 10x10 en 300x300 pixels.
- Non colliding objects.
- Raw file format (.raw) to process offline.

The Spatter Tracking C++ code has several options that can be fine-tuned to achieve the desired tracking depending on the application. The options with the standard values are depicted in Table 4-1. The values of the parameters influence the performance of the tracking, both in accuracy and computational time.

Parameter	Default value	Description
stc-threshold	0	STC: filtering threshold delay (in us). 0 means no filtering will be applied
cell-width	7	Cell width used for clustering, in pixels
cell-height	7	Cell height used for clustering, in pixels
D [max-distance]	50	Maximum distance for clusters association, in pixels
a [activation-threshold]	1	Minimum number of events in a cell to consider it as active
min-track-time	1000	Minimum time necessary for a cluster to be tracked (us)
apply-filter	1	If true, then the cell activation threshold considers only one event per pixel
min-size	10	Minimum object size, in pixels
max-size	300	Maximum object size, in pixels
untracked-threshold	5	Maximum number of times a cluster can stay untracked before being removed
static-memory	1000000	Time after which the static constraint of detected clusters is reset (us)
max-size-variation	10	Maximum size variation (in pixels)
min-dist-moving-obj	0	Allowed match two clusters minimum distance (in pixel to consider a cluster moving (else, it is considered a static object which we don't want to track)
polarity-filter	FILTER_NEG	Polarity of events to filter out for the tracking ([NO_FILTER], FILTER_NEG, FILTER_POS , SEPARATE_FILTER]
processing-accumulation-time	5000	Processing accumulation time (in us)

Table 4-1: Processing parameters for the tracking algorithm. These Options are accessible via `metavision_spatter_tracking.exe -h` command in the terminal.

The most important parameter options will be highlighted below and divided into three distinct groups: event setting, object detection and object preservation:

Event settings:

- **processing-accumulation-time**: This is the 'frame rate' at which the tracking updates the position. The lower the accumulation, the fewer events are accumulated to form a cluster and create trackers. A lower accumulation time shows less detail. For fast objects a lower accumulation time is preferable for tracking.
- **polarity-filter**: Specifies the type of events to track. By default, it tracks only positive events, but also provides the option to track On and Off polarities separately.
- **stc-threshold**: The software counterpart of the Spatio-Temporal contrast filter, discussed earlier in Section 3-6-1, is designed to filter out isolated events by preserving only the second event of a burst.

- **cell-width and cell-height**: The argument is in pixels and allows the user to define the cell width and height used. The sensor grid is divided into fixed cells. Decreasing the width and height of cells enhances the accuracy of bounding box fitting around tracked objects, albeit with a corresponding increase in computational time.

Object detection:

Forming Clusters

These parameters are involved in the initial formation and detection of clusters:

- **activation-threshold**: The minimum number of events to consider a cluster active with 1 being the lowest argument value. The lower the value of this parameter, the more sensitive the tracking is and potentially more noise gets generated. This parameter works on the pixel level.

- **min-size and max-size**: The minimum and maximum size of the object specified in pixels. These parameters are useful when dealing with multiple objects and only a specific object needs to be tracked, or when there is a small cluster of noise that couldn't be filtered out. These parameters work on the cluster level.

Filtering Clusters

These parameters are involved in refining and filtering the clusters that have been formed:

- **apply-filter**: A true or false filter, when activated, ensures that pixels registering multiple events are counted as only one event per pixel for the `activation threshold`. This parameter works on the pixel level.

- **min-track-time**: The minimum time required for tracking a detected cluster, measured in microseconds. A lower minimum track time facilitates easy detection of clusters but increases the likelihood of tracking random 'wrong' clusters that are removed shortly afterward.

Object preservation:

- **untracked-threshold**: Maximum number of times a cluster can stay untracked before being removed. Increasing it allows the tracker to longer keep it object which is useful if the object sometimes 'disappears' during the tracking.

- **max-distance**: The maximum distance for cluster association. Increasing this distance makes recognizing the object shape easier, but it could lead to merging two separate objects into one if the distance is small enough.

- **max-size-variation**: The maximum allowable size variation for recognizing the same object when the tracker compares the previous and current positions.

Parameter	Description
–x arg	X coordinates of no-track zone center
–y arg	Y coordinates of no-track zone center
–radius arg	Radius of no-track zone
–inside arg	Inside flags of no-track zone: true/1 to filter inside of circle, false/0 to filter outside

Table 4-2: Region of Interest options.

In addition to the tracking parameters, it is possible to select a region of interest as depicted in Table 4-2. It is possible to filter within or outside this user-defined radius. After implementing the tracking algorithm, users can extract the object's position and export it into a comma-separated file (csv) for offline processing. This file includes the following information within its columns:

- **Timestamp (ts):** The first column represents the timestamp for the tracked objects.
- **Cluster ID:** The second column represents the ID of the tracked cluster.
- **Position (x, y):** The third and fourth columns represent the x and y coordinates of the tracked object's position in the image.
- **Size (width, height):** The fifth and sixth columns represent the width and height of the tracked object.

Chapter 5

Experimental Setup

Chapter 6 introduces the experimental setup, providing an overview that includes a schematic representation. It also explains the processing of data from the event-based camera.

5-1 Schematic of setup

As illustrated in Figure 5-1, the acoustic levitation setup consists of two piezoelectric phased arrays, each equipped with 64 transducers from Murata with model number MA40S4S [20]. This setup is commonly employed and utilized by other researchers [50, 22, 5]. It is connected to a System-on-Module (SOM) from Zynq, specifically a PicoZed 7015. The PicoZed contains a Field-programmable gate array (FPGA) that generates PWM signals within a voltage range of -30 to 30V to facilitate the formation of standing waves. The phase shift between the bottom and top transducer arrays is set to 180 degrees.
The PicoZed is connected to two computers via a network switch. The first computer, operating on Windows 10, is connected to the Prophesee EVK4 HD event camera. It executes the metavision Spatter tracking algorithm and transmits coordinates to the PicoZed. The second computer runs Simulink to directly access PicoZed functions and is responsible for modifying the ultrasound field produced by the transducers. The levitated object is a white Polyethylene microsphere, made by Cospheric [51], with a diameter ranging between 500 and 600 micrometers and a density of 1.35g/cc.

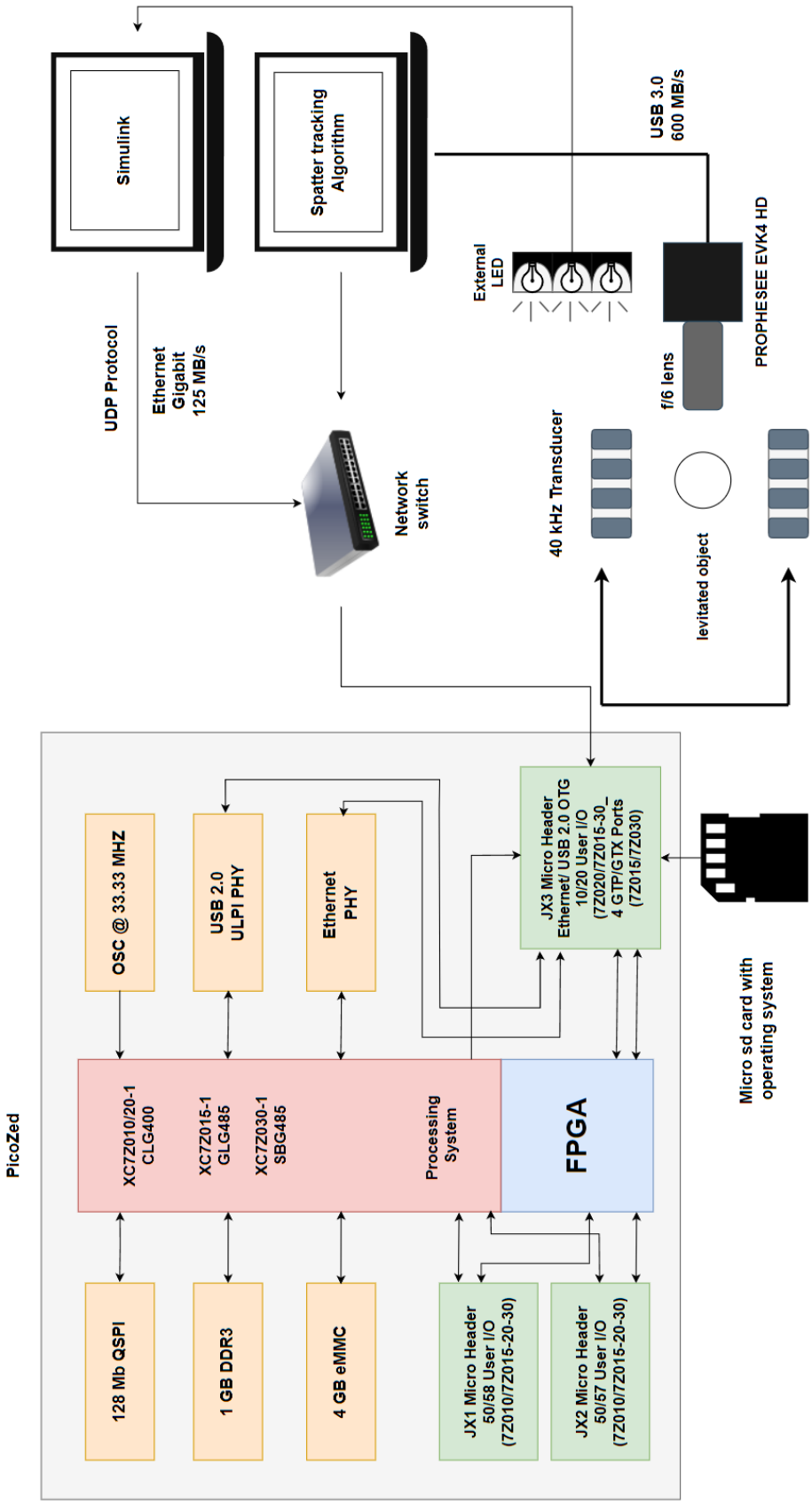


Figure 5-1: Schematic of the setup currently employed.

5-2 Processing real-time data

As shown in the block diagram (Figure 5-2), obtaining position data from the event-based camera involves the processing of raw data using PHROPHESEES proprietary SDK. This SDK includes several libraries alongside C++ code for the algorithms. This code can edited in an IDE of choice to send the object coordinates to Simulink.

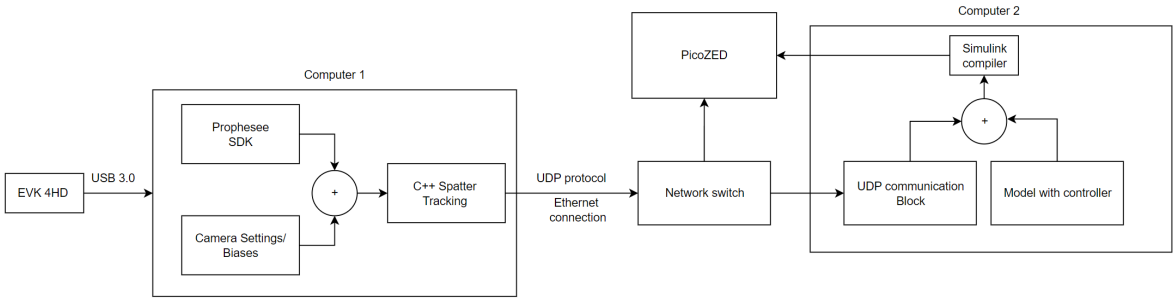


Figure 5-2: This figure depicts how the data from the camera is processed and compiled into code.

There are several protocols available for transmitting data between two computers (or the same device using localhost). In this scenario an Ethernet connection will be used to transmit data from the lab computer running a Windows 10 OS which is equipped with Simulink to the PicoZED. Two protocols will be compared: UDP (User Datagram Protocol) and TCP/IP (Transmission Control Protocol/Internet Protocol) [52]. Only the coordinate data from the Lab computer will be sent to the picoZED.

The main difference between these protocols is that UDP is a connection-less protocol. Connection-less refers to the fact that UDP does not establish a dedicated connection before transmitting data. This approach results in lower communication overhead and reduced latency, making it ideal for "real-time" applications. However, this comes with a trade-off as it affects reliability. UDP packets can be lost, arrive out of order, or even be duplicated during transmission. While UDP is achieves low latency, it is not entirely real-time and obtaining a true real-time system requires running the code directly on the PicoZED.

Conversely, TCP/IP is a connection-oriented protocol. It establishes a connection before data transmission and includes features like error checking to ensure reliable and ordered data delivery. This makes it more reliable and suitable for applications where data integrity and order are important, although it increases latency.

Due to limitations of the FPGA board, only UDP communication is supported, and as a result, it is implemented. Two important components are required for this setup to work. The first is the Boost.Asio C++ library, which is responsible for sending the data acquired from the C++ code to Simulink. The second component is the Simulink Embedded Coder Support Package for Xilinx Zynq Platform, which contains the UDP communication blocks. The exact implementation is detailed in Chapter 11.

5-3 Optics and Sensor setup

The EVK4HD uses a Sony IMX636 CMOS sensor with a resolution of 1280x720 pixels and a pixel size of $4.86 \times 4.86\,\mu$m, resulting in sensor dimensions of $3.5 \times 6.22\,$mm [53]. The camera has a CS lens mount which allows for interchangeable lenses. The current setup utilizes a telecentric lens from Edmund Optics [54], with an f-number of f/6 and a working distance of 139 mm. This telecentric lens captures objects without distortion, enabling precise measurements. The 0.75X magnification results in an image size of $8.3 \times 4.7\,$mm and a pixel size of $3.465\,\mu$m. The current setup also employs an external LED lightsource from IDT [55], allowing for continuous lightning or flashing at a user-defined frequency depending on field of view size. The maximum frequency for flashing is 5160 Hz. Using double exposure for 2 consequent flashes is also possible what it halves the maximum frequency to $2650 Hz$.

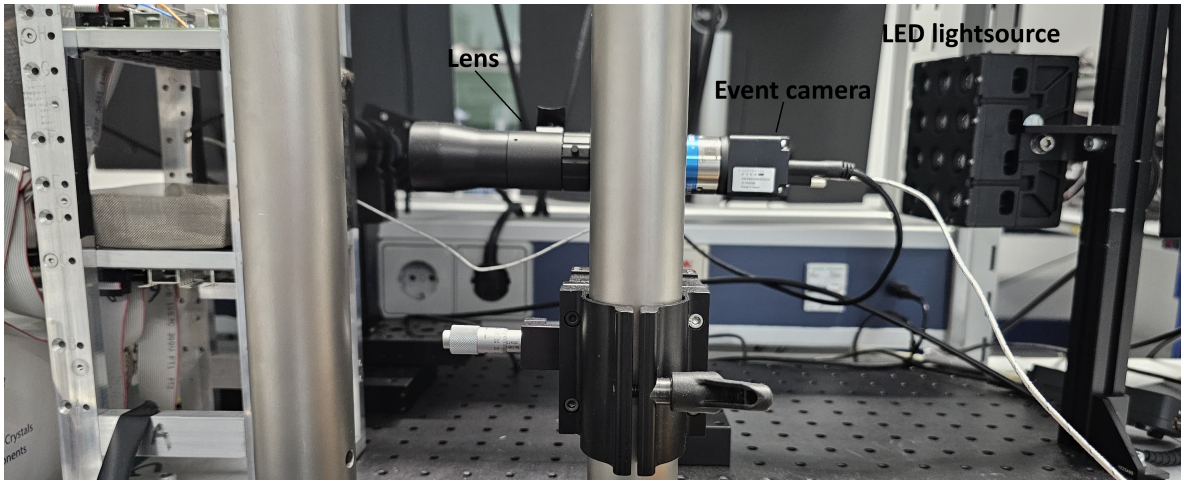


Figure 5-3: Photo of the optics used in the setup.

Chapter 6

Methodology

The previous chapters laid the theoretical groundwork for this study. Taking this knowledge into account, the methodology will be explained and divided into several steps. The first step is event camera tracking, which is divided into offline processing and a real-time part, as depicted by Figure 6-1. These steps are crucial for achieving the desired real-time tracking of the object and depend on a combination of light settings, Bias settings, and tracking algorithm settings. After achieving sufficient tracking, the process continues with applying system identification strategies and tuning the controller for the obtained plant of the system. Many observations were made that influenced the process, necessitating the need for back-and-forth iterations to achieve the desired results. The event camera Bias and tracking settings are novel and are specifically developed for the current use case. The values used for the all settings are detailed in Chapter 13.

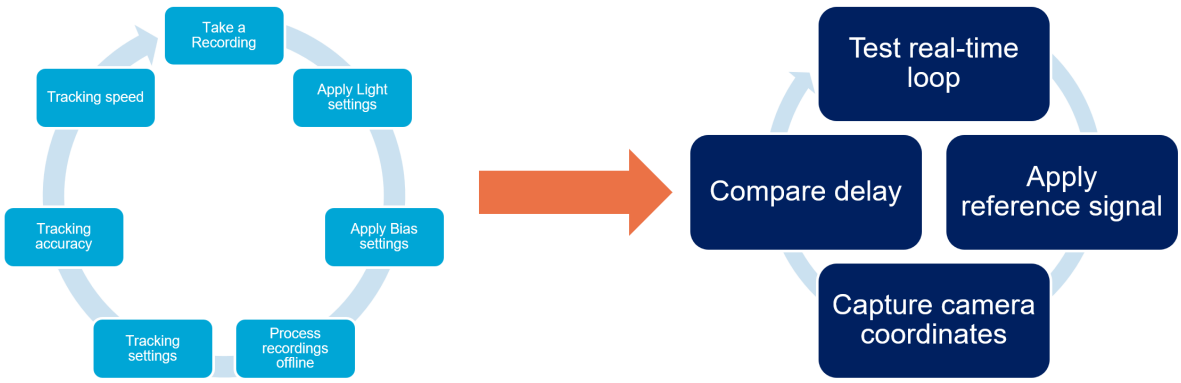


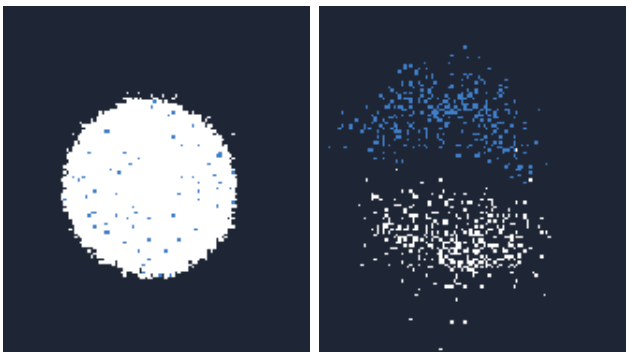
Figure 6-1: Methodology steps for the event camera and tracking: On the left, offline processing; on the right, the real-time loop.

6-1 Offline processing

The initial step involves testing the tracking settings offline by capturing recordings of the object. This allows for a detailed review, including slow-motion playback, to identify details missed during live recordings that are happening too fast to observe. Replaying these recordings aids in refining the tracking and Bias parameters. Furthermore, offline processing establishes a controlled environment, enabling the application of various Bias and tracking settings to the same recording for a more straightforward comparison

6-1-1 Light settings

The environmental light was altered by using an external LED light source in the setup. Various configurations were tried, such as: not using the light source and relying on the environmental ambient light, turning the light source on continuously, and using the flash function at different frequencies with various exposure settings. This led to two situations: one where a flash could be used to illuminate the object to generate events at a specific chosen framerate, and another where the light source was turned on continuously to better illuminate the object. The reasoning behind using a flash is that event cameras only generate events when the light intensity threshold is exceeded, and this differs for different velocities. Stationary or slow objects do not generate changes in intensity and are thus 'invisible' to event cameras. Using a flash allowed the entire object to be illuminated at the moment of the flash, even when the object was not moving or moving at a very low velocity, illustrated Figure 6-6a. However, this came with the disadvantage of losing information between the flashes. At the moment of the flash, a burst of events was generated compared to the situation with continuous illumination, where events were only generated when there was movement. Continuous illumination resulted in only a part of the object being illuminated, creating two half-moons: one with positive events in the direction of movement and the other with negative events at the tail of the object , depicted in Figure 6-2b.



(a) Using a flash **(b)** Constant illumination

Figure 6-2: Comparison of an object illuminated with a flash at 1000 Hz compared to using constant illumination while levitating in a node.

6-1-2 Bias settings

After obtaining a basic understanding, adjustments were made to the Bias settings, and their effects were observed. Both situations required different sets of Bias settings. The process of applying the Bias settings was iterative: recordings were taken, viewed offline, and then the Bias settings were adjusted accordingly.

Flash

While the use of a flash initially showed promise for effective tracking at low velocities, it unfortunately led to a significant delay (5+ ms) in tracking. This delay was caused by the high event rates resulting from the burst of events generated when the flash illuminated the entire object. In addition, the flash needs to be synced to the object movement, which adds complexity to the setup. As the number of events generated could not be further reduced even when Bias settings were applied, making improvements in future prospects difficult, the use of flash was ultimately abandoned in favor of continuous lighting. However, during this process, valuable information was gathered that could be useful for other applications and is for this reason linked in Section 13-1.

Continuous light

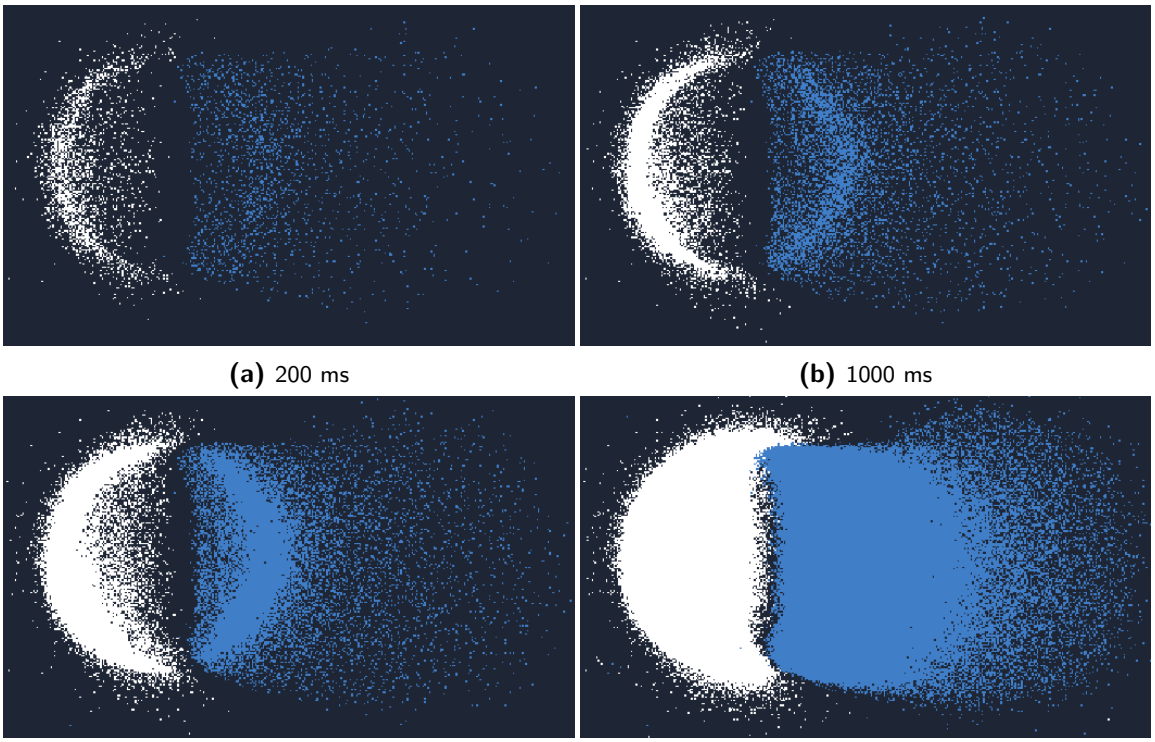
Several Bias settings were tested with the continuous light; however, changing the Bias settings was not necessary because the continuous stream of events did not create sudden bursts of noise, as in the flash scenario. However, an attempt was made to obtain better images for stationary and low-velocity situations by decreasing the `Bias_refr` to detect events more easily. Additionally, the `Bias_diff_on` and `Bias_diff_off` were lowered to make light intensity detection more sensitive to small contrast changes. Unfortunately, those settings did not seem to introduce any noticeable improvements, so the Bias settings were left on default.

6-1-3 Tracking settings

After obtaining a good recording, the tracking settings were tested offline. It is important to create the best possible recording, as this influences tracking effectiveness. Analog Bias settings could reduce the data rate, improving tracking performance in real-time. It is important to create enough events to be able to track the object without saturating the arbiter. Additionally, applying numerous tracking settings to compensate could negatively affect performance. The goal is to first establish tracking that works with a delay fitting the requirements. This can then be further improved and optimized to enhance accuracy.

The initial step involves understanding the impact of each tracking parameter and prioritizing the most crucial settings to identify any potential bottlenecks. These bottlenecks could adversely affect real-time tracking performance, potentially leading to slower processing times. The first setting adjusted was the `accumulation-time`, setting its value to 1000. This value was chosen specifically for its role in determining the sampling rate of the X and Z coordinates. The objective is to track the object's position at a rate of at least 1000 times per second, which fulfills the requirement. In theory, a higher `accumulation-time` makes tracking less

computationally intensive. However, due to the fast movement of the object, a lot of detail is lost. This loss of detail can lead to poorer tracking performance as too many events are accumulated, as illustrated in Figure 6-3. As a result, the algorithm struggles to identify a distinct shape to track. An `accumulation-time` greater than 1000 fails to meet our requirements, while an `accumulation-time` less than 1000, although it slows down performance, has the potential to reveal more detailed motion. The `accumulation-time` value strikes a balance between capturing enough events to distinguish the fast movement of the object, enabling tracking by the algorithm, and not being too low to introduce delays in real-time due to computational requirements.



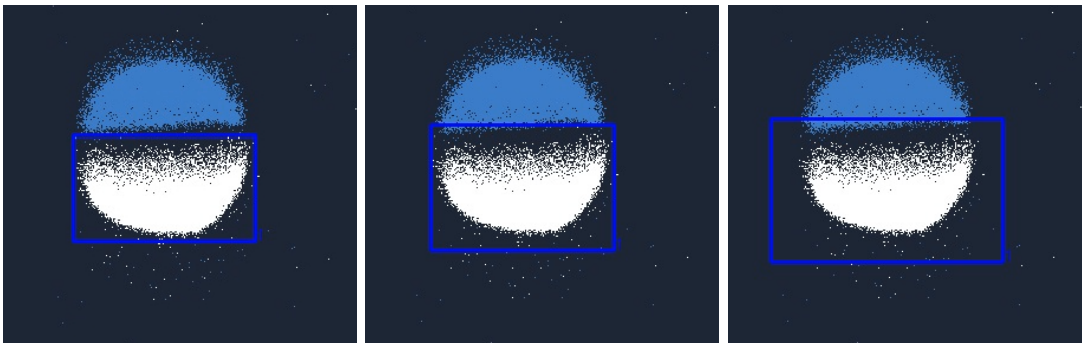
(a) 200 ms **(b)** 1000 ms

(c) 2000 ms **(d)** 5000 ms

Figure 6-3: The effect of different Accumulation times on a fast moving object.

The next setting applied was the `cell-width` and `cell-height`. The default value of 7x7 was chosen. A smaller grid size proved to be a bottleneck in tracking, and depending on the accumulation time used, it would slow down tracking performance. Despite this, it resulted in more accurate tracking, as the resolution of the bounding box was increased, as depicted in Figure 6-4. Additionally, a smaller grid size is more susceptible to noise fluctuations and random noisy events, whereas a larger grid size remains more consistent at the cost of lower resolution, potentially missing subtle movements. The grid size value is also relative to the object size being tracked, as larger objects require a larger relative grid size.

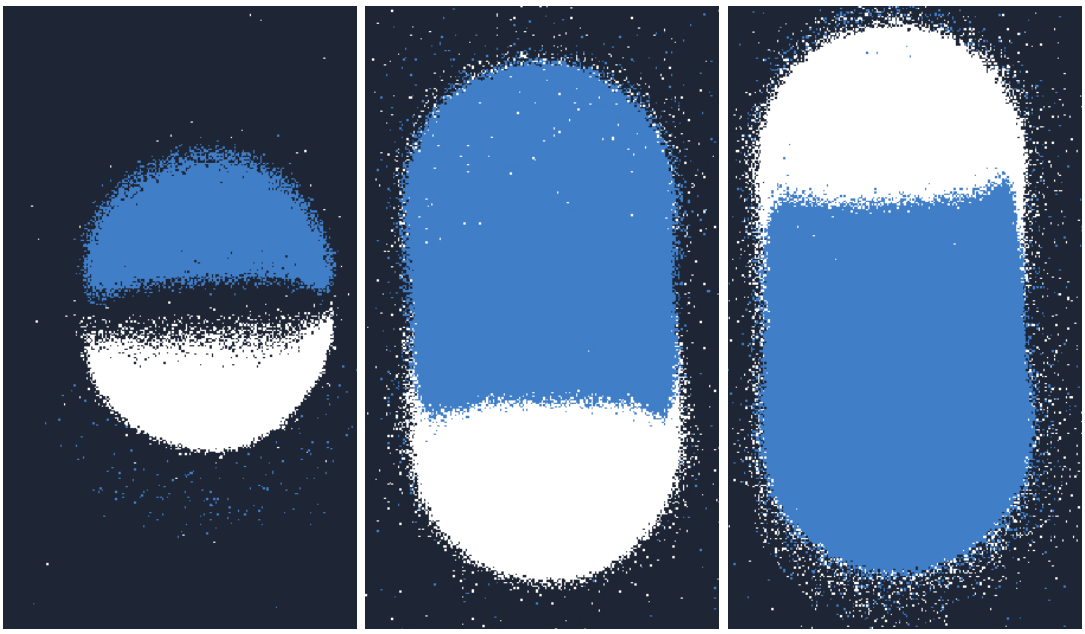
Given the relatively narrow field of view, a standard movement pattern has been established to consistently track the object. This pattern simulates realistic use cases, such as transporting objects from one point to another. At the same time, it exposes potential weaknesses of the tracker in specific worst-case scenarios. The object, which levitates at the center of the sensor, oscillates in a sinusoidal pattern. As an example, this oscillation has a frequency of



(a) Grid: 1x1 **(b)** Grid: 7x7 **(c)** Grid: 15x15

Figure 6-4: Comparison of different Grid-sizes.

10 Hz and an amplitude of 0.5 mm, as shown in Figure 6-5. The zero position, located in the middle of the sensor, is depicted in Figure 6-5a. Figure 6-5b demonstrates the object's movement towards the lower end. An accumulation time of 20 ms was deliberately chosen for this example to illustrate the movement history of the object. Following this, the object moves back to the zero position and then towards the upper end, as depicted in Figure 6-5c.



(a) Zero begin position. **(b)** Lower end position. **(c)** Upper end position.

Figure 6-5: Sinusoidal movement pattern of object.

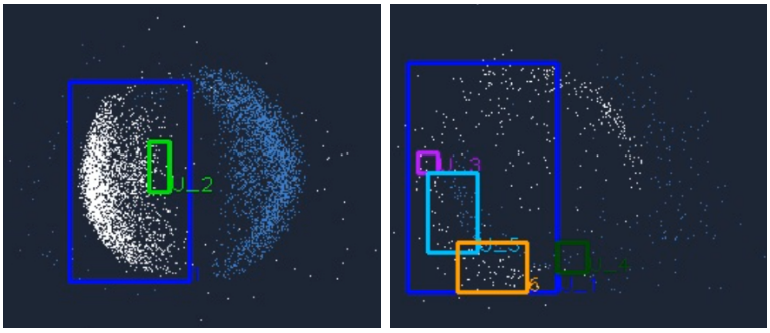
At the maximum and minimum points, several interesting phenomena occur. First, as the object moves along its path, its size and velocity decrease until it slows down and reaches the end of its path. Its velocity drops to zero before changing direction and doing the opposite. The number of events generated depends on the velocity of the object. Occasionally, at lower velocities, fewer events are generated, causing the tracker to lose sight of the object. To mitigate this, several tracking settings were applied, each to combat a different problem.

To prevent the tracker from losing sight of the object, the **minimal-track-time** of the tracker was decreased, and the **untracked-threshold** was increased to keep the object in memory until it returned to its previous size. Additionally, the **max-distance** was increased to improve cluster association during the back-and-forth periodic movement. The values are found in Table 6-1

Parameter	Value	Value Used
minimal-track-time	$\leq$ accumulation-time	1000
untracked-threshold	$\frac{2 \cdot \text{accumulation-time}}{T}$	50
max-distance	Amplitude of the object's periodic movement	400

Table 6-1: The tracker parameters are defined where T is the period of the sinusoidal input signal in milliseconds.

Next, the **min-size** parameter, with the selected value of 80, is utilized to avoid tracking smaller clusters as objects. It is recommended to set this parameter to roughly half the physical size of the object in pixels, in this case, 160 pixels. This is because only a portion of the object, which is white, is tracked. The comparison in 6-6 demonstrates instances where smaller clusters are incorrectly identified as objects. By increasing the **min-size** parameter, this can be mitigated. As the object moves, its size and velocity may vary, which could impact the tracking process. Therefore, adjustments to the **min-size** setting may be necessary.



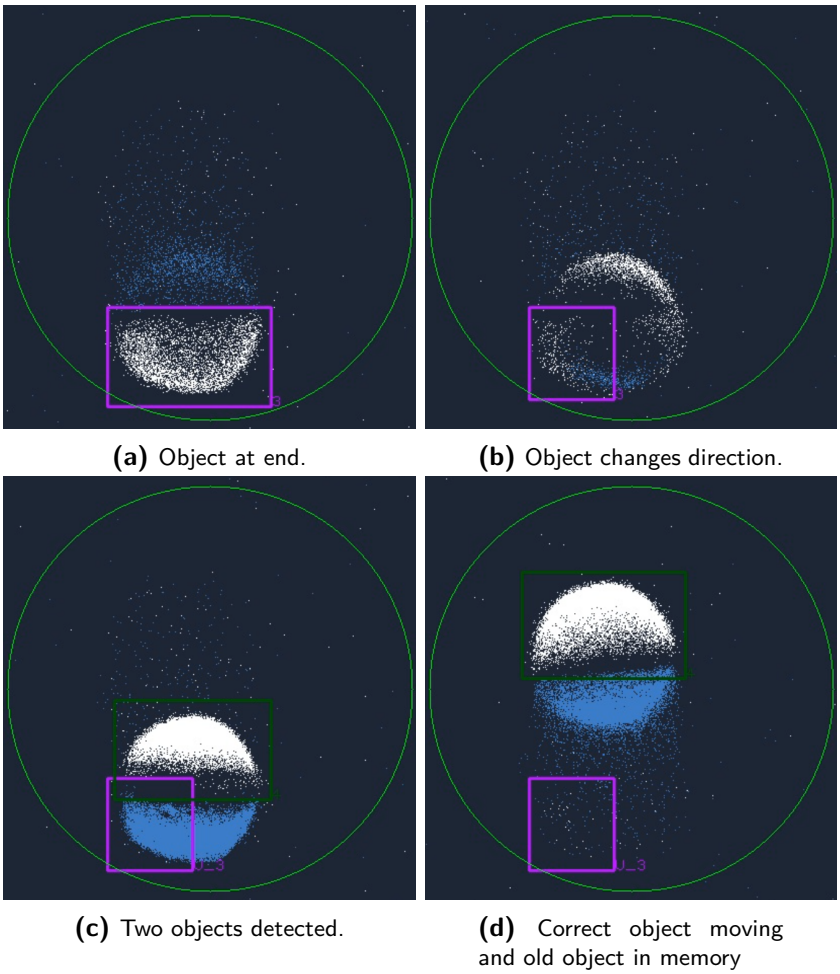
(a) Example 1 of wrong clusters.

(b) Example 2 of wrong clusters.

Figure 6-6: Examples where several smaller clusters are being tracked as objects, while the blue tracker is only the tracker of interest.

The **activation-threshold** is used to enhance tracking performance and is adjusted based on the velocity. Typically, it starts with a default value of 10. In this case, a value of 7 has been chosen. The **activation-threshold** is decreased for lower velocities to increase the tracker's sensitivity. Conversely, at higher velocities, the **activation-threshold** is increased to handle the additional events generated. Additionally, the **apply-filter** is activated to consider only one event per pixel. The **activation-threshold** scales linearly with the event rate. For instance, if the accumulation time is halved, the activation threshold is also halved. If the grid size increases, leading to more events, the activation threshold is raised to match the

relative increase in tracker area size. However, sometimes at higher velocities, two trackers appear at the same time, as depicted in Figure 6-7. The old tracker contains the old position in memory, and the new tracker follows the object perfectly. This happens at the lower end position when a group of events that should clear out is exactly as large as the **`min-size`**. This problem was solved by modifying the code to only accept the most recent tracker.



(a) Object at end.

(b) Object changes direction.

(c) Two objects detected.

(d) Correct object moving and old object in memory

Figure 6-7: This set of images illustrates a sphere moving in a sinusoidal pattern, where occasionally the tracker detects two objects when the sphere reaches zero velocity and changes direction.

Lastly, a **`Region of Interest`** was implemented, depicted by the green circle outline in Figure 6-7. This region of interest is defined by an x-value of 640 pixels, a y-value of 360 pixels, and a radius of 250 pixels, which correspond to the center of the sensor where the object is located. It's important to note that the sensor's pixel location coordinate system originates from the top left. The reasoning behind using a region of interest is to filter out events generated outside the object's movement pattern, thereby enhancing both tracking accuracy and speed. However, this approach led to only minor improvements, primarily due to the already low noise levels in the regions not of interest.

6-1-4 Tracking accuracy

The validation of algorithms is crucial to ensure their effectiveness and reliability in practical applications. In this section, we aim to validate the tracking performance of the PROPHESEE spatter tracking algorithm. Assessing the accuracy of the tracking is not straightforward. The ideal method is to compare the tracking with another calibrated camera; however, this is challenging due to the complexity of the implementation, as it would require significant time to set up and calibrate the tracking for this camera.

Two approaches were used to assess the tracking accuracy. The first method used was to use the input signal from the transducer as a reference and compare it to the output signal measured with the event camera. However, this method is less dependable due to disturbances and an imperfect sound field, making it difficult to distinguish between disturbances and camera noise/measurement errors. As this sound field will be used in the final application, this test tells more about the accuracy of the whole system and not necessarily the camera in isolation.

The second approach taken was to compare the mean of the coordinates for several scenarios to assess the consistency of the tracking. The object was recorded at different amplitudes and this was was done for various grid sizes to assess the influence of grid size on tracking accuracy, as a smaller grid size increases the resolution of the object tracker. The benefit of this method is that it is possible to compare the different grid sizes using the same data set for each amplitude and compare the tracking.

6-1-5 Tracking speed

To evaluate real-time tracking, the viewer was disabled, as it impacted performance by slowing down the recording to 1000 fps, thereby extending the duration of the real recording. With the viewer turned off, the tracking settings were applied and compared the duration of the recording to the time taken by the algorithm to complete. If the algorithm's processing time was shorter than the duration of the recording, it theoretically indicated that it was fast enough for real-time tracking. However, it's important to build in a margin of safety.

It was assumed that tracking settings were fast enough for real-time processing when the processing time for an offline recording was no more than 60% of the time required to process the original recording. This number is a buffer to take into account any delay. In the real-time loop, the event data is sent over a USB connection instead of being already available on the storage device. In addition, the coordinates are sent sequentially over a network connection, without buffering or parallel processing capabilities, introducing delays.

6-2 Real-time loop

After applying the offline processing and obtaining tracking setting parameters, the next step involved testing the system in a real-time closed loop. This step is crucial for determining the delays added by the separate components in the loop and striving to minimize them where possible. Some delays are always present, such as the mechanical delay of the object or the delay of the sound waves, and cannot be practically changed. However, other delays depend on the tracking settings and the network connection.

6-2-1 Reference signal

To measure the delay in a real-time application, a reference signal was sent via the PicoZed. The signal was a 1-minute-long sine wave with frequencies of 12 and 40 Hz and amplitudes of 0.5 mm. This reference signal alters the sound fields and directs the object to follow a predetermined path. There is no universal setting for all frequencies, so the tracking was optimized for observing the eigenfrequency of the object. The amplitude of the signal is selected to achieve a displacement within the field of view of the sensor, with the object moving back and forth at the frequency of the sine wave. Additionally, the amplitude is kept moderate to avoid introducing non-linear behavior or ejecting the object from the trap location. Following that, the tracking algorithm is applied, sending the object coordinates in real-time over UDP back to the PicoZed, where they are logged. This process generates several plots displaying the input (reference) and output signal (camera coordinates), facilitating offline examination later.

6-2-2 Delay comparison

The MATLAB Find Delay [56] function was utilized to compare the time delay between the reference input and the measured output signal. This function estimates the delay between two signals via cross-correlation and provides the delay in the number of samples, which can be converted back to milliseconds. However, this method has the disadvantage that if the delay is less than one sample in time, it rounds up to the nearest integer.

6-3 System identification

This section outlines the steps applied for system identification and how, with the help of the tracking algorithm, a Bode diagram describing the dynamics is obtained.

First, a controlled environment needed to be created. Turning on the ultrasonic transducers and the external LED generates heat, which affects the levitated object. As the temperature increases, the eigenfrequency of the object decreases. The vertical (Z) and horizontal (X) directions have different eigenfrequencies. The resonance frequency for the Z direction is approximately 40 Hz, while for the X direction, it is 14 Hz. These resonance frequencies shift slightly depending on the temperature during the measurement. Before applying a multi-sine signal, the levitator was turned on for a few minutes, and the temperature was measured with a thermocouple connected to a multimeter until it reached a steady state. The external LED was only turned on during the measurement of the multi-sine signal.

Since a multi-sine signal contains multiple frequencies and previous testing was performed at a single frequency, a recording of the multi-sine was made to observe the effects of the previously obtained tracking settings. These settings were slightly modified to prepare for real-time system identification. The final values used for the multi-sine are provided in Table 13-1.

Axis	Cell Size	Accu-mulation	Min Track Time	Apply Filter	a	D	Min Size	Untracked Threshold	ROI
Z	7x7	1000	100	1	5	400	80	12	220
X	7x7	1000	100	1	8	400	80	35	220

Table 6-2: Final tracking values used for the multi-sine for the Z and X directions.

The multi-sine signals used utilize excitation frequencies from 1 to 60 Hz, with the exact values for the Z and X direction depicted in Table 13-2. Higher frequencies resulted in delays, and 60 Hz is enough to capture the most important dynamics of the object. Selecting the right amplitude is crucial: a small displacement does not capture the dynamics of the object, while too large a displacement introduces nonlinear effects. The amplitude must also be large enough to be bigger than the noise floor level. The length of the multi-sine signal also influences the accuracy of the obtained measurement.

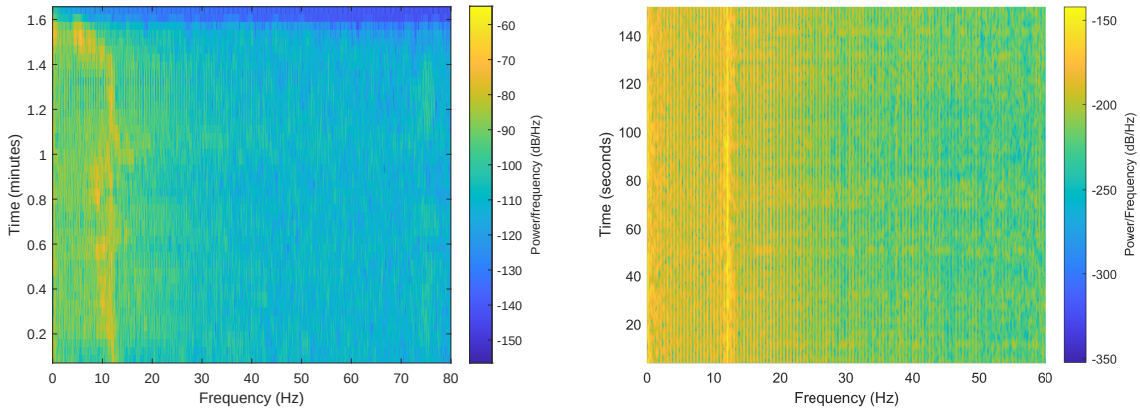
Parameter	Value Z	Value X
Amplitude input signal	1000 um	1000 um
Total time input signal	120 s	160 s
Frequency resolution	0.5 Hz	0.5 Hz
Sample frequency	1000 Hz	1000 Hz
lower bound frequency range	0.5 Hz	0.5 Hz
upper bound frequency range	60 Hz	60 Hz

Table 6-3: Multi-sine signal values used for the X and Z directions.

To verify the results of the multi-sine signal, several plots were created and compared:

In Chapter 7, the final values of these plots are presented. For clarity and ease of understanding, some of these plots are also included in this section. They serve as visual aids and provide examples of where to direct attention.

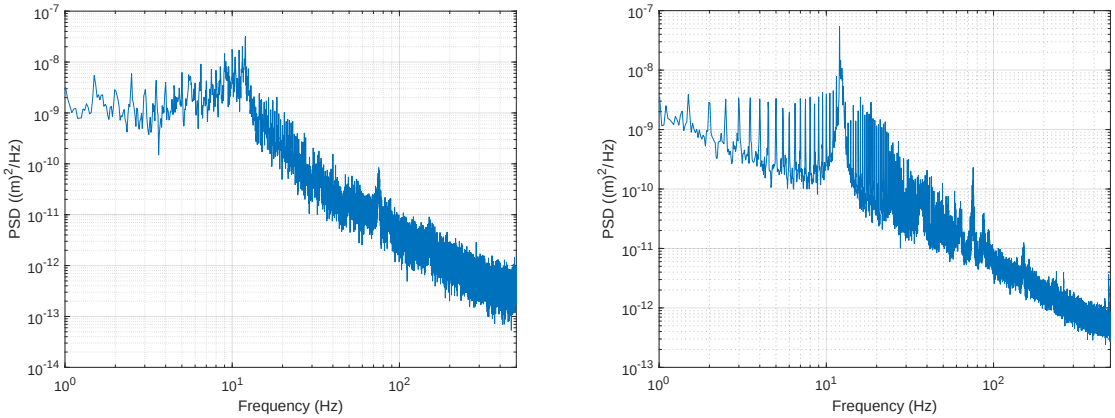
1. The first plot was a Spectrogram to show how the frequencies are distributed over time. At a glance, the eigenfrequency and excitation frequencies can be identified. As temperature influences the measurement, any temperature fluctuations become apparent. The vertical dominant eigenfrequency line shifts horizontally instead of being a straight line at one frequency. Figure 6-8a shows a Spectrogram where excitation frequencies are not visible and where the frequency shits over time, compared to Figure 6-8b where there are almost no temperature fluctuations visible and at the same time some excitation frequencies are visible.



(a) Wrong experiment Spectogram. **(b)** Correct Spectogram.

Figure 6-8: Examples of a Spectogram.

2. The second plot was the Power Spectral Density (PSD) using Welch's overlapped segment averaging. The PSD describes how the power of the signal is distributed over different frequencies, making it easy to observe the individual excitation frequencies of the multi-sine signal. This is especially useful as it gives insight into how the plant (system) reacts to an excitation signal. In addition, any higher-order harmonics become visible. Figure 6-9a shows a PSD where the resonance frequency is barely visible with a very low power, compared to Figure 6-9b where the eigenfrequency and individual excitation frequencies are visible.



(a) Wrong experiment PSD. **(b)** Correct PSD.

Figure 6-9: Examples of a PSD.

3. The third plot was a time-domain plot showing the output of the displacement over time. This was useful for verifying if the tracking worked throughout the entire signal and if there were periods where the tracking lost the object. This can be combined by overlaying the output signal over the input signal to identify any delays.

4. The final step involved creating Bode diagrams for both the vertical and horizontal directions. The frequency response depicted in these Bode diagrams provides valuable insights into the system dynamics. This system should resemble the behavior of a second order system, Figure 6-10 provides an example of an idealised system. The plant model derived from the Bode diagrams is necessary for designing a controller, which will be discussed in Section 6-4.

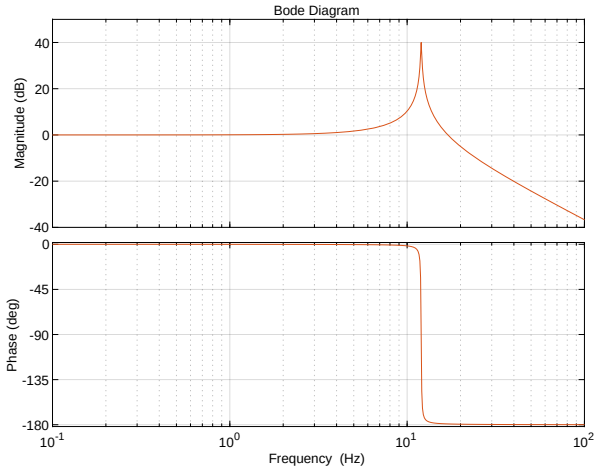


Figure 6-10: Example of an idealized second-order system.

6-4 Control

This section will outline the process of creating a closed-loop control system. The objective is to demonstrate that a controller can function effectively with an event camera. There is no perfect one-size-fits-all controller, and for this reason, several controllers can be created depending on system requirements. The controller designed was optimised for an improved step response.

The first step involved deriving the transfer function from the Bode diagrams and creating a plant. This plant can be simulated in Matlab to observe the effects of specific control actions. The extent to which we can increase the bandwidth depends on the phase margin. The phase margin in the X-direction around the eigenfrequency of 12 Hz is higher, which facilitates a higher bandwidth improvement for the X direction compared to the Z direction, where the resonance frequency resides at 38 Hz. However, it's still feasible to apply control in both directions, but initial improvement will be larger for the X-direction. For this reason, the controller will be only made for the X direction.

To create a controller and tune it for this system, MATLAB was used. A transfer function of a second-order system with an eigenfrequency of 12 Hz and a Q factor of 100 was used, depicted in Figure 6-11. Two transfer functions were created using those values, one as a 'perfect system' without delay and the other as a system with a 3.5 millisecond delay. This is convenient as it allows us to easily show the effects of delays on controller performance and how much performance can be theoretically obtained when the delays are reduced.

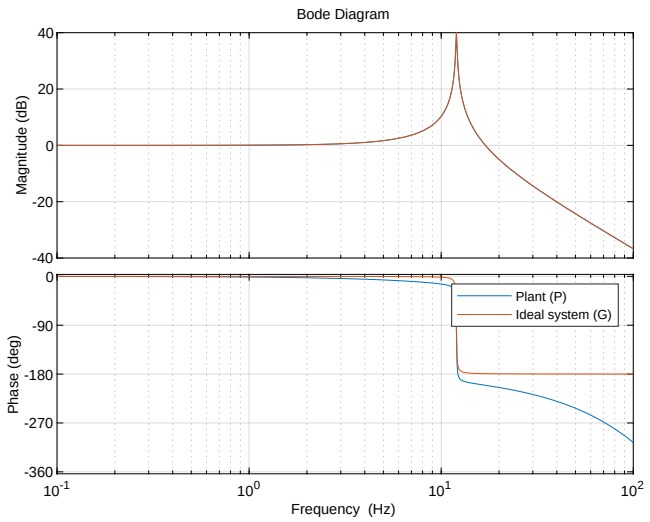


Figure 6-11: Plant of an ideal system compared to a system with a 3.5 millisecond delay.

6-4-1 Loop Shaping

To create a PID controller, loop shaping was used as a method. Loop shaping leverages the graphical nature of Bode plots to easily visualize the magnitude and phase parts of the system. When transfer functions are multiplied, it can be easily seen what happens with the magnitude and phase and at which frequency it started.

Several goals were established for control design:

1. Increasing the closed-loop bandwidth to achieve a faster response speed of the system, resulting in a lower rise-time. This is done by increasing the gain of the system to shift the magnitude plot upwards and shifting the open-loop cut-off frequency to the right. The cut-off frequency is not necessarily equal to the final closed-loop bandwidth, as $\omega_c \leq \omega_{\text{bandwidth}} \leq 2\omega_c$ [57].

2. Using derivative action to add phase margin which enhances stability. In addition, this derivative action creates artificial damping which lowers the resonance peaks and results in less overshoot.

3. Adding integral action to reduce steady-state error and increase disturbance rejection.

The following steps were used when applying loop shaping:

1. The cut-off frequency was set to 30 Hz to increase the bandwidth of the controller. This number was chosen to be 30 Hz, as the delay at this frequency was not too large and it also avoids potential coupled dynamics at around 40 Hz.

2. The PID controller values k_p, f_i, f_d, and f_t are set to 0.8400, 3, 10, and 90 respectively. These values are determined using the previously chosen cut-off frequency and are calculated using the rules of thumb from the MSD book [31], as shown in Table 6-4

P Control	PD Control	PID Control
$k_p = \left\lvert \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$	$f_d = \frac{f_c}{3},\ f_t = 3f_c$ $k_p = \left\lvert 0.33 \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$	$f_d = \frac{f_c}{3},\ f_t = 3f_c,\ f_i = \frac{f_c}{10}$ $k_p = \left\lvert 0.33 \frac{1}{G(\omega)} \right\rvert_{\omega=2\pi f_c}$

Table 6-4: Rules of Thumb: The values of f_d, f_t, and f_i are related to the value of the cut-off frequency chosen.

3. A basic controller was implemented, and the open-loop stability margins were observed. The theoretical system, using a controller with the rules of thumb, was robust and had a phase and gain margin of 51.3 degrees and 7.48 dB, respectively. The real system was still stable, but the stability margins were lower, as expected, due to the delay. The k_p value was tuned slightly to obtain a good step response, with a phase margin of 20.4 degrees and a gain margin of 7.48 dB, depicted in Figure 6-12. Figure 6-13 shows the performance of the closed loop after control was applied.

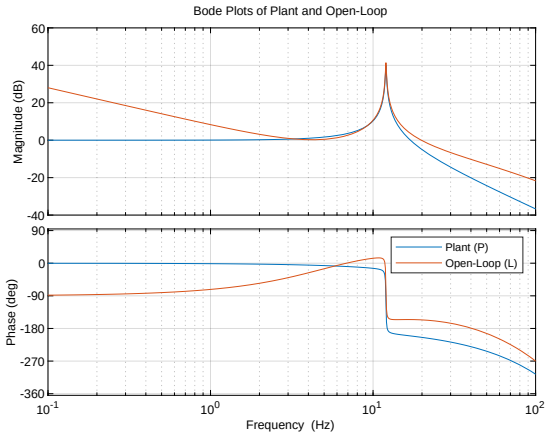


Figure 6-12: Plant vs Open-loop.

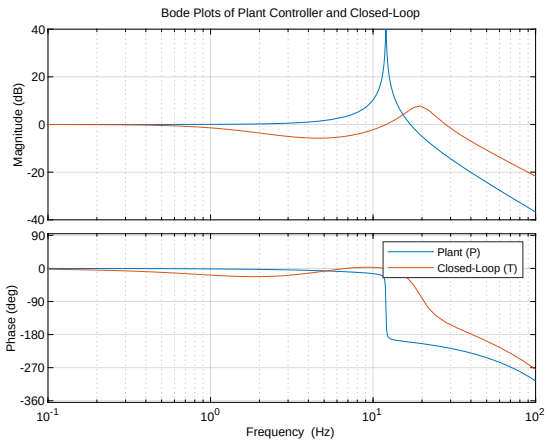


Figure 6-13: Plant vs closed-loop.

4. The closed-loop response of the system was plotted together with the sensitivity of the system. For the closed-loop system, the goal is to achieve good tracking and disturbance rejection, which involves a high open-loop gain at low frequencies, while simultaneously achieving good noise rejection at higher frequencies with a low open-loop gain. This results in a trade-off between sensitivity and complementary sensitivity, depicted in Figure 6-14.

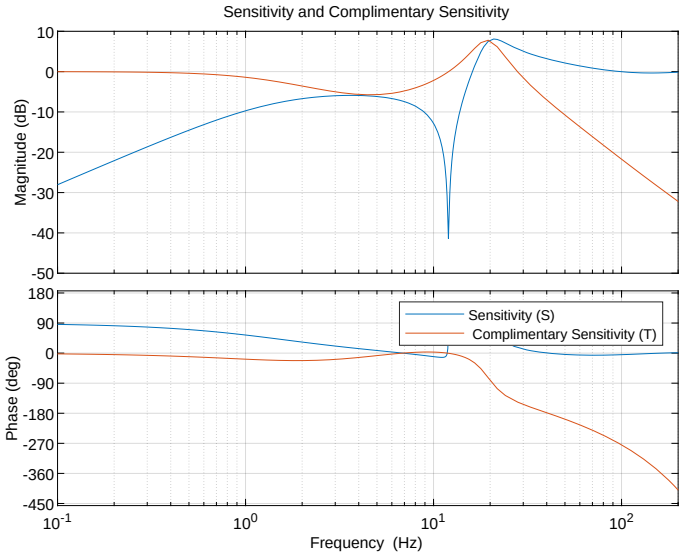


Figure 6-14: Sensitivity and complimentary sensivity.

5. To assess the stability of the system, a Nyquist plot was plotted, depicted in Figure 6-17. The number of rotations were counted to establish if the controller was stable.

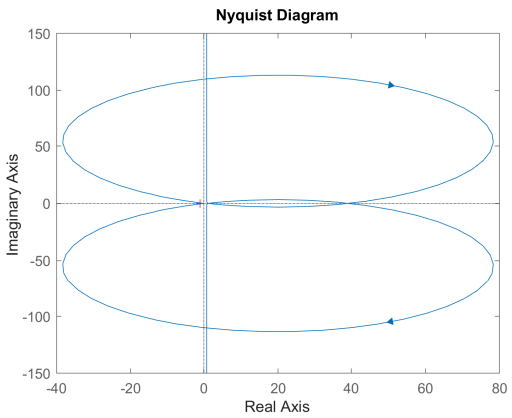


Figure 6-15: Nyquist diagram zoomed-out.

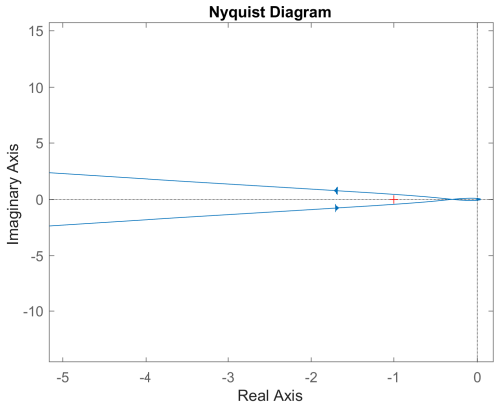


Figure 6-16: Nyquist diragram zoomed-in around -1 point

Figure 6-17: Nyquist diagrams.

6. To simulate the effect of the controller on the system, the step response of the system with and without control was plotted. This also helped in determining the final K_p value to achieve the desired step response.

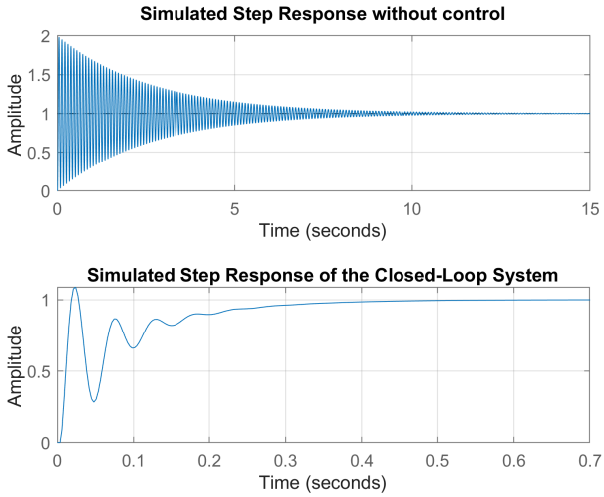


Figure 6-18: Simulated step response of system with and without control.

7. The last step was converting the transfer function of the obtained controller from a continuous system to a discrete system. The MATLAB function `c2d` [58] was used to obtain the Z-domain discrete controller transfer function, which was then imported into the Simulink setup. The closed loop was tested by applying an input square wave signal and comparing the performance of the system before and after feedback.

Chapter 7

Results and Discussion

This chapter presents the findings of this Master Thesis, organized into two sections: results and discussion. The results are divided into separate categories. The first subsection details the results of the event camera and tracking. The second, third, and fourth subsections cover the system identification process, divided into spectrograms, PSD, and Bode diagrams, respectively. The fifth subsection presents the control results of the fully integrated system, focusing on the step response results. Lastly, the discussion follows the same structure, addressing the main findings of the results. Additionally, the sub-questions formulated will be briefly answered.

7-1 Results

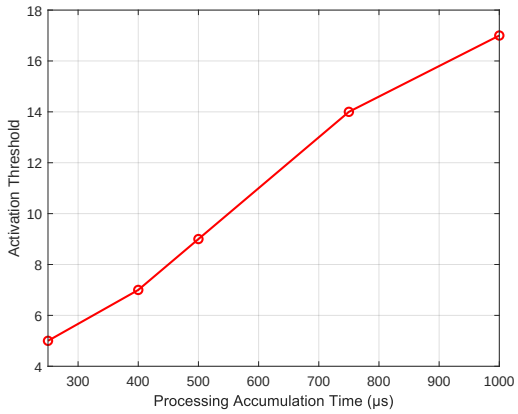
7-1-1 Event camera

Table 7-1 presents the delays introduced by the individual components of the setup.

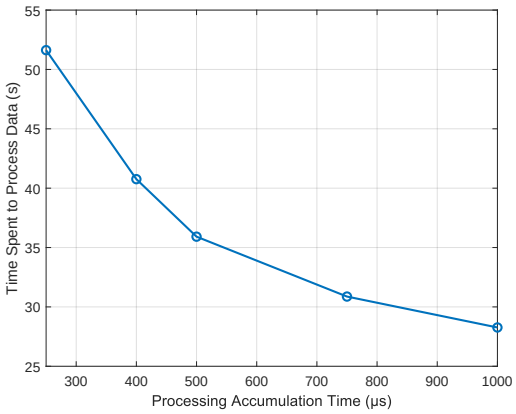
Description	**Delay (ms)**	**% Total Delay**
Fpga	0.50	14.3%
Sound Field	0.20	5.71%
Transducer	0.14	4.00%
Mechanical Delay Object (freq. dependent)	0.50	14.3%
Internal sensor delay [41]	≤ 0.20	5.71%
Event Camera USB 3.0 Connection	0.20	5.71%
Tracking Algorithm + Accumulation Time	1.56	44.6%
UDP Connection + Switch	≤ 0.40	11.4%
Total	**3.50**	**100%**

Table 7-1: Delay of individual components of the complete setup [59].

Figure 7-1a and Figure 7-1b represent both the Activation Threshold and the time to process a recording, each as a function of accumulation time.

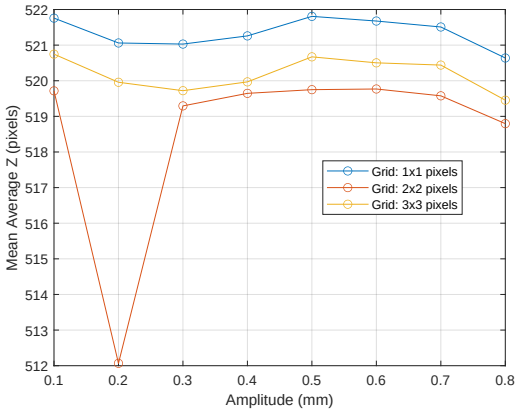


(a) Activation Threshold plotted against Processing Accumulation Time.

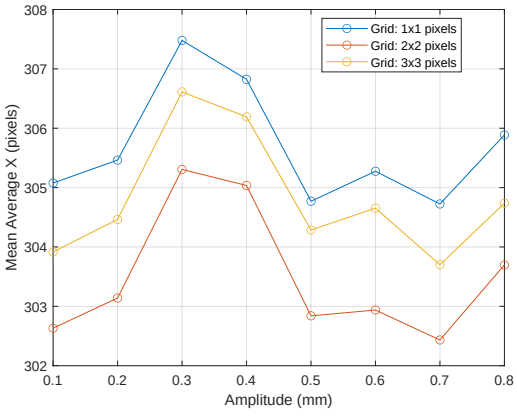


(b) Time to process a recording plotted against Accumulation Time.

Figure 7-2 presents the mean values of the average differences between the peaks and lows of the tracking coordinates for various amplitudes, emphasizing the effect of different grid sizes. For each peak and corresponding low, their average is calculated. Then, the mean of all these averages is determined to highlight the impact of different grid sizes. The mean values for the Z-direction are displayed in Figure 7-2a, while Figure 7-2b illustrates the mean values for the X-direction. The object was actuated solely in the Z-direction using a sinusoidal signal. However, due to coupled dynamics and disturbances, the observed X values are also plotted. A separate recording was made for each amplitude, but the three grid sizes use the same recording as data set. The location of the object on the sensor is given in pixels, as the grid sizes are also in pixels, with the sensor pixel location starting from the upper left corner of the sensor. The pixel size after magnification was 16.7 μm.



(a) The mean value for the Z-coordinate.

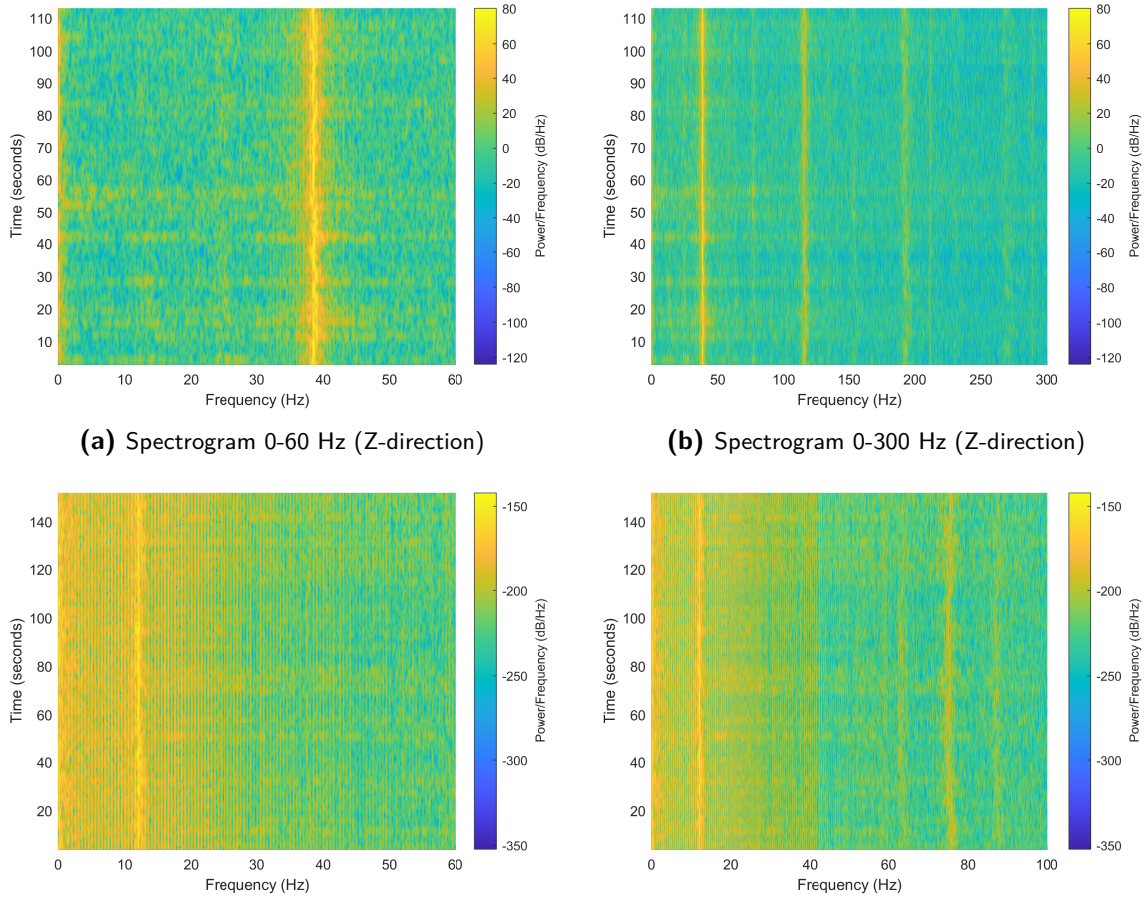


(b) The mean value for the X-coordinate.

Figure 7-2: The mean values of the object's X and Z coordinates are plotted against the amplitude for three grid sizes.

7-1-2 Spectogram

Figure 7-3 displays separate Spectrograms for the Z and X directions using a multi-sine input signal with a frequency range of 1 to 60 Hz. The total measurement duration was 110 seconds for the Z direction and 140 seconds for the X direction. The eigenfrequency in the Z direction is approximately 38.5 Hz, while for the X direction, it is 12 Hz. Higher frequency spectra are shown in Figure 7-3b and Figure 7-3d, revealing higher-order harmonics. In the X direction, these harmonics are observed at 61, 75, and 86 Hz, with no visible harmonics beyond 100 Hz. For the Z direction, vertical lines appear at 76.5, 115.5, 153.7, and 192.2 Hz.



(a) Spectrogram 0-60 Hz (Z-direction)

(b) Spectrogram 0-300 Hz (Z-direction)

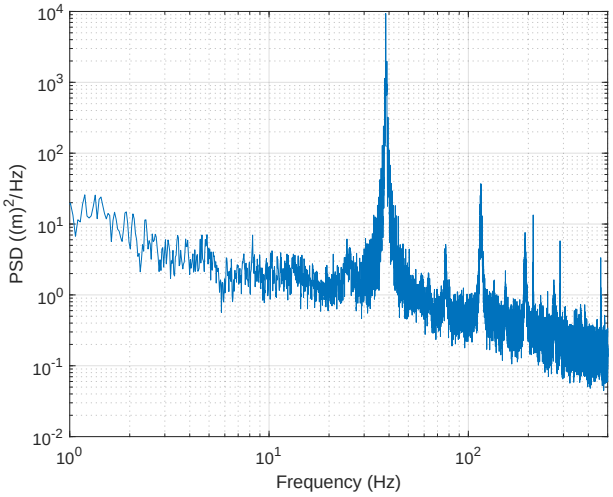
(c) Spectrogram 0-60 Hz (X-direction)

(d) Spectrogram 0-100 Hz (X-direction)

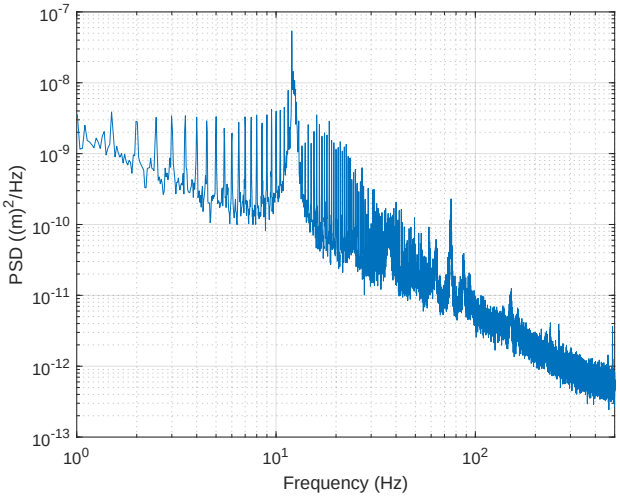
Figure 7-3: Spectrograms in X and Z directions

7-1-3 Power Spectrum Density

Figure 7-4 and Figure 7-5 illustrate a Power Spectral Density (PSD) plot using the Peter D. Welch method, showing the signal's power distribution across the frequency spectrum for the Z and X directions. Figure 7-4a and Figure 7-4b are generated without control, using a multi-sine signal. Figure 7-4a shows the Z direction where the eigenfrequency at 38.5 Hz exhibits the highest power, with individual excitation frequencies primarily visible around the resonance frequency. At lower frequencies, excitation peaks are somewhat subdued due to noise, following a 1/f curve, resembling pink noise. Additional peaks at 76.5 Hz, 115.5 Hz, 153.7 Hz, and 192.2 Hz align with the spectrogram in Figure 7-3b. Figure 7-4b shows the PSD in the X direction, where the eigenfrequency is 12 Hz, with individual excitation frequencies more visible at lower frequencies compared to the Z direction and fewer peaks beyond 100 Hz.



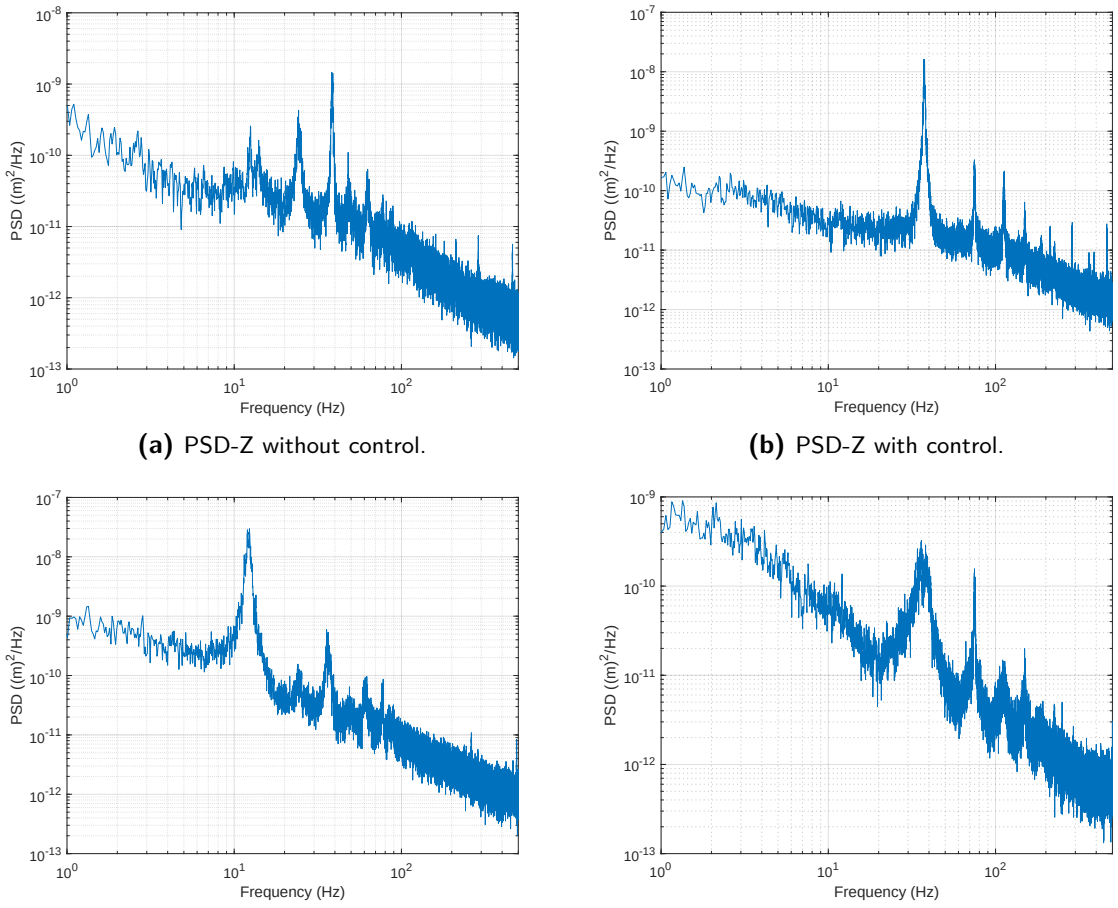
(a) PSD-Z with no control



(b) PSD-X with no control

Figure 7-4: Power Spectrum Diagrams in X and Z directions under different conditions

The PSD plots in Figure 7-5 were generated for the object levitating in a node without any displacement input signal. Figure 7-5a and Figure 7-5c were generated without any feedback control, while Figure 7-5b and Figure 7-5d utilize a closed-loop system with feedback control. With control, the Z direction shows lower power and subdued higher-order peaks, while in the X direction, the first resonance peak is subdued and a larger new broad resonance peak appears at approximately 38 Hz. Additionally, the power of the resonance peaks decreased for both the X and Z directions.



(a) PSD-Z without control.

(b) PSD-Z with control.

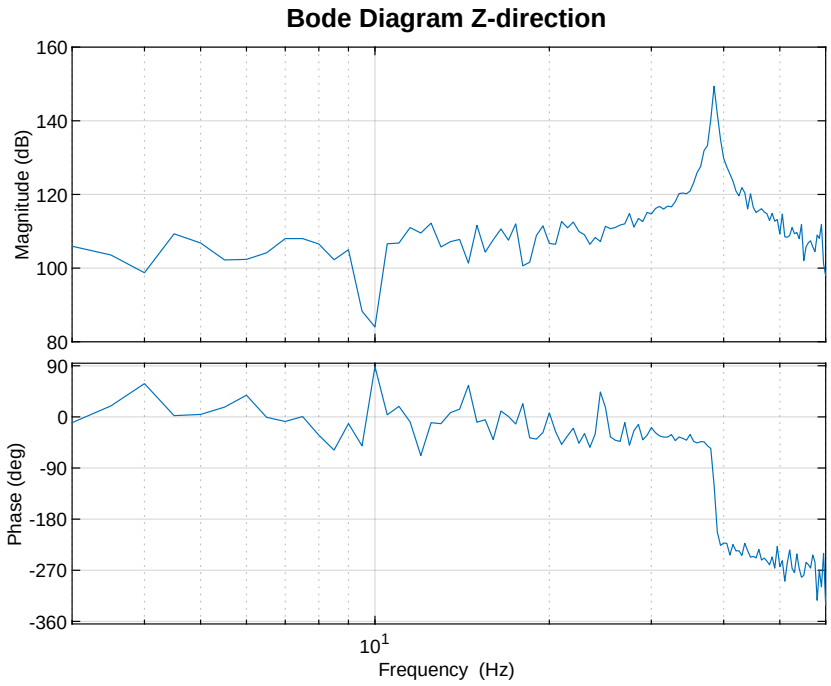
(c) PSD-X without control.

(d) PSD-X with control.

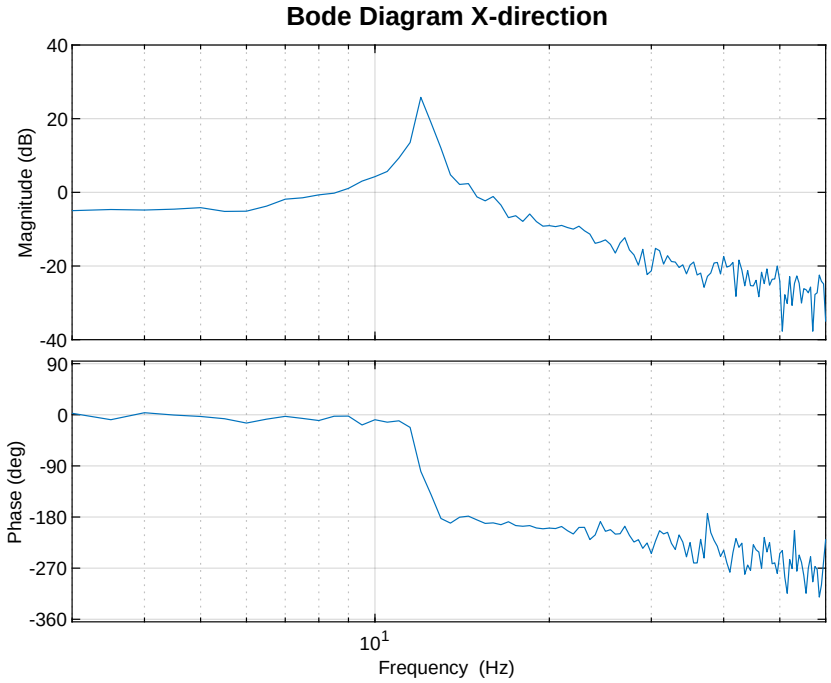
Figure 7-5: A collection of PSD diagrams was made for the object while it was levitating in a node without any input displacement.

7-1-4 Bode Diagrams

Figure 7-6 depicts two Bode diagrams made for the Z and X directions, obtained using a multi-sine. Figure 7-6a shows a resonance peak at 38.5 Hz, and Figure 7-6b shows the resonance peak at 12 Hz.



(a) Bode Diagram Z-direction.

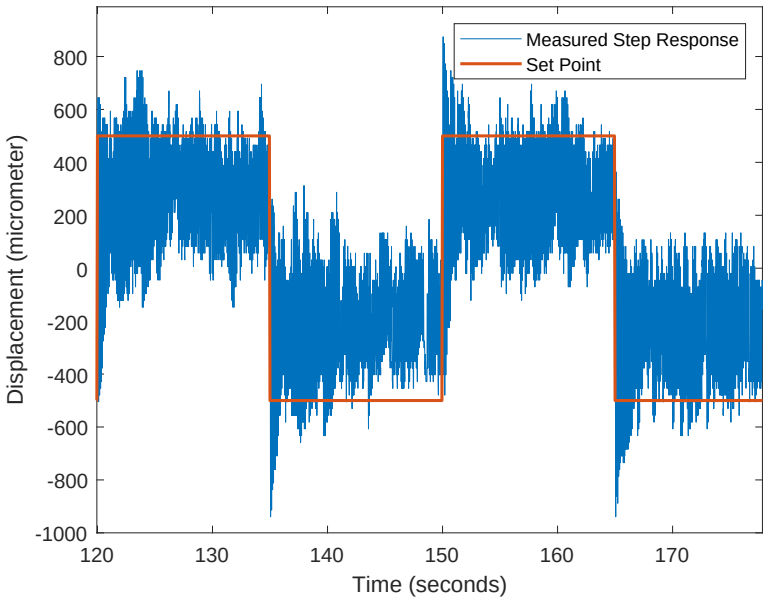


(b) Bode Diagram X-direction.

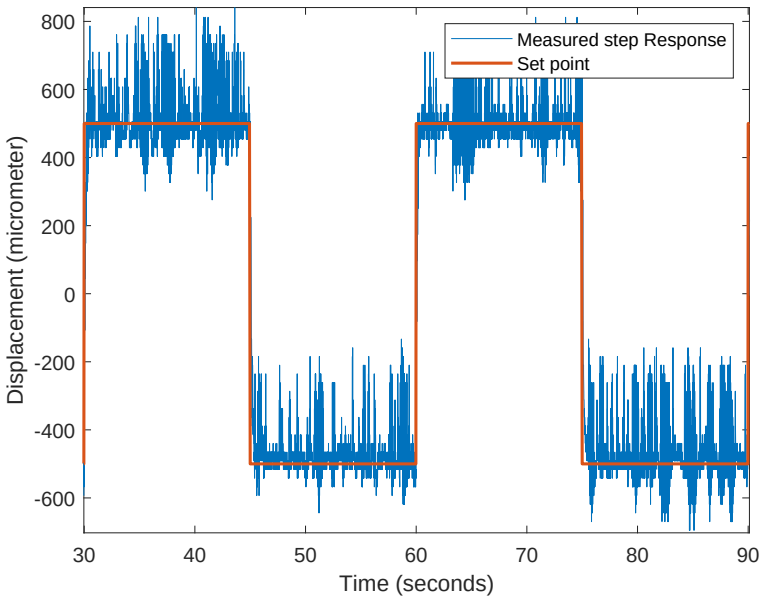
Figure 7-6: Bode diagram in Z and X direction.

7-1-5 Step Response

The final closed-loop bandwidth of the system is 40.8 Hz. Figure 7-7 presents the measured step response of the system with and without a controller, with the controller designed solely for the X-direction. The step signal used is a square wave with an amplitude of 1 millimeter and a frequency of 1/30 Hz.



(a) Step Response without control.



(b) Step Response with control.

Figure 7-7: Step response of the system with and without control.

7-2 Discussion

This section discusses the results previously presented and is organized into several subsections.

7-2-1 Event camera

During testing, the aim was to obtain enough events to capture the object for tracking while ensuring the tracking was fast and having minimal delay. The number of events generated depends on the particle (object) velocity v_p , the accumulation time t_a, and the pixel size L_{px}, as depicted in the Equation 7-1. Given the harmonic motion of the object, the velocity v_p can be described as $v_p = A_p\omega_p \sin(\omega_p t)$, where A_p is the amplitude of the motion, ω_p is the angular frequency, and t the time in seconds.

This leads to the formula general formula:

$$n_{px} = \frac{v_p t_a}{L_{px}} \tag{7-1}$$

And specifically for harmonic motion:

$$n_{px} = \frac{A_p\omega_p \sin(\omega_p t)}{L_{px}} \tag{7-2}$$

This relationship holds true when applying the tracking settings, which linearly depend on the number of events generated. These settings include accumulation time, cell width/cell height, and activation threshold. A higher accumulation time collects more events, necessitating a higher activation threshold to maintain the same event count, as depicted in Figure 7-1a. Due to rounding pixels to the nearest integer, the relationship is not completely linear. Additionally, when the grid size is changed, the activation threshold scales according to the area size ratio.

The `untracked threshold` depends mainly on the accumulation time of the object and how fast the object moves. A higher accumulation time requires a lower untracked threshold, and this setting is especially useful for harmonic motion where a predetermined path is followed. Equation 7-3 can be used to select the appropriate `untracked threshold`, where T is the period of the harmonic motion in milliseconds.

$$\text{Untracked threshold} = \frac{2 \cdot \texttt{accumulation-time}}{T} \tag{7-3}$$

The primary difference between various accumulation times is that the total processing time decreases with a higher accumulation time, mainly due to the read/write operations of the algorithm, which resembles logarithmic decay as depicted in Figure 7-1b. This conclusion was drawn using the same dataset (offline recording) for the various settings to prevent UDP delays from influencing the comparison. It was assumed that the data rate scales linearly with the event rate, which is not always the case. Depending on the object's motion, events

generated in columns require more data to be stored compared to the same number of events generated in rows.

In other literature, event cameras are often employed for high-speed tracking, with the footage being post-processed rather than used in real-time closed-loop systems. Christian E. Willert [34] found that the maximum event rate before saturating the arbiter was approximately 40×10^6 EV/s with a sampling rate of 5 kHz. Using the EVK4HD during this study confirmed that the camera can track a high number of events; however, this number decreases in real-time applications due to delays of the individual components and the tracking algorithm, as depicted in Table Table 7-1. For the current setup, the maximum number of events for real-time tracking is 6×10^6 EV/s at a sampling rate of 1 kHz, resulting in a total delay of approximately 3.5 milliseconds.

The discrepancies in event rates per second could be attributed to various factors. Some delays are inevitable, such as the mechanical delay of the object or the sound field, as these are inherent properties. Additionally, depending on the accumulation time, information from past events is also shared, increasing delays. While improvements can be made to the FPGA, the gains are far less compared to the delays introduced by the tracking algorithm and the UDP connection, as these constitute the main bulk of the delays. Minimizing UDP delay can be achieved by reducing the number of components in the setup, such as by running the algorithm directly on the FPGA instead of on a separate computer.

Tracking performance

Figure 7-2 was made using a flash instead of continuous lighting. A large part of the thesis was conducted using a flash before switching to continuous lighting. These figures do not necessarily offer insight into the accuracy but can tell something about the precision and consistency. At the same time, the reason for including these results is that they offer useful information about the way the tracking algorithm works. These results were obtained using a larger sphere of 1 mm and another lens magnification resulting in a larger pixel size of 16.7 μm versus 3.465 μm. The current, smaller field of view setup uses a grid size of 7x7, but the relative grid size is comparable to the 3x3 grid size.

Figure 7-2a shows that the mean position of a levitating object in the Z-direction stays relatively consistent across various amplitudes, even when using different grid sizes. The mean fluctuates approximately 1 pixel, which equals 16.7 μm. The input signal was a sinusoid, and the mean of this should be constant across various amplitudes, which is indeed the case. This consistency is observed despite the absence of any feedback control during the measurement, meaning the measurements were also subjected to disturbances. A smaller grid size should provide a higher resolution at the expense of being more susceptible to noise and having greater variation. This is evident as the 1x1 grid size exhibits a higher offset compared to larger grid sizes. However, the 2x2 grid size suffers more from noise compared to the 1x1 grid size, as seen in Figure 7-2a at an amplitude of 0.2 mm. This could be an outlier situation, due to noisy events. The expectation was that a larger grid size would result in a more consistent mean, but a grid size of 2x2 has a lower offset and comparable mean to that of a 3x3 grid.

Looking at the mean in the X-direction, as seen in Figure 7-2b, the mean fluctuates more. This could be attributed to the fact that the object was only actuated in the Z direction. However, due to coupling and disturbances, the object also moves in the X-direction in a

more chaotic manner and not in a consistent sinusoidal pattern like in the Z-direction. The Z-direction also experiences disturbances, but it should be less compared to the X-direction and averages out. For both directions, it seems that the grid size does not significantly help in increasing precision as the mean remains constant between different grid sizes, but only creates an offset between several grid sizes. Smaller grid sizes slow down the tracking noticeably without improving it. However to increase the minimum tracking resolution, which in turn increases the accuracy of the tracking requires smaller grid-sizes, but this still needs to be confirmed with measurements.

7-2-2 Spectogram

Both spectrograms in Figure 7-3 clearly show the eigenfrequency of their respective directions. The slight vertical line shift over time indicates possible temperature influences or sensor noise. In the Z-direction, excitation frequencies from the multi-sine are mostly not visible, except around the resonance frequency. Conversely, in the X- direction, the individual excitation frequencies from the multi-sine are more consistently visible over time.

In the Z-direction, higher-order excitation frequencies are more pronounced compared to the X direction and are approximately integer multiples of the eigenfrequency of 38.5 Hz, suggesting higher-order harmonics indicative of non-linear behavior. In contrast, in the X-direction, each harmonic is quite broad, not exactly an integer, and has relatively low power. Additionally, no harmonics are visible beyond 100 Hz in the X-direction, while in the Z direction, harmonics are visible up to 300 Hz, which is expected due to higher stiffness in the Z-direction.

7-2-3 Power Spectrum Density

Both PSDs, shown in Figure 7-4a and Figure 7-4b, clearly display the resonance peak of their respective directions. Additionally, both PSDs exhibit 1/f noise. At lower frequencies, the individual excitation frequencies are much more visible in the X direction compared to the Z direction. This can be explained by the multi-sine signal used: the amplitude of the excitation frequency was lowered, allowing power to be better distributed over other frequencies. This is also evident in the PSD, where the resonance frequency peak power is smaller relative to other excitation frequencies. Moreover, the tracking for the X direction was optimized to accommodate the lower eigenfrequency, which helps capture lower frequencies more accurately. Compared to the Z-direction, there are fewer peaks at higher frequencies in the X-direction, especially after 100 Hz, where they become almost non-existent.

Figure 7-5d clearly shows that after applying feedback control, the first resonance peak disappears in the X-direction. However, a second resonance peak is still unaddressed, depicted in Figure 7-5c. Still, the power of this second resonance peak is lower than the first peak that was damped using control. Even though the controller was only made for the X-direction, there is still some effect visible in Figure 7-5b, where all the lower peaks before the first resonance frequency disappear, which were previously visible in Figure 7-5a. This could be attributed to the coupled dynamics of the object.

7-2-4 Bode Diagrams

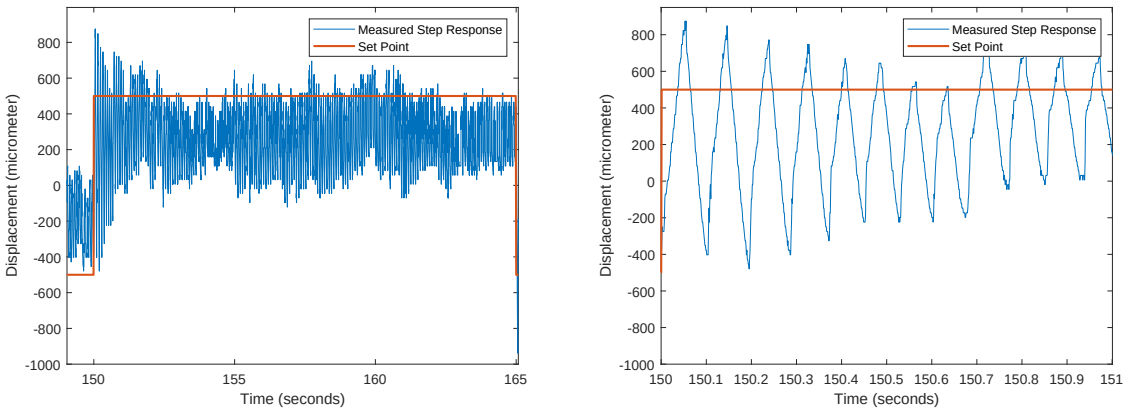
Figure 7-6a and Figure 7-6b depict the Bode diagrams of the Z and X directions, respectively, confirming that the levitated object is indeed a second-order system. Both the magnitude and phase diagrams follow second-order behavior relatively well. At lower frequencies, there is some noise, which could be attributed to sensor and pink noise. The resonance peak corresponds with the Spectrogram and PSD. The Q factor of the system in the Z-direction is about 460, and in the X-direction 20.

The delay in the system is clearly visible and increases with frequency. The total delay can be calculated at the frequency using Equation 7-4. The delay is about 50 degrees phase after the -180-degree line, resulting in a total delay in the whole system of 3.5 milliseconds when accounting for the Bode diagram in the Z-direction.

$$\text{Total Delay} = \frac{1}{\text{eigenfrequency}} \times \frac{\text{phase delay}}{360^\circ} \tag{7-4}$$

7-2-5 Step Response

Without control, the object moves according to its eigenfrequency, which is clearly visible when the step signal is applied, as depicted in Figure 7-8b. The object oscillates at this eigenfrequency until it reaches a steady position after approximately 10 seconds.



(a) Zoomed in step response.

(b) An oscillation with 12 Hz.

Figure 7-8: Step repsonse without control zoomed in.

When the controller is applied in the closed loop, as illustrated in Figure 7-9, the system follows the step input much faster, achieving a steady-state time of about 0.8 seconds. This is an improvement of 12.5 times compared to the open loop and confirms that the closed loop is working successfully in conjunction with the event camera. The total measurement duration was 12 minutes, and the object followed the step consistently. From the measurement, the system appears to be slightly overdamped. This suggests that the controller effectively damps the oscillations, but at the expense of some responsiveness. Fine-tuning the controller parameters could improve performance further, reducing the overdamping while maintaining stability.

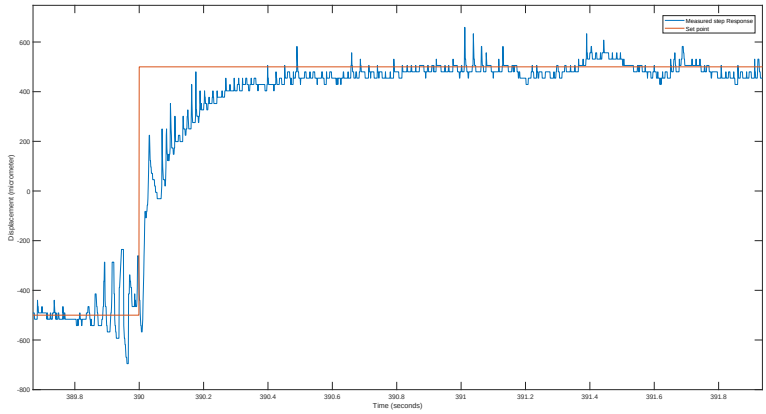


Figure 7-9: Measured step response zoom-in.

However, there seem to be some sudden spikes, illustrated in Figure 7-10. In the absence of spikes, the steady-state error is approximately 40 micrometers. These spikes can be attributed to several factors. Firstly, the open-loop cut-off frequency was chosen at 30 Hz to avoid a second resonance peak. However, this resulted in the final closed-loop bandwidth being 40.8 Hz. Unfortunately the underlying assumption that both would result in a bandwidth of 30 Hz is not true, as $\omega_c \leq \omega_{\text{bandwidth}} \leq 2\omega_c$. A 40.8 Hz bandwidth lies within a secondary excitation frequency. A PSD analysis was conducted on half a square wave where these spikes occur, depicted in Figure 7-11. There was a wide peak distributed between 33 and 39 Hz, hinting at the possible presence of a second resonance peak. This PSD fragment is short and suffers from noise, so a separate longer PSD was made of the object levitating in a zero position depicted in Figure 7-5d, which shows that the controller completely dampens the first resonance peak; however, there is a broad resonance peak around 38 Hz.

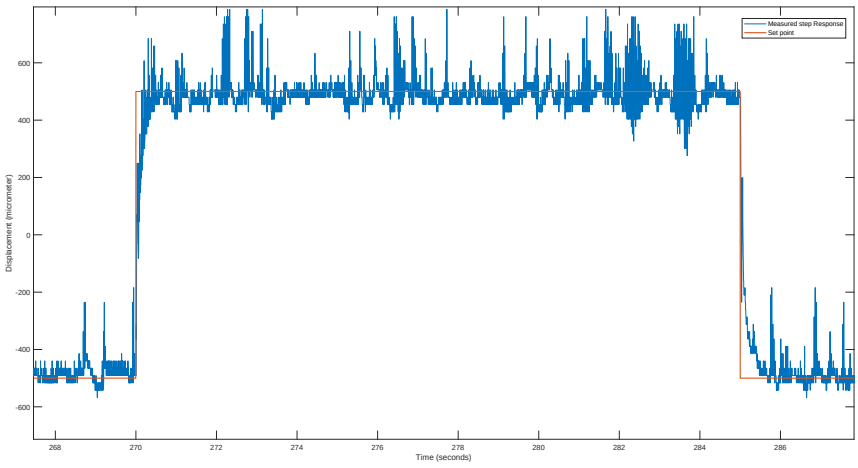


Figure 7-10: Segment of the measured steep response showing sudden noisy peaks.

In addition, a separate Bode plot in the X direction using a larger multi-sine signal also

confirms the existence of this peak, depicted in Figure 7-12. This Bode diagram was made for the system without using control. This multi-sine signal gave each frequency the same amplitude and uncovered a second peak, whereas previously, a signal was used where the amplitude of the resonance frequency was reduced. Secondly, the first resonance of the Z direction is also around this frequency, and since this is a coupled system, it could also be a contributing factor. A third possible explanation could be high-frequency amplification through the feedback loop. The controller used is slightly over damped, and the derivative action does not have a low-pass filter. If the final closed-loop bandwidth was less than 33 Hz, these spikes shouldn't have occurred, but the system currently behaves in a predictable manner, with the spikes being attributed to a chosen bandwidth that was set too high.

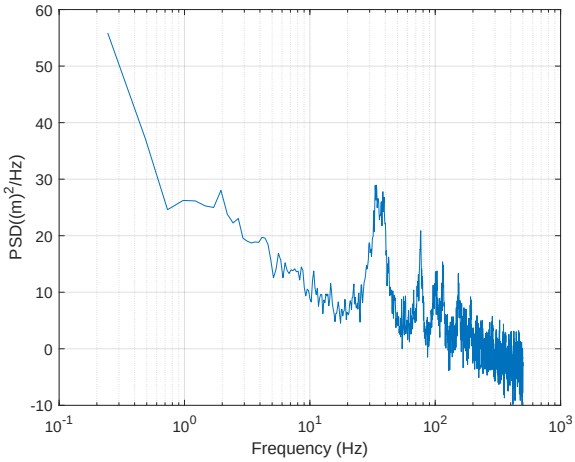


Figure 7-11: PSD of a 30 second segment.

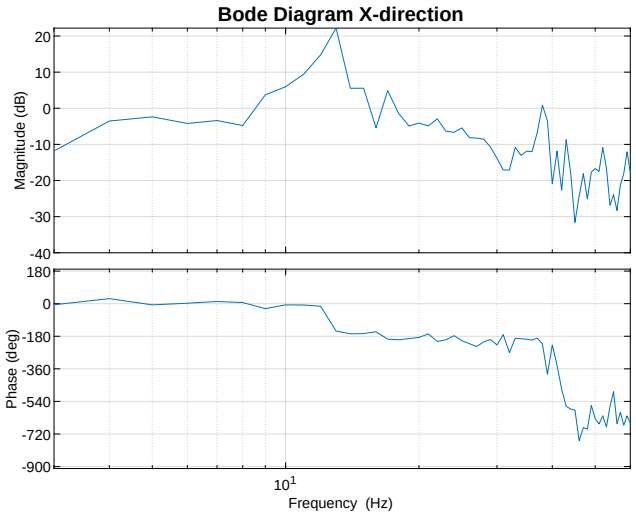


Figure 7-12: The step response was measured using a multi-sine ranging from 1 to 60Hz, with each frequency maintaining the same amplitude. A minor secondary resonance peak can be observed around the 38Hz mark.

7-3 Sub-questions

This section answers the sub-questions formulated alongside the main question and also summarizes some findings from the discussion.

- **Are there hardware limitations or constraints that need to be addressed in the integration of control and event-based cameras?**

  Yes, the current system has a delay of 3.5 ms. The delays the system experiences should be minimized. Delays affect the controller bandwidth and lower the stability margins. The stability margins are lower compared to an 'ideal' system without a delay. This reduces the potential of the system, lowering the operating bandwidth to remain stable. The bulk of the delay is due to processing, and this processing needs to be improved both software and hardware-wise.

- **How effectively and rapidly can event-based cameras measure displacement and oscillations?**

  Event cameras have the potential to record with over 10,000 fps. However, this is only for a limited number of events and this does not allow any real-time processing, only taking a recording. The current system can track up to 6×10^6 events per second (EV/s) in real-time at a sampling rate of 1 kHz.

- **What control strategies are best suited to address unwanted vibrations in acoustic levitation?**

  For now, only PID control was used, which has proven effective for the current use case. However, to achieve a higher bandwidth of 100 Hz, PID might not be sufficient, depending on the total system delay. There is always some delay in the system due to the sound field, transducers, and mechanical delay of the object. These delays lower the phase margins of the system, and for this reason, the delay of the individual components in the setup that can be influenced, such as the tracking algorithm, needs to be minimized and improved to be able to achieve higher bandwidths while remaining stable.

- **To what extent can event-based cameras enhance the accuracy of position measurements in acoustic levitation systems?**

  An exact figure of the precision still needs to be established, as the current setup lacks a second camera that can be used as a reference for comparison. However, from testing, the tracking seems consistent. In practice, the current implementation improved the step response of the system and provided insights into the dynamics of the system.

Chapter 8

Conclusions and Recommendation

Acoustic levitation is a promising technology for containerless transportation, offering the benefit of levitating materials regardless of their material properties (provided the transducers are not placed in a vacuum). The goal of this thesis was to improve the step response of acoustically levitated objects by adding artificial damping. Enhancing the step response allows for faster and more accurate transportation of objects, which can be applied to many fields, especially the high-tech industry. The proposed solution for this was to utilize an event camera to create a control feedback loop. Existing work has mainly focused on understanding the dynamics of acoustic levitation, but in this study, the dynamics of the system were actively controlled to improve performance. Consequently, the following main research question was set up:

"To what extent can control in conjunction with event-based cameras be employed to improve the step response of an acoustically levitated object?"

In order to answer the research question, the following methodology was implemented: an event camera was added to the acoustic levitation setup. Then, the spatter tracking algorithm was implemented, which allows to track the position of a levitated object and extract the Z and X coordinates. Subsequently, the event camera was utilized to apply system identification, which gave insight into the dynamics of the object. During this system identification, higher order harmonics were discovered. A second order model of the levitated object was developed, which was then used to design a feedback controller integrated with the event camera.

An event camera was successfully implemented in a closed control-loop. Due to the novel nature of event cameras, there are few existing algorithms, and event cameras often struggle in scenarios involving objects with low velocities and scenes with a high number of events. Although the Spatter tracking algorithm has the ability to track an object with good consistency, the exact tracking error has not been determined yet. Assuming a constant data rate, the performance of this algorithm scales linearly with the number of events. In this current setup, the maximum event rate achieved is 6 Mev/s in a real-time loop, compared to

40 Mev/s reported in the literature, where the difference is that the data from the literature is processed offline. The total system currently experiences a delay of 3.5 ms that impacts its full theoretical potential. While some delays are inherent due to physical constraints, the majority can be reduced with further optimization. Given the 3.5 ms delay, a controller has been optimized specifically for the X-direction.

Despite these delays, the experimental results demonstrate a substantial improvement in system performance. The bandwidth of the system improved from 18 Hz to 40.8 Hz. The settling time decreased from 10 seconds to 0.8 seconds, an improvement by a factor of 12.5, and the steady-state error was reduced by a factor of 3 compared to a setup without a controller. Additionally, the introduction of artificial damping successfully eliminated the first resonance peak, although a second higher-order resonance peak remains unaddressed.

These promising results highlight the potential of using event cameras in control loops for acoustic levitation systems. Although there are challenges to overcome, particularly concerning system delays, this work represents a significant initial step towards more precise and accurate control of levitated objects, with potential for further optimization and application.

8-1 Future Work

To continue and improve upon this research in the future, it is essential to address the limitations identified in the current work. Future research should focus on mitigating delays, implementing better event tracking algorithms, and addressing higher-order resonance peaks with improved control to fully realize the potential of acoustic levitation. These recommendations for future research are categorized to guide further improvements.

8-1-1 Event Camera and Setup Improvements

- Currently, the system lacks a secondary calibrated reference camera that could be used to evaluate the performance of the event camera and determine the tracking error. So far, the tracking consistency has only been assessed using a flash, not with the current method of constant illumination, which has yet to be conducted. While the Spatter tracking functions and is very promising, an exact figure needs to be determined before an event camera can be implemented in a real-world industrial application.
- The current setup employs a second computer to run the tracking algorithm to process the coordinates. Those coordinates need to be sent over a network connection to the FPGA. Eliminating a second computer and directly running those algorithms on the FPGA removes any network delays.
- The current algorithm is provided by Prophesee and is closed source. However, it is possible to obtain the source code by purchasing additional licenses. It would be interesting to explore comparable open-source algorithms or implement new ones. This approach has the benefit of optimizing the code, not solely depending on faster hardware. The code can also be optimized for this hardware, which goes hand in hand. Recently, Prophesee has offered an FPGA development board [60] alongside the event camera. In the future, this setup will necessitate a second camera to achieve 6DOF control, making it an interesting solution to consider.

- During this thesis, only a sphere was used, but other objects such as rectangles need to be studied extensively as they suffer from spinning. Event cameras can detect those rotational frequencies of individual pixels or the complete object as explained in Section 10-2.

8-1-2 Controller improvements

- The currently implemented controller is a PID controller that uses rules of thumb and is not fully optimized. From the dynamics, it is visible that the object has several resonance peaks. A control strategy needs to be derived to dampen these in order to further increase the bandwidth of the closed-loop system. One possibility is using higher-order control or employing multiple derivatives working in conjunction.
- Exploring other control strategies, such as a state feedback controller [61], could be beneficial. This system is a multi-input multi-output system, and to ensure stability, a robust controller can be created using H-infinity norm to counter deviations between the transfer function obtained from system identification and the real system.
- Another potentially interesting solution is an event controller [62]; event control is a new field that is still being actively developed. Current controllers work with fixed sampling rates, whereas a controller utilizing event data directly could offer significant advantages. Using an event camera in conjunction with an event controller opens new possibilities that could be used in other fields such as robotics, self-driving cars, and industry.

Chapter 9

All PROPHESEE Algorithms

9-1 Modules

Table 9-1: Comparison between METAVISION SDK and METAVISION SDK PRO [63].

Feature/Service	**METAVISION SDK**	**METAVISION SDK PRO**
Comprehensive tools and APIs (C++, Python)	✓	✓
Commercial-grade license	×	✓
Run on PC platforms (Windows, Linux)	✓	✓
Compile on custom platforms	×	✓
Source code access	×	✓
Knowledge center access	×	✓
Premium support	×	✓

Table 9-2: Advanced modules [63].

Module	**Description**
CV	Algorithms to filter, transform or extract information from events streams.
CV3D	Algorithms to localize the camera in 3d and reconstruct its environment.
Analytics	Algorithms to monitor and analyze the event stream, e.g., counting, tracking, measuring vibrations etc.
Calibration	Algorithms to calibrate an event-based camera.
ML	Python modules to manipulate event-based datasets and design event-based Neural Networks. Comes with pre-trained models and inference demos with event-based cameras.

Table 9-3: Open modules [63].

Module	Description
HAL	Hardware Abstraction Layer. Generic access to hardware features of the cameras.
Base	Basic classes and utility functions used in other modules.
Core	Generic and commonly used processing blocks.
Core ML	Generic functions for Machine Learning, event_to_video and video_to_event pipelines.
Driver	User-friendly API to ease the interaction with event-based systems, access event data and control sensor settings. This module is based on HAL and allows access to all HAL classes.
UI	Utility classes to manage on-screen display and react to system or user events.

9-2 Processors

Table 9-4: Algorithms processing events as input and outputting display-related data (e.g. frames):

Name	Module	C++ API
EventFrameDiffGenerationAlgorithm	Core	EventFrameDiffGenerationAlgorithm
EventFrameHistoGenerationAlgorithm	Core	EventFrameHistoGenerationAlgorithm
OnDemandFrameGenerationAlgorithm	CV	OnDemandFrameGeneration
PeriodicDecayFrameGeneratorAlgorithm	CV	PeriodicDecayFrameGeneration
TimeDecayFrameGenerationAlgorithm	CV	TimeDecayFrameGeneration
DenseFlowFrameGeneratorAlgorithm	CV	DenseFlowFrameGenerator
SparseFlowFrameGeneratorAlgorithm	CV	SparseFlowFrameGenerator
DominantValueMapAlgorithm	Analytics	DominantValueMap
HeatMapAsyncAlgorithm	Analytics	HeatMapFrameAsync
MapGeneratorAsyncAlgorithm	Analytics	MapGeneratorAsync
PeriodMapAsyncAlgorithm	Analytics	PeriodMapAsync
BlinkingFrameGeneratorAlgorithm	Calibration	BlinkingFrameGenerator

Table 9-5: Algorithms processing events as input and outputting some data from the event (counting, tracking etc.) [64].

Name	Module	C++ API
EventConverterAlgorithm	CV	EventConverter
FrequencyAlgorithm	CV	Frequency
PeriodAlgorithm	CV	Period
SparseOpticalFlowAlgorithm	CV	SparseOpticalFlow
PlaneFittingFlowAlgorithm	CV	PlaneFittingFlow
TimeGradientFlowAlgorithm	CV	TimeGradientFlowAlgorithT
TripletMatchingFlowAlgorithm	CV	TripletMatchingFlow
ModulatedLightDetectorAlgorithm	CV	ModulatedLightDetectorAlgorith
ActiveMarkerTrackerAlgorithm	CVD3D	ActiveMarkerTrackerAlgorith
ActiveMarkerPoseEstimatorAlgorithm	CVD3D	ActiveMarkerPoseEstimatorAlgorith
CountingAlgorithm	Analytics	Counting
DominantValueEventsAlgorithm	Analytics	DominantValueEvents
JetMonitoringAlgorithm	Analytics	JetMonitoring
PsmAlgorithm	Analytics	Psm
SpatterTrackerAlgorithm	Analytics	SpatterTracker
TrackingAlgorithm	Analytics	Tracking
BlinkingDotsGridDetectorAlgorithm	Calibration	BlinkingDotsGridDetector
DtfHighFreqScorerAlgorithm	Calibration	DtfHighFreqScorer

Table 9-6: Algorithms taking events as input and outputting the same number of events but with a transformation applied [64].

Name	Module	C++ API
FlipXAlgorithm	Core	FlipX
FlipYAlgorithm	Core	FlipY
PolarityInverterAlgorithm	Core	PolarityInverter
RotateEventsAlgorithm	CV	RotateEvents
TransposeEventsAlgorithm	CV	TransposeEvents

9-3 Filters

Table 9-7: Filters are algorithms taking events as input and outputting only the events that satisfy certain conditions [64].

Name	Module	C++ API
PolarityFilterAlgorithm	Core	PolarityFilter
RoiFilterAlgorithm	Core	RoiFilter
AntiFlickerAlgorithm	CV	AntiFlicker
ActivityNoiseFilterAlgorithm	CV	ActivityNoiseFilter
ProximityFilterAlgorithm	CV	ProximityFilter
RoiMaskAlgorithm	CV	RoiMask
SpatioTemporalContrastAlgorithm	CV	SpatioTemporalContrast
TrailFilterAlgorithm	CV	TrailFilter

9-4 Metavision Studio

The EVK4HD is also provided with a Graphical User Interface (GUI) called Metavision Studio, depicted in Figure 9-1. This program offers several options categorized into two main categories. The first category displays information and data collected about the system, while the second category allows for active changes to how the camera operates. The previously discussed settings such as the Biases, ROI and ESP functions are easily accessible using Metavision studio.

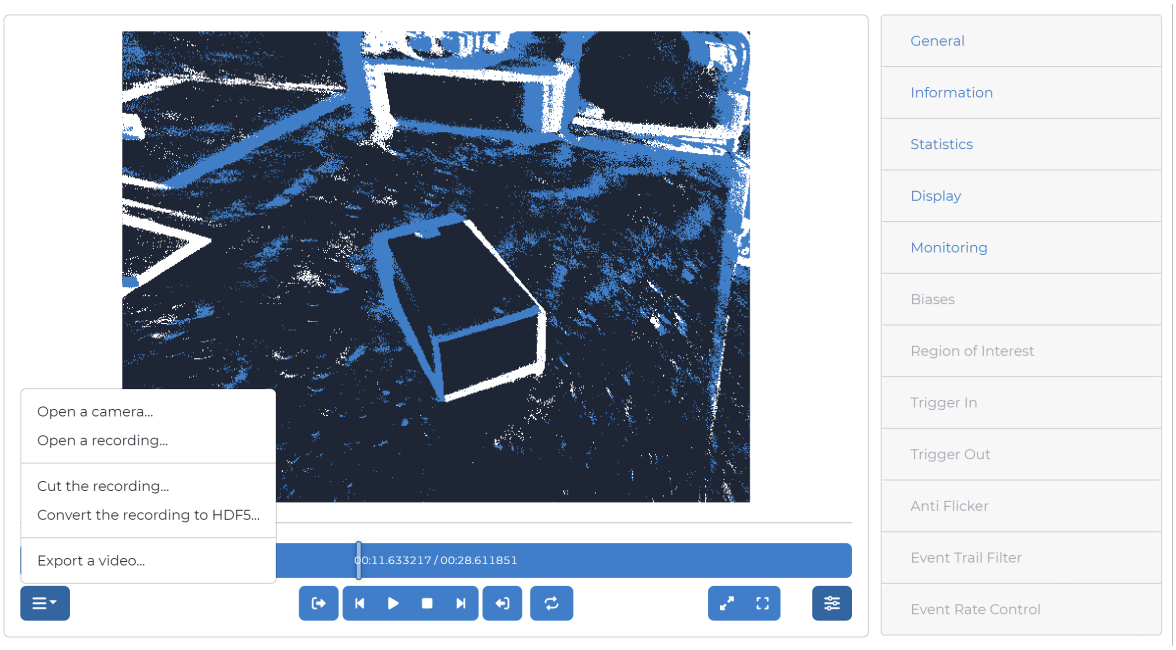


Figure 9-1: A screenshot of Metavision Studio with the available options on the right side.

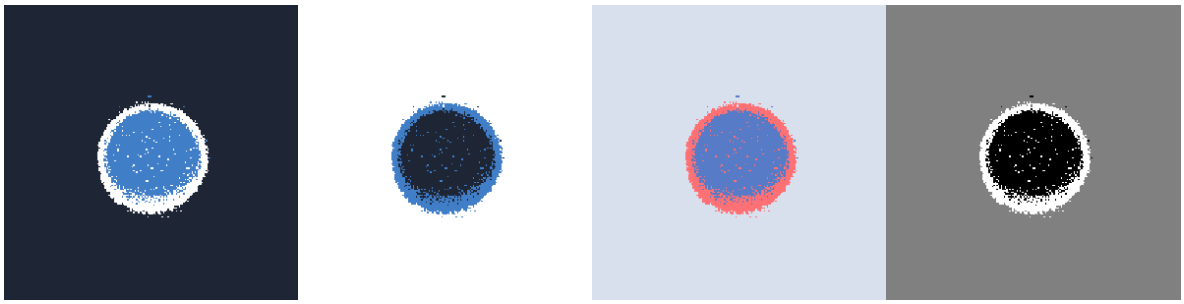
Opening MetaVision Studio reveals two buttons. The left button is for opening a camera, selecting the EVK4HD. This allows the user to see live video or make a recording. The right button is for opening a recording. A recorded file has the file extension .RAW. On the right side, there is a menu with various settings that will be discussed below:

9-4-1 Statistics

Under the tab "Statistics," two values can be seen. The first is the average Event Rate/s, and the second is this event rate/s converted to data rate MB/s.

9-4-2 Display

When opening a recording, the accumulation time and frame rate can be changed. This allows the user to slow down the recording and see 'frame by frame' each time new events are generated. The maximum frame rate achievable this way is 10,000 FPS. Additionally, it is possible to change the colors of events to four combinations depicted in Figure 9-2.



(a) Dark **(b)** Light **(c)** Coolwarm **(d)** Gray

Figure 9-2: Different color themes.

9-4-3 Monitoring

This setting displays external trigger events. There are three settings to choose from: a chart, a table, and disabling the view.

9-4-4 Biases

The Bias settings are explained in detail in Section 3-9. The default values, as well as the maximum and minimum values, are included in Table 9-8. Prophesee recommends not changing the 'bias_diff' setting as it could damage the sensor.

Table 9-8: Bias values

Bias Name	Default value	Minimum value	Maximum value
bias_diff	0	-25	23
bias_diff_on	0	-85	140
bias_diff_off	0	-35	190
bias_fo	0	-35	55
bias_hpf	0	0	120
bias_refr	0	-20	235

When the desired bias settings are selected they can be saved in a seperate file with a .bias extention. These files can be loased later in meatvisions studio or added to an algorithm. An example of how a .bias file looks is depicted in Figure 9-3:

```
1 0    % bias_diff
2 0    % bias_diff_off
3 0    % bias_diff_on
4 55   % bias_fo
5 0    % bias_hpf
6 0    % bias_refr
```

Figure 9-3: Example of a Bias file.

9-4-5 Region of Interest

The region of interest can be selected in two ways. The first and easiest way is to select the region in the video viewer with your mouse. The second, and more precise way, is entering an x and y coordinate and choosing the width and height of it, respectively.

9-4-6 Trigger In and Out

The trigger input channel can be selected. There are three options: Main, Aux, and Loopback.

9-4-7 Anti Flicker

The anti-flicker filter works in a range of 50 to 520 Hz. The filter type that can be selected is a Bandcut or a Bandpass filter. To enable the anti-flicker feature, select the status button at the bottom. The default values are shown in Table 9-9.

Table 9-9: Anti Flicker values

Options	Default value	Minimum value	Maximum value
Start threshold	6	0	7
Stop threshold	4	0	7
Duty cycle (%)	50	0.1	100

9-4-8 Event Trail Filter

Table 9-10: Event Trail Filter values

Options	Default value	Minimum value	Maximum value
STC_CUT_TRAIL (ms)	10	1	100
STC_KEEP_TRAIL (ms)	10	1	100
TRAIL (ms)	10	1	100

9-4-9 Event Rate Control

Options	Default value	Minimum value	Maximum value
Max Event Rate (MEv/s)	10	1	3200

Table 9-11: Event Rate Control values: It is not entirely clear whether the event rate is capped randomly or if only the newest events are filtered out.

Chapter 10

Supplementary Algorithms

This appendix chapter discusses two other algorithms, namely the 3D object tracking algorithm and why it is not implemented, as well as Vibration Estimation algorithm.

10-1 3D object Edge Tracking

The CV3D module [49] includes the Model 3D Detection and tracking, an algorithm for real-time tracking of a pre-known 3D model. This algorithm utilizes 2D data obtained from an event-based camera and focuses on tracking the object's edges depicted in Figure 10-1.

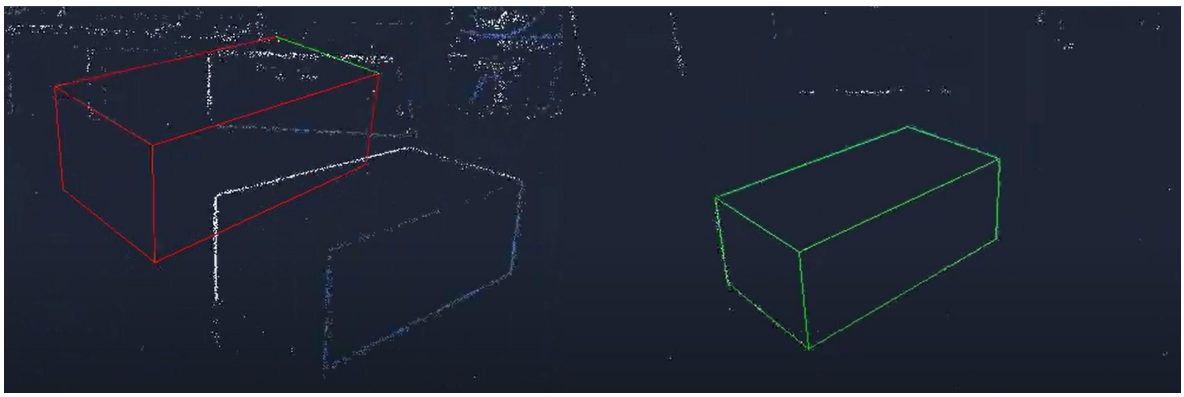


Figure 10-1: On the left: Predetermined shape (red) searching for the object to be tracked. On the right: The edges of the tracked object highlighted in green.

Requirements for the algorithm to function effectively include:

- The edges of the tracked model must be provided in a JSON file. It does not work for spheres.
- These edges should exhibit sufficient contrast to trigger the threshold of the event-based camera.

- There should be no other edges in close proximity to the object's edges.
- Excessive textures near the object's edges should be avoided.

The necessary files for the algorithm are as follows:

- A JSON file containing properties and details of the 3D object being tracked.
- Another JSON file specifying the initial camera position, which serves as the starting point for the detection algorithm.
- A JSON file containing the camera calibration information used during the process.

The detection and tracking works as follows:

1. **Edge detection:** The algorithm identifies the visible edges of the model.
2. **Creating support points:** Points are sampled along each edge, called support points depicted in Figure 10-2.

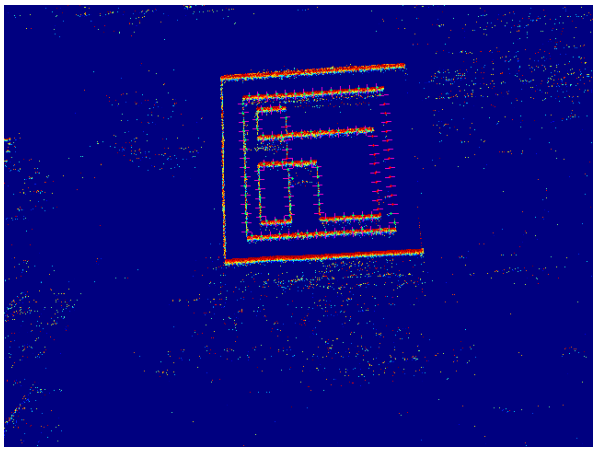


Figure 10-2: Support points along edges [49].

3. **Search for Matches:** For each support point, matches are searched for in the timesurface. A Time Surface is a snapshot from an event camera showing when each change in the scene occurred.
4. **Compute camera pose:** The previously matched support points are used in a Gauss-Newton optimization to compute the pose of the camera in relation to the model.
5. **Determine matches:** The algorithm identifies matches by looking for the pixel with the most recent timestamp along the normal of the edge. These matches are given weights based on their timestamps. The weights for the most recent and oldest timestamps can be adjusted in the pipeline parameters.
6. **Confirm Matches:** The matches are confirmed by verifying that the optical flow at the location of the match aligns with the expected flow, which is determined by the normal of the edge. Optical flow is the perceived motion of scene objects.

To enhance tracking performance, you have the flexibility to adjust the following parameters:

- `-n-us N_US`: This parameter alters the number of event changes before each tracking attempt is initiated.
- `-n-events N_EVENTS`: This parameter controls the time interval between each tracking attempt.

When this algorithm was tested, it exhibited poor performance primarily due to the round shape of the object initially used for tracking. This result was expected, and the algorithm also struggled when attempting to use a polygon shape. Additionally, it had difficulty with small rectangular chips because of the low contrast between their edges caused by their flat shape. Furthermore, there aren't many parameters that the user can change, making it constrained.

10-2 Vibration Estimation Algorithm

The Analytics API incorporates an algorithm designed to estimate the frequency of a vibrating object. It not only calculates the frequency for each event but also identifies the dominant frequency across all pixels. This algorithm can be used to estimate at which frequency the object oscillates with certain displacements. This is a useful property when looking at a mass-spring-damper system. In Figure 10-3, you can see a visual representation. Every individual pixel can be selected to determine its local frequency, while the algorithm highlights the dominant frequency with a box. This is also very useful while tuning the `bias_fo` and `bias_hpf` biases, as it shows at which frequencies noise is generated, making the selection of the filter band easier.

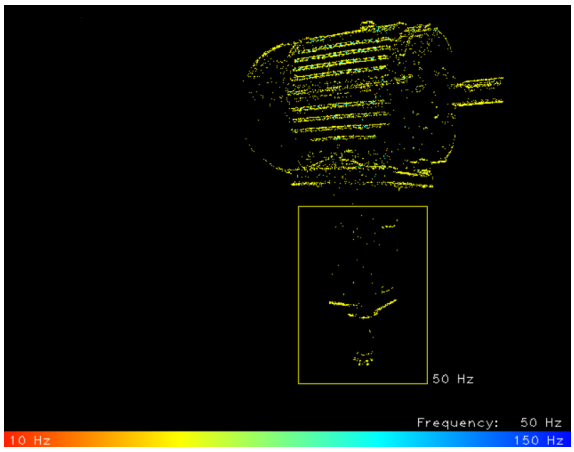


Figure 10-3: Example of vibration estimation on a motor, showing the frequency of each pixel [65].

Chapter 11

Network and Simulink implementation

This appendix chapter will show the implementation of the modified spatter tracking algorithm, where the BOOST.Aiso library will be incorporated. The base code will be modified to extract the object coordinates and then send them to Simulink for further processing.

11-1 Boost.Aiso

To send the position data from the Prophesee C++ code, the Boost.Asio [66] library is used. This library is useful for socket programming and enables a cross-platform client-server model. A socket provides a means for two programs or processes to communicate over a network, as depicted in Figure 11-1 , where the server sends the data and the client receives the data, or vice versa. In this case, the C++ code sends the data to the client, the Simulink model. To communicate and send/receive information to the correct socket, the IP address and port number play a crucial role. The following code snippets will explain how to achieve this.

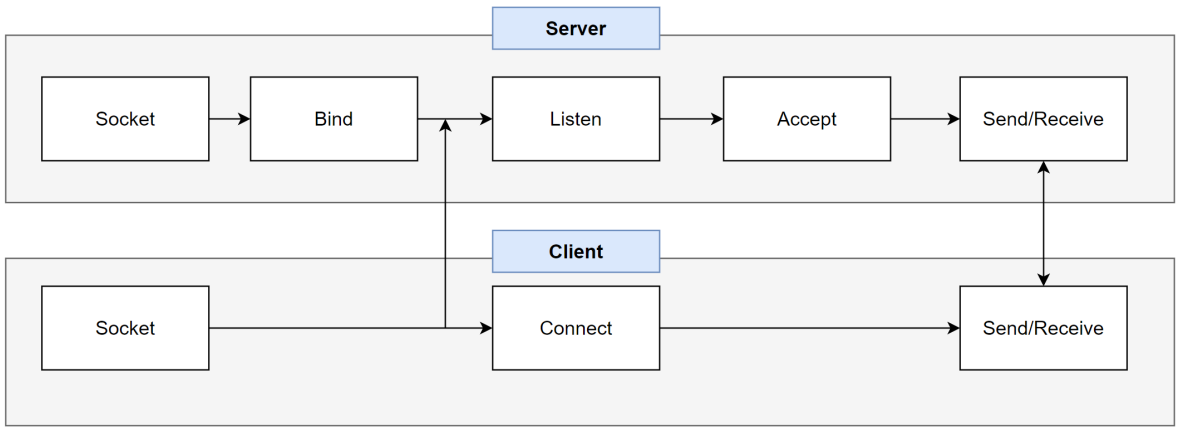


Figure 11-1: Network communication visualized [67].

This code snippet includes the necessary `C++` libraries, including `Boost/asio` version 1.84.0 and Boost/array version `1.77.0` [68].

```
1 //networking libraries
2 #include <boost/asio.hpp>
3 #include <boost/array.hpp>
```

Figure 11-2: C++ libraries.

Line two begins with creating a global `io_service` object, providing an event processing loop that handles asynchronous operations. Next, a global UDP socket is established making use of the `boost::asio::ip::udp::socket` class. After this, a `remote_endpoint` is created to specify the remote server to connect to.

```
1 // Create a global io_service object
2 boost::asio::io_service io_service;
3
4 // Create a global UDP socket
5 boost::asio::ip::udp::socket udp_socket(io_service);
6
7 // Create a global endpoint object
8 boost::asio::ip::udp::endpoint remote_endpoint;
```

Figure 11-3: UDP socket creation.

Figure 11-4 is the most important part of the code where the coordinates are extracted from the tracker. The `Pipeline::tracker_callback` function processes the tracker data. For each tracker, a `DataPacket` struct is created to hold the x and y coordinates. These coordinates are directly assigned to the `DataPacket` structure without any casting since they are already in the appropriate format. The `DataPacket` structure is designed to hold these coordinates efficiently, reducing the data size and lowering latency.

The function also includes a check for empty trackers. If the trackers vector is empty, the function returns immediately, avoiding unnecessary processing. This is a form of error handling that ensures the function operates correctly even when there are no trackers to process, which can happen at the beginning of the tracking algorithm before the first tracker is formed. If there are trackers, the function proceeds to create `DataPacket` objects and add them to the `packets` vector. And if there are multiple trackers at the same time, they are added to the same vector that then becomes a 4x1 vector instead of a 2x1. In the Simulink code that receives the coordinates, there is a method to only extract the useful coordinates.

```
struct DataPacket {
    double x;
    double y;
};

 /// [TRACKING_TRACKER_CALLBACK_BEGIN]
 void Pipeline::tracker_callback(Metavision::timestamp ts,
                          const std::vector<Metavision::EventSpatterCluster>
                              &trackers) {
// In your callback function
    if (trackers.empty())
        return;

    const auto& tracker = trackers.back();
    auto packets = std::vector<DataPacket>{};
    for (const auto& tracker: trackers) {
        DataPacket packet;
        packet.x = tracker.x;
        packet.y = tracker.y;
        packets.push_back(packet);
    }
```

Figure 11-4: Modified `Pipeline::tracker_callback` function.

This code is implemented inside the main function and is used to initialize the network connection. Line two sets the remote endpoint created earlier to the loopback IP address and port using the `boost::asio::ip::udp::endpoint` class.

```
 // Initialize the endpoint to the loopback IP address and a specific port
     number
    remote_endpoint = boost::asio::ip::udp::endpoint(boost::asio::ip::address
        ::from_string("192.168.2.2"), 9999);

    // Open the socket
    udp_socket.open(boost::asio::ip::udp::v4());
```

Figure 11-5: Connecting to IP of other computer.

11-2 Simulink

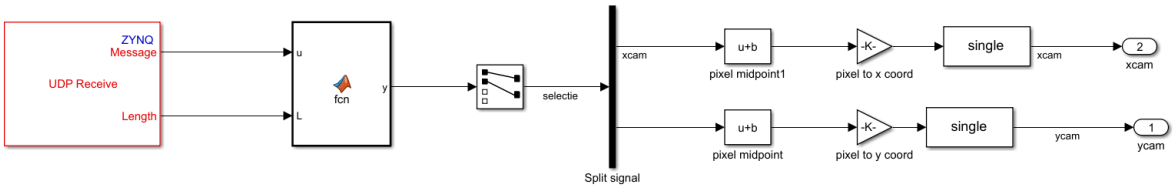
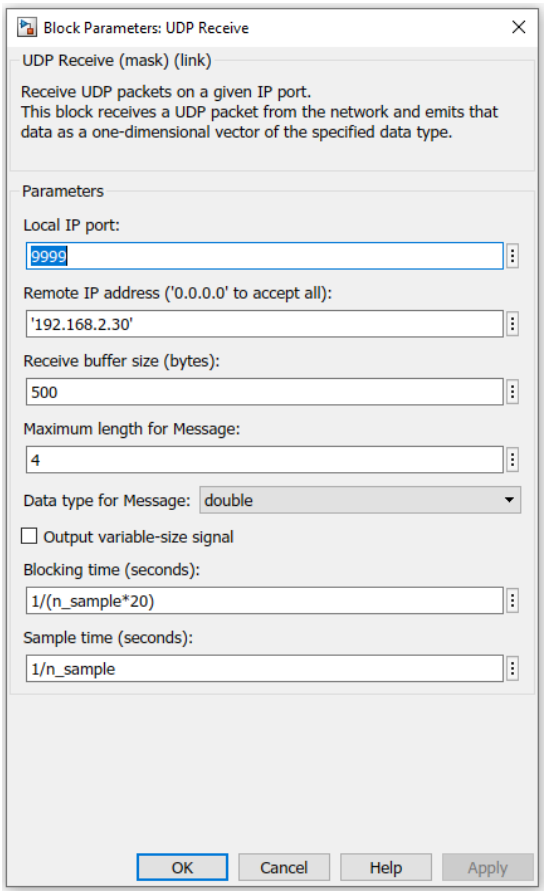
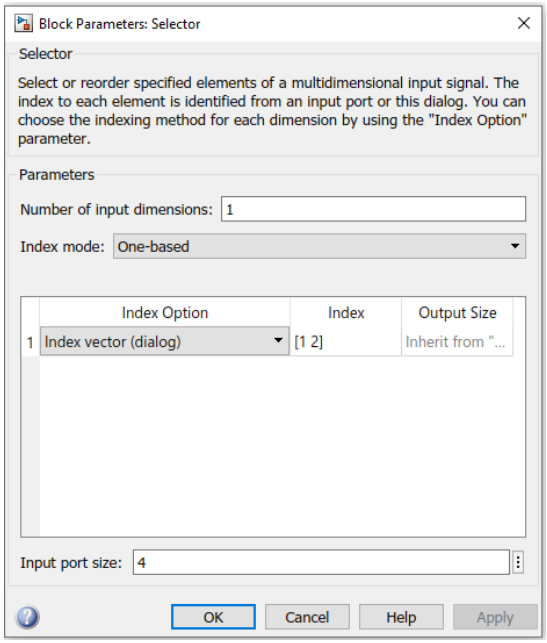


Figure 11-6: Simulink setup for network connection.

Figure 11-6 illustrates al the simulink blocks used for the network connection. To receive data over UDP, a server needs to be set up first as depicted in Figure 11-7a. When sending data between two programs on the same machine, 'localhost' or '0.0.0.0' is typically used as the IP address. For programs on different machines, the public IP address of the receiving machine is utilized. Both the sender and receiver should employ the same port number, which must remain open and unblocked by any firewalls for a successful connection and to minimize delays.



(a) UDP receive block



(b) Block selecting first or last 2 parameters.

Figure 11-7: UDP receive and parameter selector blocks.

The receive buffer size, measured in bytes, depends on the size of the data being sent. In this case, the maximum length of the message corresponds to the number of separate messages sent; for example, the x and y coordinates result in a length of 2. The data received is in the `uint16` format, resulting in a buffer size of 8 bytes, although 4 bytes should suffice. The blocking time refers to the duration the Simulink block must wait before receiving data and should ideally be minimized to ensure rapid data transmission. Additionally, the sample time should match the frequency of the data for optimal performance. The `Data` port outputs the received data and can be connected to other blocks to read it. In the case where there are two trackers at the same time, a simple Simulink function is written to only receive the most recent tracker, Figure 11-8. The data received is normally a 1x2 vector, but when there are two trackers, the data is sent in a 1x4 vector, and only the last two elements are taken.

```
function y = fcn(u,L)

y = u;

if L uint16(3)

elseif L uint16(2)
    y([12]) = [u(end-1); u(end)];
else
end

end
```

Figure 11-8: Code used in the Simulink block.

To separate the data and extract the x and y coordinates, a demux is used. The event camera provides the position as the location in the sensor using a coordinate system that works from the top left of the sensor, depicted in Figure 11-9. The conversion to a coordinate system with (0,0) at the center of the sensor is depicted in Figure 11-10. In addition, the pixel size conversion is done afterwards, and these coordinates are then sent to the controller.

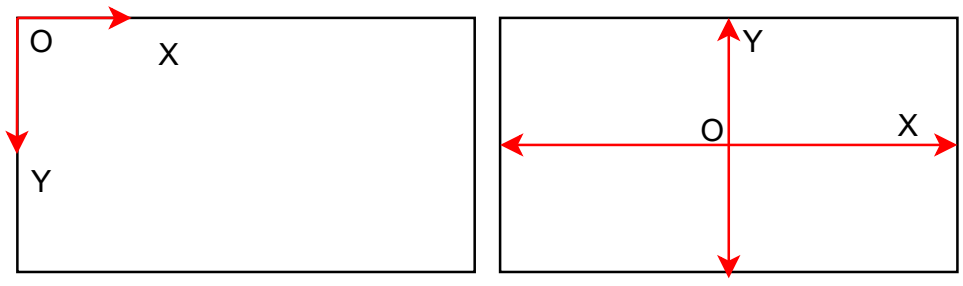
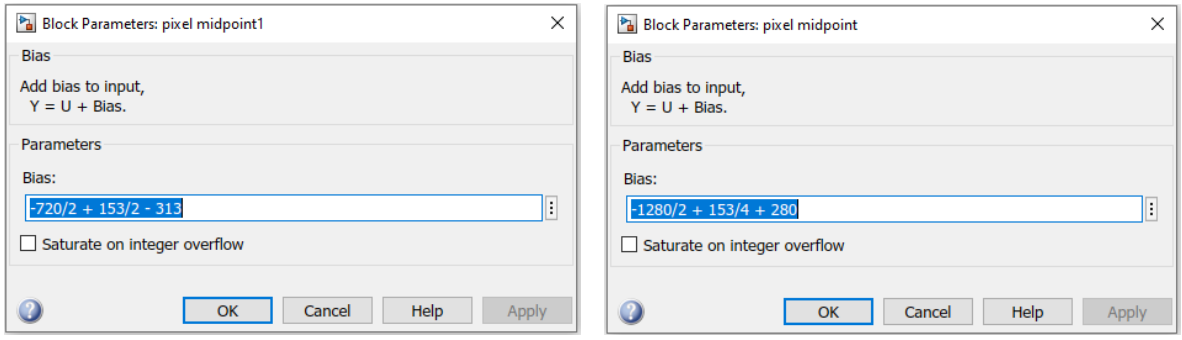


Figure 11-9: On the left is the standard coordinate system, and on the right is the converted coordinate system.



(a) Conversion X. **(b)** Conversion Y.

Figure 11-10: Modified values for coordinate conversion.

Chapter 12

Project Software Setup

This section will provide a step-by-step guide on the process of installing the software used in conjunction with the Prophesee EVK4 event camera. The files required for installation are supplied in a separate folder in case any of the links do not work anymore. Additionally, the BIOS settings used alongside other settings are detailed.

12-1 Locating Code

When you install the PROPHESEE SDK in the designated folder, you'll discover code examples provided in both Python and C++. The C++ examples, which come with CMake files, facilitate easy building of the projects. Additionally, essential third-party libraries from Boost and OpenCV, necessary for the PROPHESEE libraries, are included along with the SDK. All the algorithms are located in the SDK folder and are divided depending on the module. The C++ examples folders contain CMake files for easy compilation and linking of the necessary libraries. However, it's important to note that the PROPHESEE libraries are only available as .dll and .lib files. Unfortunately, the SDK doesn't support static linking, making it impossible to create standalone .EXE files.

The SDK folder is located using the following path shown below in Figure 12-1:

```
1 C:\Program Files\Prophesee\share\metavision\sdk
```

Figure 12-1: SDK folder file path.

The algorithm in question used is found in the following folder below in Figure 12-2.

```
C:\Program Files\Prophesee\share\metavision\sdk\analytics\cpp_samples\
    metavision_spatter_tracking
```

Figure 12-2: Spatter tracking file path

12-2 Installing Integrated Development Environment (IDE)

In the current setup, an Integrated Development Environment (IDE) is utilized to compile the code and generate an .EXE file that runs on the same computer. While any IDE can be used, the current configuration leverages Microsoft Visual Studio as the IDE and Microsoft Visual Studio Code as the text editor for code editing. Visual Studio is installed primarily for its C++ capabilities. Both Microsoft Visual Studio and Visual Studio Code can be installed from Microsoft's website. The community versions are selected to install. While logging in is possible it is not required.



Figure 12-3: Visual studio and studio code installation button screenshot from the website of microsoft [69].

In visual studio The desktop development with C++ option under the section Desktop and mobile needs to be selected and can be installed using the ''install while downloading" button at the bottom right corner.

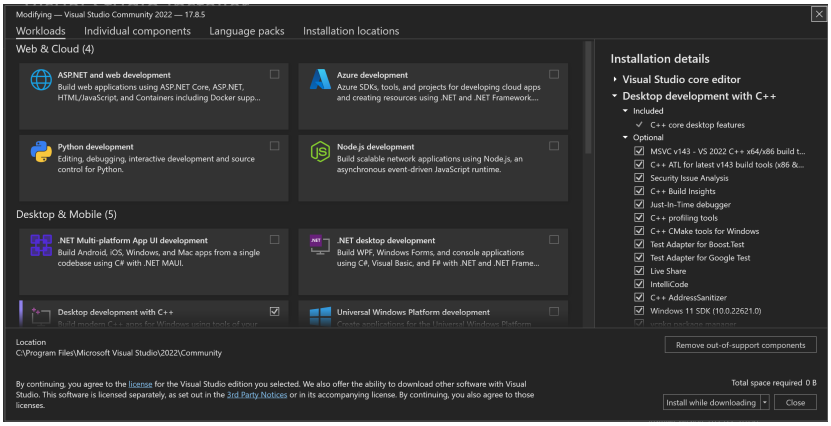


Figure 12-4: Screenshot from installation menu of Visual Studio.

After this a few exentions are installed form the extension store in Visual studio and are listed in Figure 12-5.

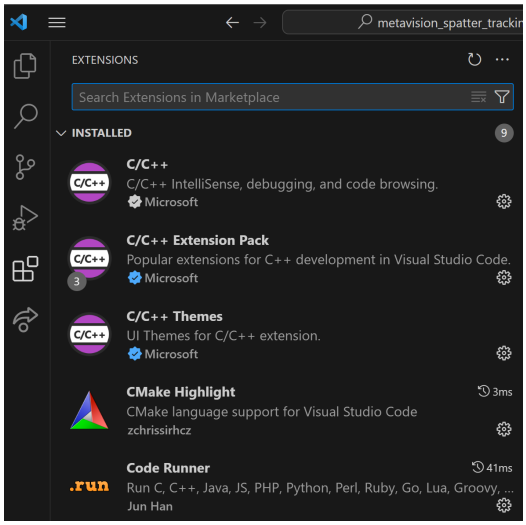


Figure 12-5: Visual studio extensions used.

12-3 Installing C++ compiler

The compiler used is MinGw distributed by MSYS2. This provides the necessary tools to compile and debug code. The compiler is installed from the website of MSYS2 [70]. Addtional instructions are avaliable on the website of microsoft. [71]

Installation instructions are also available on the website, but summarized below. Some tools need to installed using the UCRT dekstop manager depicted in Figure 12-6.

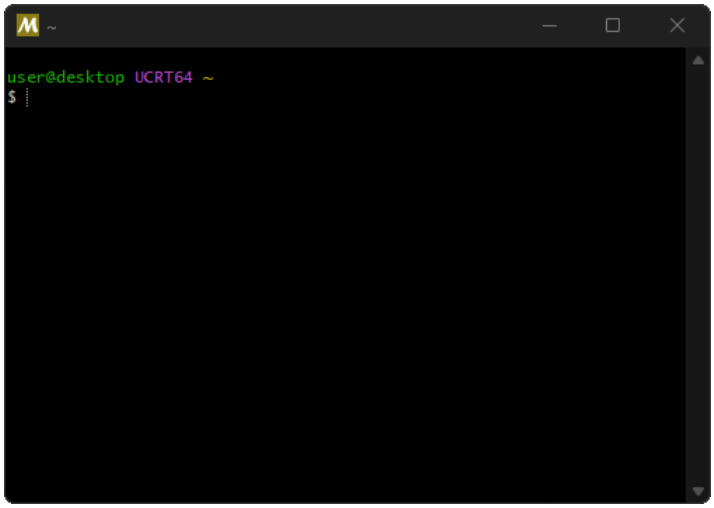


Figure 12-6: UCRT64 terminal environment.

The following command is used to install packages and is typed in the UCRT64 environment

terminal.

```
$ pacman -S mingw-w64-ucrt-x86_64-gcc
```

Figure 12-7: UCRT64 terminal command 1.

After typing the command the following will be displayed to proceed with the installation type 'Y' followed by pressing the enter key.

```
resolving dependencies...
looking for conflicting packages...

Packages (15) mingw-w64-ucrt-x86_64-binutils-2.41-2
          mingw-w64-ucrt-x86_64-crt-git-11.0.0.r216.gffe883434-1
          mingw-w64-ucrt-x86_64-gcc-libs-13.2.0-2 mingw-w64-ucrt-x86_64-gmp
              -6.3.0-2
          mingw-w64-ucrt-x86_64-headers-git-11.0.0.r216.gffe883434-1
          mingw-w64-ucrt-x86_64-isl-0.26-1 mingw-w64-ucrt-x86_64-libiconv
              -1.17-3
          mingw-w64-ucrt-x86_64-libwinpthread-git-11.0.0.r216.gffe883434-1
          mingw-w64-ucrt-x86_64-mpc-1.3.1-2 mingw-w64-ucrt-x86_64-mpfr
              -4.2.1-2
          mingw-w64-ucrt-x86_64-windows-default-manifest-6.4-4
          mingw-w64-ucrt-x86_64-winpthreads-git-11.0.0.r216.gffe883434-1
          mingw-w64-ucrt-x86_64-zlib-1.3-1 mingw-w64-ucrt-x86_64-zstd
              -1.5.5-1
          mingw-w64-ucrt-x86_64-gcc-13.2.0-2

Total Download Size: 49.38 MiB
Total Installed Size: 418.82 MiB

:: Proceed with installation? [Y/n]
[... downloading and installation continues ...]
```

Figure 12-8: UCRT64 terminal output of command 1.

The same process is repeated again, typing the command in Figure 12-9

```
$ pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

Figure 12-9: Terminal command 1.5

To finish the installation press the enter key.

```
$ pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
warning: base-devel-2022.12-2 is up to date -- skipping
:: There are 19 members in group mingw-w64-ucrt-x86_64-toolchain:
:: Repository ucrt64
1) mingw-w64-ucrt-x86_64-binutils
2) mingw-w64-ucrt-x86_64-crt-git
3) mingw-w64-ucrt-x86_64-gcc
4) mingw-w64-ucrt-x86_64-gcc-ada
5) mingw-w64-ucrt-x86_66-gcc-fortran
6) mingw-w54-ucrt-x86_66-gcc-libgfortran
7) mingw-w54-ucrty-x8664-gcc-libg
8) mingw-w54-ucret-yx8664-gcc-objective
9) mingu-ucret-yx8664-gdb
10) mungu-ucret-yx8664-dgb-multiarch
11) mungu-ucret-yx8664-header-git
12) mungu-ucret-yx8664-libgccjit
13) mungu-ucret-yx8664-libmangle-git
14) mungu-ucret-yx8664-libwinpthread-git
15) mungu-ucret-yx8666-make
16) mungu-ucretyyxx864-pkgconf
17) munguucretyyx864-tools-git
18) mnguucretyyx864-winpthreds-git
19) mnguucretyyx864-winstorecompat-git

Enter a selection (default=all):
```

Figure 12-10: UCRT64 terminal command 2.

To confirm if everything was installed correctly the following command is typed in the terminal

```
gcc --version
```

Figure 12-11: Command to confirm gcc version.

and should result in the following, confirming successful the installation.

```
C:\Users\Moham>gcc --version
gcc (Rev3, Built by MSYS2 project) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Figure 12-12: UCRT64 terminal output from command 2

The last step is adding the bin folder to the environement path in windows. This is done as follows.

Search for "Edit environment variables for your account." in the windows settings Under "User variables," locate and select the "Path" variable, then click "Edit." Choose "New" and add the destination folder of MinGW-w64 that you noted during installation. If you followed the default settings, this folder should be located at:

```
C:\msys64\ucrt64\bin.
```

Figure 12-13: Msys64 file location.

Click "OK" to save the updated PATH. Remember to reopen any console windows for the changes to take effect.

12-4 Installing Cmake

Cmake can be installed from [72]. In the distribiuted Cmake file in the prohphesee sdk the minimum supported cmake version is stated to be 3.5 and higher. In practise, errors occured while compiling. Version 3.21.1 has been found to work reliably for some unknown reason and solved this problem.

The process of adding the Cmake bin folder to the environemnt variables is repeated again with the only difference the location of the cmake Bin folder being the following:

```
C:\Program Files\CMake\bin
```

Figure 12-14: Cmake file location.

12-5 compiling code

To compile the CMake files, the command line is used, and the following three commands are typically employed. It's crucial to create the build folder in the correct directory to avoid compilation errors.

```
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build . --config Release
```

Figure 12-15: CMD commands for compiling exe

To access the help menu with all prompts type the following:

```
C:\Userlocation\metavision_spatter_tracking\build\Release>
   metavision_spatter_tracking.exe -h
```

Figure 12-16: CMD command to get help menu.

12-6 Bios settings

To achieve peak performance on the computer performing the tracking algorithm and to minimize sources introducing delays, several BIOS settings are changed to prevent power management and other power-saving features built into the processor from altering the performance. The following settings were adjusted in the BIOS:

Under the "Power" tab, the following two options were available:

1. **Fan speed:** The CPU fan speed can be increased, resulting in better airflow and allowing the CPU to sustain higher clock speeds. This comes at the expense of increased fan noise and potential reliability issues due to fan bearing wear. The standard fan speed enabled was 20/50 with no attached unit.
2. **Intel Speed Shift Technology:** Enables the operating system to select the appropriate processor performance automatically. This setting was turned off.

Under the "Performance" tab, the following two options were available:

1. **Intel SpeedStep Technology:** The system adjusts processor voltage and core frequency dynamically, decreasing average power consumption and heat production. This setting was turned off.
2. **Intel C-state Control**: Enables the CPU's ability to enter and exit low-power states. This setting was turned off.

Additionally, beyond BIOS adjustments, enabling the algorithm executable to run in real-time via the task manager can further minimize the risk of software-induced delays, thereby optimizing system performance for the tracking algorithm.

Chapter 13

Methodology Settings

13-1 Bias settings

The selected flashing frequency was set to 1000 Hz. This choice aimed to capture 1000 snapshots every second, resulting in 1000 coordinates per second. Several exposure settings were tested to strike a balance between providing ample light and avoiding unnecessary events.

Initially, the **bias_hpf** function (high-pass filter) was increased until it reached a value of 90, effectively reducing most of the noise. As the flash generates events, the goal was to minimize noise and maintain a low event count. The predominant noise occurred at low frequencies, which was filtered away.

The **bias_fo** (low-pass filter) was decreased to its maximum value of 55. This decreased the pixel response time, thereby improving the tracking of fast movements.

The **bias_diff_off** was increased to 90: The focus was solely on the sphere's position for controller coordinates, only one polarity was necessary. This was achieved by increasing the contrast detection level for the less important polarity. Specifically, adjusting **bias_diff_off** eliminated the negative polarity (blue) and retained only the positive polarity (white).

13-2 Multisine settings

Axis	Cell Size	Accu-mulation	Min Track Time	Apply Filter	a	D	Min Size	Untracked Threshold	ROI
Z	7x7	1000	100	1	5	400	80	12	220
X	7x7	1000	100	1	8	400	80	35	220

Table 13-1: Final tracking values used for the multi-sine for the Z and X directions.

Parameter	Value Z	Value X
Amplitude input signal	1000 um	1000 um
Total time input signal	120 s	160 s
Frequency resolution	0.5 Hz	0.5 Hz
Sample frequency	1000 Hz	1000 Hz
lower bound frequency range	0.5 Hz	0.5 Hz
upper bound frequency range	60 Hz	60 Hz

Table 13-2: Multi-sine signal values used for the X and Z directions.

13-3 Controller settings

Table 13-3: PID Controller Parameters

Parameter	k_p	f_i	f_d	f_t
Value	0.8400	3	10	90

Controller Transfer function:

$$C(s) = \frac{0.0007092s^2 + 0.04605s + 0.84}{9.382 \times 10^{-5}s^2 + 0.05305s} \tag{13-1}$$

Discrete Z-domain Simulink controlle:r

$$C[z] = \frac{7.56z^2 - 14.74z + 7.188}{z^2 - 1.568z + 0.5681} \tag{13-2}$$

Bibliography

[1] "Precision and microsystems engineering - tu delft." https://www.tudelft.nl/me/over/afdelingen/precision-and-microsystems-engineering-pme (Accessed on 19 June 2024).

[2] J. Cecil, M. B. Bharathi Raj Kumar, Y. Lu, and V. Basallali, "A review of micro-devices assembly techniques and technology," *The International Journal of Advanced Manufacturing Technology*, vol. 83, pp. 1569–1581, Apr. 2016.

[3] X. Shan, Y. Li, H. Liu, and T. Huang, "Residual Vibration Reduction of High-Speed Pick-and-Place Parallel Robot Using Input Shaping," *Chinese Journal of Mechanical Engineering*, vol. 35, p. 16, Feb. 2022.

[4] S. Borkar, P. Ghutke, W. Patil, S. Joshi, and S. Sorte, "A Review of Pick and Place Robots for the Pharmaceutical Industry," in *2023 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP)*, pp. 1–6, Apr. 2023. ISSN: 2157-0485.

[5] W. Beasley, B. Gatusch, D. Connolly-Taylor, C. Teng, A. Marzo, and J. Nunez-Yanez, "Ultrasonic Levitation with Software-Defined FPGAs and Electronically Phased Arrays," in *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, (Colchester, United Kingdom), pp. 41–48, IEEE, July 2019.

[6] S. Santesson and S. Nilsson, "Airborne chemistry: acoustic levitation in chemical analysis," *Analytical and Bioanalytical Chemistry*, vol. 378, pp. 1704–1709, Apr. 2004.

[7] K. Hasegawa and K. Kono, "Oscillation characteristics of levitated sample in resonant acoustic field," *AIP Advances*, vol. 9, p. 035313, Mar. 2019.

[8] S.-T. Kang and C.-K. Yeh, "Potential-well model in acoustic tweezers," *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 57, pp. 1451–1459, jun 2010.

[9] R. Hirayama, G. Christopoulos, D. Martinez Plasencia, and S. Subramanian, "High-speed acoustic holography with arbitrary scattering objects," *Science Advances*, vol. 8, p. eabn7614, June 2022. Publisher: American Association for the Advancement of Science.

[10] J. R. Gao, C. D. Cao, and B. Wei, "Containerless processing of materials by acoustic levitation," *Advances in Space Research*, vol. 24, pp. 1293–1297, Jan. 1999.

[11] T. Fushimi, T. L. Hill, A. Marzo, and B. W. Drinkwater, "Nonlinear trapping stiffness of mid-air single-axis acoustic levitators," *Applied Physics Letters*, vol. 113, p. 034102, July 2018.

[12] N. Pérez, M. A. B. Andrade, R. Canetti, and J. C. Adamowski, "Experimental determination of the dynamics of an acoustically levitated sphere," *Journal of Applied Physics*, vol. 116, p. 184903, Nov. 2014.

[13] M. A. B. Andrade, S. Polychronopoulos, G. Memoli, and A. Marzo, "Experimental investigation of the particle oscillation instability in a single-axis acoustic levitator," *AIP Advances*, vol. 9, p. 035020, Mar. 2019.

[14] R. C. Dorf and R. H. Bishop, "MODERN CONTROL SYSTEMS SOLUTION MANUAL," 2011.

[15] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-Based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 154–180, Jan. 2022. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[16] A. Kundt, *Ueber eine neue Art akustischer Staubfiguren und über die Anwendung derselben zur Bestimmung der Schallgeschwindigkeit in festen Körpern und Gasen*, vol. 203. 1866.

[17] J. W. Strutt, *The Theory of Sound*, vol. 1 of *Cambridge Library Collection - Physical Sciences*. Cambridge: Cambridge University Press, 2011.

[18] L. V. King, "On the Acoustic Radiation Pressure on Spheres," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 147, no. 861, pp. 212–240, 1934. Publisher: The Royal Society.

[19] L. P. Gor'kov, "On the Forces Acting on a Small Particle in an Acoustical Field in an Ideal Fluid," *Soviet Physics Doklady*, vol. 6, p. 773, Mar. 1962. ADS Bibcode: 1962SPhD....6..773G.

[20] L. Murata Manufacturing Co., "MA40S4S Ultrasonic Sensors , note = https://www.murata.com/en-eu/products/productdetail?partno=ma40s4s (accessed on 1 november 2023),."

[21] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders, *Fundamentals of Acoustics*. John Wiley & Sons, Jan. 2000. Google-Books-ID: FecSEAAAQBAJ.

[22] A. Marzo, T. Corkett, and B. W. Drinkwater, "Ultraino: An Open Phased-Array System for Narrowband Airborne Ultrasound Transmission," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 65, pp. 102–111, Jan. 2018. Conference Name: IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control.

[23] R. van Hoorn, "Self system identification of a massive mimo phased array of ultrasound transducers," 2023.

[24] S. Ghenna, F. Giraud, C. Giraud-Audine, and M. Amberg, "Vector Control of Piezoelectric Transducers and Ultrasonic Actuators," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 4880–4888, June 2018.

[25] M. A. Andrade, N. Perez, and J. Adamowski, "Review of Progress in Acoustic Levitation," *Brazilian Journal of Physics*, vol. 48, Dec. 2017.

[26] V. K. Gupta, G. Shanker, and N. K. Sharma, "Experiment on fluid drag and viscosity with an oscillating sphere," *American Journal of Physics*, vol. 54, pp. 619–622, July 1986. ADS Bibcode: 1986AmJPh..54..619G.

[27] T. Tsujino, K. Nakashima, and T. Fujii, "Application of h_∞ control and closed loop identification to a magnetic levitation system," *Asian Journal of Control*, vol. 1, pp. 283–296, oct 2008.

[28] S. Roefs, "Increasing the stability of levitating non-spherical solid objects," master's thesis, Delft University of Technology, 2019.

[29] M. A. B. Andrade, N. Perez, F. Buiochi, and J. C. Adamowski, "Matrix method for acoustic levitation simulation," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 58, pp. 1674–1683, Aug. 2011.

[30] J. Matouš, A. Kollarčík, M. Gurtner, T. Michálek, and Z. Hurák, "Optimization-based Feedback Manipulation Through an Array of Ultrasonic Transducers," *IFAC-PapersOnLine*, vol. 52, pp. 483–488, Jan. 2019.

[31] R. M. Schmidt, G. Schitter, A. Rankers, and J. van Eijk, *The Design of High-Performance Mechatronics*. IOS Press, 2020.

[32] W. Heemels, "Reset and switching control." Eindhoven University of Technology, n.d. https://heemels.tue.nl/research/reset-and-switching-control.

[33] Prophesee, "Temporal precision of prophesee sensor's readout." https://support.prophesee.ai/portal/en/kb/articles/gen4-readout-process (Accessed: 18 June, 2024).

[34] C. E. Willert, "Event-based imaging velocimetry using pulsed illumination," *Experiments in Fluids*, vol. 64, no. 98, 2023.

[35] Prophesee, "Introduction to event-based vision sensor video." https://support.prophesee.ai/portal/en/kb/articles/introduction-to-event-based-vision-sensor (Accessed on 19 June 2024).

[36] Prophesee, "Transfer latency." https://support.prophesee.ai/portal/en/kb/articles/evk-latency (Accessed on 19 June 2024).

[37] Prophesee, "Prophesee data encoding formats." Prophesee Documentation. https://docs.prophesee.ai/stable/data/encoding_formats/index.html (Accessed on 19 June 2024).

[38] Y. Wu, F. Paredes-Vallés, and G. C. H. E. de Croon, "Lightweight Event-based Optical Flow Estimation via Iterative Deblurring," May 2024. arXiv:2211.13726 [cs].

[39] Prophesee, "Event data visualization and representation video." https://support.prophesee.ai/portal/en/kb/articles/event-data-visualization-and-representation (Accessed on 20 May 2024).

[40] PROPHESEE Metavision For Machines, "Metavision Training Videos | Sensor Settings - Bias Tuning & Region of Interest (ROI)," Aug. 2023. https://www.youtube.com/watch?v=7c7n5DMuatY&t=17m3s.

[41] Prophesee, "Bias tuning & region of interest (roi) video." https://support.prophesee.ai/portal/en/kb/articles/bias-tuning-region-of-interest-roi-31-8-2023 (Accessed on 19 June 2024).

[42] Prophesee, "Bias tuning application note." https://support.prophesee.ai/portal/en/kb/articles/bias-tuning-flow (Accessed: 18 June, 2024).

[43] Prophesee, "Prophesee's metavision® image deblur solution for smartphones is now production-ready, seamlessly optimized for the snapdragon 8 gen 3 mobile platform." https://www.prophesee.ai/2024/02/27/metavision-image-deblur-production-ready/ (Accessed on 19 June 2024).

[44] Prophesee, "Prophesee Announces Collaboration with Qualcomm." hhttps://www.prophesee.ai/2023/02/27/prophesee-qualcomm-collaboration-snapdragon/ (Accessed on 19 June 2024).

[45] Prophesee, "Metavision sdk modules and packaging." https://docs.prophesee.ai/stable/modules.html (Accessed: 18 June, 2024).

[46] Prophesee, "Sdk pipelines guide." https://docs.prophesee.ai/stable/guides/sdk_pipelines/index.html# (Accessed: 18 June, 2024).

[47] Prophesee, "Generic tracking using c++." https://docs.prophesee.ai/stable/samples/modules/analytics/tracking_generic_cpp.html?highlight=generic%20tracki (Accessed: 18 June, 2024).

[48] Prophesee, "Spatter tracking using c++." https://docs.prophesee.ai/stable/samples/modules/analytics/tracking_spatter_cpp.html?highlight=spatter (Accessed: 18 June, 2024).

[49] Prophesee, "Model 3d detection and tracking using c++." https://docs.prophesee.ai/stable/samples/modules/cv3d/model_3d_tracking_cpp.html (Accessed: 18 June, 2024).

[50] A. Marzo, S. A. Seah, B. W. Drinkwater, D. R. Sahoo, B. Long, and S. Subramanian, "Holographic acoustic elements for manipulation of levitated objects," *Nature Communications*, vol. 6, p. 8661, Oct. 2015. Number: 1 Publisher: Nature Publishing Group.

[51] Cospheric, "White polyethylene microspheres, polymer beads, particles - opaque." https://www.cospheric.com/WPMS_opaque_white_polymer_spheres.htm (Accessed: 18 June, 2024).

[52] "TCP/IP and UDP Comparison - MATLAB & Simulink - MathWorks Benelux." https://nl.mathworks.com/help/instrument/tcpip-and-udp-comparison.html.

[53] "Sony to Release Two Types of Stacked Event-Based Vision Sensors with the Industry's Smallest*1 4.86μm Pixel Size for Detecting Subject Changes Only|News Releases|Sony Semiconductor Solutions Group." https://www.sony-semicon.com/en/news/2021/2021090901.html.

[54] E. Optics, "0.75x in-line illumination silvertl telecentric lens." https://www.edmundoptics.com/p/075x-in-line-illumination-silvertl-telecentric-lens/30608/ (Accessed: 18 June, 2024).

[55] I. Europe, "Idt high speed cameras." https://idteurope.eu/idt-cameras/ (Accessed: 18 June, 2024).

[56] The MathWorks, Inc., *Find Delay Between Signals*, 2024. https://nl.mathworks.com/help/signal/ref/finddelay.html (Accessed: Jun. 24, 2024).

[57] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, ch. 6. Prentice-Hall, 6 ed., 2010.

[58] The MathWorks, Inc., *Convert model from continuous to discrete time - MATLAB c2d*, 2024. https://nl.mathworks.com/help/ident/ref/dynamicsystem.c2d.html (Accessed: Jun. 24, 2024).

[59] V. Bos and P. G. Steeneken, "Vibration reduction for acoustic levitation." Unpublished.

[60] Prophesee, "Metavision amd kria starter kit imx636." https://www.prophesee.ai/event-based-metavision-amd-kria-starter-kit-imx636/ (Accessed: Jun. 19, 2024).

[61] S. H. ŻAKJ and C. A. MACCARLEY, "State-feedback control of non-linear systems†," *International Journal of Control*, vol. 43, pp. 1497–1514, May 1986. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207178608933554.

[62] E. Aranda-Escolástico, M. Guinaldo, R. Heradio, J. Chacon, H. Vargas, J. Sánchez, and S. Dormido, "Event-Based Control: A Bibliometric Analysis of Twenty Years of Research," *IEEE Access*, vol. 8, pp. 47188–47208, 2020. Conference Name: IEEE Access.

[63] Prophesee, "Metavision sdk modules and packaging." https://docs.prophesee.ai/stable/modules.html#chapter-modules-and-packaging-open (Accessed: 18 June, 2024).

[64] Prophesee, "Algorithms overview." https://docs.prophesee.ai/stable/algorithms.html (Accessed: 18 June, 2024).

[65] "Vibration Estimation using C++ — Metavision SDK Docs 4.4.0 documentation." https://docs.prophesee.ai/stable/samples/modules/analytics/vibration_cpp.html.

[66] Boost C++ Libraries, "Boost Version 1.84.0 Release Notes." https://www.boost.org/users/history/version_1_84_0.html (Accessed: 18 June, 2024).

[67] M. Tahir, "Socket programming in c++ using boost::asio." https://www.codeproject.com/Articles/1264257/Socket-Programming-in-Cplusplus-using-boost-asio-T (Accessed: 18 June, 2024).

[68] Boost C++ Libraries, "Boost Version 1.77.0 Release Notes." https://www.boost.org/users/history/version_1_77_0.html (Accessed: 18 June, 2024).

[69] Microsoft, "Meet the visual studio family." https://visualstudio.microsoft.com/ (Accessed: 18 June, 2024).

[70] MSYS2, "Msys2 software distribution and building platform for windows." https://www.msys2.org/ (Accessed: 18 June, 2024).

[71] Visual Studio Code, "Configuring C++ development with MinGW." https://code.visualstudio.com/docs/cpp/config-mingw (Accessed: 18 June, 2024).

[72] Kitware, "Cmake version 3.5 download." Kitware. https://cmake.org/download/ (Accessed June 18, 2024).