WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science & Engineering

Real-time Analysis of Aerosol Size Distributions with the Fast Integrated Mobility Spectrometer (FIMS)

by
Daisy Wang

A thesis presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Master of Science

December 2023
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

I extend my deepest appreciation to my advisor, Professor Jeremy Buhler, for his outstanding mentorship and remarkable patience. His dedication to guiding me has surpassed all my expectations, offering invaluable feedback on every aspect of my work. His approachability and professional insight into tackling any problem have profoundly shaped my academic journey. I am truly honored and privileged to have conducted my master's research under his guidance.

Special thanks to Marion Sudvarg, whose unwavering support was instrumental throughout my research. His readiness to assist with debugging and fielding questions smoothed the progress of my project and strengthened my confidence. Working alongside him provided a vital lesson in academic professionalism.

I must also express my sincere gratitude to Professor Jian Wang and Jiaoshi Zhang for their willingness to impart their technical expertise and for their consistent support throughout my research. Their guidance has been fundamental to my academic journey, and their trust in my work has greatly encouraged my scholarly development.

Daisy Wang

Washington University in St. Louis
December 2023

ABSTRACT OF THE THESIS

Real-time Analysis of Aerosol Size Distributions with the

Fast Integrated Mobility Spectrometer (FIMS)

by

Daisy Wang

Master of Science in Computer Science

Washington University in St. Louis, 2023

Professor Jeremy Buhler, Chair

The Fast Integrated Mobility Spectrometer (FIMS) has emerged as an innovative instrument in the aerosol science domain. It employs a spatially varying electric field to separate charged aerosol particles by their electrical mobilities. These separated particles are then enlarged through vapor condensation and imaged in real time by a high-speed CCD camera. FIMS achieves near 100% detection efficiency for particles ranging from 10 nm to 600 nm with a temporal resolution of one second. However, FIMS' real-time capabilities are limited by an offline data analysis process. Deferring analysis until hours or days after measurement makes FIMS' capabilities less valuable for probing dynamic, rapidly changing environments. Our research aims to address this limitation by developing a real-time data analysis pipeline for FIMS, allowing for adaptive aerosol measuring, eliminating lengthy delays between data collection and analysis, and boosting FIMS' potential for aerosol research. The pipeline is written in C++, making it suitable for deployment even in low-power embedded systems. The design also allows for easy future upgrades like new data types or machine learning integrations. Benchmarks confirm its efficiency. All real-time components operate within established limits, yielding results that are consistent with traditional offline methods. The real-time capabilities of this pipeline significantly extend FIMS's utility in dynamic, rapidly changing environments.

Chapter 1

Introduction

1.1 Role and Challenges of FIMS

Atmospheric aerosols wield a significant influence over air quality, public health, and climate dynamics. Their impact is intricately linked to their size. For instance, while larger particles scatter light more effectively, smaller ones often absorb it. This differential interaction with sunlight can either warm or cool the atmosphere, leading to tangible climate effects. On the health front, larger particles are mostly trapped in the upper respiratory tract, but finer aerosols penetrate deeper, posing heightened respiratory risks and persisting longer within the atmosphere. Thus, understanding the size distribution of these aerosols, particularly in the sub-micrometer spectrum, is fundamental to fully comprehending their implications for health, atmospheric quality, and climate.

The Fast Integrated Mobility Spectrometer (FIMS) represents a significant advancement in aerosol measurement tools. Designed to assess aerosol size distributions, it accurately measures particle sizes from 10nm to 600nm. What distinguishes FIMS from other instruments is its one-second time resolution for sub-micron aerosols, enabling it to track swift aerosol dynamics, such as those during nucleation events or particle formation processes. Therefore, FIMS is frequently utilized in mobile setups, such as on aircraft or vehicles, effectively capturing the detailed spatial and temporal fluctuations of atmospheric aerosols.

Research leveraging FIMS has illustrated its capabilities. In a study focused on the boundary layer aerosol concentration in the Amazon basin [27], researchers sought to understand how vertical transport during rainfall sustains the aerosol concentrations. The G-1 aircraft, equipped with an array of instruments, was deployed for measuring aerosol dynamics. Among these tools, FIMS captured the submicrometre particle size spectrum with a 1 Hz time resolution. The data from FIMS helped in discerning average concentrations of various aerosol particle sizes and their vertical distribution patterns during the wet season. Another research focused on the interaction between ultrafine aerosol particles, specifically those smaller than 50 nanometers, and deep convective clouds (DCCs) in the Amazon rainforest [6]. FIMS offered an accurate size distribution of ultrafine particles, which served as a benchmark for validating the assumptions and outcomes of their model.

While FIMS has both high measurement time resolution and high accuracy, its full potential is curtailed by latency in data processing. Although designed for mobile scenarios, FIMS relies on static measurement strategies with predefined measuring routes, followed by post-measurement analysis that is typically performed hours or even days later. This delay between data collection and analysis impedes real-time detection of swiftly changing aerosol environments which could be of great value for science research. It could also restrict the immediate identification and correction of instrument anomalies during data collection.

1.2 Objectives

The primary objective of this thesis is to transform FIMS into a tool for real-time atmospheric analysis. We seek to develop a computational pipeline that eliminates the lag between collection and interpretation, providing immediate access to processed, actionable data. This goal requires tackling several challenges: establishing a data flow that enables rapid processing; ensuring the accuracy of real-time outputs against traditional post-processed results; and maintaining consistent performance reliability, characterized by stable latency and throughput, to prevent system blockage and data loss, especially on low-power platforms.

By utilizing real-time analytical results, we intend to enable more adaptive, efficient, and reliable aerosol measurement. With real-time capability, the system should offer:

Swift Response to Environmental Shifts: With the ability to process data on the fly, researchers can detect environmental changes immediately. This responsiveness allows for the adjustment of measurement strategies in real time, ensuring that transient events are captured and analyzed as they occur, rather than being potentially overlooked due to delayed analysis.

Immediate Anomaly Detection: Real-time analysis is not just about capturing data efficiently; it also plays a crucial role in quality control. By monitoring the instrument's performance in real time, any deviations from expected behavior—whether due to malfunctions or calibration drift—can be identified.

Informative Decision-making: Real-time analysis transcends the basic interpretation of raw sensor data by providing context and detailed insights that inform decision-making processes. This immediacy allows researchers to make informed decisions on the spot, which is critical for adapting to evolving atmospheric conditions.

Efficiency in Research: The capability to analyze data in real time eliminates the dependence on post-processing and accelerates the research workflow. Immediate data availability enables researchers to refine their experimental approach dynamically, adapting to the data as it is collected and maximizing the efficacy of their research activities.

The potential applications of such a real-time approach range from environmental monitoring to industrial applications.

For instance, in pollution monitoring [13], integrating real-time data pipelines with UAV systems could enhance their field effectiveness. Instead of waiting for lab analyses, operators could pinpoint pollution hotspots on the fly, capturing timely and pertinent data. Such immediate feedback could offer the public and officials timely alerts when pollutant levels rise critically. The immediate validation of data quality could further streamline the monitoring process, ensuring that only reliable data informs these decisions. Thus, real-time pipelines might hold the key to making air pollution monitoring more responsive and actionable.

Another example is in the oil and gas industry [2]. According to domain experts, drones, bolstered by advanced remote sensing capabilities, are set to redefine the oil and gas sector by amplifying precision, efficiency, safety, and environmental stewardship. Introducing real-time pipelines to these drones could fine-tune their detection capabilities. For instance, real-time analysis might help swiftly identify oil spills, enabling prompt containment actions and limiting environmental damage. Monitoring pipelines in real time could detect their issues in a more timely manner, preventing potential leakages and resource waste.

1.3 Contributions

This thesis enhances the functionality and utility of FIMS by developing a real-time analysis pipeline. The pipeline, built using C++ for efficiency and cross-platform compatibility, is structured with a multithreaded architecture that enables simultaneous, coordinated processing of distinct tasks. Key components of the pipeline include integrated sensor data parsing, image processing, and data inversion computation. To validate the effectiveness of our pipeline, we measure its accuracy vs. the previous offline FIMS analysis pipeline and evaluate its throughput and latency to determine the viability of conducting real-time analysis with limited computational resources.

Additionally, we design a strategy to integrate our real-time pipeline with the existing LabVIEW-based control system for FIMS. This integration involves redesigning an existing data-saving process performed by LabVIEW to enable it to safely run concurrently with real-time processing. We assess our design's performance, particularly its multitasking capabilities, through tests aimed at ensuring effective operation in portable and resource-limited environments. We note that integration and its performance evaluation were conducted in a simulated environment, as our collaborators have not yet commissioned FIMS' production hardware platform and LabVIEW environment We hope to validate our design integrated with FIMS under actual field conditions on a research flight in April 2024.

Finally, we consider the potential for extending this real-time analytical approach to other areas. We examine the implications for industries and fields that could benefit from swift and reliable data analysis, including the use of UAVs for pollution monitoring and surveillance in the oil and gas sector. This examination highlights the broader applications of real-time aerosol measurement technology and its possible impact on environmental monitoring practices.

Chapter 2

Background

2.1 Measuring Particle Size Distribution

In the field of aerosol science, measuring particle size distributions involves several steps: collecting samples, segregating aerosols by size, and quantifying them in specific size ranges over set intervals.

1. **Sample Collection**
   Aerosol particles are initially collected from the atmosphere using methods such as impaction, where particles are directed onto a surface, or filtration, involving air drawn through filters. The collected samples are then conditioned by controlling factors like humidity or temperature to preserve particle integrity before separation.

2. **Particle Separation**
   Following collection, samples undergo size-based separation. Commonly used methods include:

   - Differential Mobility Analyzer (DMA) [10, 17]: Particles are charged and then passed through an electric field. The field separates particles based on their electrical mobility, which is related to their size.
   - Aerodynamic Particle Sizer (APS) [26]: This method sorts particles by their aerodynamic diameter, using the time it takes for particles to settle under gravity or to be drawn through a known distance.
   - Centrifugal Methods: These use centrifugal force to separate particles based on their mass and size.

3. **Particle Counting and Sizing**
   After the separation process, particles of varying sizes are counted. Larger particles

can often be detected directly, but smaller particles typically require condensation for accurate counting. In this context, Condensation Particle Counters (CPC) [5, 21, 31] are commonly used. CPCs utilize a working fluid to create a supersaturated environment, enabling each particle to act as a condensation nucleus and grow to a detectable size. This technique allows the detection range of CPCs to extend to particles of size less than 3 nm.

Particle counting is generally conducted using two methods:

- Optical Methods: These methods involve illuminating particles and measuring the light they scatter. The intensity and pattern of this scattered light provide insights into the particle sizes.
- Electrical Low Pressure Impactor [21]: Often used following DMA, this method relies on measuring the electrical charge of particles, which correlates with their size. The charge's magnitude provides a basis for counting and sizing the particles.

4. **Data Analysis**
   Finally, the collected data are analyzed to determine particle concentration across different size ranges. This step often involves sophisticated algorithms and calibration to enhance accuracy and may consider variables like particle shape and composition.

2.2 Prior Aerosol Sizing Instruments

The Scanning Mobility Particle Sizer (SMPS) [30] has been the benchmark tool for measuring particle size distributions. The core of the SMPS system consists of a Differential Mobility Analyzer (DMA) coupled with a Condensation Particle Counter (CPC) [7]. It functions by classifying and counting particles of varied sizes sequentially. By adjusting voltages, the DMA selectively allows particles of specific mobilities, and hence sizes, to pass. Post-classification, these particles grow into detectable droplets, ready for optical detection. This stepwise ”scanning” from the smallest to the largest particle sizes can take the system between 1 to 10 minutes for a complete scan. Although SMPS provides accurate size distributions, its inherent scanning mechanism can pose challenges in environments demanding real-time aerosol property changes.

The Fast Mobility Particle Sizer (FMPS), inspired by the foundational principles of the Electrical Aerosol Spectrometer (EAS) from the University of Tartu [24], offers a method for assessing particle size distributions within the 5–560 nm range, boasting a time resolution of 1 second. In contrast to the Scanning Mobility Particle Sizer (SMPS), which sequentially determines particle sizes by varying voltages, the FMPS forgoes such voltage scans. Particles first undergo a controlled charging process and are then classified based on their electrical mobility using a central electrode system. This results in particles with different mobilities landing on different segments of a detector, achieving simultaneous size measurement. Such concurrent detection makes the FMPS suitable for environments with rapidly shifting aerosol properties. However, at low particle concentrations, its sensitivity can decrease due to the inherent noise in its electrometer setup, leading to size underestimations by 40–50% [15].

Another popular instrument class used for aerosol size distribution is Optical Particle Counters (OPCs), which harness light scattering for real-time particle measurement. As particles intersect a laser beam, photodetectors capture their scattered light, revealing their size. Unlike SMPS's sequential approach, OPCs have multiple channels that concurrently detect different particle sizes, providing near-instant readings across diverse sizes. However, OPCs typically focus on particles larger than 100 nm, and their measurements can be influenced by the particle's refractive index, which in turn is affected by size, chemical composition, and morphology—factors often uncertain for ambient aerosols.

Each aerosol measurement instrument has inherent strengths and drawbacks. While the SMPS is known for its precision, it does not offer rapid real-time data. The FMPS, although quick, might sacrifice accuracy in scenarios with low particle concentrations. OPCs, on the other hand, excel at real-time measurements but can be challenged by smaller particles and variations in refractive index. Recognizing the need for an instrument that marries swift response time with consistent accuracy, especially for fine particle measurements, the Fast Integrated Mobility Spectrometer (FIMS) [28, 29] was conceived. Building on principles similar to the SMPS, which utilizes electrical mobility for particle separation, the FIMS employs a spatially varying electric field. This design permits simultaneous measurement of particles ranging from 8 to 600 nm in a 1-second time frame. This makes FIMS particularly suited for transient environments such as in-flight or car-mounted measurements, or in tracking rapidly evolving events. Furthermore, the relative differences between the SMPS and FIMS in average particle size and total concentration are within 6.5% [32], aligning well with the established performance of SMPS.

2.3 Overview of FIMS Instrument Working Process

The Fast Integrated Mobility Spectrometer (FIMS) system comprises three main components: a separator, a condenser, and a detector. These components are arranged sequentially to form a rectangular channel, as illustrated in Figure 2.1. The process of deriving aerosol size distribution involves three primary steps: imaging particle positions, processing images to obtain FIMS response, and inverting aerosol size distribution from FIMS responses. The latter two steps are elaborated in [32].

Image Particle Positions in the Cross-section of the Channel

1. Initially, particles are charged and then introduced into a separator. Here, a spatially varying electric field is employed ensuring particles of different electric mobilities follow distinct trajectories when passing through this field, leading to their separation.
2. These separated particles proceed to a condenser, where they are enlarged through vapor condensation. This process of enlargement makes the particles more easily detectable.
3. At the exit of the condenser, the enlarged particles are illuminated by a laser light sheet and subsequently imaged by a CCD camera. A sample image is shown in Figure 2.2.
4. Particle counts and positions extracted from these images are used to determine particle sizes and concentrations.

Process Images to Obtain FIMS Response

1. The captured particle images are processed to separate overlapping particles and identify individual particle coordinates.
2. The detected position of each particle corresponds to an instrument response mobility, denoted as Z_p^*. This relationship is established using simulated particle trajectories and the Langevin equation [8]. The positions are converted into response mobilities and then response diameters D_p^* via the Stokes-Millikan equation [8].

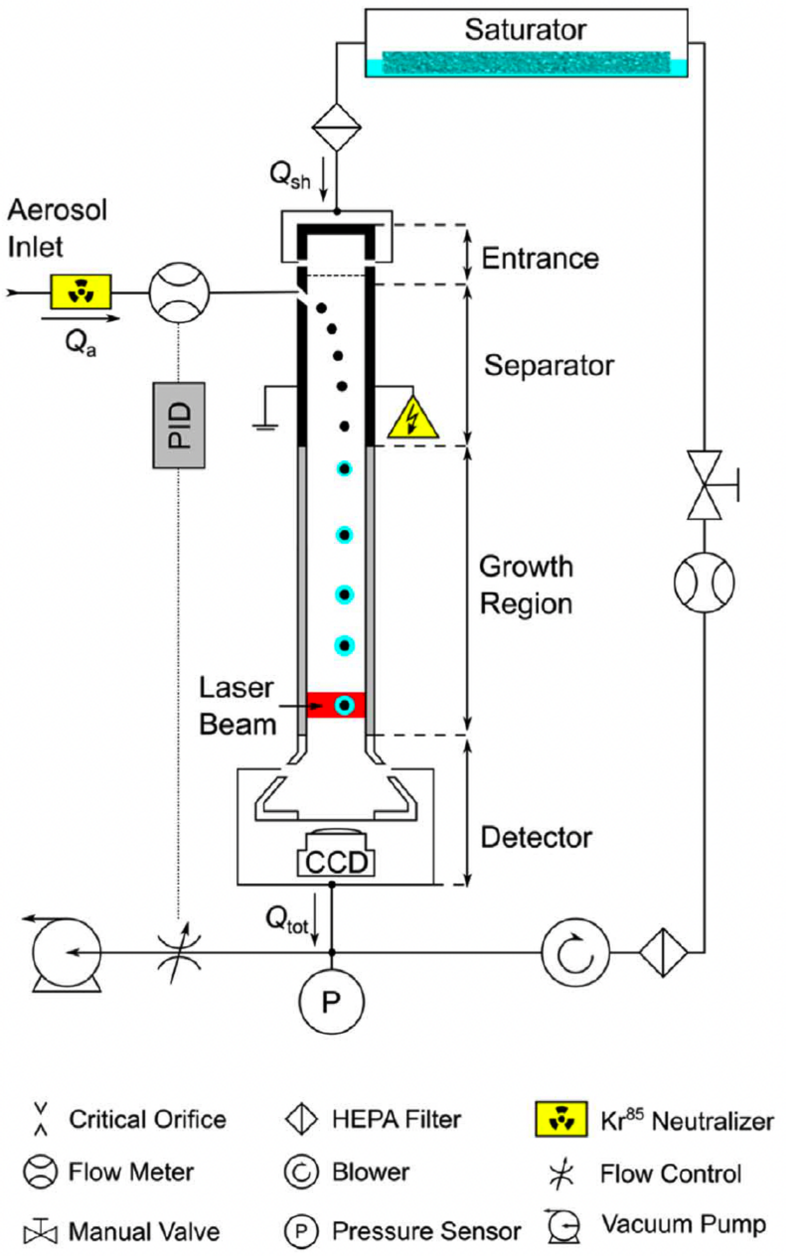

Figure 2.1: Image from Wang, Pikridas, et al. 2017, *Schematic of the FIMS*

3. Particle residence time in FIMS, varying with particle mobility due to the parabolic flow profile, is calculated as described in [19]. The arrival time at the separator is deduced by subtracting the residence time from the detection time, grouping particles into specific time intervals.

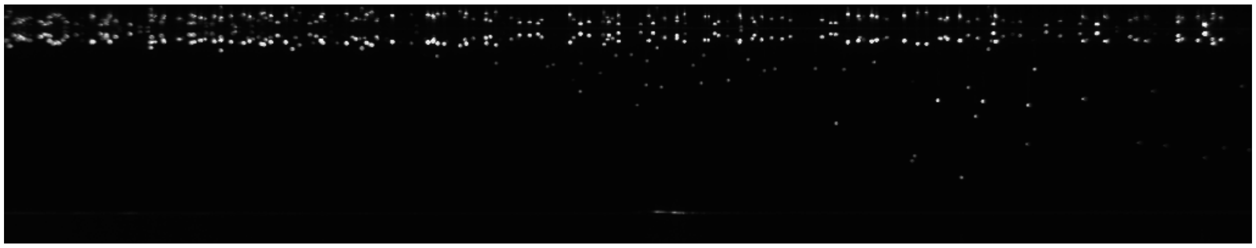

Figure 2.2: A sample image taken by CCD Camera *rotated 90 degrees counterclockwise*

4. Particles detected in each time interval are categorized into diameter channels. For each channel i ($i = 1, 2, 3, \ldots, l$), the FIMS response R_i quantifies the number of particles detected between the boundaries $D^*_{p,i-1/2}$ and $D^*_{p,i+1/2}$ within the interval. Thirty channels ($l = 30$) evenly spaced on a logarithmic scale are utilized, covering a size range of 10–600 nm. The overall FIMS response, denoted as $\vec{R}$, is the histogram of countsR_i across all channels for a given time interval.

Inversion of Aerosol Size Distribution from FIMS Responses

The inversion process is essential in converting FIMS response data, which provides instrument-related aerosol diameters, into accurate size distributions. It corrects for factors like charging rate, multiple charging, Brownian motion, and transfer function width that influence the number of observed particles differently at different mobilities.

1. The relationship between the response vector $\vec{R}$ and the actual aerosol size distribution $\vec{n}$ can be encapsulated by an inversion matrix, represented as $\mathbf{M}$. This matrix accounts for factors such as charging probability, Brownian motion, and the standardized width of the transfer function. The vector $\vec{n}$ quantifies the number of particles within each size range. The relationship is mathematically formulated as $\vec{R} = \mathbf{M} \times \vec{n}$.

2. To determine $\vec{n}$, the optimized Twomey inversion method [18] is employed, rather than linear inversion methods that struggle with the non-linearity and complexity inherent in aerosol size distributions [25]. In contrast, Twomey's non-linear method offers improved stability, error tolerance, and adaptability, effectively handling the varied patterns and dynamics in aerosol data to yield reliable and accurate results. This non-linear iterative technique calculates $\vec{n}$ given $\mathbf{M}$ and $\vec{R}$.

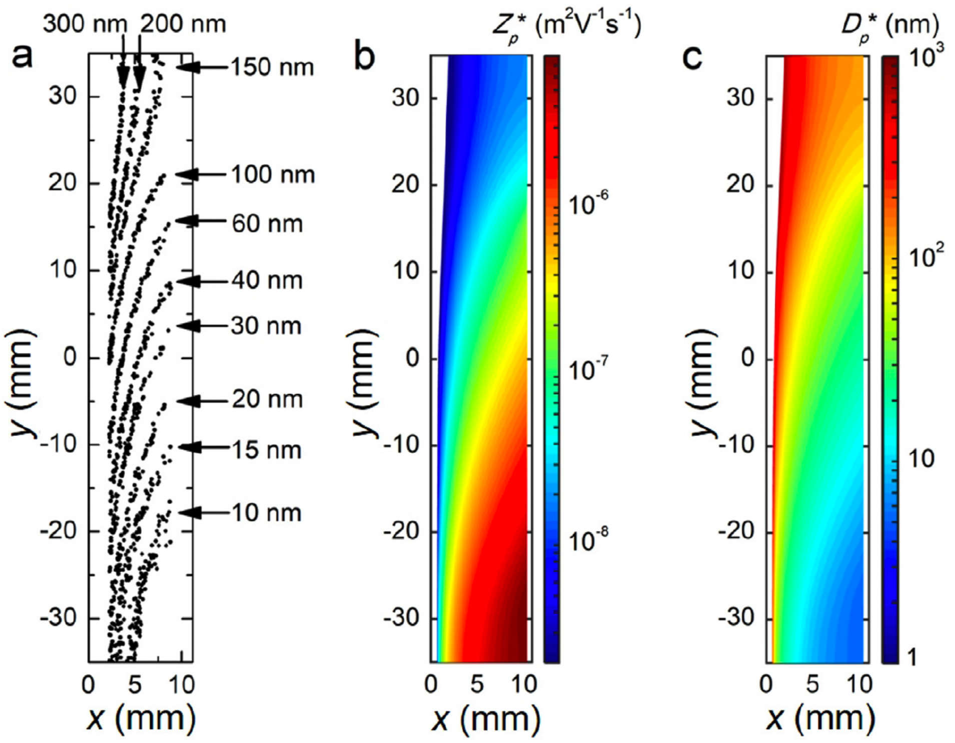

Figure 2.3: Image from Wang, Pinterich, et al. 2018, *FIMS Instrument Response*. (a) The FIMS camera-recorded locations of 10–300 nm monodisperse particles within the viewing window after mobility separation and growth. (b) and (c) show maps of instrument response mobility Z_p^* and diameter (D_p^*) derived from particle trajectories and positions at the separator exit simulated using the Langevin equation.

2.4 Current FIMS Aerosol Sizing Process

The current approach to FIMS aerosol sizing is a two-step process: online measurements conducted in flight, and offline analyses, carried out hours or days after the online phase. This approach requires significant storage due to the large volume of images generated by measurement.

2.4.1 Online Measurement

In the online phase, the primary focus is on directly measuring aerosols and recording associated data:

- Sensor metrics, including temperature, pressure, flow rate, and voltage, are logged at a 2Hz frequency with timestamps and stored in a CSV file.
- Aerosols processed through the FIMS are charged and amplified. Their representations are then captured by a high-speed CCD camera operating at 10Hz, with the resulting images saved in binary format.

Data is written to long-term storage (currently a USB nonvolatile memory stick) for subsequent analysis.

2.4.2 Offline Analysis

After the online measurement phase, the offline phase uses the stored data to complete the latter two steps of the FIMS pipeline: processing the images to obtain particle response distributions for each measurement time point, and inverting these distributions to compute aerosol size distributions. These analytical steps are presently performed in Matlab.

2.4.3 Limitations

The current FIMS aerosol sizing process, comprising online measurement and subsequent offline analysis, imposes a time gap between in-flight data collection and post-flight analysis, as well as a a need for storage to manage the large volume of image data. In dynamic aerosol environments, such as atmospheric research or pollution monitoring, the delay in computing aerosol size distributions can result in missed opportunities to observe transient atmospheric phenomena.

To avoid delays in analysis, we seek to construct a real-time analysis pipeline for FIMS. By enabling quicker data processing and interpretation, such a pipeline would reduce the interval between data acquisition and analysis, facilitating more responsive decision-making and adaptability to environmental changes. Moreover, it would streamline the overall analysis process, possibly even alleviating the need for extensive data storage if the computed particle size distributions can be stored instead of the raw images.

Chapter 3

Real-Time Analysis Pipeline Design

3.1 Overview of the Real-Time Pipeline

Traditional aerosol sizing techniques typically separate the processes of online data collection and offline analysis. This delay between measurement and analysis diminishes the utility of FIMS when investigating dynamic, rapidly evolving environments. A real-time system, on the other hand, can simultaneously measure and analyze data, immediately providing insights into aerosol size distribution. However, it needs to handle the complexities of simultaneous data acquisition and analysis. A key challenge in developing a real-time analysis pipeline for FIMS lies in managing the inherent asynchrony among various tasks including image reading and processing, housekeeping data reading and parsing, and synchronizing these processed datasets for subsequent data inversion.

We address this challenge by segmenting the workflow into three parallel, asynchronous components: an Image Processor for real-time imaging inputs, a Housekeeping Data Reader for sensor data, and a Data Inversion Component that combines outputs from the first two components to produce the final result. These components function asynchronously, each with its unique operational frequency. To harmonize their outputs, we utilize two global data queues, storing housekeeping data and detected particle data, respectively.

The responsibilities of each component are as follows.

Housekeeping Data Reader

The Housekeeping Data Reader serves as an adaptive interface, accommodating data from various housekeeping sensor versions. Each type of housekeeping sensor collects and delivers a slightly different set of parameters in a specific format. The Reader identifies the sensor type, parses the required data from each sensor's format accordingly, and standardizes the output into a consistent Housekeeping instance format.

- collects sensor readings such as temperature, pressure, flow rate, and voltage at a consistent frequency of 2Hz
- parses and structures the gathered data, placing it into a queue for the next steps in processing

Image Processor

The Image Processor's primary function is to discern particles within images, calculate each particle's inlet time, and allocate each particle to the appropriate bin for further instrument response computation.

- handles images taken at 10 Hz from the CCD camera as particles pass through the FIMS detector
- identifies particles in the images, translating their coordinates into real-world parameters
- determines the residence time of each particle, estimating its entry time into the instrument
- sorts identified particles into 1-second time bins stored in a chronological queue.

Data Inversion Component

The data inversion component synthesizes the processed outputs from the Image Processor and Housekeeping Data Reader, integrating data to derive an accurate aerosol size distribution.

- monitors the housekeeping and particle queues, advancing when certain conditions are satisfied

- extracts the housekeeping data, assessing parameters like voltage, temperature, and pressure to ascertain the need for inversion matrix recalculation
- retrieves and processes data from the foremost particle bin in the queue, generating corresponding FIMS response (a instrument mobility response histogram of all particles in the bin)
- applies the refined Twomey inversion technique [18] to transform the particle histograms into accurate size distributions at a 1 Hz frequency

3.2 Queuing Model of Real-time Pipeline

To assess the feasibility of the real-time pipeline, we first analyzed its queueing model, focusing on the efficiency of each system component. Our goal is to identify any potential bottlenecks that might hinder performance, particularly in resource-limited conditions.

A queuing model of the real-time pipeline is depicted in Figure 3.1.

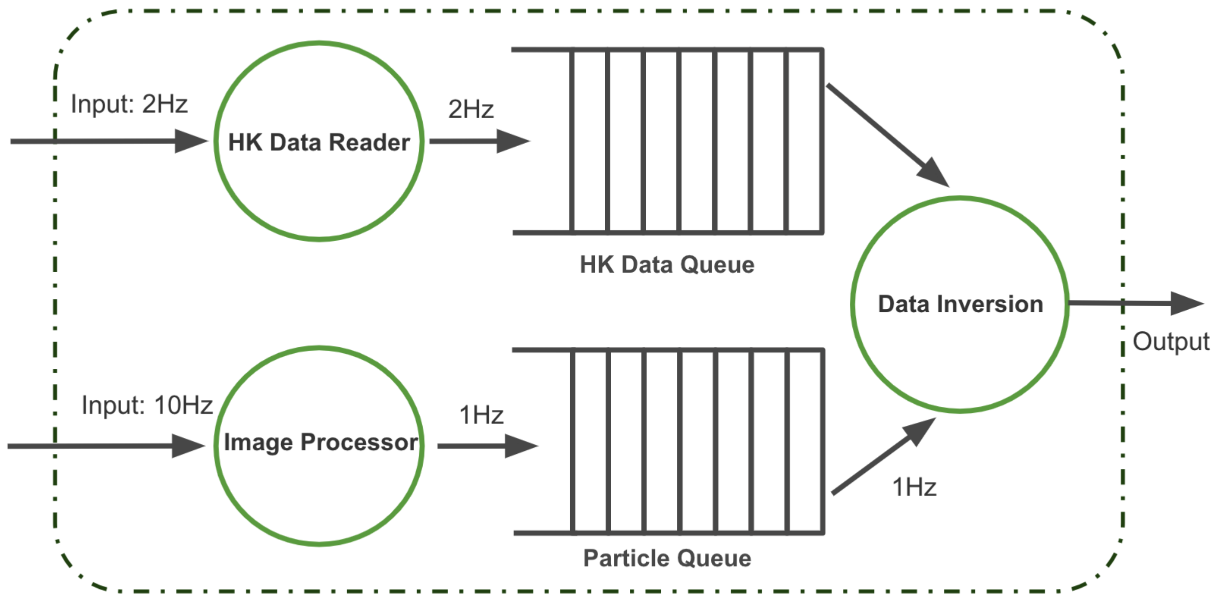

Figure 3.1: *Real-time Queue Model*

3.2.1 Arrival and Service Rates

In the queuing model of the real-time pipeline, the balance between data arrival rates (λ) and input queue service rates (μ) is critical for system efficiency. The optimal functioning of the pipeline requires service rates that meet or surpass these arrival rates.

Given the arrival rates:

- Raw Housekeeping (HK) data: arrives at $\lambda_{\text{HK}} = 2$ tasks/s
- Image data: arrives at $\lambda_{\text{Image}} = 10$ tasks/s

The corresponding service rates:

- HK Data Reader: must service more than $\mu_{\text{HK}} = 2$ tasks/s
- Image Processor: must service more than $\mu_{\text{Image}} = 10$ tasks/s

Maintaining these service rates ensures efficient data handling in the subsequent queues:

Housekeeping Data Processing: The HK data reader processes incoming raw data at a frequency of 2 Hz. This translates to parsing and queuing data at a rate of two entries per second into the subsequent HK data queue, leading to an arrival rate of 1Hz of the subsequent queue.

Image Processing: The Image Processor operates at a 10 Hz frequency, analyzing images to identify particle positions and calculate their resident times inside the FIMS. The *inlet time*, i.e., the time of arrival at the instrument, for each particle is then calculated using the formula:

$$\textit{inlet time} = \textit{frame time} - \textit{residence time}.$$

Particles are binned into 1-second bins based on their calculated inlet times. The output queue of image processing holds a sequence of time bins, into which particles are placed as they are discovered by image processing. A new bin is added to the queue whenever a particle's inlet time is newer than the newest bin already on the queue; otherwise, the particle is added to an existing bin. Except in the case of extremely sparse data, we expect that at least one new particle will be found each second, so bins are added to the particle queue at a typical rate of one per second.

Assuming residence times range from *min resident time* to *max resident time*, a bin that receives particles with inlet times between a given start and end could accumulate particles from image frames observed at times in the interval

$$[\textit{start time} + \textit{min resident time}, \textit{end time} + \textit{max resident time}].$$

Data Inversion Processing: The Data Inversion component processes the contents of a time bin on the queue and then removes the bin from the queue. During this process, the component also interpolates Housekeeping (HK) data, utilizing two data sets: one immediately preceding and the other immediately succeeding the time frame of the bin. Assuming bins appear on the queue at the full rate of one per second, the data inversion stage must maintain an average service rate of at least one inversion per second to maintain operational efficiency and prevent the system from becoming a bottleneck.

However, if the inversion stage removes a time bin from the queue before all particles associated with the bin have been detected, the late-arriving particles for that bin will be lost. Hence, we must impose an *intentional delay* between time bin creation and inversion to avoid losing these particles. Ideally, inversion should not consume a time bin until it has existed for at least *resident_time_max* seconds; we analyze the practical consequences of different delays in Section 4.2.2.

3.2.2 Performance Analysis on Different Platforms

Offline Analysis Method in MATLAB

The original offline analysis code, fully implemented in MATLAB, processes images in batches. Each batch consists of 600 images from a single binary file. After processing each batch, results are saved to text files before proceeding to the next file. Upon completion of all image processing, a data inversion step is performed where all detected particles are loaded into memory for further analysis. The MATLAB implementation was tested on a MacBook Pro with a 2 GHz Quad-Core Intel Core i5 running MacOS Ventura. The table below presents the average processing times for different tasks: image processing, data inversion[1], and Housekeeping (HK) data parsing. These times are measured per image, per inversion, and per set of HK data, respectively.

Platform	**Image Process Time (ms)**	**Data Inversion Time (ms)**	**HK Data Parsing Time (ms)**
MacBook	33	138	< 1

Table 3.1: Mean Processing Time of Offline Analysis Method

Transition to Real-Time Analysis in C++

While MATLAB is excellent for data analysis, algorithm development, and prototyping, its slower execution speed, resource intensity, and lack of fine-grained control over system resources make it less ideal for real-time applications. We therefore reimplemented the analysis pipeline in C++. This decision was driven by C++'s compatibility across different platforms, including embedded systems, its efficiency, and its ability to handle low-level operations, deterministic behavior, and robust support for multithreading and concurrency. Our C++-based real-time analysis pipeline utilizes OpenCV for image processing and the Eigen library for data inversion. The C++ implementation of the particle detection and channel wall position detection algorithms was based on work by undergraduate research assistant Jeffrey Gong.

Real-Time Analysis Pipeline Performance Evaluation

To evaluate whether the real-time pipeline consistently meets its throughput goals across diverse systems, we tested it on three distinct platforms:

[1]On data inversion: This process includes recalculating the inversion matrix, a step necessary when operational conditions such as temperature, pressure, and set voltage undergo significant changes. In vertical measurements, these conditions may vary more frequently due to substantial changes in temperature and pressure, while variations are less common in horizontal measurements. For this analysis, a worst-case scenario is assumed, where the inversion matrix is recalculated every time, to provide a comprehensive understanding of processing times under varying conditions. This assumption is also applied to all subsequent running time measurements.

- **Raspberry Pi 4 (Linux):**
    - **Processor:** 1.5 GHz Quad-Core Cortex-A72 (ARM v8)
    - **Purpose:** To evaluate performance on energy-efficient systems, typical in embedded applications.
- **MacBook Pro (MacOS):**
    - **Processor:** 2 GHz Quad-Core Intel Core i5
    - **Purpose:** To assess functionality on consumer-grade MacOS laptops, offering insights into everyday computing environments.
- **Desktop PC (Windows):**
    - **Processor:** 3.6 GHz 12-Core i7-12700k
    - **Purpose:** To test on a high-performance Windows platform, representing advanced computational capabilities.

We used wall-clock time to gauge the duration from the start to the finish of each execution. Although wall-clock time may include periods of preemption and waiting and is susceptible to external factors not directly related to the process, it provides a straightforward and universally applicable metric for assessing real-world performance across systems with varying hardware and software. We utilized `std::chrono::high_resolution_clock` to capture these timings for multiple runs and get the mean processing times for each task across these varied platforms. The results are as follows:

Platform	**Image Process Time (ms)**	**Data Inversion Time (ms)**	**HK Reading Time (ms)**[2]
Raspberry Pi	39	41	< 1
MacBook	10	17	< 1
Desktop PC	15	9	< 1

Table 3.2: Mean Processing Time Per Input Across Different Platforms

Using these mean service times, we calculate the service rates for each component using the formula:

$$\text{Service Rate}(\mu) = \frac{1}{\text{Mean Process Time (s)}}$$

[2]The Housekeeping (HK) Reading process operates at a high speed, often too rapid to be accurately measured by the high_resolution_clock with its 1ms resolution. As a result, the durations for this process are frequently undetectable and recorded as 0 ms. To reflect this, we denote the speed of the HK Reading process as being less than 1 ms ($< 1ms$)

The service rates for each platform are detailed in Table 3.3. The results indicate that, on all platforms, the service rates for each component of the pipeline comfortably exceed the data arrival rates.

Platform	Image Process Rate (ops/s)	Data Inversion Rate (ops/s)	HK Reading Rate (ops/s)
Raspberry Pi	25	24	> 1000
MacBook	97	59	> 1000
Desktop PC	68	111	> 1000

Table 3.3: Service Rate Across Different Platforms

Finally, to assess whether the system's processing work fits within the periods dictated by data arrival rates, we analyzed each component's utilization, defined as the ratio of the arrival rate (λ) to the service rate (μ):

$$\rho = \frac{\lambda}{\mu}$$

A utilization near or above 1 indicates a bottleneck. A utilization rate significantly lower than 1 indicates that the system's capacity is not being fully utilized, suggesting that there is spare processing power available that could potentially be harnessed for additional tasks or more efficient operations.

The analysis results, as shown in Figure 3.4, reveal that service rates for all of the pipeline components — Image Processing, Data Inversion, HK Reading — comfortably exceed arrival rates on all platforms. However, the Image Processor shows a much higher utilization compared to Data Inversion and HK Reader, especially on the Raspberry Pi (0.4, compared to $0.041 < 0.002$ for the other two components.) This suggests that the Image Processor could be the potential bottleneck when running on a platform with constrained resources or in a busy multitasking environment. The real-time pipeline could therefore benefit from optimizing the Image Processor for better overall performance.

Platform	Image Processing Utilization	Data Inversion Utilization	HK Reading Utilization
Raspberry Pi	0.4	0.041	< 0.002
MacBook	0.103	0.017	< 0.002
Desktop PC	0.147	0.009	< 0.002

Table 3.4: Utilization Across Different Platforms

Comparison to Matlab

We compared the running time of the C++ implementation to that of the original Matlab code on our MacBook platform. The results, detailed in Table 3.5, show that the C++-based real-time pipeline outperforms the MATLAB version in terms of processing speed.

Implementation	**Image Process Time (ms)**	**Data Inversion Time (ms)**	**HK Reading Time (ms)**
MATLAB	33	138	< 1
C++	10	17	< 1

Table 3.5: Running Time Comparison between MATLAB and C++

3.3 Worst-Case Running Times

Examining the average running times, as we did above, is necessary to assess the practicality of the FIMS real-time analysis pipeline. However, understanding the worst-case running time (WCRT) is equally important, especially in a real-time system context. Although the FIMS pipeline operates under soft deadlines, where missing deadlines does not cause system failure, frequent misses of these soft deadlines result in a degradation of the quality of service. For instance, they may delay the availability of real-time analysis results, impacting the timeliness of the data provided to users. In this section, we explore the worst-case running time of the pipeline and diagnose potential problems that cause long worst-case running time.

To determine the worst-case running time (WCRT) of the pipeline, we employed the Raspberry Pi 4 Model B as our test platform given its minimalistic operating environment and the inherent stability of the Linux operating system. This device is powered by a 4-core, 64-bit Cortex-A72 (ARM v8) CPU and operates under Linux 5.10.103. We measured both the Worst Case Wall-clock Time and the Worst Case CPU Time to learn the pipeline's WCRT.

Wall-clock Time: The wall-clock time measures the total elapsed time for each task, providing a real-world measurement of the time taken from initiation to completion. In a real-time setting, this measure encompasses all aspects of the system's operation, including potential delays from external sources and multitasking overhead. The wall-clock time was captured using `std::chrono::high_resolution_clock` from the beginning of each task to the end of each task.

CPU Time: CPU time is the time during which the CPU is actively processing the task, discounting any periods of idleness or waiting for resources. We used `getrusage()` to capture the CPU time from the end of last execution to the end of this execution. The CPU time is then determined by summing the user time, which represents the direct time spent by the CPU executing the process, and the system time, indicating the time the system spends on behalf of the process.

Initial Observations and Concerns

In our initial assessment of the Raspberry Pi 4 Model B, we observed that while the average running times for image processing (50.4ms) and data inversion (39.1ms) were within acceptable limits, the worst-case scenarios revealed potential issues with system stability. Notably, peak running times reached as high as 90.0ms for image processing and 93.0ms for data inversion. This is particularly concerning for image processing, as it approaches the image inter-arrival time of 100ms. These findings are illustrated in Figure 3.2.

In scenarios with denser particle distribution in images, the worst-case performance could be worse than what we observed, which elevates the risk of missed deadlines, especially in high-demand operational environments. Although our system is designed to operate under soft deadlines, where missed deadlines do not immediately result in system failure, the observed inconsistencies could still significantly impact the timeliness and reliability of data analysis. Therefore, optimizing the stability of the system to mitigate these risks is necessary to ensure consistent performance and reliable data processing.

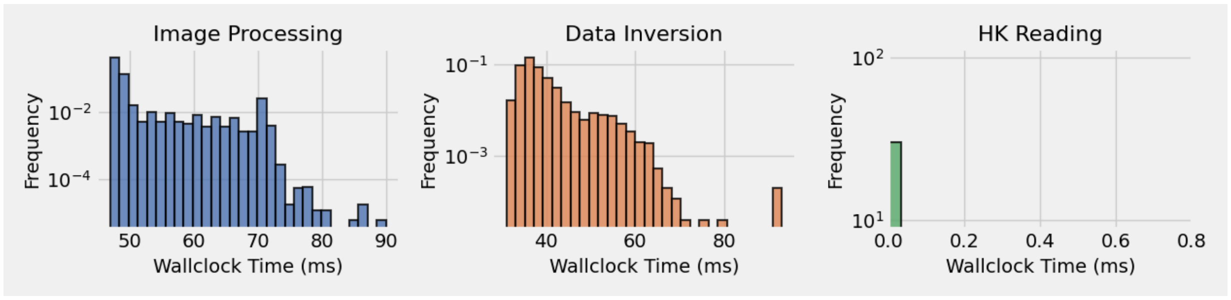

Figure 3.2: *Wall-clock Running Time Using Default Scheduler*

Diagnostic Analysis

To investigate the root cause of the instability observed in the real-time pipeline's performance, we initially hypothesized that frequent preemptions by higher-priority threads were responsible. To test this hypothesis, we conducted a retest using a real-time scheduler

(SCHED_FIFO). In this setup, we assigned the highest priority to our real-time processes and removed the default CPU usage limit of 95%. Surprisingly, this adjustment did not lead to the expected improvements. Instead, the worst-case running times worsened, with image processing and data inversion times escalating to 131ms and 120ms, respectively. These results are illustrated in Figure 3.3.

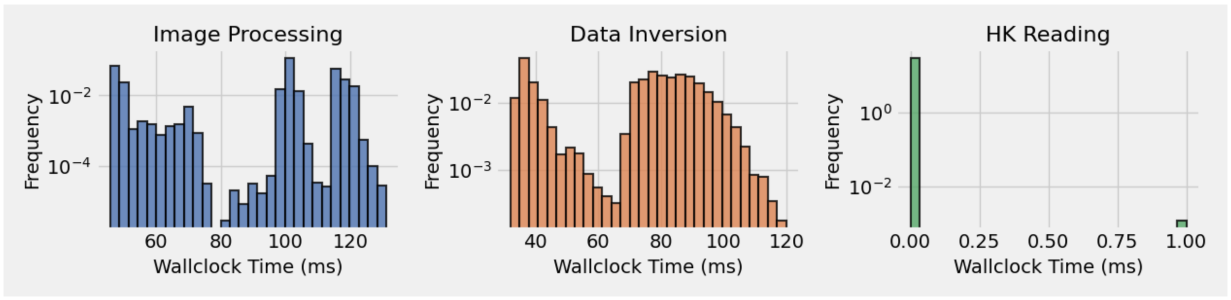

Figure 3.3: *Wall-clock Running Time Using Real-time Scheduler*

Seeking a deeper understanding of the performance issues, we turned to CPU time analysis using the `getrusage()` function, focusing on user and system times. This analysis revealed similar patterns of instability: under the default scheduler, the worst-case CPU times were 88ms for image processing and 91ms for data inversion (Figure 3.4). Switching to the real-time scheduler, these times increased to 130ms and 117ms (Figure 3.5). The alignment of CPU time with wall-clock time indicated that preemption was not the primary issue.

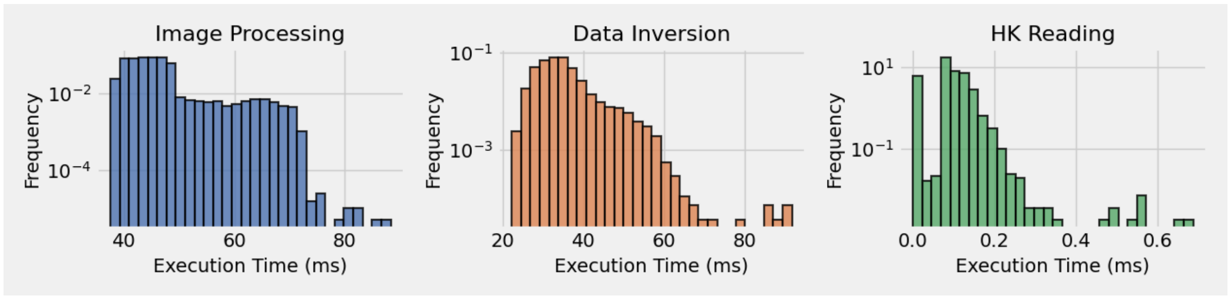

Figure 3.4: *CPU Time Using default Scheduler*

To further our investigation, we utilized Kernel Shark for a detailed analysis of process execution times. Figures 3.6 and 3.7 present snapshots from Kernel Shark, with purple bars representing the image processing thread and green bars for the data inversion thread. Figure 3.6 highlights the variable execution times for both processes. In contrast, Figure 3.7 shows significantly prolonged execution times for image processing, with intervals where

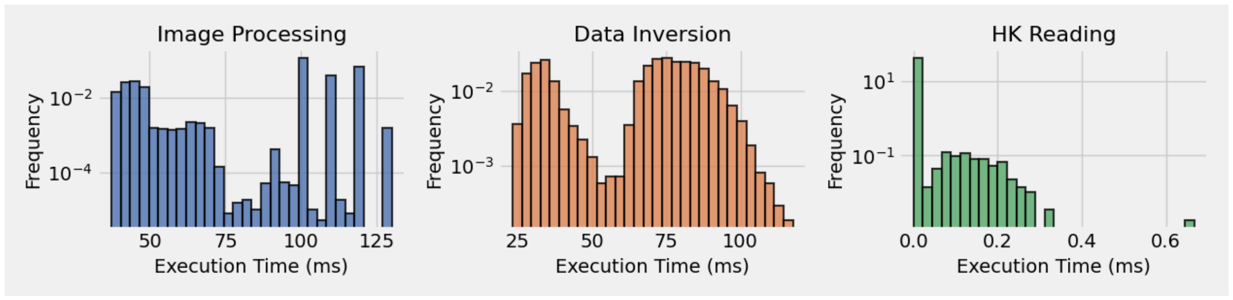

Figure 3.5: *CPU Time Using Real-time Scheduler*

the bars are nearly contiguous. A notable observation from these figures is the concurrent prolongation of both image processing and data inversion times, suggesting a correlation in their execution patterns.

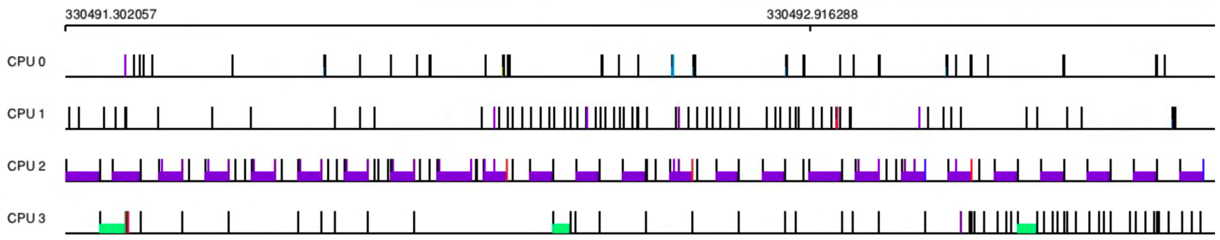

Figure 3.6: *Kernel Shark Captured Varying Execution Time*

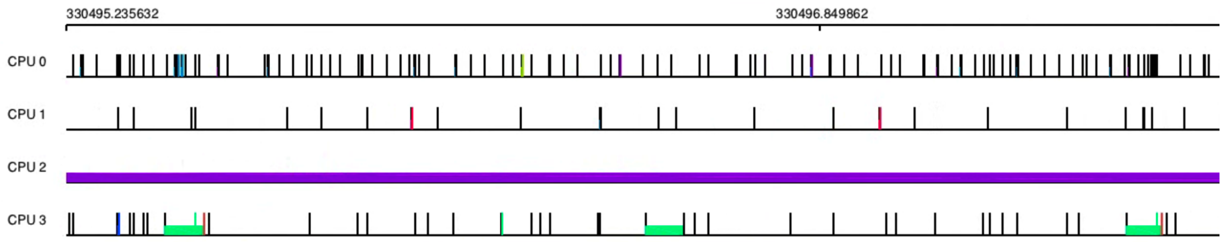

Figure 3.7: *Kernel Shark Captured Prolonged Execution Time*

This concurrent extension in processing times led us to hypothesize that CPU throttling might be the underlying cause of the observed instability in execution times.

Addressing CPU Throttling

After identifying CPU throttling as a potential factor affecting system performance, we disabled it to reassess the efficiency of our pipeline. This adjustment led to a significant improvement in system stability. With the default scheduler, as illustrated in Figure 3.8, image processing times stabilized between 30ms and 41ms, and data inversion times ranged from 31ms to 52ms. When the real-time scheduler was enabled, shown in Figure 3.9, image processing times varied between 30ms and 37ms, and data inversion times were between 31ms and 48ms. On average, the image processing times improved to 31ms, and data inversion times to 37ms across both scheduling strategies. These averages, similar to those observed with the default scheduler, underscore the beneficial impact of disabling CPU throttling in enhancing overall system performance. Additionally, in our implementation where CPU throttling was disabled, the processor's temperature peaked at a safe 47.2°C, significantly below the Raspberry Pi's safety threshold of 80°C. This indicates that even without CPU throttling, the device operated well within its thermal limits. Furthermore, it's important to note that Raspberry Pi systems are designed to automatically reduce CPU frequency under high-temperature conditions, a critical safety measure independent of user configurations. Therefore, the decision to disable CPU throttling for improved system efficiency was effective and did not introduce overheating risks.

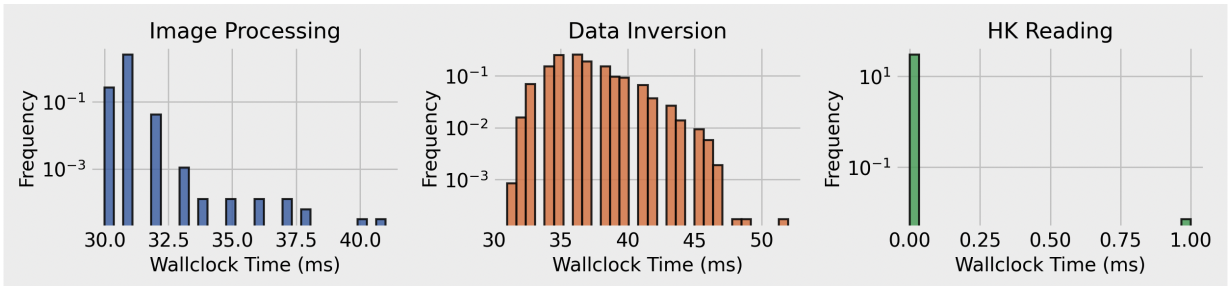

Figure 3.8: *Wall-clock Running Time Using Default Scheduler Post-Disabling CPU Throttling*

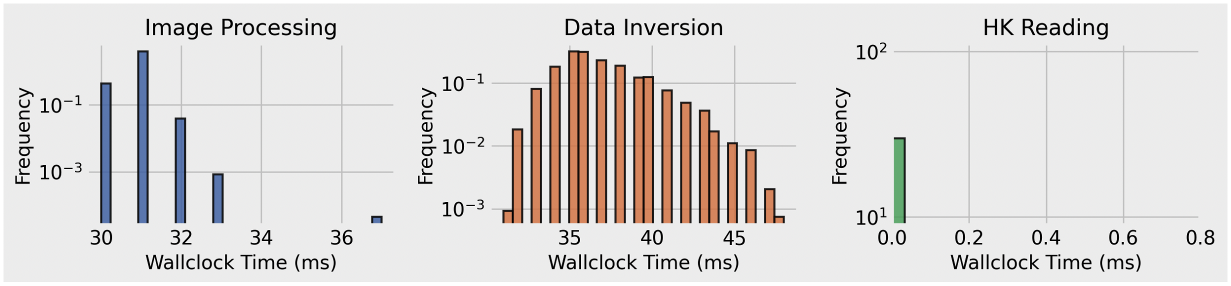

Figure 3.9: *Wall-clock Running Time Using Real-time Scheduler Post-Disabling CPU Throttling*

The improved stability was also reflected in the corresponding CPU times, as demonstrated in Figures 3.10 and 3.11, which illustrate the performance under both the default and real-time schedulers. In each scenario, the peak CPU times for image processing settled at around

43.4ms, with an average of 31.5ms. For data inversion, the peak times were approximately 56.5ms, averaging at 38.0ms[3].

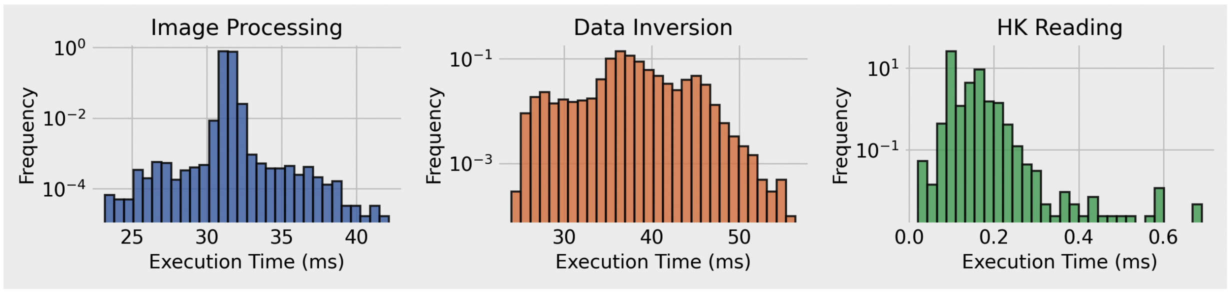

Figure 3.10: *CPU Execution Time Using Default Scheduler Post-Disabling CPU Throttling*

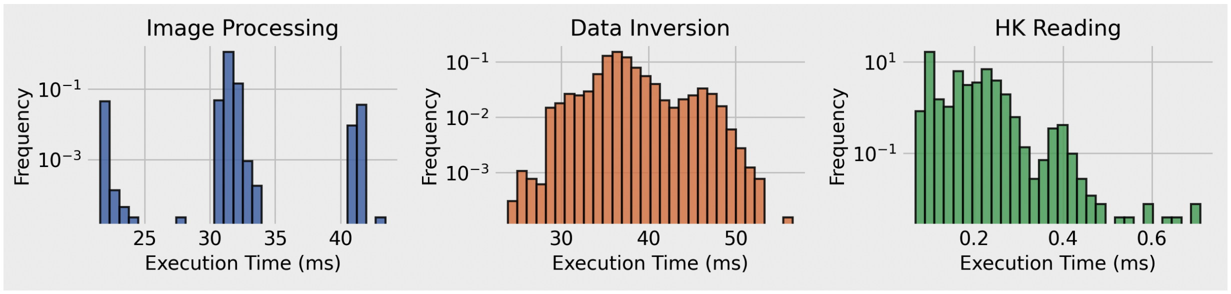

Figure 3.11: *CPU Execution Time Using Real-time Scheduler Post-Disabling CPU Throttling*

3.4 Using the Real-Time Analysis Pipeline in a Multitasking Scenario

We have optimized and stabilized running time for the real-time pipeline when operating in an isolated environment, free from additional processing tasks with a relatively clean background. However, in practical applications, the pipeline is likely to function in a multitasking scenario. This involves simultaneous operations with upstream processes — responsible for reading images and housekeeping (HK) data from sensors — and downstream processes that handle result visualization.

[3]It is important to note that since we measured the wall-clock time from the start to the end of each execution, it does not include the context-switching costs between executions. Our CPU time measurement accounts for these switching costs. Therefore, it is not surprising that the CPU time is slightly longer than the wall-clock time.

In such real-world settings, the pipeline must not only process data in real time but also contend with the demands of concurrent tasks. Particularly challenging are scenarios involving intensive disk-writing activities, where a dedicated thread continuously saves images to disk. This multitasking environment may significantly affect the performance of the real-time pipeline.

In this, section we examine the pipeline's performance under multitasking conditions. Our approach entails integrating the pipeline with LabVIEW for upstream data input, coupled with a modified mechanism for disk-writing operations. The primary objective is to identify and mitigate any potential performance degradation resulting from these concurrent operations.

3.4.1 Interfacing LabVIEW with the Real-time Analysis Pipeline

The FIMS instrument, inclusive of its sensors and CCD camera, operates on LabVIEW, which excels in handling complex system engineering tasks and facilitates rapid hardware interfacing. A seamless integration between LabVIEW and the real-time analysis pipeline is essential to ensure immediate data acquisition, crucial for keeping the pipeline in sync with the data collection systems.

Our main challenge was to establish a robust connection between LabVIEW's data acquisition features and our C++-based real-time analysis pipeline. After considering various options, we opted to leverage LabVIEW's capability to interface with Dynamically Linked Libraries (DLLs) written in C/C++. This choice was driven by the efficiency of DLLs in data transmission, particularly their ability to pass data pointers directly, thereby minimizing the overhead associated with data copying. LabVIEW's direct interaction with these libraries enables it to execute C++ functions encapsulated within. This inter-process communication method is advantageous in real-time systems due to its low overhead.

Data Transfer Mechanism

The real-time pipeline interfaces with LabVIEW through a DLL. This DLL features two external functions: one for handling the transmission of images and timestamps from LabVIEW, and the other for processing HK data. These functions serve as communication bridges, allowing data to be efficiently exchanged between LabVIEW and the pipeline. The data flow is as shown in Figure 3.12.

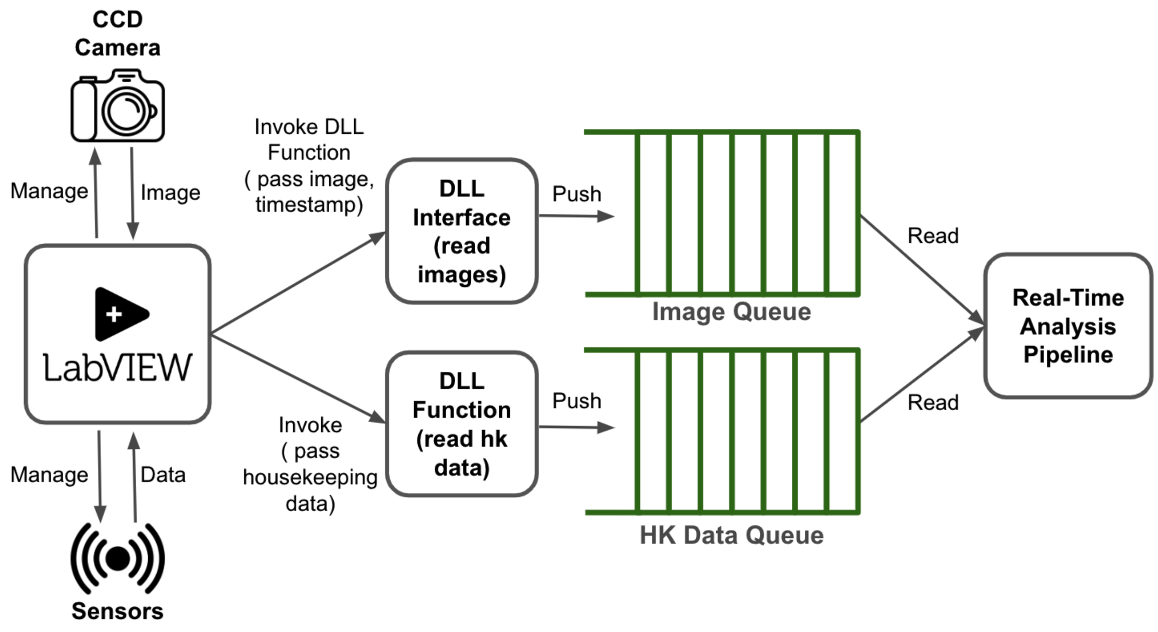
CCD Camera
Manage
Image
Invoke DLL Function (pass image, timestamp)
DLL Interface (read images)
Push
Image Queue
Read
Real-Time Analysis Pipeline
LabVIEW
Invoke (pass housekeeping data)
DLL Function (read hk data)
Push
HK Data Queue
Read
Manage
Data
Sensors

Figure 3.12: *LabVIEW integration Data Flow*

Image Transfer: Images captured at a frequency of 10Hz are sent from the cameras to LabVIEW, which then invokes the DLL function, passing a pointer to the image and its corresponding timestamp. The DLL retrieves this data and places it into an image queue, subsequently signalling the C++ pipeline of the new image's availability through a condition variable.

Housekeeping Data Transfer: In parallel, LabVIEW transmits HK data to the pipeline at a frequency of 2Hz. When new data arrives, LabVIEW triggers the corresponding DLL function, which then processes and stores the data in the hk data queue, alerting the C++ pipeline for its retrieval.

3.4.2 Optimizing Image Saving Process

An essential component of the FIMS system is the reliable backup of captured images to disk. In its current configuration, FIMS utilized LabVIEW to capture images from the camera and save them to disk or to a USB stick in real time. However, this approach occasionally loses images before they can be saved to long-term storage. This is likely due to the execution of image capture and saving operations being handled sequentially by LabVIEW, using a finite circular queue to hold images. Delays incurred by backups writing buffered file data to the storage device can result in images being overwritten in the queue before they are saved. Moreover, delays in saving images could potentially block execution of the real-time analysis pipeline.

To address these concerns, we decoupled the image-saving function from LabVIEW and from our pipeline. In this revised setup:

1. When LabVIEW captures an image, it invokes a DLL function, passing the image pointer.
2. The DLL function places the image and its timestamp into two distinct queues. One queue is dedicated to real-time analysis, and the other is for disk storage.

This separation ensures that the processes of real-time analysis and image saving can operate independently from LabVIEW and from each other, using separate CPU threads. Consequently, this change minimizes the impact of disk writing operations on the real-time performance of our pipeline. The revised data flow is shown in Figure 3.13.

We note that delays incurred in writing data to long-term storage could still result in lost images, given that we still utilize a finite queue to store image data pending this write. However, any such failures are now isolated from the rest of the analysis process and can be addressed with separate OS- and filesystem-specific mitigations.

3.4.3 The Real-time Result Display Unit

Adding real-time analysis capabilities to FIMS in turn requires prompt results visualization to provide users with immediate insights into the prevailing aerosol size distribution. By

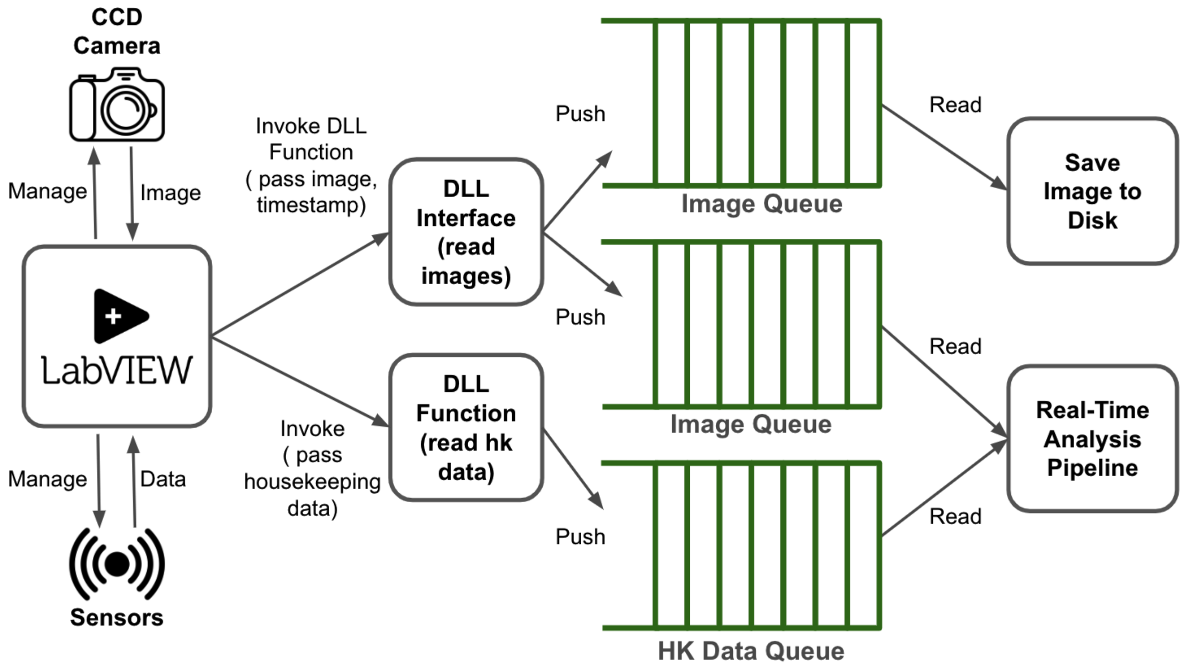
CCD Camera
Manage
Image
Invoke DLL Function (pass image, timestamp)
DLL Interface (read images)
Push
Image Queue
Read
Save Image to Disk
LabVIEW
Push
Image Queue
Read
Real-Time Analysis Pipeline
Invoke (pass housekeeping data)
DLL Function (read hk data)
Push
HK Data Queue
Read
Manage
Data
Sensors

Figure 3.13: *Data Flow with Image Saving*

looking at the visualization, users can quickly understand the current particle size in the atmosphere. This is important for researchers who might need to make quick decisions, like changing an aircraft's direction for better sample collection.

FIMS may be used in different scenarios that necessitate different approaches for visualizing results. For example, in field test scenarios where the instrument is stationary, researchers may prefer to access data remotely from the laboratory. To facilitate this, results can be transmitted via TCP/IP using socket communication, allowing lab personnel to remotely monitor aerosol size distributions. In contrast, onboard testing scenarios, such as those on aircraft, require may direct display of results on machine doing the analysis for swift decision-making. Additionally, there may be scenarios where accessing real-time results via a mobile app is desirable. Addressing these varied requirements will be part of future work.

In the current testing phase, we have crafted a Python-based application to visualize real-time results. This application operates using FIFO for communication on Linux machines and sockets for interaction with Windows systems or different Linux machines. The primary objective of this development phase is to thoroughly evaluate the process flow, from data collection to the presentation of results, and to explore the form of displaying real-time aerosol data.

As shown in Figure 3.14, the visualization updates in real-time to provide immediate insight into the particle size distribution. The result is shown in the form of a heatmap: the colours show particle concentration; the vertical axis shows particle size; and the horizontal axis shows time. The display keeps updating every second, moving left and adding the newest data on the right. The particle size distribution is our direct output from the analysis module, and we can also transform these results into integrated surface area or volume concentrations.

The display unit is categorized as a dynamic component within the system architecture, capable of supporting various presentation modalities, including local and remote visualization, and versatile display formats like scrolling heatmaps. To maintain the focus of our performance evaluation, this unit will not be included.

3.4.4 Multitask Running Time Assessment and Optimization

FIMS typically operates with a LabVIEW control platform on Windows. However, accurate execution time measurement of specific threads on Windows poses significant challenges due to the complex scheduler, the presence of numerous background processes, and the dynamic management of thread priorities. These factors can introduce variability in runtime measurements, particularly in a multi-threaded environment like the real-time pipeline for FIMS.

An additional challenge is that the FIMS instrument is still under calibration and so is not yet ready for integrated testing and deployment with its actual processing hardware. Given these circumstances, we have elected to study multitasking behavior of the FIMS pipeline on a Raspberry Pi 4. This approach allows us to produce results now and also circumvents the measurement challenges posed by the Windows environment. The Pi also offers more precise and granular control over CPU time measurements for individual threads for performance measurement.

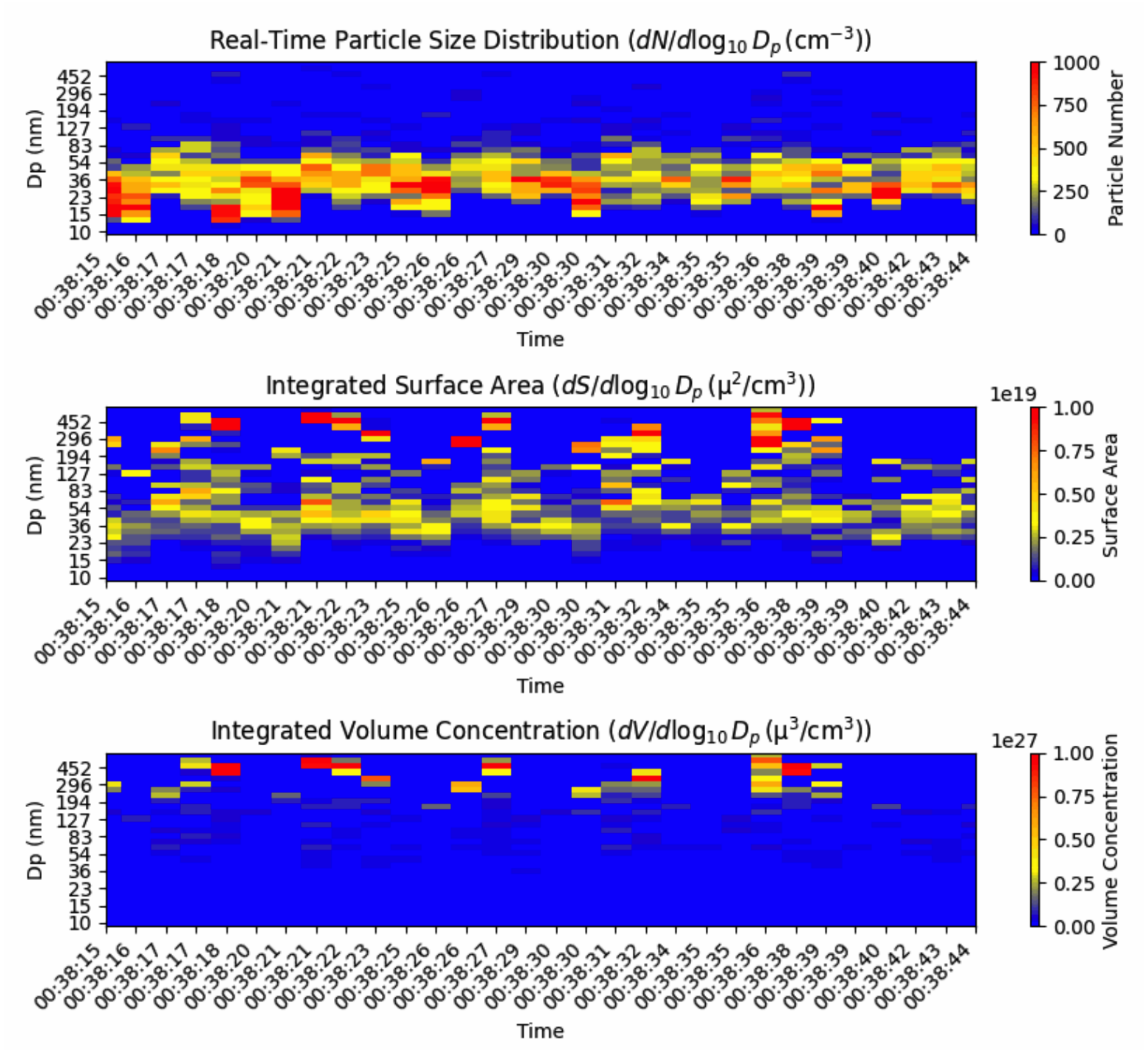

Figure 3.14: Real-time Result Visualization *(This is an animation that scrolls to the right every second)*

3.4.5 Simulation Setup and Execution

The simulation uses a C++ program to replace LabVIEW, handling the image and housekeeping (HK) data acquisition. All images and HK data are preloaded into memory. The program then periodically invokes the DLL interface, transmitting an image every 100ms and HK data every 500ms. This setup mimics the pipeline's real-world operating conditions.

There are six simulation components, each running in a separate thread to simulate concurrent operations:

- **Image Acquisition:** Interacts with the DLL interface to feed images into two separate queues at a rate of 10Hz. One queue is designated for image processing; the other, for image saving.
- **HK Acquisition:** Queues raw HK data by invoking the DLL interface at a frequency of 2Hz.
- **Image Processor:** Processes images as soon as they are available in its queue, subsequently pushing the results to a particle data queue.
- **HK Data Reader:** Processes new sets of raw HK data from its queue, pushing the processed data to a separate HK data queue.
- **Data Inversion Component:** Retrieves data from both the particle and processed HK data queues to perform data inversion tasks.
- **Image Saver:** Responsible for retrieving images from its queue and saving them to disk.

These threads operate under the SCHED_FIFO real-time scheduling policy. Thread priorities are as follows: Image Acquisition, HK Acquisition, and Image Processor (99, the highest priority), HK Reader (98), Data Inversion (97), and Image Saver (90). CPU throttling is disabled for consistent performance.

Each thread is allocated to a specific processor of the 4-core Raspberry Pi to minimize interference and optimize performance. The Image Saver operates on processor 0; Image and HK Acquisition, and the HK Reader on processor 1. The Image Processor is assigned to processor 2, and Data Inversion is handled by processor 3. Processors 1, 2, and 3 are isolated from the general scheduler to ensure dedicated processing for these tasks.

Figure 3.15 presents the CPU execution times for the six components involved in our simulation. To simplify the measurement process, the execution time for each component was captured using `clock_gettime()` instead of `getrusage()`[4]. The CPU execution time calculates the duration from the end of one execution cycle to the end of the next.

The CPU execution times for the Image Processing, Data Inversion, and HK Reading components, averaging 29.0ms, 39.85ms, and 0.08ms respectively, align closely with standalone test results. Any minor discrepancies may result from differing measurement methods. Additionally, the Image Saving, Image Acquisition, and HK Acquisition components demonstrate minimal execution times, averaging 1.01ms, 0.04ms, and 0.11ms, respectively.

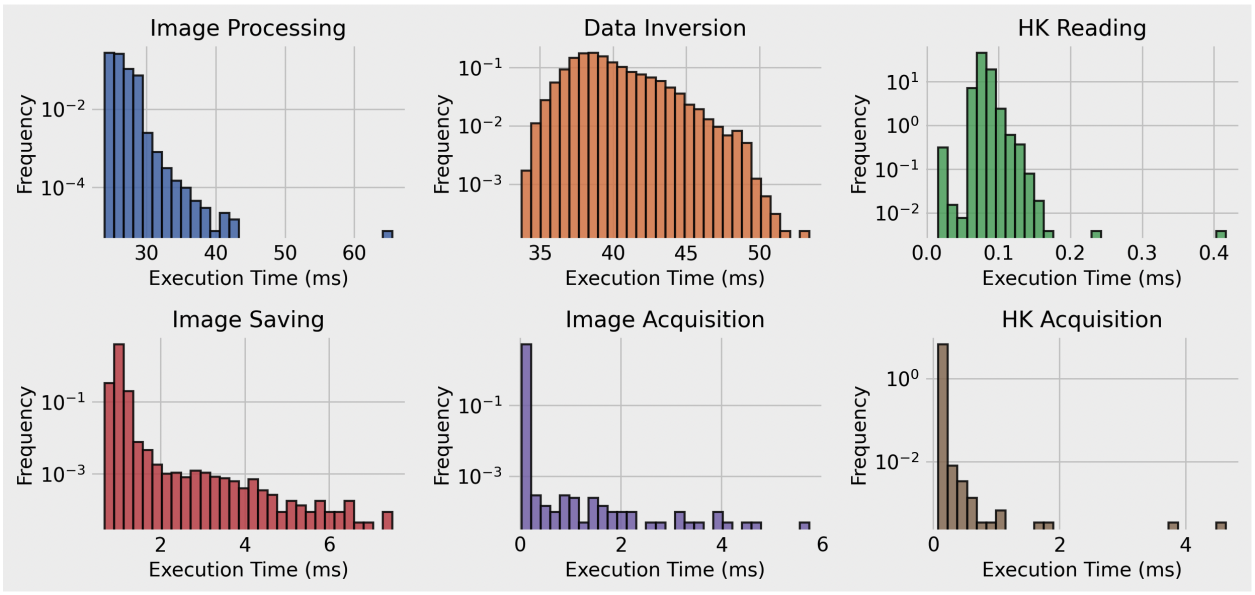

Figure 3.15: *Execution Time of All Components in Multitasking Scenario*

The wall-clock time analysis, as shown in Figure 3.16, highlights a significant anomaly in the Image Saver thread. It demonstrates unusually prolonged execution times, reaching as high as 4650.0ms. We suspect this excessive delay might stem from several factors. Firstly, the operating system's multilayer buffering for disk I/O operations plays a role. As data is written to disk, it's buffered across multiple OS layers, creating larger data blocks. This process, while generally efficient, can lead to bottlenecks, especially with the high-volume data transfers involved in continuous image saving. When these buffers fill up, additional data writes are temporarily halted, potentially increasing execution times. Secondly, the inherent

[4]The `clock_gettime()` function primarily measures thread-level CPU time, focusing on the precise tracking of a thread's execution. In contrast, `getrusage()` offers a comprehensive view of resource usage, distinguishing between user and system CPU times. `getrusage()` may also impose more overhead. We observed that the total CPU time obtained from `getrusage()`, which sums user and system times, may be slightly higher than the time measured by `clock_gettime()` for the same task under similar conditions.

limitations of the Raspberry Pi's SD card write speed may also contribute to these delays, particularly noticeable when saving image files. We plan to conduct further investigations to confirm these hypotheses.

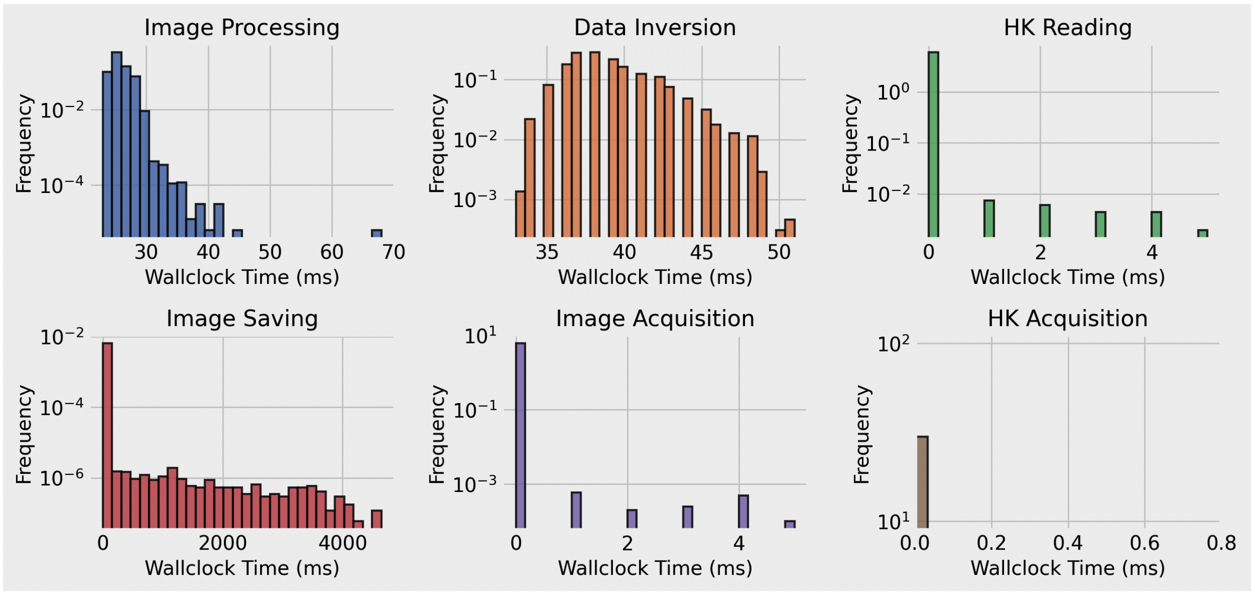

Figure 3.16: *Wallclock Time of All Components in Multitasking Scenario*

The stability of the real-time pipeline in multitasking conditions is evaluated by examining the inter-execution time, defined as the duration between the start of one execution cycle and the start of the next. This metric is captured using `std::chrono::high_resolution_clock` and is crucial for assessing the consistency of the pipeline's performance. Figure 3.17 illustrates this analysis.

In our simulation, Image Acquisition and HK Acquisition serve as input components. They independently initiate the data flow within the system, queuing raw data without dependency on outputs from preceding processes. The inter-execution times for these components exhibit minor fluctuations around their preset durations, set at 100ms for Image Acquisition and 500ms for HK Acquisition using the `std::wait_until()` function. Specifically, HK Acquisition shows variability around 500 ms ± 5 ms, and Image Acquisition around 100 ms ± 10 ms. Despite using high-precision clocks and FIFO scheduling at high thread priority, inherent hardware and operating system constraints can introduce variability in loop execution timing. Downstream components, Image Processing and HK Reading, exhibit similar inter-execution time patterns, as they process data as soon as it is made available by the upstream components. The Data Inversion component, however, shows larger fluctuations.

This is attributed to the compounded effect of execution time variability and the need to synchronize data from two differently timed sources, often leading to extra waiting periods for data alignment.

Despite these variations, the average inter-execution times align closely with the expected durations, indicating overall stability in system performance. An exception is noted in the Image Saver component, which exhibits extended worst-case inter-execution times, a consequence of the operating system's buffering mechanism affecting disk I/O activities. This observation underscores the importance of isolating the Image Saver from other processes to prevent I/O-intensive operations from adversely impacting the real-time pipeline's efficiency and responsiveness.

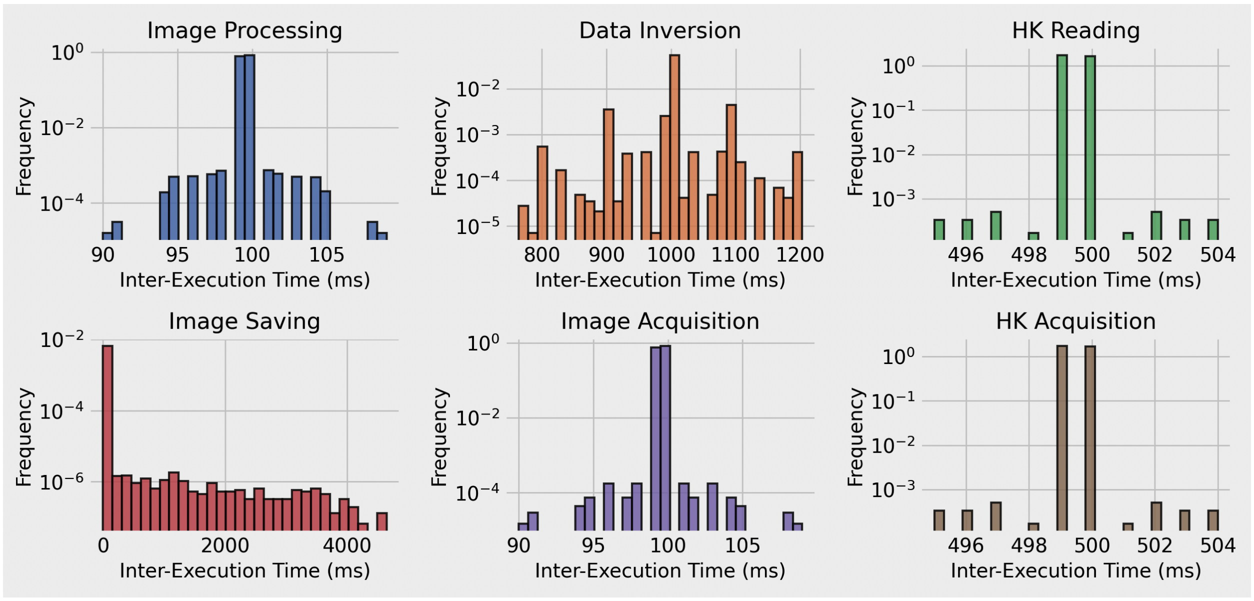

Figure 3.17: *Inter-Execution Time of All Components in Multitasking Scenario*

Chapter 4

Accuracy Evaluation of the Real-Time Pipeline

In this chapter, we shift our focus from the computational performance of our FIMS real-time pipeline to a detailed evaluation of its accuracy. In aerosol science, establishing an exact ground truth for size distribution is a complex task. Typically, results from a Scanning Mobility Particle Sizer (SMPS) are used as a benchmark. In our case, we refer to the offline analysis method's results, which, as reported by Wang, Pinterich, et al. [32], closely align with the SMPS findings. Therefore, the offline analysis outcomes will serve as our standard for comparison. Our approach includes comparing total detected particle numbers, particle counts (actually concentrations) over time, and size distributions over time to gauge the real-time pipeline's accuracy. Additionally, we assess the impact of the pipeline's intentional delay between image processing and inversion.

To assess accuracy, we utilized a dataset of 12,000 images collected over a 20-minute period from an actual FIMS measurement session. This dataset captures the typical operational conditions of the FIMS and is thus well-suited for our evaluation. Its real-world origin ensures that our assessment reflects the pipeline's practical application.

4.1 Accuracy Assessment

4.1.1 Comparison of Total Particle Counts

We start by comparing the number of particles identified in the size range of 10nm to 600nm between our real-time method and the traditional offline analysis. In the provided data samples, the offline method detected a total of 6,014,338 particles. In comparison, our real-time system identified 6,020,977 particles, which is only a slight difference of 0.11%. Despite this small discrepancy in the total count, the detailed distribution across various sizes is consistent between the two methods. As depicted in Figure 4.1, where the y-axis represents particle counts and the x-axis indicates particle size, the curves from both methods closely overlap throughout the entire size range, confirming the accuracy of our system's results.

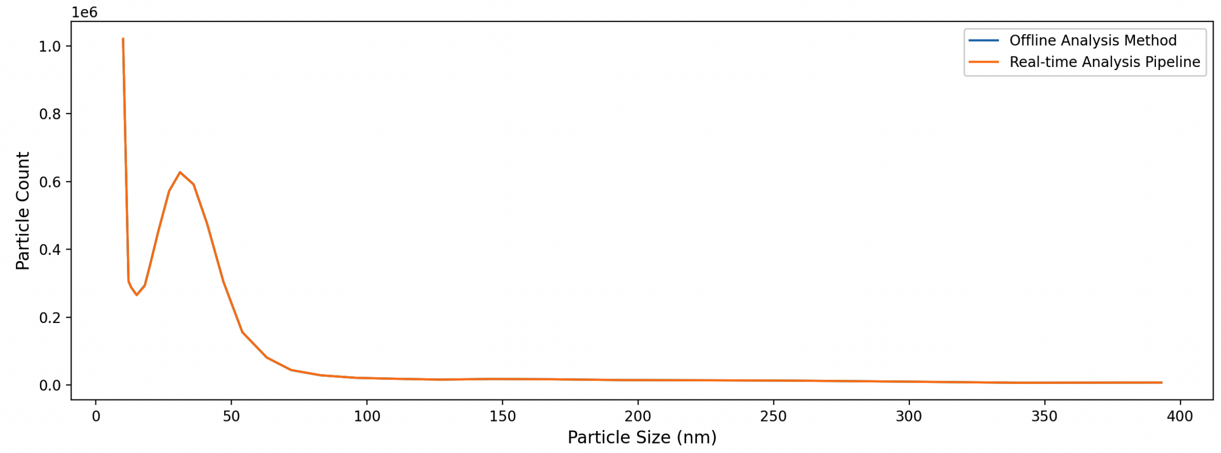

Figure 4.1: *Comparison of Total Detected Particles at Different Sizes*

The minor discrepancy in particle counts observed between the offline method implemented in MATLAB and the real-time method in C++ can be primarily attributed to differences in how each platform handles decimal precision. Despite using the same particle detection algorithm, these slight variations in decimal precision can lead to notable differences in the detected results:

- **Particle Proximity Analysis:** In our algorithm, when two particles are closer than a certain threshold distance, one is removed to avoid double counting. The way MATLAB and C++ handle decimal precision can lead to slight variations in calculating this proximity. For instance, MATLAB might measure a pair of particles as being just under 3 pixels apart, while C++ might calculate the distance as slightly over 3 pixels. This small difference could result in MATLAB considering it as a single particle, whereas C++ sees them as two separate entities.

- **Coordinate Calibration and Boundary Assessment:** The algorithm calibrates particle coordinates relative to detected channel walls, and a predefined viewing window is used to filter out particles lying outside this range. Minor differences in decimal precision between MATLAB and C++ can affect whether a particle is considered inside or outside this window, leading to inconsistencies in counting the particles.

- **Size Binning and Boundary Conditions:** After calibration, particles are categorized into 30 different size bins based on a one-second interval. The slight discrepancies in how MATLAB and C++ handle the decimal precision of particle coordinates can result in a particle being placed in adjacent bins in each platform. Such variations, though minor, can influence the overall distribution of particles across the bins and subsequently affect the total particle count post-inversion.

4.1.2 Comparison of Particle Concentrations over Time

We also measured the number of particles (more precisely, the concentration in particles per cubic centimeter) detected every second over a span of 20 minutes to evaluate our method's accuracy over time. The chart in Figure 4.2 shows the counts of particles on the y-axis and the time on the x-axis. Both real-time and offline methods are represented. Again, the two lines closely mirror each other throughout the duration.

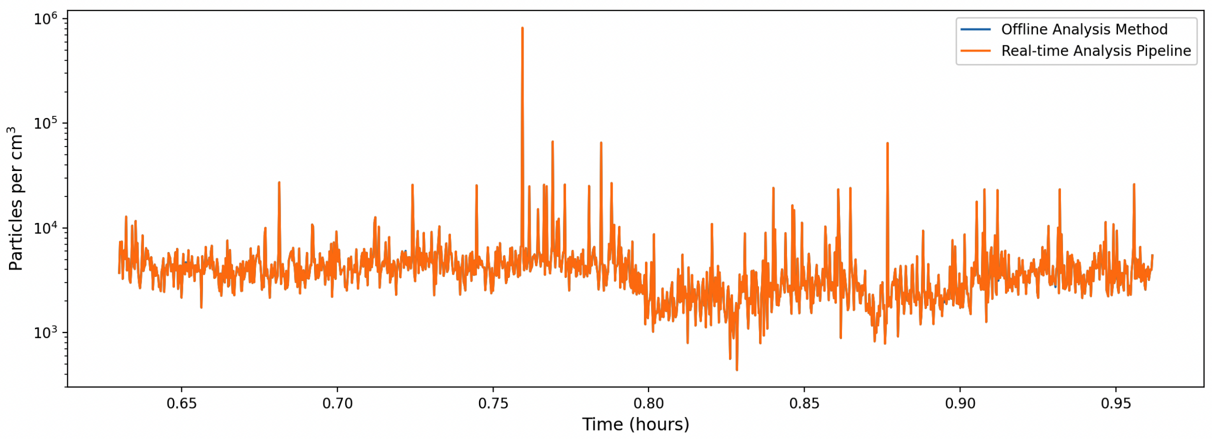

Figure 4.2: *Comparison of Particle Concentration Across Time*

4.1.3 Comparison of Size Distribution Over Time

We further looked at how the Particle Size Distribution changed every second over a twenty-minute period, comparing the real-time method to the offline one. The heatmap in Figure 4.3 displays this comparison. On this heatmap, the y-axis shows the particle size, the x-axis represents time, and the different colors indicate particle concentrations. Both methods showed similar patterns over time. Another chart in Figure 4.3 measures the similarity of the size distribution every second. The average similarity score was very close to 1, at 0.9999, meaning the two methods were almost identical. These minor differences can be attributed to the variance in floating-point arithmetic precision across different programming platforms.

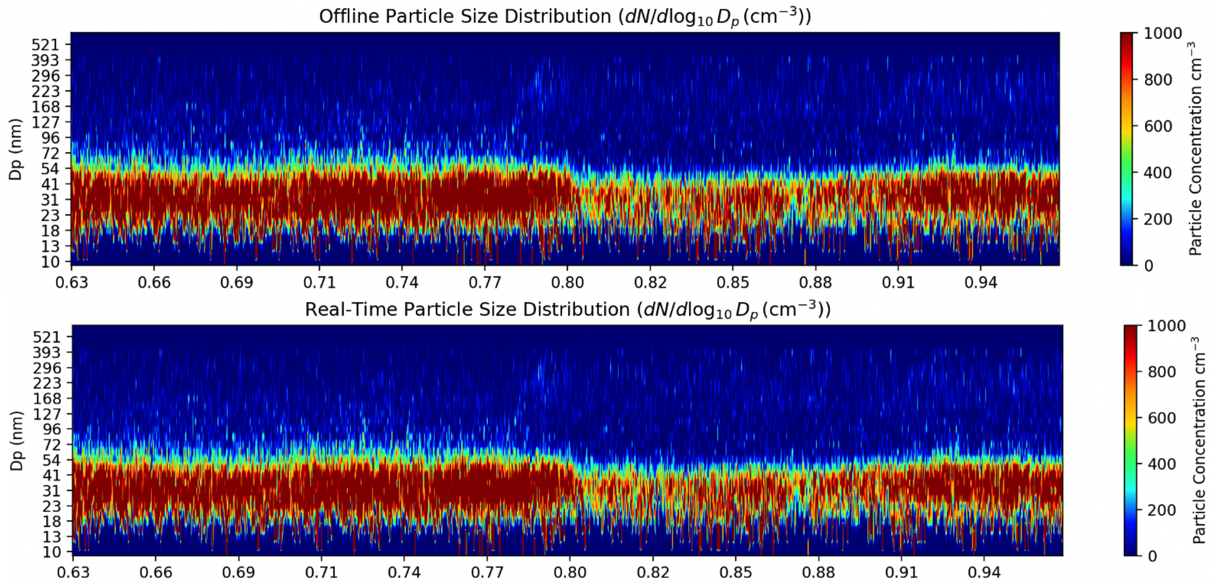

Figure 4.3: *Particle Size Distribution Across Time*

These comparisons across different metrics – total particle counts, temporal concentrations, and size distribution over time – demonstrate a high degree of alignment between the real-time and offline methods underscoring the real-time pipeline's reliability.

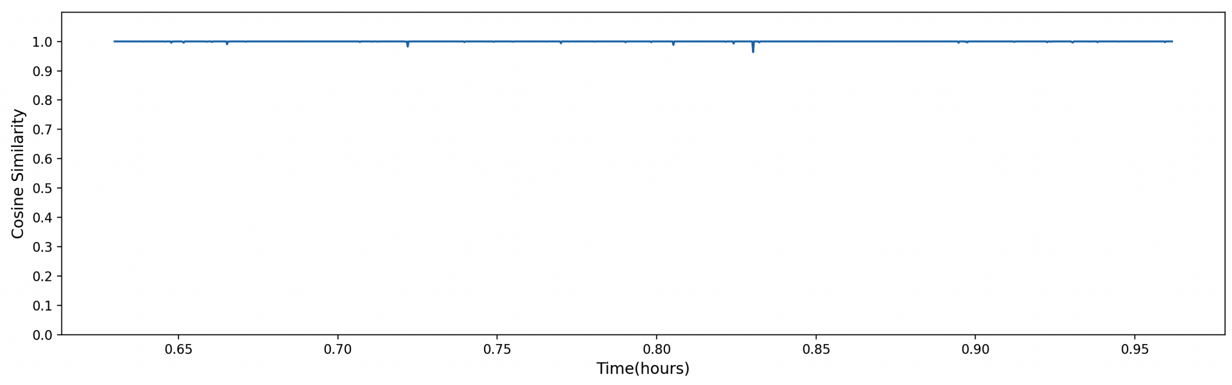

Figure 4.4: *Particle Size Distribution Cosine Similarity Across Time*

4.2 System Latency Analysis

4.2.1 Particle Residence Time

Particles entering FIMS traverse through its separator and condenser before being recorded by the CCD camera. The total residence time of a particle in FIMS is the sum of the durations spent in the separator (t_s) and the condenser (t_c). However, due to the non-uniform flow velocity profile in these sections, particles with different electrical mobilities experience varying trajectories and residence times. This results in each particle having a unique total residence time at the point of image capture at the end of the condenser. To account for this, we calculate each particle's total residence time and determine its entry time into the separator by subtracting the residence time from the detection time.

As per Olfert et al.[19], the residence time in the separator (t_s) can be calculated as:

$$t_s = \frac{\tilde{x} \cdot a^2}{Z_p^* \cdot V} \tag{4.1}$$

where a is the channel width (distance between electrodes), $\tilde{x}$ is the particle's motion in the a direction, Z_p^* represents the electrical mobility, and V is the applied voltage. The residence time in the condenser (t_c) is expressed as:

$$t_c = \frac{l_c \cdot a \cdot b}{6Q_t} \left[\tilde{x} \cdot (1 - \tilde{x})\right] \tag{4.2}$$

where Q_t is the flow rate, l_c is the condenser length, and b is the channel width (perpendicular to a). The total residence time for a particle from entering the separator to detection is t_s+t_c.

Figure 4.5 shows the residence times of particles in the separator and condenser as a function of their final location under one specific operating condition. A notable time discrepancy of a few seconds is observed between particles located near the centre and the edge of the gap. By calculating each particle's entry time into the separator and grouping them into timestamped intervals in the particle queue, we effectively account for these variations.

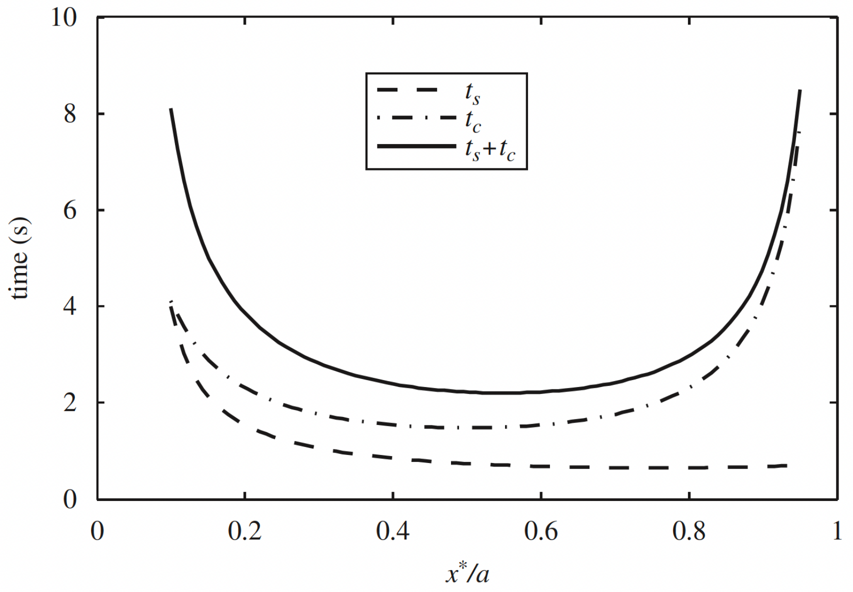

Figure 4.5: Image from Olfert et al.[19] *The time each particle spends in the separator (t_s), the condenser (t_c), and the total ($t_s + t_c$) as function of the particle location, $\tilde{x^*} = x^*/a$, under a specific operating condition.*

The range of particle residence times varies with different operating settings. Our analysis, derived from the test dataset, reveals that these times span from 1.4 to 2.3 seconds (refer to Figure 4.6).

4.2.2 Intentional System Delay and Result Accuracy

To give FIMS sufficient time to detect all particles that arrived at the instrument's inlet at a given time, we must introduce an "intentional system delay" between image processing and final inversion. This delay allows particles that are detected in distinct images, but whose

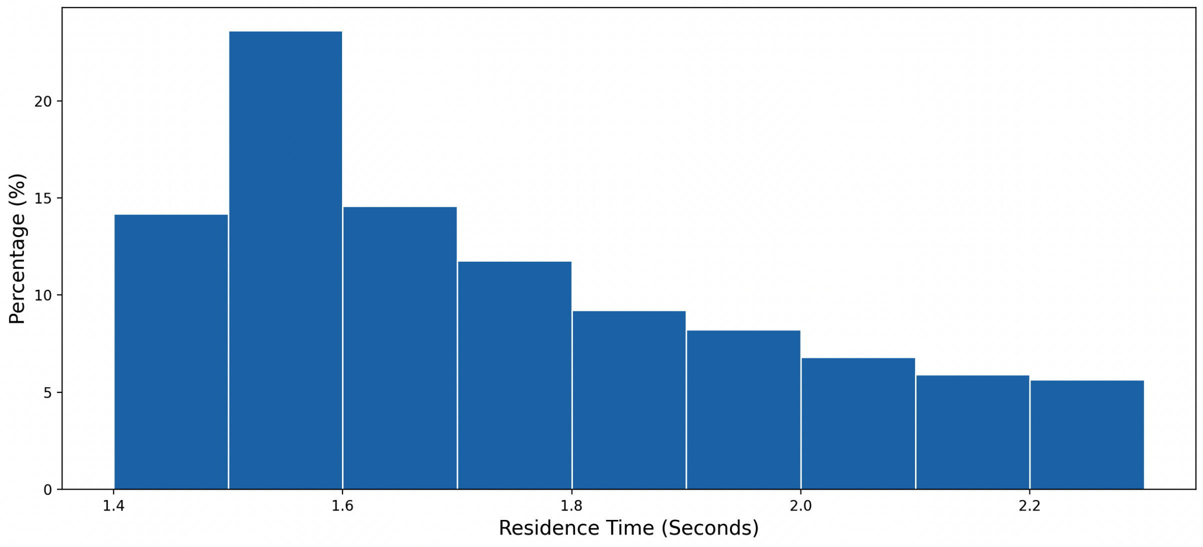

Figure 4.6: *Residence Time Distribution*

differing residence times imply that their inlet arrival times are the same, to be binned together for inversion. An intentional delay δ means that a time bin b placed on the particle queue (i.e., at the output of the image processing phase) at time t will not be dequeued by inversion until at least time $t + \delta$. During the period δ, particles detected in any image may be placed in bin b if their computed arrival times fall within b's time window. In contrast, a detected particle whose arrival time belongs to a time bin that has already been dequeued must be discarded and will not be counted toward the final measurement for that time bin.

Offline analysis has an effectively unlimited intentional delay, since it completes all image processing prior to any inversion and so never discards detected particles. In practice, we found that implementing a 2.5-second delay in our pipeline yielded results effectively identical to that of the offline pipeline. Of course, this means that the output of our real-time pipeline is delayed by 2.5s vs. the actual state of the atmosphere at the time of measurement! We therefore considered the tradeoff between length of intentional delay and result accuracy.

We tested intentional delays of 2.5s, 2.0s, 1.5s, and 1.0s. Figure 4.7 displays the total particles of different sizes detected at each delay duration. The graph shows that delays above 2.0s result in only minor differences in particle counts. However, when the delay is reduced below 1.5s, there is a substantial decline in detected particles. Specifically, a 2.0-second delay misses about 0.88% of particles, a 1.5-second delay omits 21.9%, and a 1.0-second delay results in a significant loss of 65.4% of particles.

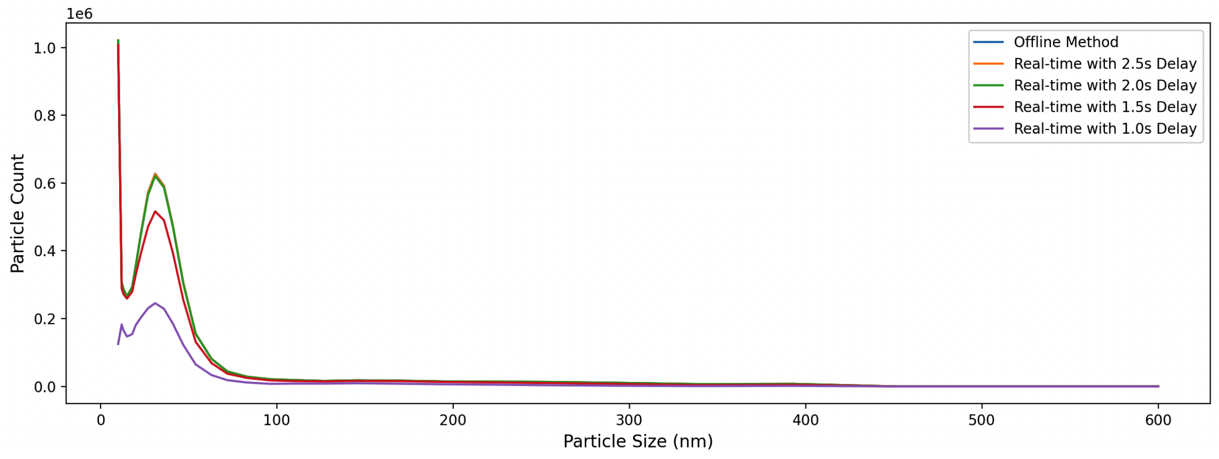

Figure 4.7: *Total Detected Particle Counts Comparison with Different System Delays*

Further analysis can be seen in Figure 4.8, which presents detected particle concentrations across time for each delay setting. Delays of 2.0s or longer align closely with the offline method. In contrast, delays of 1.5s or less show marked discrepancies.

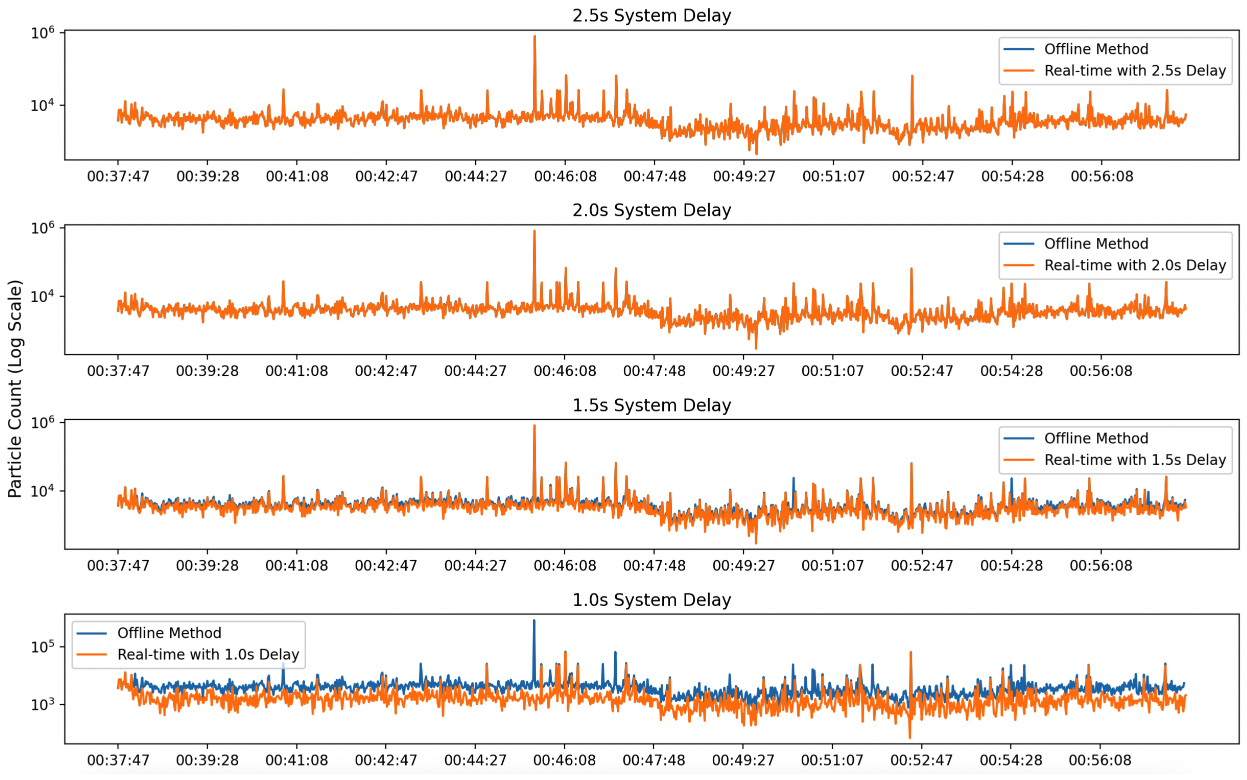

Figure 4.8: *Detected Concentration Comparison Cross Time with Different System Delays* (cm^{-3})

Comparisons of particle size distributions further underscore these findings. As depicted in Figure 4.9, a shortened delay leads to notable reductions in particle concentrations. Moreover, the cosine similarity graph of per-second size distribution, Figure 4.10 highlights a consistent performance for 2.5s and 2.0s delays but shows irregularities at 1.5s and deteriorates significantly at 1.0s.

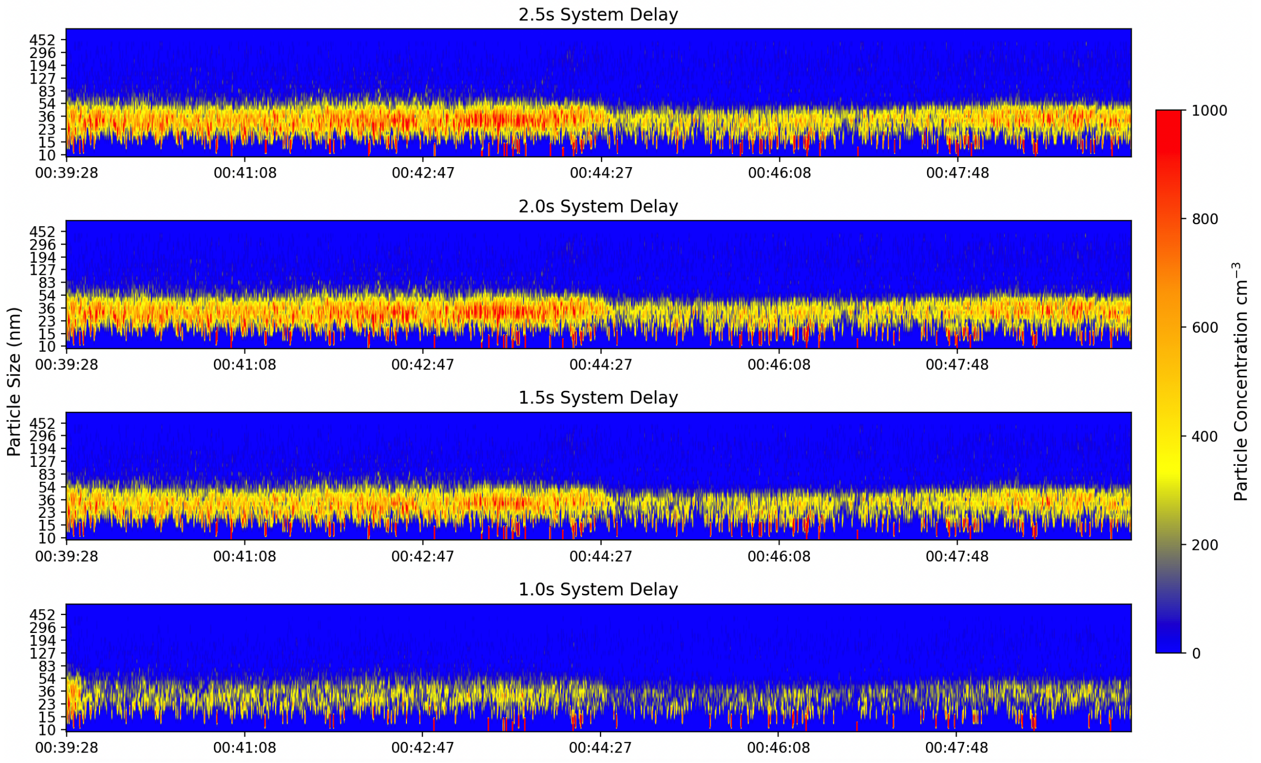

Figure 4.9: *Particle Size Distribution Comparison with Different System Delays* $dN/d\log_{10}D_p\,(\mathrm{cm}^{-3})$

Our assessment underscores the importance of intentional system delay, necessitated by the particle residence times, for accurate particle detection. The tests demonstrate a delicate balance between the system delay and overall data accuracy. We find that implementing a delay of at least 2.5 seconds maximizes accuracy. However, maintaining a high degree of accuracy is still feasible with a minimum delay of 2.0 seconds. Conversely, reducing the delay to 1.5 or 1.0 seconds results in a significant drop in accuracy.

In various practical scenarios, the significance of a 2.5-second delay in the FIMS system can vary. For aircraft equipped with FIMS that typically travel at approximately 100-200 m/s, this delay corresponds to covering a distance of 250 to 500 meters.

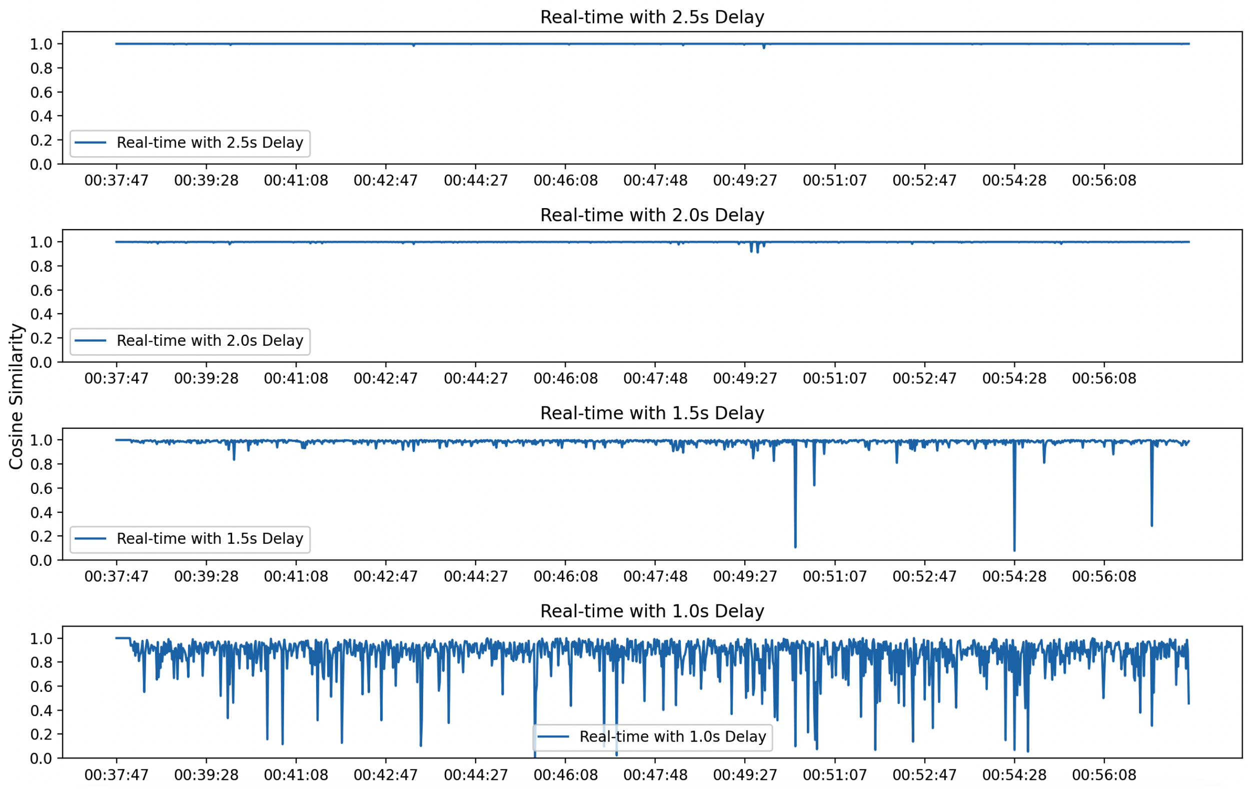

Figure 4.10: *Comparison of Particle Size Distribution Cosine Similarity to Offline Method with Different System Delays*

In tasks such as tracking a pollution plume from a ground source, the 2.5-second delay implies that the aircraft might travel nearly half a kilometer before FIMS data indicates the need for a course correction. This situation requires heightened pilot vigilance to adjust the course and compensate for the distance covered during the delay. Predictive models may be helpful in rapidly changing environments to effectively mitigate the impact of the system delay.

In other monitoring scenarios, like detecting the boundaries of an air mass, the delay results in a gap in real-time data updates. However, alternative methods, such as monitoring temperature or pressure changes, could offer a more efficient way to obtain immediate information. These quicker indicators can be employed to overcome the limitations imposed by the delay in the FIMS system. Overall, the impact of the FIMS delay varies with the specific application.

A more compact version of FIMS is currently in development, which is expected to significantly reduce residence times within its separator and condenser components. Such an advancement could permit a reduction in the intentional system delay.

Chapter 5

Particle Detection Algorithm Redesign

Recognizing the Image Processor as the most demanding component in the real-time pipeline, we initiated efforts to enhance its efficiency. Our focus centered on redesigning the particle detection algorithm to enhance efficiency and reduce its high utilization rate.

5.1 Conventional Particle Detection Algorithm

The original offline particle detection methodology incorporates intensity-based thresholding to separate possible overlapped particles, binary transformations to highlight discernible regions, and specific algorithms to locate particles and eliminate redundant detections. The pseudocode for this method is outlined in Algorithm 1. For each of a descending series of thresholds, (ten of them in the original implementation), the algorithm binarizes the image using that threshold, labels connected components in the resulting binary image, and finally extracts each component from the original image as a potential particle. However, it only retains new particles whose region's maximum intensity is less than the preceding threshold. To eliminate redundant particle identifications, the algorithm keeps only one of any pair of particles located whose centroids are closer than three pixels.

Limitations
Using a series of predefined masks to separate overlapped particles and using specific distances to examine duplicate counting particles has the following limitations:

1. **Computational Overhead:** Applying multiple thresholds sequentially can be computationally expensive.

Algorithm 1 Conventional Particle Detection Algorithm

```
 1: function PARTICLE_DETECTION(image, intensity_thresholds)
 2:     Ensure intensity_thresholds are in descending order
 3:     Initialize a list particles_coordinates to store coordinates of detected particles
 4:     for each threshold in intensity_thresholds do
 5:         Create a binary image (bw_image) from image using the threshold
 6:         Identify connected regions in bw_image
 7:         Apply the identified region to original image
 8:         Extract weighted_centroids and max_intensity for each region of image
 9:         for each region in image do
10:             if region's max_intensity is below the previous threshold then
11:                 Append region's weighted_centroids to particles_coordinates
12:             end if
13:         end for
14:     end for
15:     Perform pair-wise comparison of detected particles in particles_coordinates
16:     Retain only one particle when two are positioned within three pixels of each other
17:     return particles_coordinates
18: end function
```

2. **Constraints of Predefined Thresholds:** The effectiveness of using ten predefined thresholds might limit the variability of the particles. It might not adapt well to identify all the overlapped particles.

3. **Risk of Over-filtering:** The filtering mechanism, designed to eliminate duplicate particle detections using a rigid "3-pixel distance" criterion, risks excluding genuine particles, especially in densely packed scenarios. When particles are fewer than three pixels apart, the algorithm retains only one, potentially erroneously omitting legitimate particles in closely clustered arrangements.

5.2 Find-peak Particle Detection Algorithm

In the particle detection task, identification of scattered particles as well-separated regions in a threshold image is achieved using `cv::connectedComponents()` in OpenCV or `bwconncomp()` in Matlab. The primary challenge is efficiently detecting and resolving overlapping particles.

We have observed that the brightness distribution of individual particles resembles a mountain peak, with the brightest intensity at the center and decreasing towards the edges, as exemplified in Figure 5.1. Leveraging this pattern, we propose a two-category classifier to differentiate between peak and valley areas in an overlapped particle region. Assuming that overlapped particles can always be separated along a line, we exploit the mountain peak pattern to classify pixels in the overlapped region. Pixels in the valley area will be assigned a zero intensity, and all valley points will form a line to separate the overlapped particles (see Figure 5.2 for an example). After the separation of the overlapping particles, they can be considered as individual particles. The centroid coordinates of all particles can then be obtained with only a single thresholding step.

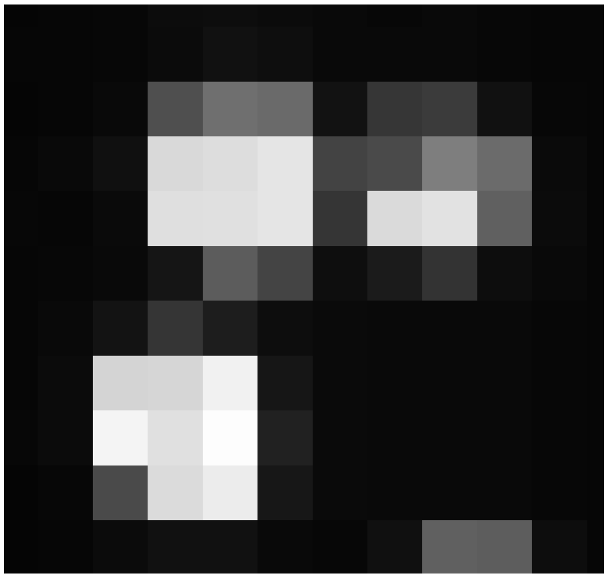
Figure 5.1: *Overlapped Particles*

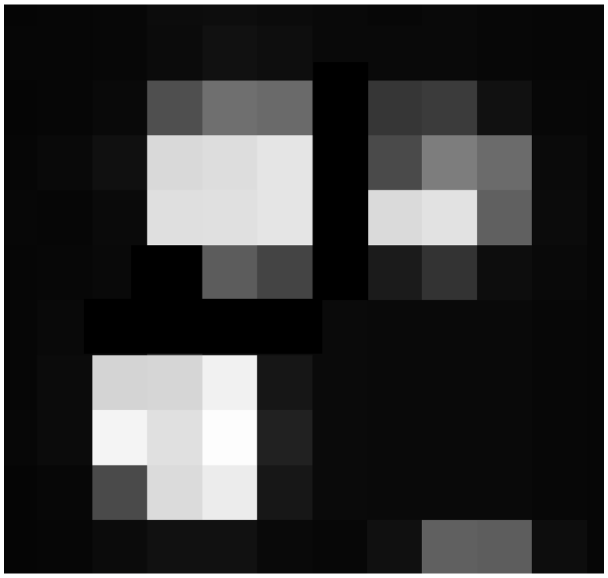
Figure 5.2: *After Separation*

5.3 Redesigned Particle Detection Steps

To address this two-category classification problem, the simplest idea is to adopt a linear classifier. Specifically, a 7x7 image patch M centered on each pixel to be classified is obtained, and a difference matrix D is generated by subtracting the intensity of the center pixel from the intensities of its neighbors, $D = M - v$ where v is the intensity of the center pixel. D_v is the vector obtained by flattening the matrix D. Thus the linear classifier can be expressed as $c = WD_v + b$.

Ideally, the parameters W and b should be obtained through training. However, since there is a lack of training data with ground truth, the classifier parameters need to be tuned manually. In this case, tuning W become impractical as there are 49 parameters in W. To address this issue, it is possible to assume that all elements of W are equal to one and to use only one dimension of information to classify the pixels.

Algorithm Design for Classification:

1. Given the matrix representation of a 7×7 image patch centered on a pixel:

$$M = \begin{bmatrix} m_{00} & m_{01} & \dots & m_{06} \\ m_{10} & m_{11} & \dots & m_{16} \\ \vdots & \vdots & \ddots & \vdots \\ m_{60} & m_{61} & \dots & m_{66} \end{bmatrix} \tag{5.1}$$

2. The difference matrix D can be computed as:

$$D = M - v \tag{5.2}$$

3. Where v is the intensity of the central pixel. The matrix D can be flattened to form a vector:

$$D_v = [d_{00}, d_{01}, \dots, d_{66}] \tag{5.3}$$

4. Let's introduce a weight vector W:

$$W = [w_{00}, w_{01}, \dots, w_{66}] \tag{5.4}$$

5. Then, the output c of the linear classifier can be defined as:

$$c = W \cdot D_v + b \tag{5.5}$$

6. To construct a more explainable classifier, let's consider two primary dimensions:
   (a) The number of neighboring pixels with higher intensity, represented as $\sum_{ij}(D_{ij} > 0)$.
   (b) The number of neighboring pixels with lower intensity, represented as $\sum_{ij}(D_{ij} < 0)$.
7. Thus, the classifier can be defined as a combination of two simplified classifiers:

$$c_1 = \sum_{ij}(D_{ij} > t_1) + b_1 \tag{5.6}$$

$$c_2 = \sum_{ij}(D_{ij} < t_2) + b_2 \tag{5.7}$$

$$c = \begin{cases} 1 & \text{if } c_1 > 0 \text{ and } c_2 > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

   Here, t_1 and t_2 are thresholds that can be tuned for optimization.

Particle Centroid Computation: After separation of overlapping particles, the centroids of individual particles are computed using the `cv::connectedComponents()` function.

5.4 Performance Comparison

5.4.1 Comparison on Overlapped Particle Detection

Assessing the absolute accuracy of particle identification algorithms presents a significant challenge, primarily due to the difficulty in establishing a definitive ground truth in particle imagery. Nonetheless, when focusing on the precision of detecting non-overlapping particles, both our original and new algorithms exhibit comparable effectiveness. The original algorithm relies on MATLAB's `bwconncomp()` function for detecting connected components, while our newly developed find-peak algorithm uses OpenCV's `cv::connectedComponentsWithStats()` for the same purpose. The primary difference between these two algorithms lies in their distinct methods of handling overlapped particle separation.

To assess the methods' accuracy in distinguishing overlapped particles, we devised an experimental setup where we overlapped two images to simulate possible particle conjunctions. This process involves the following steps:

1. Apply both algorithms to two separate images (referred to as image 1 and image 2).
2. Create a composite image by superimposing image 1 and image 2, thus generating potential overlapped particle scenarios.
3. Run both algorithms on the composite image.
4. To validate the results, we examine each connected area within the composite image and track the particle count as identified from image 1 and image 2 against the count from the composite image within the area. A match in total particle count between the sum of individual images and the composite image indicates the successful identification of overlapped particles. Conversely, a discrepancy signals the presence of unidentified (false negative) or erroneously identified (false positive) particles.

Our analysis covered two distinct datasets: one sparse set with an average of 40 particles per image within the region of interest and one dense set with an average of 155 particles per image. Each dataset comprised 600 images. The results were as shown in Table 5.1.

Table 5.1: Comparison of Overlapped Particle Detection of the Two Algorithms

Algorithm	Dataset One (%)		Dataset Two (%)	
	True Positive	False Positive	True Positive	False Positive
Original	64.351	0.084	57.605	0.037
Find-Peak	71.770	1.697	72.366	4.238

The analysis of the results indicates that the find-peak algorithm demonstrates a higher true positive rate, particularly noticeable in densely populated image sets. However, this increased accuracy in detecting true positives is accompanied by a higher rate of false positives. On the other hand, the conventional algorithm, while exhibiting a lower true positive rate, also maintains a lower false positive rate. This comparative analysis highlights the distinct strengths and limitations of each algorithm.

Currently, the calibration of the find-peak algorithm is conducted manually. This suggests an opportunity for further enhancement: incorporating machine learning techniques could significantly improve its precision. The potential integration of machine learning could not only automate the calibration process but also refine the algorithm's ability to distinguish between true and false positives more effectively.

5.4.2 Comparison on Particle Size Distribution Detection

In the field of aerosol science, obtaining a definitive ground truth for particle size distribution is often impractical. Hence, instead of measuring accuracy relative to ground truth, we examine the deviations of the find-peak algorithm from the conventional one. The data set for this study comprises 12,000 sparse images collected over 20 minutes using FIMS.

We first compare raw particle counts and distributions $\vec{R}$ of particles grouped by mobility, as computed each second by the two algorithms prior to inversion. The conventional method identified 33,553 total particles, whereas the find-peak algorithm detected 33,832 particles — a difference of 0.8%. Per-second comparisons are illustrated in Figure 5.3 for counts and Figure 5.4 for distributions. The average cosine similarity of the latter was 0.989.

We next compare post-inversion results, which reveal a larger discrepancy. The find-peak algorithm registered 5,861,742 particles after inversion, compared to the 6,020,977 particles reported by the conventional method, indicating a 2.4% difference. The corresponding per-second counts and discrepancies between post-inversion size distributions are given in Figures 5.5 and 5.6.

While average cosine similarity in post-inversion distributions between the two algorithms was high, at 0.993, there were occasional large discrepancies observed. These discrepancies typically appear in bins computed by from very few raw particles, where the inversion process may be particularly unstable or simply produces small values that amplify relative deviations. Further study is needed to determine how significant these discrepancies actually are.

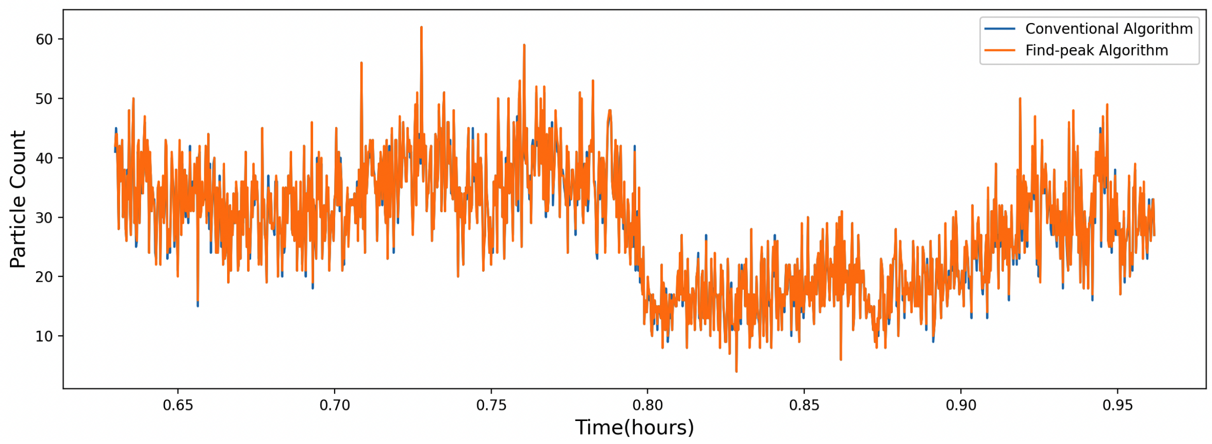

Figure 5.3: *Comparison of Detected Particle Counts from Images Between the two Algorithms*

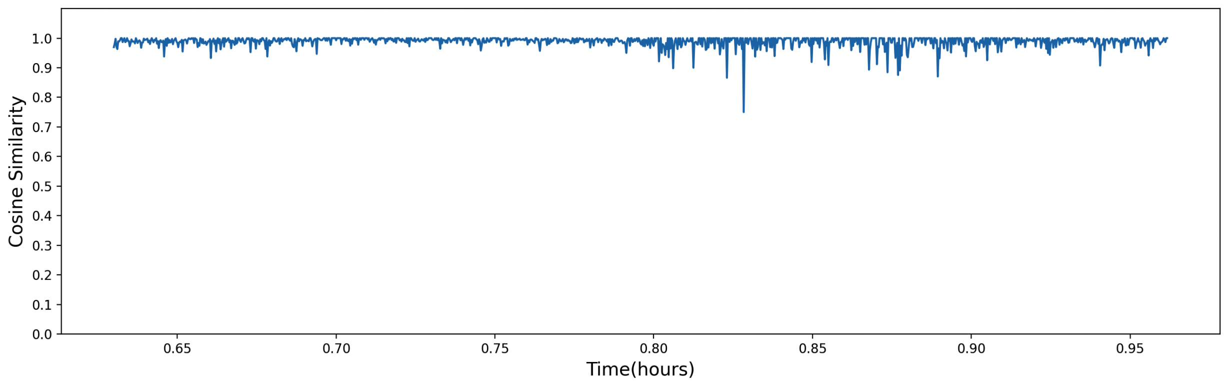

Figure 5.4: *Cosine Similarity of Particle Size Distribution Between the Two Algorithms Before Inversion*

Notably, there is a discrepancy in particle counts before and after the inversion process between the two methods. Specifically, more particles were detected pre-inversion using the Find-Peak algorithm compared to the conventional thresholding method. However, post-inversion, the scenario was reversed, with fewer particles being identified by the Find-Peak algorithm. This variation in counts suggests underlying complexities in the inversion process.

To provide context, let us revisit the fundamental equation that underpins our analysis. The key relationship is mathematically represented as follows:

$$\vec{R} = \vec{M} \times \vec{n} \tag{5.9}$$

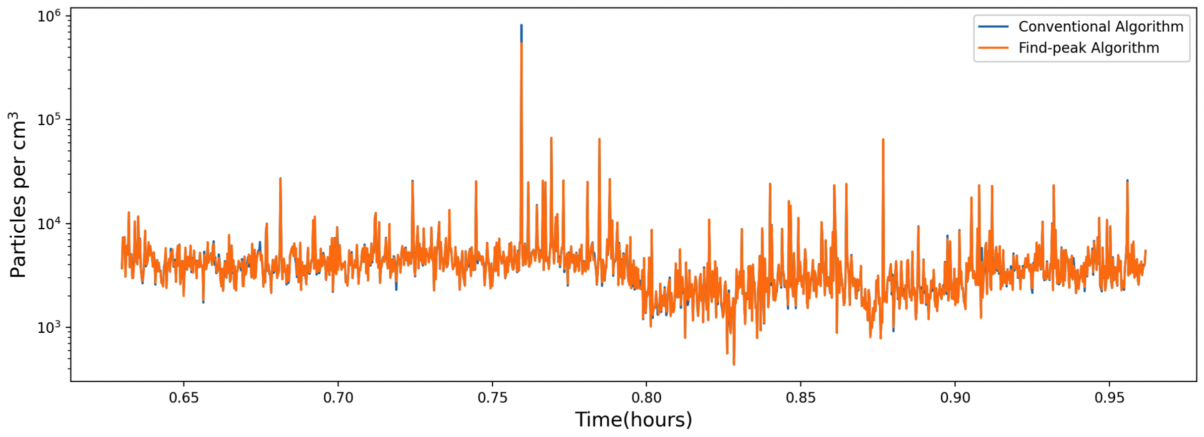

Figure 5.5: *Comparison of Detected Particle Counts Between the two Algorithms*

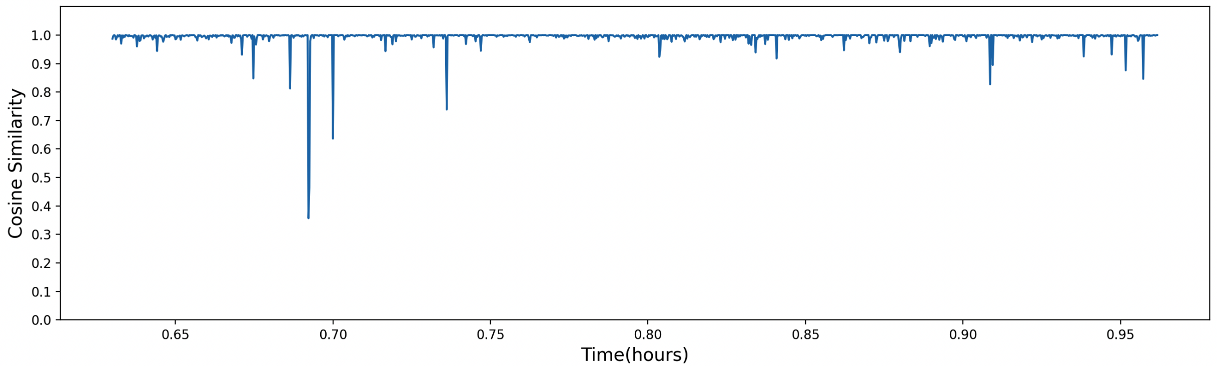

Figure 5.6: *Cosine Similarity of Particle Size Distribution Between the Two Algorithms*

In this equation:

- $\vec{R}$ represents the particle count histogram from a 1-second interval, measuring from the corresponding images.
- $\vec{M}$ is the inversion matrix.
- $\vec{n}$ denotes the particle distribution we aim to determine.

The inversion process and the observed discrepancies can be understood through two main points:

1. **Influence of Particle Size and Detection**: Due to their lower probability of being charged, smaller particles are less likely to be detected. This suggests that an equal

count of particles detected before inversion might represent a larger actual number of smaller particles, compared to larger ones, once the data is inverted. In our analysis, it is possible that the Find-Peak algorithm, while detecting more larger particles and having a higher total particle count, may overlook smaller particles that the conventional method picks up. As a result, despite a lower count of detected particles before inversion, the conventional method might reveal a higher particle count post-inversion.

2. **Non-Linearity of the Twomey Inversion Process**: The Twomey inversion process, which utilizes a chi-square test for iterative refinement, introduces non-linear characteristics. This non-linear relationship between $\vec{R}$ and $\vec{n}$ suggests that a higher particle count in images ($\sum \vec{R}$) does not straightforwardly translate to a proportionally higher particle distribution post-inversion ($\sum \vec{n}$). This non-linearity complicates direct correlations between pre- and post-inversion counts.

5.4.3 Comparison on Running Time

To assess the performance of the two algorithms in terms of running time, we conducted tests on a Raspberry Pi 4 Model B, which has a 4-core, 64-bit Cortex-A72 (ARM v8) CPU running at 1.50GHz and 4GB of RAM. We test with Linux 5.10.103. The execution time for each algorithm was measured using the `getrusage()` function, which provided the combined user and system time. The results are illustrated in Figure 5.7 for the conventional particle detection algorithm and in Figure 5.8 for the redesigned find-peak algorithm.

Our analysis revealed that the conventional algorithm had an average CPU time of 31.5ms, while the find-peak algorithm reduced the average time to 8.0ms, a decrease of around 75%.

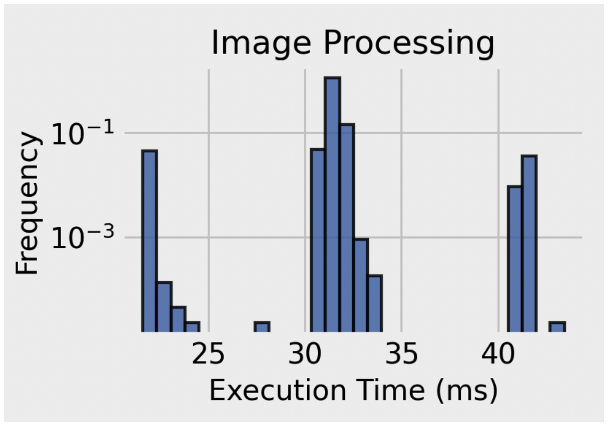

Figure 5.7: *Execution Time of Conventional Algorithm*

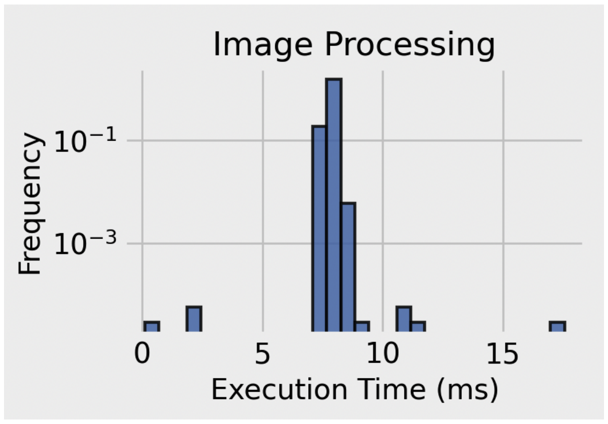

Figure 5.8: *Execution Time of Find-peak Algorithm*

5.5 Limitations and Future Updates of Find-peak Algorithm

5.5.1 Limitations

While the find-peak particle detection algorithm simplified the process by eliminating the need for repeated thresholding and connected components analysis, it introduced new limitations, primarily due to its reliance on parameters that lack ground truth validation:

1. **Manual Parameter Tuning:** The algorithm relies on manually tuned parameters, which might not yield the best results. Given the diverse nature of particle imaging scenarios, these parameters may not be effective in all situations.

2. **Simplified Classification:** The classifier, in its current form, does not capture all the relevant details. Its accuracy is likely lower than what could be achieved with a more sophisticated linear classifier. This limitation stems from its reliance on simplified criteria.

5.5.2 Directions for Future Optimization

Considering these limitations, future enhancements of the classifier should focus on a more empirical approach:

- **Automated Parameter Optimization:** Developing methods to automatically update classifier parameters could lead to more robust and universally applicable solutions.
- **Alignment with True Data Components:** A key improvement would be aligning the primary dimensions used by the classifier with the true principal components of the data space.
- **Comprehensive Classifier Models:** Expanding beyond linear classifiers to more comprehensive models would likely increase the accuracy of particle detection.

For these enhancements, training the classifier with a substantial dataset is essential, although this introduces the challenge of pixel-level data labelling, which is time-consuming. Nevertheless, with enough training data, we could even explore employing neural networks for particle identification, potentially automating and significantly improving the particle separation process.

Chapter 6

Adaptive Performance Optimization of the Real-time Pipeline in Resource-Constrained Environments

This chapter describes work done as part of collaborative research with Marion Sudvarg for the paper "Harmonic Elastic Scheduling," which was submitted to RTAS 2024. The study utilized the Fast Integrated Mobility Spectrometer (FIMS) as a real-world application to validate elastic scheduling models within dynamic, resource-constrained environments.

The elastic real-time scheduling model [20, 23] is a paradigm for dynamically adapting task utilizations in response to fluctuating system demands. Elastic scheduling selectively decreases task utilizations within predetermined ranges, thereby limiting overall system utilization in exchange for some expected degradation of task result quality. Our role in the collaboration was to integrate the FIMS real-time pipeline into the elastic framework, so as to enable dynamic scheduling of FIMS tasks using an elastic scheduler. We therefore investigated mechanisms to dynamically reduce our pipeline's CPU utilization and the corresponding impacts on its functional integrity and accuracy.

6.1 Utilization of a Real-Time Task

The concept of utilization (U) is fundamental in the management of real-time tasks[5]. It is defined as the ratio of a task's execution time (C) to its period (T), represented by:

[5] In context of queueing models, utilization typically measures the proportion of time resources are busy serving requests, focusing on efficiency and capacity, as analyzed in Chapter 3.

$$U = \frac{C}{T} \tag{6.1}$$

where:

- C is the worst-case execution time (WCET) of the task.
- T is the period of the task.

The system-wide utilization is the sum of the utilizations of all individual tasks in a real-time system. It provides an overview of the total resource demand of all tasks relative to the available resources. The system-wide utilization is given by the equation:

$$U_{\text{total}} = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{6.2}$$

where:

- U_{total} is the total utilization of the system.
- n is the number of tasks in the system.
- C_i represents the worst-case execution time (WCET) of the i-th task.
- T_i is the period of the i-th task.

This equation is crucial for determining whether a set of real-time tasks can be scheduled without exceeding the system's resource capacity and ensuring that all tasks can meet their deadlines.

In the context of the FIMS real-time analysis pipeline, we assess the worst-case execution times (WCET) of each component on a Raspberry Pi 4. The WCET for the image processing component, data inversion, and HK reading are as follows:

Platform	Image Process Time (ms)	Data Inversion Time (ms)	HK Reading Time (ms)
WCET	42.3	56.4	0.7

Table 6.1: Worst-Case Running Time of Each Component

6.2 Utilization Optimization Strategies for Real-time Pipeline

Given these parameters, the utilization of the FIMS pipeline on a Raspberry Pi 4 is calculated as:

$$U_{\text{image\_process}} = \frac{C_{\text{image\_process}}}{T_{\text{image\_process}}} = \frac{42.3}{100} = 0.423 \tag{6.3}$$

$$U_{\text{data\_inversion}} = \frac{C_{\text{data\_inversion}}}{T_{\text{data\_inversion}}} = \frac{56.4}{1000} = 0.056 \tag{6.4}$$

$$U_{\text{hk\_reading}} = \frac{C_{\text{hk\_reading}}}{T_{\text{hk\_reading}}} = \frac{0.7}{500} = 0.001 \tag{6.5}$$

$$U_{\text{total}} = U_{\text{image\_process}} + U_{\text{data\_inversion}} + U_{\text{hk\_reading}} = 0.48 \tag{6.6}$$

From the calculated utilization, the image processor is the most resource-intensive, accounting for 88.0% of the total system utilization. To enhance the efficiency of the real-time pipeline, focusing on the image processor is crucial. Reducing its workload can significantly optimize the pipeline's overall utilization. We propose several strategies to achieve this:

1. **Implementation of the Find-Peak Algorithm:** The particle detection phase in the image processor is the most computationally demanding. Replacing the current algorithm with the find-peak algorithm, as detailed in Section 3.3, offers potential reductions in computational load at some cost to accuracy.

2. **Compression of the Original Particle Detection Algorithm:** The current method uses ten masks for particle separation. We propose to experiment with reducing this

number, assessing the impact on system utilization and measurement accuracy at each step.

3. **Adjustment of Imaging Frequency:** Modifying the frequency of image processing operations can also contribute to workload reduction. By increasing the time intervals between image processing events, we aim to decrease the overall resource demand, analyzing how this adjustment affects the accuracy of aerosol size distribution measurements.

These strategies are designed to reduce the total utilization of the real-time pipeline to make is more adaptive in resource-constrained environments. The balance between efficient resource usage and maintaining accuracy will be explored in the next section.

6.3 Quantitative Assessment of Utilization Reduction Methods

To assess the impact of various strategies proposed for reducing the utilization of our real-time pipeline, we will employ a custom error metric. This metric combines two key measures: the Mean Normalized Absolute Error (MNAE) and the Mean Cosine Similarity (MCS). MNAE will evaluate the magnitude of deviation from the baseline, while '1 - MCS' will assess the distributional differences. This combined metric provides a comprehensive view of the accuracy trade-offs involved in each utilization reduction method.

Error Metric Formulation
We define our Error Metric, taking equal weighting for MNAE and MCS, as follows:

$$\text{Error Metric} = \alpha \cdot \text{MNAE} + \beta \cdot (1 - \text{MCS}) \tag{6.7}$$

where we take $\alpha = 0.2$ and $\beta = 0.8$ for the following assessment.

Mean Normalized Absolute Error (MNAE)
The MNAE is calculated using the equation:

$$\text{MNAE} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{m} \left| \text{baseline}_{ij} - \text{adjusted}_{ij} \right|}{\sum_{j=1}^{m} \text{baseline}_{ij}} \right) \tag{6.8}$$

where:

- baseline_{ij} is the particle count in the j-th size bin of the i-th interval in the baseline result,
- adjusted_{ij} is the particle count in the j-th size bin of the i-th interval in the adjusted result,
- n is the total number of intervals measured,
- m is the total number of size bins in each interval (30 in our case).

Mean Cosine Similarity (MCS)
To assess the similarity in particle size distribution between baseline and adjusted results, we use the following formula:

$$\text{MCS} = \frac{1}{n} \sum_{i=1}^{n} \text{Cosine Similarity}(\text{baseline}_i, \text{adjusted}_i) \tag{6.9}$$

where:

- baseline_i represents the particle size distribution in the i-th interval for the baseline method,
- adjusted_i represents the particle size distribution in the i-th interval for the adjusted method,
- n is the total number of intervals measured.

This analytical approach ensures a balanced evaluation of each proposed utilization reduction strategy, aiding in the determination of the most effective method for our real-time pipeline under resource-constrained environments.

6.4 Adopting the Find-Peak Algorithm

The find-peak algorithm, introduced in Chapter 4, offers a more efficient method for particle detection in images with certain compromises on accuracy.

Replacing the conventional thresholding algorithm with the find-peak algorithm resulted in a significant decrease in the worst-case execution time (WCET) of the image processing component, and consequently, a reduction in its utilization. The utilization for image processing improved from 0.423 to 0.129, while the total system utilization dropped from 0.48 to 0.189. This change aligns the real-time pipeline with even more resource-constrained environments like a busy Raspberry Pi with only 0.2 of total available CPU utilization.

The accuracy impact of this change was quantified using Mean Cosine Similarity (MCS) and Mean Normalized Absolute Error (MNAE). The MCS decreased to 0.993 compared with the original result, and the MNAE increased to 0.056. The total error, as defined by our combined metric, was calculated to be 0.017. Table 6.2 summarizes these changes.

Algorithm	$WCET_{\text{Image Process}}$	$U_{\text{Image Process}}$	U_{total}	MCS	MNAE	Error
Thresholding	42.3	0.423	0.48	1	0	0
Find-Peak	12.9	0.129	0.189	0.993	0.056	0.017

Table 6.2: Comparison of Utilization and Accuracy Metrics for Thresholding and Find-Peak Algorithms

However, this method does not provide continuous adjustment according to the resource availability of the environment, which limits its potential contribution to elasticity.

6.5 Reducing Thresholds in Particle Detection

Reducing the number of thresholds in the particle detection algorithm within the real-time pipeline is another strategy aimed at enhancing computational efficiency. This approach focuses on decreasing the computational workload of the image processor, thereby improving overall system utilization.

Table 6.3 demonstrates the effects of reducing the number of thresholds from 10 to 1 on both the utilization and accuracy of the system. The total system utilization declines from 48%

with 10 thresholds to a much lower 16.9% with just one threshold, indicating a substantial increase in efficiency. These efficiency gains come at a cost to accuracy. The Mean Cosine Similarity (MCS) and Mean Normalized Absolute Error (MNAE) indicate a slight decrease in accuracy, and the total error remains relatively low even with minimal thresholds. The relation of utilization and error is shown in Figure 6.1, which can be roughly described as:

$$y = -0.15 * x^2 + 0.05 * x + 0.01 \tag{6.10}$$

as plotted in the Figure 6.2.

$ThresholdsNum.$	$WCET_{\text{Image Proc.}}$	$U_{\text{Image Proc.}}$	U_{total}	MCS	MNAE	Error
10	42.3	0.423	0.48	1.000	0.000	0.000
9	38.2	0.382	0.442	0.999	0.016	0.004
8	35.9	0.359	0.419	0.998	0.020	0.005
7	32.9	0.329	0.389	0.998	0.025	0.007
6	29.9	0.299	0.359	0.997	0.034	0.009
5	27.1	0.271	0.331	0.996	0.037	0.011
4	24.8	0.248	0.308	0.995	0.040	0.012
3	21.3	0.213	0.273	0.994	0.048	0.014
2	16.7	0.167	0.227	0.994	0.047	0.014
1	10.9	0.109	0.169	0.994	0.052	0.015

Table 6.3: Impact of Reducing Thresholds on Utilization and Accuracy

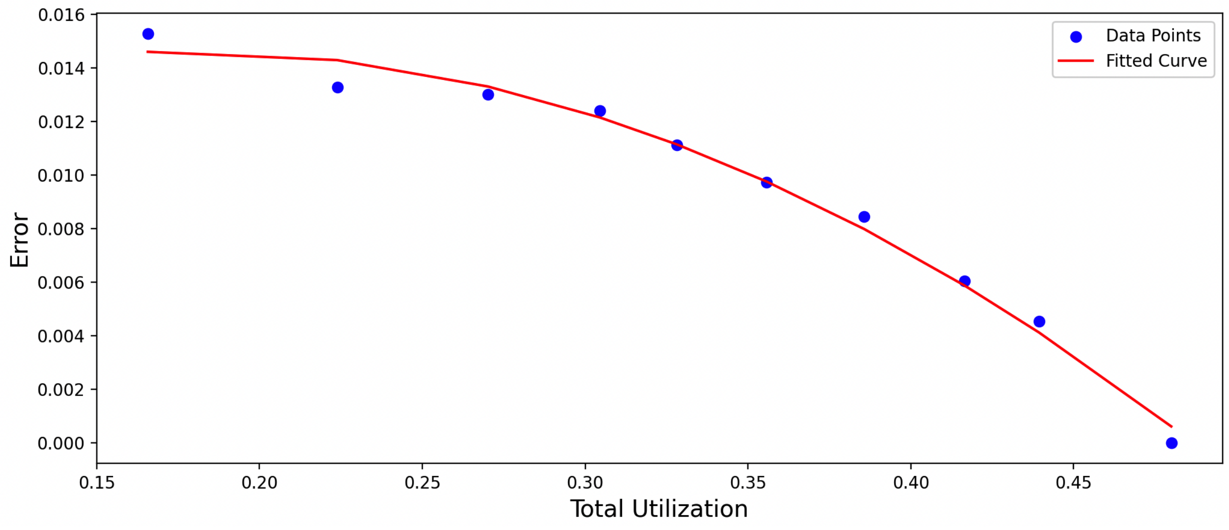

Figure 6.1: *Impact of Threshold Numbers on Utilization and Error Metrics*

In summary, the threshold reduction strategy works well in scenarios with sparse particles, and it offers an efficient approach to adapt the system's performance to different operational constraints with only minor accuracy decreases. We note, however, that this analysis was conducted using a dataset of images with sparsely populated particles. In scenarios with densely populated particle images, reducing thresholds might lead to more pronounced accuracy reductions.

6.6 Adjusting Image Processing Frequency

Adjusting the frequency at which images are processed offers a different way to make the FIMS system more efficient. Unlike previous methods, this strategy does not reduce the amount of work the image processor does. Instead, it spreads out the work over a longer time, effectively slowing down the process to reduce its demand on the system's resources.

To test this, we used a series of images captured every 100ms. By combining, or 'stacking', these images, we simulated longer processing times. For example, stacking two images mimics an image taken over a 200ms period, stacking three images represents 300ms, and so on. The Image Processor will adjust its processing frequency accordingly.

Table 6.4 shows how changing the time taken to process each image impacts both the system's workload and its ability to accurately measure particle sizes. As we increase the processing time, the system's workload decreases, but this also reduces the accuracy of our particle measurements. This effect is measured using two metrics; both metrics, MCS and MNAE show that as processing time increases, accuracy slightly declines.

One key finding is that a processing time of below 300ms seems to offer the best compromise. It significantly reduces the workload on the system while maintaining a reasonable level of accuracy. Going beyond 300ms does not offer much additional benefit in terms of reducing workload but does start to have a more noticeable impact on accuracy. The relationship between Error (y) and utilization (x) can be roughly described by an exponential function with a quadratic expression in the exponent. The model is defined as:

$$y = 1.06 * x^2 - 0.96 * x + 0.22 \tag{6.11}$$

$Duration_{\text{Image Proc.}}$	$WCET_{\text{Image Proc.}}$	$U_{\text{Image Proc.}}$	U_{total}	MCS	MNAE	Error
100 ms	42.3	0.423	0.48	1.000	0.000	0.000
200 ms	42.3	0.216	0.269	0.979	0.152	0.047
300 ms	42.3	0.141	0.198	0.974	0.226	0.066
400 ms	42.3	0.106	0.163	0.964	0.315	0.092
500 ms	42.3	0.085	0.142	0.960	0.246	0.081
600 ms	42.3	0.071	0.128	0.948	0.382	0.118
700 ms	42.3	0.06	0.117	0.938	0.364	0.122
800 ms	42.3	0.053	0.110	0.938	0.331	0.116
900 ms	42.3	0.047	0.104	0.926	0.405	0.140
1000 ms	42.3	0.042	0.1	0.924	0.413	0.142

Table 6.4: Impact of Processing Duration on System Utilization and Accuracy

as plotted in the Figure 6.2.

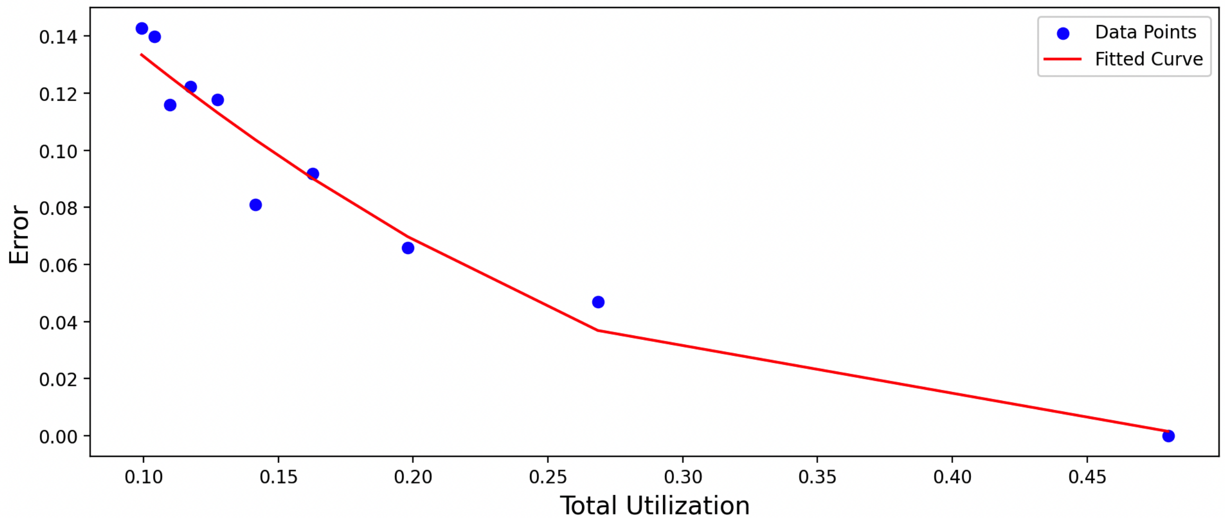

Figure 6.2: *Impact of Image Processing Duration on Utilization and Error Metrics*

In conclusion, while it is beneficial to slow down image processing to a certain extent, there is a limit to how much this helps. After a point, further slowing down does not significantly reduce the workload but does start to compromise the quality of our particle size measurements. Thus, it is crucial to find the right balance, considering the specific requirements and limitations of the environment where FIMS will be used.

Chapter 7

Conclusion and Future work

7.1 Conclusion

In this thesis, we have described the development of a real-time analysis pipeline for the Fast Integrated Mobility Spectrometer (FIMS), a tool for measuring aerosol size distributions in the atmosphere. The FIMS instrument, traditionally reliant on post-measurement analysis, has been re-engineered to provide real-time particle distribution insights.

A core contribution of this thesis involved translating FIMS's algorithmic operations into a multithreaded C++ framework, enabling the efficient processing of atmospheric data in real time. The pipeline's multithreaded design allows for simultaneous image processing and numerical computations, including data inversion. We also analyzed the efficiency and effectiveness of the real-time analysis pipeline, focusing on both its performance and accuracy under various conditions. To analyze the pipeline's performance, we examined its behavior in isolated environments, in multitasking scenarios, and under resource constraints.

Our investigation began with a focus on the pipeline's computational components, especially the image processor, which emerged as the most computationally intensive element. We explored several methods to optimize its utilization, including implementing the find-peak algorithm, adjusting operational frequencies, and reducing the number of thresholds in particle detection. Each method demonstrated varying degrees of impact on system efficiency and accuracy, with the find-peak algorithm significantly reducing computational load at a limited cost to accuracy, though it lacked flexibility to adapt dynamically to varying resource constraints. Adjusting the image processing frequency is a viable strategy for reducing resource utilization, but extending the processing period beyond 300ms leads to a noticeable decrease in accuracy. Meanwhile, reducing the number of thresholds significantly improves the pipeline's efficiency, but it only maintains high accuracy in scenarios with sparsely distributed particles.

In terms of the system's multitasking capabilities, we observed that certain processes, like image saving, were heavily influenced by the operating system's multilayer buffering mechanism, leading to prolonged execution times. This finding highlighted the importance of isolating such processes to prevent them from impacting the performance of critical components.

Accuracy assessment was another critical component of this research. We utilized a dataset of 12,000 images collected over 20 minutes to evaluate the real-time pipeline's accuracy against the conventional offline method. This comparison spanned various metrics, including total particle counts, concentration over time, and size distribution. The findings revealed a high degree of similarity between the real-time and offline methods, underscoring the reliability of the real-time pipeline.

System latency is a significant factor affecting the accuracy of particle detection. We investigated the effects of varying intentional system delays on the accuracy of particle detection, finding that delays shorter than 1.5 seconds significantly compromised accuracy. This study highlighted the delicate balance between system delay and data accuracy, emphasizing the importance of carefully calibrating delay duration to ensure reliable data collection.

Overall, this thesis detailed the development and assessment of a real-time pipeline for aerosol measurement, highlighting both its efficiency and effectiveness. By integrating a real-time pipeline, FIMS now offers immediate data interpretation, crucial for dynamic environmental studies. This advancement significantly enhances FIMS's capabilities, making it a more versatile tool in atmospheric research, particularly for studying rapidly changing aerosol environments.

7.2 Future Work

7.2.1 Realizing the Potential of Real-Time Analysis

In future work on FIMS, we will focus on advancing data reliability, streamlining resource utilization, and melding human discernment with automated precision for enhanced decision-making processes.

Real-time Anomaly Detection and Correction

One area of future focus will be the integration of an anomaly detection feature into the FIMS real-time analysis pipeline. This advancement will address potential errors in instrument setup, which can compromise data reliability. Anomaly detection in real-time would enable immediate corrective actions, thereby preventing the costly process of re-measurement, particularly in data gathered via aircraft.

The development of this feature will involve creating new algorithms [4], drawing on either parametric statistical methods or machine learning techniques. These algorithms must be capable of promptly identifying and rectifying any discrepancies in data collection. The incorporation of such anomaly detection would not only bolster the trustworthiness of the aerosol data but also enhance the overall efficiency of atmospheric data-gathering operations.

Optimizing Resource Allocation

In traditional approaches, aerosol data collection paths are typically preset based on specific research objectives, such as examining size distributions at certain altitudes and locations. However, this method can miss unforeseen yet scientifically significant events, like abrupt shifts in aerosol distribution within a particular area. These events are often discovered only during post-collection analysis, by which time the chance to gather more detailed observations has passed.

The integration of real-time analysis into FIMS enhances our ability to detect significant environmental events as they unfold. By leveraging a combination of sensor fusion, pattern recognition, and predictive modelling techniques, we can promptly identify unexpected variations in aerosol size distributions during data collection. This approach allows us to adopt a more flexible and responsive strategy for data capture. For example, when a spike in certain particle sizes is detected, indicating a specific type of particulate, relevant instruments [1] can be activated for further analysis and verification to better understand the event. Predictive models further enhance this approach by projecting the possible evolution of these events, guiding ongoing data collection efforts. Another application is data collection path adjustment in response to localized events. When such an event is detected, the path of the research aircraft or vehicle can be immediately redirected to focus on the specific area. Such an adaptive method would facilitate deeper and more immediate insight into atmospheric conditions and phenomena.

Dynamic Path Planning for Data Collection

Adapting to the real-time demands of scientific research and operational constraints is a fundamental aspect of dynamic measurement path planning, particularly when using ground vehicles for aerosol data collection. The unpredictable nature of field conditions presents a considerable challenge.

Dynamic path planning should endeavor to achieve a delicate balance among several factors — the constraints of the environment, the quality and integrity of the data, and the efficiency of energy use — all while managing logistical and operational considerations [11]. To address these complexities, we propose the development of a multi-objective optimization strategy. This strategy will evaluate parameters, including fuel consumption, the scientific value of captured data, and current environmental conditions, to dynamically optimize data collection routes.

Implementing such a multifaceted solution will enable us to adjust our strategies responsively as field conditions evolve. For instance, in the event of detecting an aerosol plume, such as one arising from a sudden industrial discharge or a natural occurrence like a wildfire, the system should promptly recalibrate its path, aiming to predict the plume's trajectory while maintaining a balance between the necessity for adequate data collection and the limitations posed by environmental and resource constraints. This targeted approach ensures that we capture extensive and valuable information about the aerosol event.

Human and Machine Teaming

Another future work direction is improving aerosol measurements by teaming up human expertise with machine efficiency. Our approach will involve human operators in aircraft-based measurement systems. Operators will use their judgment to make final decisions, guided by real-time recommendations from our human-in-the-loop model. This model will incorporate findings from anomaly detection and multi-objective optimization algorithms to suggest the best courses of action.

We will make sure that human decisions are not just the endpoint but also feed back into improving the system. By doing so, the operators' experiences and choices will help to fine-tune the algorithms, making them smarter over time. This feedback loop ensures that the technology adapts to real-world complexities, enhancing our ability to measure aerosols accurately and efficiently.

7.2.2 Other Areas for Real-Time Sensor Analysis Pipelines

We seek to extend the functionality of our real-time analysis pipeline beyond its current scope, targeting other applications that rely on UAVs and other mobile, automated sensing platforms.

In pollution monitoring [13], the adoption of real-time pipelines can significantly enhance the operational efficacy of UAV systems. Real-time data processing capabilities would enable on-the-spot identification of pollution concentrations, providing urgent data for immediate public and regulatory response. Such advancements promise to make pollution monitoring a more dynamic and effective tool for environmental protection.

Similarly, in the oil and gas industry [2], the introduction of real-time analysis to drone technology promises substantial improvements in precision and safety. Real-time data processing can lead to quicker detection and response to oil spills and pipeline issues, thereby mitigating environmental risks and preserving valuable resources.

References

[1] H. Alali, Y. Ai, Y.-L. Pan, G. Videen, and C. Wang. A collection of molecular fingerprints of single aerosol particles in air for potential identification and detection using optical trapping-raman spectroscopy. *Molecules*, 27(18):5966, 2022.

[2] S. Asadzadeh, W. J. D. Oliveira, and C. R. D. Souza Filho. UAV-based remote sensing for the petroleum industry and environmental monitoring: State-of-the-art and perspectives. *Journal of Petroleum Science and Engineering*, 208:109633, 2022.

[3] J. D. Cerro, C. C. Ulloa, and A. Barrientos. Unmanned aerial vehicles in agriculture: A survey. *Agronomy*, 11(2):203, 2021.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[5] Y.-S. Cheng. Condensation particle counters. *Aerosol measurement: Principles, techniques, and applications*, pages 381–392, 2011.

[6] J. Fan, D. Rosenfeld, Y. Zhang, S. E. Giangrande, Z. Li, L. A. T. Machado, S. T. Martin, Y. Yang, J. Wang, P. Artaxo, H. M. J. Barbosa, R. C. Braga, J. M. Comstock, Z. Feng, W. Gao, H. B. Gomes, F. Mei, C. Pöhlker, M. L. Pöhlker, and U. Pöschl. Substantial convection and precipitation enhancements by ultrafine. *Science*, 359(6374):411–418, 2018.

[7] H. Fissan, C. Helsper, and H. Thielen. Determination of particle size distributions by means of an electrostatic classifier. *Journal of Aerosol Science*, 14(3):354–357, 1983.

[8] S. K. Friedlander. Smoke, dust and haze: Fundamentals of aerosol behavior, 1977.

[9] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica. D3: a dynamic deadline-driven approach for building autonomous vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 453–471, 2022.

[10] P. Intra and N. Tippayawong. An overview of differential mobility analyzers for size classification of nanometer-sized aerosol particles. *Songklanakarin Journal of Science & Technology*, 30(2), 2008.

[11] M. Jones, S. Djahel, and K. Welsh. Path-planning for unmanned aerial vehicles with environment complexity considerations: A survey. *ACM Computing Surveys*, 55(11):1–39, 2023.

[12] M. Jones, S. Djahel, and K. Welsh. Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey. *ACM Computing Surveys*, 55(11):1–39, 2023.

[13] J. Jońca, M. Pawnuk, Y. Bezyk, A. Arsen, and I. Sówka. Drone-Assisted Monitoring of Atmospheric Pollution—A Comprehensive Review. *Sustainability*, 14(18), 2022.

[14] J. Kangasluoma and M. Attoui. Review of sub-3 nm condensation particle counters, calibrations, and cluster generation methods. *Aerosol Science and Technology*, 53(11):1277–1310, 2019.

[15] B. P. Lee, Y. J. Li, R. C. Flagan, C. Lo, and C. K. Chan. Sizing Characterization of the Fast-Mobility Particle Sizer (FMPS) Against SMPS and HR-ToF-AMS. *Aerosol Science and Technology*, 47(9):1030–1037, 2013.

[16] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 85–96. IEEE, 2014.

[17] C. Loizidis, M. Costi, N. Lekaki, S. Bezantakos, and G. Biskos. Improved performance of differential mobility analyzers with 3d-printed flow straighteners. *Journal of Aerosol Science*, 145:105545, 2020.

[18] G. R. Markowski. Improving Twomey's Algorithm for Inversion of Aerosol Measurement Data. *Aerosol Science and Technology*, 7(2):127–141, 1987.

[19] J. S. Olfert, P. Kulkarni, and J. Wang. Measuring aerosol size distributions with the fast integrated mobility spectrometer. *Journal of Aerosol Science*, 39(11):940–956, November 2008.

[20] J. Orr, C. Gill, K. Agrawal, J. Li, and S. Baruah. Elastic Scheduling for Parallel Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 6(1):05:1–05:14, 2019.

[21] J. Pagels, A. Gudmundsson, E. Gustavsson, L. Asking, and M. Bohgard. Evaluation of aerodynamic particle sizer and electrical low-pressure impactor for unimodal and bimodal mass-weighted size distributions. *Aerosol Science and Technology*, 39(9):871–887, 2005.

[22] G. Rohi, O. Ejofodomi, and G. Ofualagba. Autonomous monitoring, analysis, and countering of air pollution using environmental drones. *Heliyon*, 2020.

[23] M. Sudvarg, S. Baruah, and C. Gill. Elastic scheduling for fixed-priority constrained-deadline tasks. In *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 11–20. IEEE, 2023.

[24] H. Tammet, A. Mirme, and E. Tamm. Electrical aerosol spectrometer of Tartu University. *Atmospheric Research*, 62(3-4):315–324, 2002.

[25] S. Twomey. Comparison of constrained linear inversion and an iterative nonlinear algorithm applied to the indirect estimation of particle size distributions. *Journal of Computational Physics*, 18(2):188–200, 1975.

[26] J. Volckens and T. M. Peters. Counting and particle transmission efficiency of the aerodynamic particle sizer. *Journal of Aerosol Science*, 36(12):1400–1408, 2005.

[27] J. Wang, R. Krejci, S. Giangrande, C. Kuang, H. M. Barbosa, J. Brito, S. Carbone, X. Chi, J. Comstock, F. Ditas, J. Lavric, H. E. Manninen, F. Mei, C. Pöhlker, M. L. Pöhlker, J. Saturno, B. Schmid, R. A. Souza, S. R. Springston, and S. T. Martin. Amazon boundary layer aerosol concentration sustained by vertical transport during rainfall. *Nature*, 539(7629):416–419, 2016.

[28] J. Wang, M. Pikridas, T. Pinterich, S. R. Spielman, T. Tsang, A. McMahon, and S. Smith. A Fast Integrated Mobility Spectrometer for rapid measurement of submicrometer aerosol size distribution, Part II: Experimental characterization. *Journal of Aerosol Science*, 113:119–129, August 2017.

[29] J. Wang, M. Pikridas, S. R. Spielman, and T. Pinterich. A fast integrated mobility spectrometer for rapid measurement of sub-micrometer aerosol size distribution, Part I: Design and model evaluation. *Journal of Aerosol Science*, 108:44–55, June 2017.

[30] S. C. Wang and R. C. Flagan. Scanning Electrical Mobility Spectrometer. *Aerosol Science and Technology*, 13(2):230–240, November 1989.

[31] W. Wang, X. Zhao, J. Zhang, Y. Yang, T. Yu, J. Bian, H. Gui, and J. Liu. Design and evaluation of a condensation particle counter with high performance for single-particle counting. *Instrumentation Science & Technology*, 48(2):212–229, 2020.

[32] Y. Wang, T. Pinterich, and J. Wang. Rapid measurement of sub-micrometer aerosol size distribution using a fast integrated mobility spectrometer. *Journal of Aerosol Science*, 121:12–20, June 2018.