ABSTRACT

Title of dissertation: MODEL-BASED DESIGN OPTIMIZATION AND SIMULATION FOR DYNAMIC DATA-DRIVEN APPLICATION SYSTEMS

Honglei Li, Doctor of Philosophy, 2020

Dissertation directed by: Professor Shuvra S. Bhattacharyya
Dept. of Electrical and Computer Engineering,
and Institute for Advanced Computer Studies
University of Maryland, College Park

Dynamic Data Driven Application Systems (DDDAS) are an important class of systems in which computations on data and instrumentation components for acquiring the data are incorporated within a feedback control loop. In DDDAS, the modeling and data-driven adaptation of instrumentation is incorporated as an important aspect of the design process. Due to its potential to enhance capabilities of accurate analysis, dynamic decision making, and scalable simulation, the DDDAS paradigm plays an increasingly important role in innovative systems for a wide variety of applications. This thesis develops new model-based, software design tools to support the design and implementation of DDDAS. The methods are developed in the context of two application domains in which DDDAS principles are highly relevant — multispectral/hyperspectral video processing,

and wireless-integrated factory automation systems.

Recent advances in multispectral and hyperspectral video capture technology along with system design trade-offs introduced by these advances present new challenges and opportunities in the area of DDDAS for video analytics. Video analytics plays an important role in a wide variety of defense-, monitoring- and surveillance-related systems for air and ground environments. In this context, multispectral video processing is attracting increased interest in recent years, due in part to technological advances in video capture. Compared with monochromatic video, multispectral video offers better spectral resolution, and different bands of multispectral video streams can enhance video analytics capabilities in different ways.

Video processing systems that incorporate multispectral technology involve novel trade-offs among system design complexities such as spectral resolution, equipment cost, and computational efficiency. The design space of multispectral video processing systems is enriched by considering only the required subset of spectral bands to process as a parameter that can be adjusted dynamically based on data characteristics and constraints involving accuracy, communication, and computation.

Based on this view of selectively-processed bands from multispectral video data, we introduce in this thesis a novel system design framework for dynamic, data-driven video processing using lightweight dataflow (LD) techniques. Our proposed framework, called LDspectral, applies LD, which is an approach for model-based design of signal and information processing systems. LD facilitates efficient and reliable real-time implementation. LD is "lightweight" in the sense that it is based on a compact set of application programming interfaces, and can be integrated relatively easily into existing design processes. We

develop a framework for adaptively configuring multispectral video processing configurations in LDspectral, and develop a prototype implementation using LD methods that are integrated with OpenCV, which is a popular library of computer vision modules.

We demonstrate and evaluate the performance of LDspectral capabilities using a background subtraction application. As compared to a standard video processing pipeline, the capabilities in LDspectral for optimized selection and fusion of spectral bands enhance trade-offs that can be realized between video processing accuracy and computational efficiency. Using the DDDAS paradigm, the elements of sensor measurements, statistical processing, target modeling, and system software are analyzed by frequency bands, video analytics, environmental analysis, and dataflow techniques, respectively.

In this thesis, the LDspectral framework is also extended to hyperspectral video, which offers great spectral resolution and has significant potential to enhance the effectiveness of information extraction from image scenes. An important challenge in the development of hyperspectral video systems is managing the high computational load and storage requirements required to process the large volumes of data that are acquired by these systems.

We also investigate DDDAS-inspired methods in context of distributed, smart factory systems that are equipped with wireless communication capability. We refer to this class of systems as wireless-integrated factory systems (WIFSs). An important challenge in the development of this class of systems is ensuring reliable, low latency communication under the harsh wireless channel conditions of factory environments.

To support the application of the DDDAS paradigm in WIFSs, we develop a model-based software tool for design space exploration. We refer to this tool as the Wireless-

Integrated factory System Evaluator (WISE). WISE supports the rapid simulation-based evaluation of interactions among the placement of factory subsystems, the partitioning of factory subsystems into nodes of a wireless network, the performance of the wireless network, and overall factory system performance. WISE also incorporates a new graphical model called the cyber-physical flow graph, which provides integrated modeling for the flow of physical entities (such as parts that are processed in a factory) and the flow of information. The cyber-physical flow graph also models distributed flows in which information is communicated across multiple network nodes.

Model-Based Design Optimization and Simulation Techniques for Dynamic, Data-Driven Applications Systems

by

Honglei Li

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Shuvra S. Bhattacharyya, Chair/Advisor
Professor Charalampos Papamanthou
Professor Manoj Franklin
Professor Matthew J. Hoffman
Professor Derek A. Paley

Dedication

To my wife, my parents and all my friends

Acknowledgements

Firstly and foremost, I would like to thank Professor Shuvra Bhattacharyya for his great support, guidance, suggestion, and patience throughout my Ph.D. study as my research advisor. His insightful advice and instructions helped me finish my research and thesis. Without his invaluable guidance and continuous help, this dissertation would not have been possible.

I also want to thank my committee members, Professor Franklin, Professor Papamanthou, Professor Hoffman and Professor Paley for providing useful and insightful feedbacks and suggestions to my research.

I am grateful to Professor Mihaela van der Schaar, Doctor Erik Blasch, Professor Zhu Li, Doctor Kishan Sudusinghe, Mr. Richard Candell, Mr. Yongkang Liu, Mr. Mohamed Kashef, Doctor Shuoxin Lin, Doctor Yanzhou Liu, Doctor Lin Li, Doctor Jiahao Wu, Mr. Lei Pan, Mr. Jing Geng, Doctor Kyunghun Lee, Doctor Adrian Sapio, Mr. Eung Joo Lee, Mr. Jing xie, Mr. Xiaomin Wu, Ms. Yaesop Lee and other colleagues and research project collaborators for their generous help and support. I would also like to thank Marshall Plan Foundation and Fachhochschule Salzburg for providing a great opportunity for study in Austria.

Finally I would like to thank my wife and parents for their love and support during my Ph.D study.

The research underlying this thesis was supported in part by U.S. Air Force Office of Scientific Research under DDDAS Program, and National Institute of Standards and Technology.

Table of Contents

List of Figures vi

List of Tables viii

1 Introduction 1
1.1 Multispectral and Hyperspectral Video Stream Processing 1
1.1.1 LDspectral for Multispectral Video Processing 3
1.1.2 Extensions for Hyperspectral Processing 5
1.2 Wireless-Integrated Factory Automation Systems 6
1.3 Contributions of this Thesis 8
1.4 Outline of this Thesis 9

2 The LDspectral Framework for Multispectral Image and Video Processing 10
2.1 Introduction 11
2.2 Methodology 14
2.3 Results 20
2.4 Summary 25

3 Extensions to LDspectral for Dynamic Data-Driven Video Processing 27
3.1 Introduction 28
3.2 System Design 29
3.2.1 Run Time System 31
3.2.2 Band Subset Processing 33
3.2.3 Band Subset Selection 37
3.3 Results 40
3.3.1 Experimental Setup 40
3.3.2 Example Images 41
3.3.3 Accuracy Evaluation 43
3.3.4 Execution Time Evaluation 47
3.4 Summary 48

4 Hyperspectral Video Processing on Resource-Constrained Platforms 49
4.1 Introduction 50
4.2 Design Methods 54
4.3 Results 63
4.4 Additional Experiments 68
4.5 Summary 71

5 Design Space Exploration for Wireless-Integrated Factory Automation Systems 74
5.1 Introduction 75
5.2 Related Work 77
5.3 Design Flow of Cosimulator 78
5.3.1 Model-Based Architecture 79

5.3.2 Designer Input . . . 80
5.3.3 Factory Dataflow Graphs . . . 81
5.3.4 Network Mapping and Configuration . . . 83
5.3.5 Lower Level Models and Auto-generation . . . 84
5.4 Dataflow and Wireless Communication Models . . . 86
5.4.1 Cyber Physical Flow Graph . . . 86
5.4.2 Communication Link Modeling . . . 88
5.4.3 Autogeneration Example . . . 89
5.5 Results . . . 93
5.5.1 Factory Layout Parameters . . . 93
5.5.2 Experiment Parameters . . . 95
5.5.3 Variation of Communication Delay with N_p . . . 97
5.5.4 Variation of Communication Delay with Both N_m and N_p . . . 98
5.5.5 Varying the Distance Parameters d_x and d_y . . . 100
5.5.6 Varying the Wi-Fi Manager Algorithm . . . 101
5.5.7 Shared Access Point across Pipelines . . . 101
5.6 Additional Experiments . . . 103
5.7 Summary . . . 106

6 Conclusions and Future Work 108
6.1 Conclusions . . . 108
6.2 Future Work . . . 109
6.2.1 Generalization of LDspectral . . . 109
6.2.2 Multiple Objective Optimization Using WIFS . . . 111

Bibliography 112

List of Figures

2.1 An illustration of multispectral image capture. 11
2.2 An example of a single video frame in the employed multispectral data set. Images 1–6 show the 6 visible bands, Image 7 corresponds to the near-infrared band, and Image 8 is the corresponding foreground result that is derived using LDspectral. 13
2.3 Block diagram of the design flow in LDspectral. 16
2.4 Block diagram of band subset processing in the background subtraction system. 18

3.1 Block diagram of the LDspectral Run-time System Model (LRSM). . . . 32
3.2 Dataflow representation of the band subset processing (BSP) subsystem shown in Figure 3.1. 34
3.3 An example from Benezeth's dataset that is used to illustrate the techniques for fusion and background subtraction that are employed in LDspectral: the scene, 7 bands, and foreground fused image. 42

4.1 A hyperspectral cube of a scene in a dataset that we use in our experiments in this chapter. 51
4.2 Experimental setup for applying LDspectral to Android-based hyperspectral video processing implementation. 56
4.3 Dataflow graph for an example configuration of the first-version HVPS. . 58
4.4 Multithreaded version of Fig. 4.3 for mapping onto the targeted Android platform. 59
4.5 Multithreaded version of Fig. 4.3 for $P = 6$, and $Q_p = Q_b = 3$. 61
4.6 Frame rate and accuracy for different N_b and different multithreading configurations. 65
4.7 Variation in measured accuracy ($F_{measure}$) for different values of N_b. . . . 68
4.8 Variation in battery consumption and execution time per frame for different values of N_b. 69
4.9 Results from applying Algorithm 2 under specific operational constraints: $(C_r, f_M) = (1.3, 0.9)$. 72
4.10 Results from applying Algorithm 2 under specific operational constraints: $(C_r, f_M) = (5.0, 0.95)$. 73

5.1 An illustration of the new design flow involved in applying WISE for WIFS design space exploration. 79
5.2 Communication link modeling in WISE. 89
5.3 An example of a Factory Dataflow Graph. 90
5.4 Autogenerated CPFG. 91
5.5 The network model that is autogenerated by WISE for the example associated with Figure 5.3 and Figure 5.4. 92
5.6 Factory layout example. 95
5.7 Variation in average communication delay with N_p. 98

5.8 Variation in average communication delay with $N_p = N_m = K$. 99
5.9 Histogram of average communication delay with varying d_x, d_y. 100
5.10 Variation in average communication delay with the Wi-Fi manager algorithm and number of machines N_m. 102
5.11 Variation in average communication delay with N_p when a single, shared access point is used across all pipelines. 103
5.12 Measured relationship between the average communication delay and the distance between adjacent network nodes for a specific factory configuration. 104
5.13 Measured relationship between the average packet retransmission rate and the distance between adjacent nodes. 105
5.14 Measured relationship between the average communication delay and the selected rate control algorithm for an inter-node spacing of 20 meters. . . 106
5.15 Relationship between the average packet retransmission rate and the rate control manager algorithm. 107

List of Tables

2.1 Accuracy results for different one- and two-band combinations using LDspectral, and (in the last three rows) the results from [1] with three different algorithms. 23
2.2 Execution times for different single- and dual-band combinations. The results here are given in milliseconds. Each entry in the table represents the average time to perform background subtraction (including the entire processing chain shown in Figure 2.4 on a single input image). 23
2.3 Derived values for PBC parameters (rounded to tenths). 24

3.1 Accuracy results for different one- and two-band subsets using LDspectral with both PLF and FLF. In each off-diagonal table entry, the top value corresponds to PLF, and the bottom value corresponds to FLF. 44
3.2 Results of incremental band subset construction using the LBSS algorithm in LDspectral. 45
3.3 Accuracy improvement compared with results from [1] using the same multispectral dataset. 46
3.4 Variation in execution time for different numbers of processed bands and different fusion modes. The units of execution time in this table are milliseconds/frame. 47

4.1 Derived energy consumption, execution time, and CPU usage on Android device for sequential version. 64
4.2 Statistics on the measured frame rates for different operational configurations. The unit for each entry in the table is frames per second (fps). . . . 67
4.3 Frame rate (frames per second) for different multithreading configurations. 70

5.1 Simulation parameters. 96

List of Abbreviations

AARF Adaptive Auto Rate Fallback

AARFCD Collision detection for adaptive auto rate fallback

ADB Android Debug Bridge

AWGN Additive White Gaussian Noise

AMRR Adaptive Multi Rate Retry

BSP Band Subset Processing

BSS Band Subset Selection

CNS Communication Network Simulation

CPFG Cyber-physical Flow Graph

DDDAS Dynamic Data Driven Application and System

DICE DSPCAD Integrative Command Line Environment

DRSM Dual-Rail Single Machine

FIFO First-in-first-out

FLF Feature-level Fusion

FSM Finite State Machine

GDT Global Dataflow Time

GMM Gaussian Mixture Model

HVPS Hyperspectral Video Processing System

LBSS LDspectral Band Subset Selection

LD Lightweight Dataflow

LIDE LIghtweight Dataflow Environment

LRSM LDspectral Run-time System Model

NDK Native Development Kit

PBC Pairwise Band Combination

PLF Pixel Level Fusion

RIA Receive Interface Actor

SIA Send Interface Actor

TLFS Tau Lide Factory Simulation

WIFS Wireless-Integrated Factory System

WISE Wireless-Integrated factory System Evaluator

Chapter 1

Introduction

Dynamic Data Driven Application Systems (DDDAS) are an important class of systems in which computations on data and instrumentation components for acquiring the data are incorporated within a feedback control loop [2]. In DDDAS, the modeling and data-driven adaptation of instrumentation is incorporated as an important aspect of the design process. Due to its potential to enhance capabilities of accurate analysis, dynamic decision making, and scalable simulation, the DDDAS paradigm plays an increasingly important role in innovative systems for a wide variety of applications [3]. This thesis develops new model-based, software design tools to support the design and implementation of DDDAS. The methods are developed in the context of two application domains in which DDDAS principles are highly relevant — multispectral/hyperspectral video processing, and wireless-integrated factory automation systems.

1.1 Multispectral and Hyperspectral Video Stream Processing

The additional spectral bands available in multispectral and hyperspectral video streams offer the potential for more accurate knowledge extraction, but also increase costs associated with real-time processing, energy consumption, and storage requirements. In this thesis, we develop data-driven models and methods that address these trade-offs to systematically perform design optimization of multispectral and hyperspectral video pro-

cessing systems. For concreteness, we develop and demonstrate these methods in the context of a specific video processing application — that of background subtraction, which is widely-used in many application areas that require automated detection of moving targets.

Multispectral imaging is related to hyperspectral imaging in that both provide increased spectral discrimination compared to traditional imaging methods. The difference is primarily in the number of bands employed and the degree of spectral resolution (e.g., see [4]). Whereas multispectral imaging generally refers to a number of bands in the range of about 3–10, hyperspectral imaging uses significantly larger numbers of bands — e.g., hundreds, thousands or more — and narrower bandwidths.

Video analytics plays an important role in a wide variety of defense-, monitoring- and surveillance-related systems for air and ground environments. In this context, multispectral and hyperspectral video processing is attracting increased interest in recent years, due in part to technological advances in video capture. Compared to monochromatic video, multispectral/hyperspectral video streams offers better spectral resolution, and different bands of such video streams can enhance video analytics capabilities in different ways. For example, the infra-red bands can provide better separation of shadows from objects, and improved spatial resolution in scenes that are impaired by fog or haze [5].

In the remainder of this section, we first introduce a novel design framework that we have developed in this thesis for multispectral video processing systems. Then we introduce extensions that we have made to the design framework to the more complex domain of hyperspectral video processing.

1.1.1 LDspectral for Multispectral Video Processing

Multispectral video acquisition technology introduces novel opportunities and challenges for applying the paradigm of DDDAS to design and implementation of video analytics systems. The subset of available multispectral bands that is stored and processed, and the hardware and software configurations that are used to perform the processing introduce a complex design space.

The most effective operating point in the design space associated with a multispectral video processing system is in general dependent on the specific application scenario and data characteristics that are encountered at a given point in time during system operation. For example, when system accuracy is of greatest importance, it may be desirable to operate on the full set of available bands, while in situations where resource constraints are critical (e.g., due to failures in certain subsystems or limited energy capacity), it may be most effective to select a proper subset of the available bands and process the selected bands in a way that optimizes accuracy subject to the given resource limitations.

Based on this view of selectively-processed bands from multispectral video data, we introduce in this thesis a novel system design framework for dynamic, data-driven video processing. A central part of our framework is the application of model-based design methods based on dataflow techniques to represent and transform the functionality of multispectral video processing systems. This allows us to leverage existing knowledge on dataflow techniques, which are employed for design optimization in a wide variety of signal processing application areas, including speech processing, wireless communications, and video processing (e.g., see [6]).

The approach that we develop in this thesis supports the development of new DDDAS methods to dynamically select subsets of multispectral bands to process, and dynamically reconfigure the dataflow within the targeted video processing system to achieve the required processing on the selected subset of bands. As discussed previously, DDDAS is a paradigm that unifies computational and instrumentation aspects of applications systems, and thereby promotes deeply integrated approaches to modeling, sensing, control, and data processing. DDDAS principles have great relevance to aerospace applications (e.g., see [7, 8, 9]). In the methods developed in this thesis, we model and optimize video processing trade-offs across algorithm and implementation aspects through the DDDAS paradigm. Using the DDDAS paradigm, our proposed methods are designed to apply performance data that is collected through execution time instrumentation, and adapt video processing configurations dynamically according to their trade-off models, and according to constraints on real-time performance.

We refer to our proposed new approach for multispectral image processing as *LDspectral*, where "LD" here stands for *lightweight dataflow* [10, 11]. LD is a lightweight design methodology that facilitates cross-platform prototyping, experimentation and design optimization of signal processing systems. Lightweight dataflow is "lightweight" in the sense that it is based on a compact set of application programming interfaces that can be retargeted to different platforms and integrated into different design processes relatively easily. The *lightweight dataflow environment* (*LIDE*) is a software tool that supports the lightweight dataflow design methodology, and that we apply in this work [11].

We prototype LDspectral using LIDE together with OpenCV, and present results of extensive experimentation with this prototype to demonstrate the utility of LDspec-

tral. OpenCV provides a large library of software components for video processing (e.g., see [12]), including specialized capabilities that are complementary to the capabilities of LIDE for model-based design and implementation. In particular, the dataflow-based embedded software components (*actors*) that we employ to implement LDspectral incorporate calls to relevant OpenCV functions to perform specific image processing operations.

We demonstrate and evaluate the performance of LDspectral capabilities using a background subtraction application, along with a recently-introduced data set for experimenting with multispectral background subtraction techniques [1]. As compared to a standard image processing pipeline, the dynamic, integrated adjustment of data flow and spectral band selection provides systematic trade-off optimization among computational efficiency and multispectral video processing accuracy.

1.1.2 Extensions for Hyperspectral Processing

In Section 1.1.1, we introduced the complex design space that is involved in development of advanced systems for multispectral video processing. This design space becomes even more complex in the case of hyperspectral video processing, where the number of spectral bands involved may be orders of magnitude higher. To help address this challenge, we have developed extensions for optimized hyperspectral processing to the LDspectral system discussed in Section 1.1.1.

In particular, we incorporated new methods for dynamic system-level reconfiguration and multithreading acceleration into LDspectral. These methods enable efficient processing of hyperspectral images on resource-constrained platforms. In our developed

multithreaded configuration, image partitioning and image stitching are incorporated to partition the image into subframes, thereby helping to increase the concurrency of the image analysis operations. To balance the load on different threads, a *distributor subsystem* is designed to allocate subsets of bands to processing threads on the target platform. We design table-driven algorithms to efficiently adapt real-time embedded processing configurations based on accuracy and performance under constraints involving accuracy and energy consumption. To demonstrate the effectiveness of the proposed hyperspectral system, we experiment with its application to background subtraction on an Android platform.

While the methods for efficient multispectral and hyperspectral video processing developed in the thesis are demonstrated concretely in the context of background subtraction, the underlying approaches are more general and can be adapted to other video analysis contexts. Investigation of such adaptations is a useful direction for future work.

1.2 Wireless-Integrated Factory Automation Systems

The incorporation of increasing amounts of wireless communication capabilities in manufacturing systems brings increasing requirements for stable wireless networks that can simultaneously provide reliable and low latency communications. There exist many considerations in the design of industrial wireless communication systems including latency and reliability, as mentioned above, as well as additional factors, such as energy consumption and scalability.

A variety of different standards may be considered for industrial wireless networks;

some examples are IEEE80211a/ac/b/g/n. The design of factory communication systems involves many parameters, such as the modulation type, coding rate, data rate and rate control manager. These parameters in general have complex interactions among themselves, as well as with physical aspects of a factory layout. In this thesis, we have explored model-based methods that enable efficient design space exploration across the complex parameter combinations and interactions that are involved in design and optimization of factory communication systems.

In particular, we have developed a novel software tool for model-based design space exploration of factory systems that are integrated with wireless communication capabilities. The tool involves a novel graphical representation called a cyber-physical flow graph, which captures both the flow of information and physical entities, such as parts that transported across assembly lines.

The proposed tool supports not only the autogeneration of different factory simulation models with different layouts, but also the dynamic reconfiguration of wireless network parameters. Factory control models and wireless communication models are integrated by systematically interfacing between dataflow and discrete event simulation engines, respectively. The autogeneration of low-level simulation code is from high-level abstract models, which allows designers to rapidly experiment across alternative configurations and iteratively explore the design space. In this thesis, we demonstrate the usage of the proposed new tools for factory system design space exploration through extensive experiments with different factory layout parameters and various wireless network configurations.

1.3 Contributions of this Thesis

In this thesis, we have developed new model-based software tools to support prototyping, design space exploration, and optimization of Dynamic Data Driven Applications Systems (DDDAS). We have focused concretely on two application domains in which DDDAS principles are highly relevant — multispectral/hyperspectral video processing, and wireless-integrated factory automation systems.

The specific contributions of this thesis are summarized as follows. First, we have developed a novel framework called LDspectral for design and implementation of multispectral and hyperspectral video processing systems based on lightweight dataflow (LD) techniques. LDspectral enables the efficient implementation of image processing and video analytics in part through its support for adapting image analysis functionality to operate on subsets of selected multispectral or hyperspectral bands. The methods developed in LDspectral also provide flexible optimization and adaptation of video processing configurations based on dynamically changing performance objectives and operational constraints.

Second, we have developed a novel software tool for model-based design space exploration of factory systems that are integrated with wireless communication capabilities. The tool, called Wireless-Integrated factory System Evaluator (WISE), integrates the design perspectives of physical factory layouts, factory process flows, and wireless communications, including protocol functionality and channel characteristics.

1.4 Outline of this Thesis

The remainder of this thesis is organized as follows. Chapter 2 introduces our first-version development of the LDspectral framework, which is targeted to multispectral image and video processing. Chapter 3 presents extensions to LDspectral that make the framework more useful in the context of DDDAS. Chapter 4 extends LDspectral to support hyperspectral image and video processing, and to enable efficient implementation on resource-constrained platforms. Chapter 5 presents a software tool, called the Wireless-Integrated factory System Evaluator (WISE), for applying DDDAS methods to the area of smart factory systems that are equipped with wireless communication capability. Chapter 6 summaries the thesis and discusses possible directions for future work that are motivated by the thesis.

Chapter 2

The LDspectral Framework for Multispectral Image and Video Processing

In this section, we present our first-version development of the LDspectral framework, which was introduced in Chapter 1. This first-version of LDspectral is targeted to multispectral image and video processing.

Image and video processing applications, such as object detection, image classification, and object tracking, are playing increasingly important roles in numerous areas. In these areas, traditional imagery, with only RGB sensors, is commonly used. However, the restriction to use of RGB sensors can result in loss of information with the loss being particularly pronounced under certain conditions. For example, analysis that is restricted to RGB bands may perform poorly on images that are captured from dark environments, whereas use of near infra-red (NIR) spectral bands can significantly enhance the potential for accurate analysis of images captured from such environments. In situations like these, multispectral imaging technology offers more information such as that provided by NIR bands. Thus, multispectral image and video processing technology has gained significant popularity during recent years. However, design and implementation of multispectral image and video processing systems presents important challenges, including the problem of efficient knowledge extraction from the significantly increased volumes of data that are involved in multispectral systems compared to RGB systems. In this chapter, we present the first-version of the LDspectral framework, which facilitates design and

implementation of efficient multispectral image and video processing systems.

Material in this chapter was published in preliminary form in [13]

2.1 Introduction

Multispectral sensor technology is used in a variety of applications for monitoring and surveillance in ground and air environments, such as land cover classification, and thermal mapping. Figure 2.1 provides an illustration of multispectral image capture in which a camera captures seven different bands for the same scene. Multispectral images typically cover three to ten spectral bands that range from near infrared to visible light. In recent years, advances in sensor technology have helped to increase the effectiveness and decrease the cost of multispectral imaging systems, and make these systems practical to employ in an increasing variety of applications (e.g., see [14]).

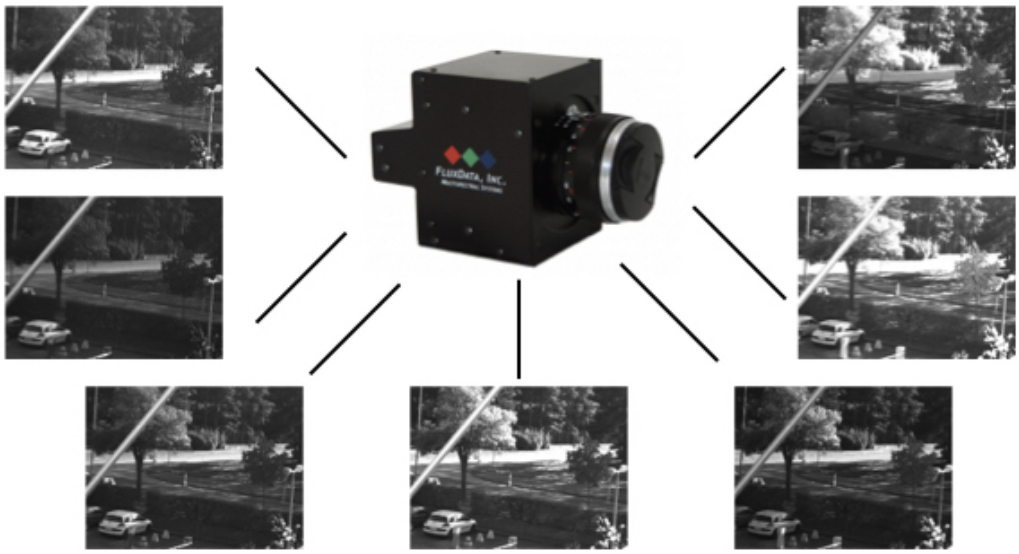

Figure 2.1: An illustration of multispectral image capture.

Like hyperspectral imaging, multispectral imaging provides increased spectral dis-

crimination compared to traditional imaging methods. However, multispectral imaging systems employ much fewer spectral bands — in the range of 3 to about 10 — while hyperspectral imaging systems can involve hundreds, thousands or even more bands [14]. In addition to being more numerous, the bands employed in hyperspectral imaging systems also have narrower bandwidths. Although the contributions of this chapter are introduced in the context of multispectral imaging systems, they have the potential for adaptation to hyperspectral systems. Investigating such adaptations is a useful direction for future work.

Benezeth et al. [1] present a publicly available collection of multispectral video sequences that includes ground truth annotation of moving objects. They also apply this data set to demonstrate improvements in background subtraction accuracy when using multispectral video streams compared to RGB streams. Additionally, they provide an evaluation of alternative background subtraction techniques that operate on multispectral video.

We demonstrate the capabilities of the proposed DDDAS-motivated LDspectral system using the multispectral data set introduced in [1]. Figure 2.2 shows an example of the data associated with a single video frame within the employed multispectral data set. Specifically, Figure 2.2 shows 7 different images corresponding to the 7 different bands for the same scene and the foreground result of this scene.

Our work on LDspectral is different from the methods discussed in [1] in its emphasis on integrating DDDAS methods into multispectral video processing, and specifically, on supporting flexible optimization involving the subset of available multispectral bands that is processed, and the associated trade-offs between accuracy and computational cost.

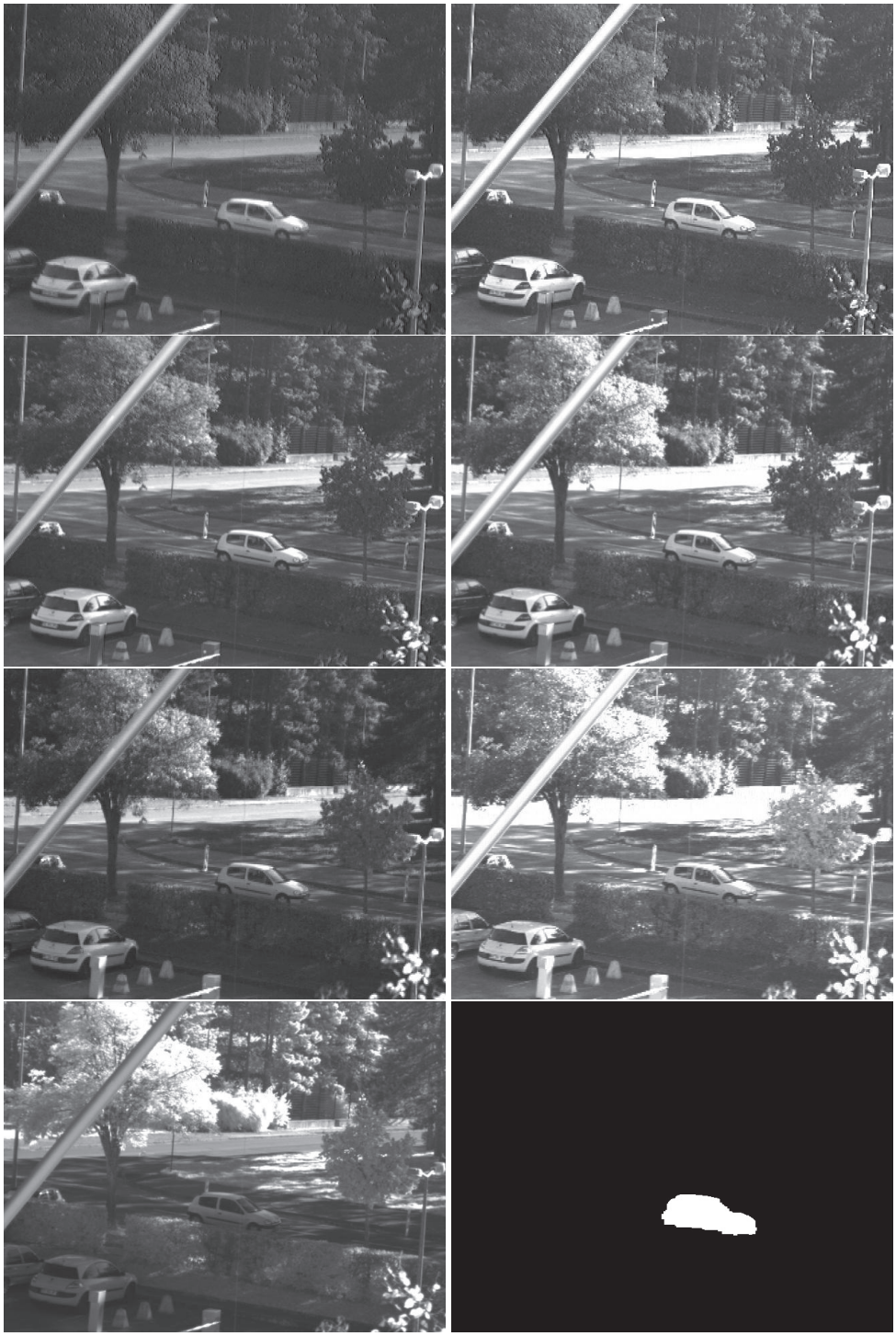

Figure 2.2: An example of a single video frame in the employed multispectral data set. Images 1–6 show the 6 visible bands, Image 7 corresponds to the near-infrared band, and Image 8 is the corresponding foreground result that is derived using LDspectral.

Additionally, pixel-level fusion in the front-end of the processing chain for background subtraction is investigated. Pixel-level fusion improves computational efficiency, and reduces the execution time costs incurred by incorporating additional bands (i.e., larger subsets of the available bands) into the video processing pipeline.

While pixel-level fusion is applied in our demonstration of LDspectral, the LDspectral framework does not require use of pixel-level fusion, nor any other specific form of multi-band processing. This flexibility allows for integration and experimentation with alternative methods for fusion and analysis of video data across multiple bands (e.g., see [15, 16, 5]) that may enhance the available operating points and overall system adaptivity in terms of accuracy, throughput, and other relevant metrics. Exploration of such alternative methods in the context of LDspectral is an interesting direction for further study to develop multi-spectral image fusion systems with user interaction [17].

The developments in this thesis provide new models and methods that are promising for integration in cloud-computing frameworks for information fusion, such as the class of frameworks reviewed in [18]. Exploration of such integration is another useful direction for further investigation.

2.2 Methodology

Compared to traditional imaging method, multispectral imaging provides increased spectral discrimination which can exploit increasing spectral resolution and spectral diversity. Conventional approaches assume that all of the available bands are employed for the video processing tasks. When system accuracy is of the greatest importance, it may

be desirable to use all bands. However, it may be most effective to select a proper subset of all the available bands in situation where resource constraints are critical due to failures in certain subsystems or limited energy capacity.

In the DDDAS-driven video processing system design problem that we target in this thesis, we assume the availability of multispectral data that comes from a set $Z = \{B_1, B_2, \ldots, B_N\}$ of spectral bands, where N denotes the total number of available bands. In resource- or heavily performance-constrained scenarios where it may not be desirable or feasible to process all bands, this leads a problem of strategically selecting a subset $S \in 2^Z$, where 2^Z is the power set of Z — that is, the set of all subsets of Z.

We assume that we are given a constraint C_r (in units of time) on execution time performance for a particular video processing scenario. Our problem then is to select the set $S \in 2^Z$ to store and process, and the associated strategy to process this selected subset of bands such that video analysis accuracy is maximized subject to the constraint C_r. In this Thesis, we focus on the former aspect of this problem — the selection of $S \in 2^Z$ — while laying a foundation for incorporating the second aspect as a useful direction for future work.

Figure 2.3 illustrates our first version system design for LDspectral, which is designed to address the design optimization problem described above. Here, video processing configurations are re-evaluated periodically with the period of re-evaluation being equal to the value of the *reconfiguration interval parameter* T_r. Lower values of T_r correspond to the possibility for more frequent reconfiguration at the expense of increased overhead due to more frequent operations for reconfiguration management. The reconfiguration management overhead includes computations for dynamically determin-

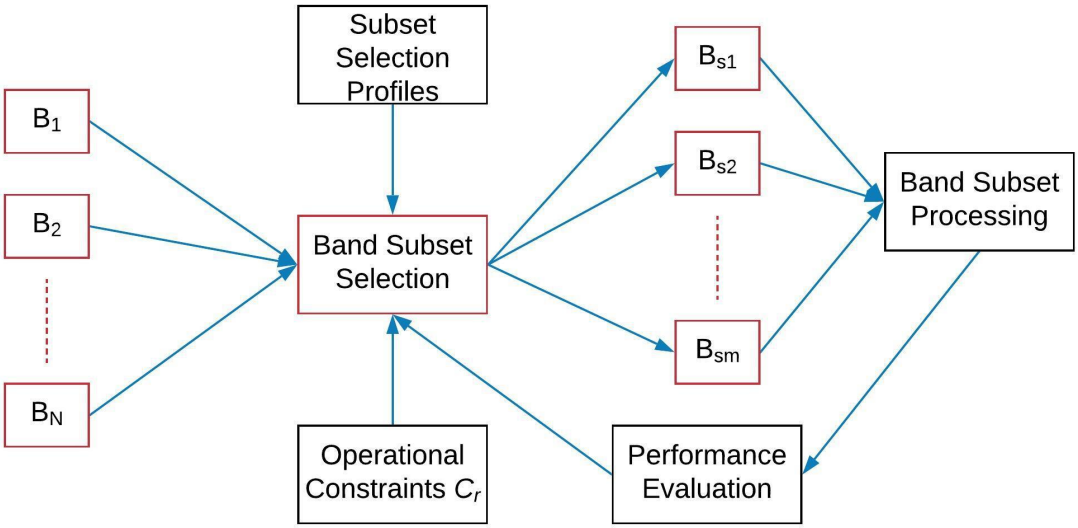

Figure 2.3: Block diagram of the design flow in LDspectral.

ing whether or not to reconfigure the system, and determining and applying the new operational parameters, including the band subset S, when reconfiguration is to be performed.

The block in Figure 2.3 labeled Band Subset Selection (BSS) is invoked at time intervals determined by the reconfiguration interval parameter T_r, subject to application specifications. The BSS block attempts to optimize the subset of bands that is to be employed during the next interval of video processing. In this optimization process, offline data (Subset Selection Profiles) pertaining to the effectiveness of selected subsets of bands is considered along with recent results from performance evaluation, and the current operational constraint C_r.

The output of Band Subset Selection is a vector indicating the bands $S = \{B_{s1}, B_{s2}, \ldots, B_{sm}\}$ ($m \leq N$ or equivalently, $S \subset Z$) that are to be processed during the next video processing interval.

We perform pixel-level fusion, where the selected bands in a given multi-spectral

image are combined pixel-by-pixel into a single image. In our fusion approach, each pixel in the combined image is derived from a weighted sum of the corresponding pixels in the individual bands. Compared to feature-level fusion, pixel-level fusion can have significantly reduced computational cost since features are extracted from the combined image rather than separately from each individual band (e.g., see [19, 20]). On the other hand, feature-level fusion allows for optimization of feature extraction algorithms for each band [21]. Extension of the LDspectral framework to include feature-level fusion and adaptive selection between pixel- and feature-level fusion is a useful direction for future work.

The video processing functionality performed on the selected bands is represented by the block in Figure 2.3 labeled Band Subset Processing. Further discussion on band subset processing in this work is given in Section 2.3.

We demonstrate the importance of careful band subset selection, which is a core aspect of the LDspectral design methodology discussed in Section 2.2. We demonstrate this through a case study involving background subtraction. The two metrics that we consider in this evaluation are the accuracy $F_{measure}$ (defined in Section 2.3) of the background subtraction (foreground extraction) results, and the average execution time t_{ave} to extract the foreground. We focus on quantifying trade-offs consisting of singleton (one-band) and two-band subsets, and demonstrate significant variations in performance trade-offs among different subsets. Analysis and optimization of band subset selection trade-offs among larger subsets (i.e., where the subset size exceeds 2) are motivated through this preliminary study as useful directions for future work.

The band subset processing subsystem for this multispectral background subtrac-

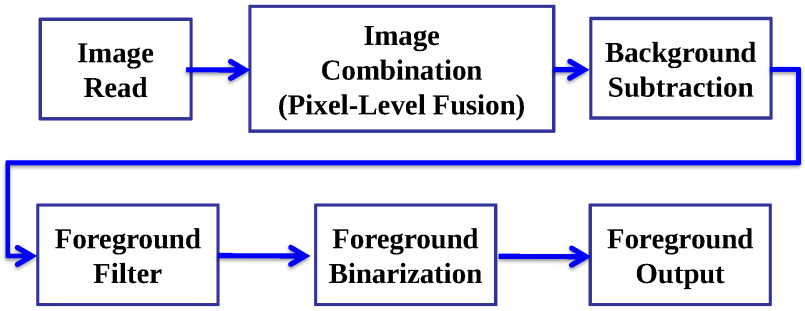

Figure 2.4: Block diagram of band subset processing in the background subtraction system.

tion case study is illustrated in Figure 2.4. In the context of this case study, this illustration represents the internal functionality associated with the block in Figure 2.3 that is labeled Band Subset Processing. In the dataflow graph subsystem depicted in Figure 2.4, each actor reads a pointer to an image from its input buffer, and outputs a pointer to the image that results from the image processing operation performed by the actor.

We use LIDE to develop a prototype implementation of the band subset processing subsystem in Figure 2.4, and we apply calls to selected OpenCV functions in some of the actors within this implementation. The *Image Read* actor in Figure 2.4 is used to inject a stream of pointers to successive images into the subsystem so that background subtraction can be performed separately on each image that is referenced (pointed to) in the stream. At the output of the Image Read actor, each image contains a set of m separate components, where each component corresponds to one of the selected spectral bands (i.e., an element of the set S, as defined in Section 2.2). The *Image Combination* actor then performs pixel-level fusion to combine the components associated with the selected bands into a singled "fused" image. More details on the fusion operation performed by this actor are discussed in Section 2.3.

The *Background Subtraction* actor then computes an initial background subtraction result and passes the extracted foreground through the image pointer produced on its output. The core background subtraction operation applied by this actor is carried out by the OpenCV function called `BackgroundSubtractorMOG2`, which applies a Gaussian mixture model(GMM) [22, 23].

The *Foreground Filter* actor in Figure 2.4 is designed to remove noise from the output of the Background Subtraction actor. In the Foreground Filter actor, we use two morphological operations — erosion and dilation — through their respective implementations in OpenCV. Intuitively, the erosion function helps to remove objects in the foreground that are smaller than the filter-size (a parameter of the erosion function), and dilation helps to more completely identify boundaries of detected objects. Erosion may lead to distortion in object boundaries; dilation is applied after erosion as a corrective operation to address this potential for distortion.

The *Foreground Binarization* actor takes the output of the the foreground filter, and converts it into a binary form, where each pixel is classified as being either a foreground or background pixel. This conversion is performed by applying a threshold, and classifying pixels as foreground whenever the corresponding pixel values exceed the threshold. The specific threshold that is employed is determined empirically (off-line) in an effort to enhance classification accuracy. The resulting binary image is then processed by the Foreground Output actor to store the classification results for each image as a separate file in a given output directory. The files generated in this output directory are indexed so that they can easily be matched up with their corresponding input frames from the given multispectral data set.

We performed experiments applying LDspectral with the background subtraction subsystem shown in Figure 2.4. These experiments were performed using a laptop computer equipped with an AMD A8-4500M CPU, 4GB RAM, and the Ubuntu 14.04 LTS operating system. Results from these experiments are discussed in the following section.

2.3 Results

In our experiments involving band subset selection in conjunction with background subtraction, we applied the novel data set for multispectral background subtraction that has been published recently by Benezeth et al. [1]. From this data set, we experimented with multispectral video input that contains 1102 images, where each image contains separate components in 7 different spectral bands. Among these 7 bands, 6 are in the visible spectrum and the remaining one is in the near-infrared spectrum. We divided this set of images into 735 images (approximately 2/3) for training and 367 images for testing.

Here, the training phase is applied to optimize the performance of each two-band subset. Given a band subset $\{b_{s1}, b_{s2}\}$, training is used to optimize the relative weightings for these bands when they are fused in the Image Combination actor described in Section 2.2. More specifically, suppose that x_1 and x_2 are two corresponding pixel values (pixel values at the same image coordinates (a,b)) in bands b_{s1} and b_{s2}, respectively, and let y denote the pixel value at coordinates (a,b) in the output of the Image Combination actor. Then y is derived by:

$$y = \alpha \times x_1 + (1-\alpha) \times x_2, \tag{2.1}$$

where α ($0 \leq \alpha \leq 1$) is a parameter of the Image Combination actor that is used to control the relative weightings of the two input bands. We refer to this parameter α as the *pairwise band combination* (*PBC*) parameter.

Based on this formulation of pixel-level fusion for a two-band subset, our training phase is used to optimize the image combination parameter α. This training process is carried out for each distinct pair $\{b_{s1}, b_{s2}\}$ of bands to yield a corresponding PBC parameter value $A(s1, s2)$ that controls the relative weighting of pixels when combining bands b_{s1} and b_{s2}.

For each distinct pair $\{b_{s1}, b_{s2}\}$ of bands, the training phase involves performing an exhaustive search across $\alpha \in \{0, 0.1, 0.2, \ldots, 1\}$, and then selecting a value for the PBC (with ties broken arbitrarily) that leads to the highest average accuracy for the background subtraction subsystem of Figure 2.4. This selected value is then used in the testing phase to assess the accuracy produced by using the band subset $\{b_{s1}, b_{s2}\}$ for background subtraction.

The measure of accuracy employed in these experiments is the harmonic mean performance measure of background subtraction accuracy, which is motivated, for example, in [1]. This measure is defined as

$$F_{measure} = 2 \times \frac{recall \times precision}{recall + precision}. \tag{2.2}$$

Here, *precision* and *recall* are defined by:

$$precision = \frac{n_c}{n_f}, \text{ and } recall = \frac{n_c}{n_g},$$

where n_c is the number of correctly classified foreground pixels, n_f is the number of pixels

classified as foreground, and n_g is the number of foreground pixels in the ground truth.

Table 2.1 and Table 2.2 show experimental results using the offline analysis capabilities of LDspectral to evaluate processing trade-offs among different one- and two-band combinations (i.e., where the set of selected bands is restricted to contain only one or two elements). Table 2.1 shows the background subtraction accuracy that is experimentally observed for different one- and two-band combinations, while Table 2.2 shows the processing times for different combinations. In each of these tables, the diagonal entries give the results for single-band processing, while each entry at row a and column b when $a \neq b$ gives the results from joint processing of the bands indexed by a and b. In each of these tables, elements below the diagonal are not shown since they are symmetric with respect to the diagonal. As mentioned above, we employ 1102 images in each of these experiments. These 1102 images form the complete set of images from the employed multispectral data set [1] that have ground truth available as part of the data set.

From Table 2.1, we see that the accuracy provided by LDspectral is significantly higher on average compared to the results presented in [1] for the same video data set. This demonstrates the effectiveness of LDspectral in optimizing the accuracy of background subtraction.

Experimentally derived data of the form shown in Table 2.1 and Table 2.2 can be used as the subset selection profiles to guide band subset selection, as illustrated in Figure 2.3. Additionally, the results in Table 2.2 define lower limits on how short the reconfiguration interval T_r (See Section 2.2) can be.

Table 2.3 shows the optimized values for the PBC parameters that were derived through the training procedure for processing of two-band subsets. The rows and columns

Table 2.1: Accuracy results for different one- and two-band combinations using LDspectral, and (in the last three rows) the results from [1] with three different algorithms.

band	1	2	3	4	5	6	7
1	0.934	0.940	0.944	0.945	0.943	0.943	0.933
2		0.931	0.942	0.936	0.942	0.937	0.930
3			0.939	0.939	0.943	0.940	0.939
4				0.929	0.940	0.932	0.930
5					0.942	0.938	0.937
6						0.919	0.925
7							0.886
Mahalanobis distance							0.689
Spectral angle							0.897
SID similarity							0.896

Table 2.2: Execution times for different single- and dual-band combinations. The results here are given in milliseconds. Each entry in the table represents the average time to perform background subtraction (including the entire processing chain shown in Figure 2.4 on a single input image).

band	1	2	3	4	5	6	7
1	62.5	68.4	67.2	66.9	67.5	68.4	66.5
2		62.3	68.2	68.4	68.0	69.7	68.2
3			62.8	67.4	66.9	68.5	66.9
4				63.1	67.4	69.0	66.8
5					63.0	68.8	66.8
6						62.2	68.4
7							63.4

Table 2.3: Derived values for PBC parameters (rounded to tenths).

band	2	3	4	5	6	7
1	0.7	0.4	0.6	0.3	0.7	0.9
2		0.3	0.7	0.4	0.5	0.8
3			0.9	0.5	0.7	0.9
4				0.4	0.5	0.9
5					0.5	0.9
6						0.8

of Table 2.3 correspond, respectively, to x_1 and x_2 in Equation 2.1. For example, when S consists of Bands 2 and 3, we use $\alpha = 0.3$, and when S consists of Bands 1 and 4, we use $\alpha = 0.6$. The diversity of the values in this table demonstrates the utility of optimizing the PBC parameter separately for each two-band subset rather than using a balanced weighting ($\alpha = 0.5$) or some other uniform PBC parameter setting for all subsets.

We also see from Table 2.3 that the optimized α values are relatively high when x_2 is taken to be Band 7, which is the near-infrared band. This results in correspondingly low weightings given to Band 7. This trend matches intuitively with the data in Table 2.1, which shows that Band 7 in isolation has significantly lower accuracy compared to all of the other one-band subsets.

Overall, the results in Table 2.1 and Table 2.2 help to motivate the utility of careful selection of band combinations as there is significant variation in accuracy among different pairs of bands. The results also help to quantify the trade-off — in terms of increased execution time — when a single band is augmented with a second band to help increase background subtraction accuracy. For the implemented pixel-level fusion approach, this increase is found to be relatively low (within 13% in all cases). This is because increasing the number of bands increases the computational load for only a small subset of the ac-

tors in Figure 2.3 and Figure 2.4 — in particular, the actors for band selection and image combination.

2.4 Summary

In this chapter, we have introduced a novel system design framework for dynamic, data-driven processing of multispectral video streams using lightweight dataflow techniques. The framework is motivated by the need for efficient and accurate video processing in a wide variety of systems for air and ground environments. This framework is designed to incorporate selection of subsets of bands as a core, front-end step in the video processing process. This emphasis on band subset selection opens up a large design space for data-driven adaptation that influences key metrics, including accuracy and computational efficiency. We have demonstrated a prototype implementation of LDspectral applied to a background subtraction application. Through experiments with this prototype on a relevant data set, we have demonstrated the utility of flexible, optimized band subset selection in the navigation of operational trade-offs for multispectral video processing systems.

The LDspectral framework uses pixel level fusion with weighting coefficients that are optimized to combine selected bands. With the optimized weighting coefficients, the accuracy is significantly improved compared to the accuracy provided by single-band input. In addition to or instead of pixel level fusion, we can incorporate feature level or decision level fusion in LDspectral. Chapter 3 extends LDspectral by adding capabilities of feature level fusion and dynamic reconfigurability. Chapter 4 further extends LDspec-

tral by adding support for hyperspectral image and video processing.

Chapter 3

Extensions to LDspectral for Dynamic Data-Driven Video Processing

In Chapter 2, we introduced the LDspectral framework for design and implementation of multispectral video processing systems. We demonstrated methods to select pairs of multispectral bands to process in a manner that optimizes accuracy while providing streamlined computational requirements due to the use of only a small subset of the available bands.

In this chapter, we improve the LDspectral framework in two major aspects. First, we develop a greedy algorithm that considers band subsets of arbitrary size rather than limiting the design space to only two-band subsets. More specifically, the algorithm is designed to select a subset of bands that contains a pre-specified number of bands, where the pre-specified number is any integer between 1 and the total number of available bands. The objective of the algorithm is to optimize video analysis accuracy subject to the given constraint on the number of bands. In addition to selecting the bands, the algorithm optimizes the weighting coefficients of the bands.

Second, we introduce feature level fusion into LDspectral, which improves the flexibility of multispectral information processing. Feature level fusion is applied in LDspectral using a pooling strategy for decision making.

The distinguishing aspects of the contributions in this chapter compared to the related work include their (1) focus on integrating DDDAS methods into trade-off opti-

mization between accuracy and real-time performance in multispectral video processing systems, and (2) emphasis on supporting flexible optimization involving the subset of available multispectral bands that is processed, and the associated algorithm and dataflow configurations.

Material in this chapter was published in preliminary form in [24]

3.1 Introduction

In the extraction of knowledge from the diverse channels provided by multispectral and hyperspectral imaging sensors, image fusion is an important class of algorithms. Liu et al. present a comparative study of different multiresolution algorithms for image fusion [17]. Bhateja et al. develop a non-subsampled contourlet transform approach for multispectral image fusion in medical applications [25]. Wei et al. propose an image fusion method for multispectral and hyperspectral images that is based on a sparse representation, and results in less spectral error and spectral distortion compared to related fusion techniques [26]. Chen et al. develop an approach for fusing low-spatial-resolution hyperspectral images and high-spatial-resolution multispectral images of the same scene using pan-sharpening methods [27].

Benezeth et al. have performed an extensive experimental investigation on the application of multispectral video processing to the detection of moving objects [1]. Benezeth's contributions also include a publicly available dataset with foreground truth for experimenting with multispectral background subtraction techniques. Uzkent, Hoffman, and Vodacek have developed a DDDAS framework for controlling hyperspectral

data collection [28]. Sobral et al. proposed an online stochastic tensor decomposition algorithm for robust background subtraction. Sobral's results demonstrate that red-green-blue (RGB) features are not sufficient to handle color saturation, illumination variations and problems due to shadows, while incorporating six visible spectral bands together with one near-infra-red band helps to address these limitations [29]. Reddy et al. present a multispectral video visualization method, and propose in this context a fusion technique to retain color, texture, relative luminance and sharpness [30]. Recently, Aved et al. [31] applied a difference criteria to weight hyperspectral bands.

The design methodologies and tools developed in this chapter are largely complementary to the methods surveyed above in the area of image fusion, and in our experimental evaluation (Section 3.3), we apply the dataset mentioned above that has been introduced by Benezeth et al.

3.2 System Design

As motivated in Section 3.1, this chapter develops new capabilities in LDspectral, which is a software tool for optimized design and implementation of multispectral video processing systems. The objective of LDspectral is to enable efficient, dynamic processing across the available bands based on constraints imposed by the given operational scenario, and instrumentation data collected from the underlying embedded platform.

In the class of DDDAS-driven video processing systems that is targeted by LDspectral, the input data comes from a set $Z = \{B_1, B_2, \ldots, B_N\}$, where N is the total number of available spectral bands. The multispectral image stream with this number of bands

is processed by a given dataflow graph $G = (V, E)$, where V is the set of graph vertices (*actors*), which correspond to functional modules, and E is the set of edges. Each edge $e \in E$ corresponds to a first-in-first-out (FIFO) communication channel that buffers data as it passes from the output of one actor to the input of another. The actors and edges in G have associated sets of parameters P_v and P_e, respectively. Parameters of an edge may include a Boolean "activation parameter", in the spirit of Boolean parametric dataflow [32]. Such activation parameters allow edges to be enabled and disabled. In this context, disabling an edge means effectively removing the associated connection (between the edge's source and sink actors) from the graph. Such use of dynamic parameter adjustment can be used to configure dataflow within the system model.

We assume a given constraint C_r (in units of time) on execution time performance for a particular video processing scenario. The run-time system for video processing is equipped with instrumentation for periodically determining the execution time performance of the current video processing configuration. The problem addressed by LDspectral is to select the subset $S \in 2^Z$ of spectral bands to store and process, and an assignment of valid parameter values for all dataflow graph parameters in $(P_v \cup P_e)$ such that video processing accuracy is maximized subject to the real-time constraint specified by C_r. Here, 2^Z denotes the power set of Z.

The LDspectral tool addresses novel video processing design spaces introduced by multispectral image acquisition technology. It enables efficient experimentation and data-driven optimization of video processing configurations for multispectral video analytics. The remainder of this chapter discusses details on design optimization models and methods employed in LDspectral, and demonstrates the tool through a case study involving

background subtraction for moving object detection.

3.2.1 Run Time System

Figure 3.1 shows a block diagram of the run-time system model that is targeted by LDspectral. We refer to this model as the *LDspectral Run-time System Model* (*LRSM*). In the LRSM, Dataflow Configuration Profiles refer to performance profiles of alternative actor configurations. These profiles provide estimates of accuracy and execution time for alternative algorithmic configurations associated with selected functional modules in the given video processing application system. The profiles are determined at design time, through simulation or through instrumented execution on the targeted embedded platform.

Similarly, the Subset Selection Profiles provide estimates of trade-offs between accuracy and execution time for different subsets of spectral bands. Each entry in this collection of profiling data corresponds to a subset $S \in 2^Z$ of the available spectral bands, and provides estimates of the achievable accuracy and the execution time cost when subset S is used as input for the core video processing functionality (and the remaining bands $(Z - S)$ are discarded or ignored). In the current version of LDspectral, one subset is selected for each cardinality value in the range of $1, 2, \ldots, N$, where N is the total number of available bands in the multispectral sensor subsystem. An entry is then stored within the Subset Selection Profiles for each of these selected subsets. Methods used to select and evaluate these subsets are discussed further in Section 3.2.3.

The Subset Selection Profiles and Dataflow Configuration Profiles are used at run-time to adapt algorithmic and dataflow parameters associated with the core video process-

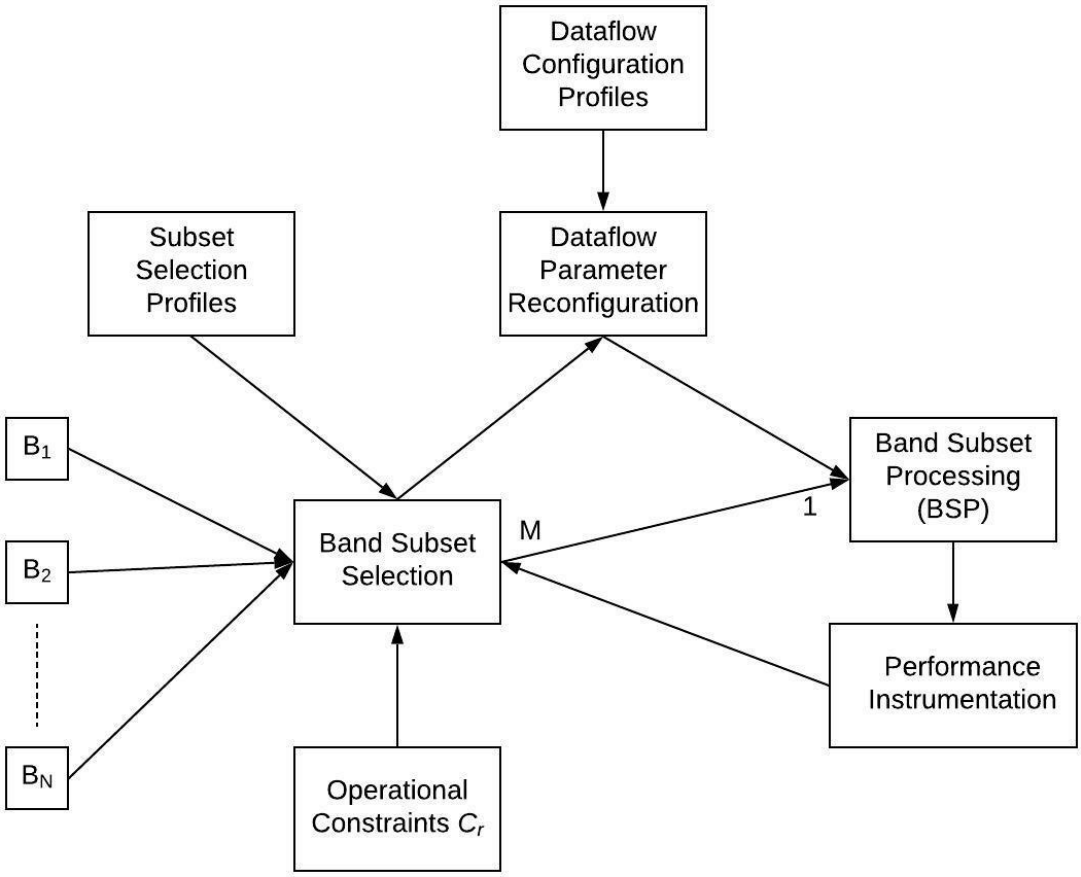

Figure 3.1: Block diagram of the LDspectral Run-time System Model (LRSM).

ing functionality of the targeted embedded system. This core functionality is represented by the block in Figure 3.1 labeled *Band Subset Processing* (*BSP*). Details about the BSP subsystem are discussed in Section 3.2.2. The dynamic, data-driven adaptation of system parameters in the LRSM is performed by the blocks in Figure 3.1 that are labeled Dataflow Parameter Reconfiguration and Band Subset Selection. *Dataflow Parameter Reconfiguration* is performed using techniques that involve parameterized dataflow [33] and Boolean parametric dataflow [32]. The *Band Subset Selection* block takes as input design time information provided by the Subset Selection Profiles, and run-time information derived from Performance (execution time) Instrumentation. Band Subset Selection produces as output the subset $\sigma(i) \in 2^Z$ of multispectral bands that are to be processed in the next iteration i of LRSM execution. This subset is taken from among the entries in the Subset Selection Profiles as the band subset that provides the highest accuracy while satisfying the current Operational Constraint C_r.

3.2.2 Band Subset Processing

This section details the BSP subsystem, which was introduced in Section 3.2.1 as one of the blocks in Figure 3.1. A dataflow representation of the BSP subsystem is illustrated in Figure 3.2, including pixel-level fusion (PLF) and feature-level fusion (FLF) [34].

The BSP subsystem in LDspectral is designed through integrated used of the lightweight dataflow environment (LIDE) and OpenCV. LIDE is a model-based tool for design and implementation of embedded software and firmware using coarse-grained

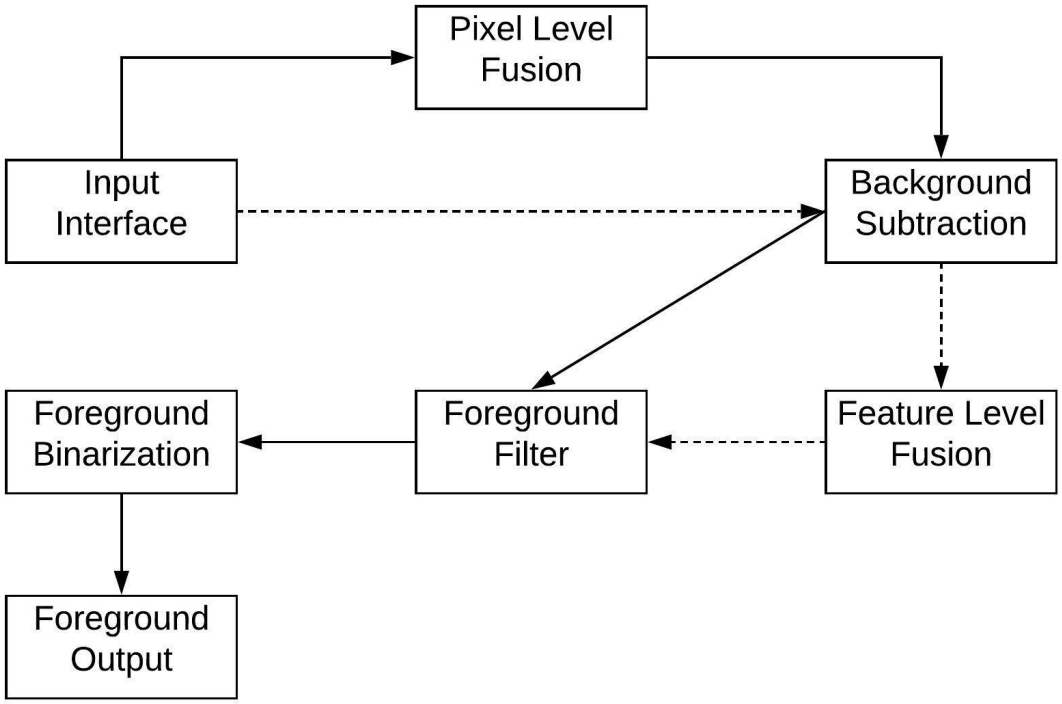

Figure 3.2: Dataflow representation of the band subset processing (BSP) subsystem shown in Figure 3.1.

dataflow representations [11, 35]. OpenCV is a computer vision software framework that includes a large library of software components for video processing (e.g., see [12]). In the integrated use of LIDE and OpenCV employed in BSP, actors in LIDE incorporate calls to relevant OpenCV functions that perform selected image/video processing operations. This approach provides an efficient means for integrating model-based system design techniques with the large library of image/video processing implementations in OpenCV.

As shown in Figure 3.1, the BSP subsystem consists of several actors. The Input Interface actor provides an interface for accessing and operating on input image frames for a given invocation I of the BSP subsystem. These input frames correspond to the selected subset of spectral bands that are to be accessed during invocation I.

The *Background Subtraction* actor computes an initial background subtraction result that is further refined in later stages of the BSP subsystem. The core operation applied by this actor is carried out by the OpenCV function called `BackgroundSubtractorMOG2`, which applies a Gaussian Mixture Model (GMM) [36, 22, 23].

The *Foreground Filter* actor is used to remove noise from the output of the Background Subtraction actor. Such noise can result from the moving of background objects, such as trees that are shaken by the wind. This actor applies two morphological operations — erosion and dilation — through their implementations in OpenCV. The erosion function removes objects in the foreground that are smaller than the filter-size (a parameter of the BSP subsystem), while the dilation function corrects distortion at foreground object boundaries that results from the erosion operation.

The *Foreground Binarization* actor takes the output of the foreground filter, and converts it into a binary form, where each pixel is classified as being either a foreground or background pixel. This actor applies a threshold that is determined empirically (off-line) to optimize classification accuracy. In BSP, the OpenCV function called `cvThreshold` is employed for foreground binarization.

The BSP subsystem provides two different fusion methods — PLF and FLF — to fuse the individual images from different bands in the subset of selected bands. PLF is applied to the input image before applying background subtraction, while FLF is applied to the result of preliminary background subtraction from each band. Using the configurable dataflow capabilities in the BSP subsystem (represented by the dashed edges in Figure 3.2), designers or the LDspectral run-time system can select flexibly between PLF and FLF.

For a band subset with two elements, a "pairwise band combination" parameter α is used to configure PLF in the BSP subsystem. The value of α must be a real number in the range $[0,1]$. The parameter α is used to configure the fusion operation by:

$$y = \alpha \times x_1 + (1-\alpha) \times x_2, \tag{3.1}$$

where x_1 and x_2 are two corresponding pixel values (at the same image coordinates) in the two input bands, and y is the pixel value at same coordinate in the output.

This fusion approach is extended to band subsets having arbitrary size N using an N-dimensional vector $\alpha(N) = (\alpha_1, \alpha_2, \ldots, \alpha_N)$, where $\sum_i \alpha_i = 1$. A vector $\alpha(N)$ of this form is referred to a *PLF weight vector*. When subsets of bands are constructed incrementally,

as they are constructed in LDspectral, the vectors $\{\alpha(N)\}$ can be computed efficiently using grid search. More details about the grid search approach employed in this work are provided in Section 3.2.3.

Following [1], we apply a pooling strategy for FLF:

$$Z_t(s) = \begin{cases} 1 & \sum_i Y_{i,t}(s) > \rho \\ 0 & otherwise \end{cases}, \tag{3.2}$$

where $Y_{i,t}$ represents the input image for fusion at frame t and spectral band i; Z_t represents the tth output frame derived by FLF; and ρ, called the *majority* parameter, provides a threshold for the fusion operation. The symbol s in Equation 3.2 represents a given pixel index. The value of ρ ranges from 1 (a logical OR operation) to the total number bands (a logical AND operation). Each binary pixel value $Z_t(s)$ in the fused result represents a prediction about whether the pixel corresponds to foreground (1) or background (0).

3.2.3 Band Subset Selection

As described in Section 3.2.1, the Subset Selection Profiles in Figure 3.1 are derived at design time to provide a set, called *bandseq*, of strategic multispectral input configurations (subsets of the available multispectral bands) that are made available to the LRSM for dynamic, data-driven adaptation. We first discuss the approach used in LDspectral for deriving Subset Selection Profiles based on PLF, and then the approach is extended to incorporate both PLF and FLF.

The derived set of Subset Selection Profiles *bandseq* contains one carefully-selected subset of bands for each cardinality value in the range of $1, 2, \ldots, N_b$, where N_b is the total

number of available bands in the multispectral sensor subsystem. Thus, *bandseq* can be viewed as a sequence or array whose *i*th element is the subset of selected bands that has cardinality *i*. Along with each subset of bands, an optimized PLF weight vector is derived to heuristically maximize the accuracy of PLF for the associated subset of bands.

Algorithm 1 provides a pseudocode sketch of the algorithm employed in LDspectral to derive the Subset Selection Profiles *bandseq* along with the array of associated PLF weight vectors *alpha*. For each i, $bandseq[i]$ is derived to be an i-element set of selected multispectral bands. The algorithm presented here is a greedy algorithm in that for each $j = 2, 3, \ldots, N_b$, $bandseq[j]$ is derived by extending $bandseq[j-1]$ with one band from $(bands - bandseq[j-1])$, where *bands* represents the complete set of available bands in the multispectral video processing system. Thus, $bandseq[j][k] = bandseq[j-1][k]$ for $k < j$.

The weight vector for each $bandseq[j]$ is derived using the constraint that:

$$alpha[j][k] = g \times alpha[j-1][k] \text{ for } k < j, \text{ and some coefficient } g \in [0,1]. \tag{3.3}$$

A grid search is then performed, using a training dataset for evaluation, to optimize the value of g. This evaluation, represented by the call to *evaluateBSP* in Algorithm 1, is performed by invoking the BSP subsystem (Figure 3.2) on all images in the training dataset to assess the average accuracy using the given band subset and PLF weights. Accuracy evaluation is performed in terms of the harmonic mean performance measure $F_{measure}$. The average $F_{measure}$ computed across the training set is returned from the call to *evaluateBSP*. This metric is discussed in more detail in Section 3.3.3.

Since the components of $alpha[j-1]$ sum to 1 (see Section 3.2.2) and the components of $alpha[j]$ must also sum to 1, the last component of $alpha[j]$ can be derived during the grid search as

$$alpha[j][j] = (1-g). \tag{3.4}$$

The constraint in Equation 3.3 is imposed during the search process to reduce the search complexity. Investigating efficient ways to relax this constraint and achieve more thorough search, while keeping the overall time required for optimization in an acceptable range, is a useful direction for future work.

In our experiments, we use a grid spacing (the *grid_spacing* parameter in Algorithm 1) of 0.1.

Algorithm 1

```
parameter N_b: number of available spectral bands
parameter bands: set of spectral bands
parameter grid_spacing: granularity for grid search
output bandseq[N_b]: sequence of selected bands
output alpha[N_b]: weight vectors for band subsets
unprocessed = bands
processed = ∅
for i = 1; i ≤ N_b; i++ do
    f_M = −1
    for u ∈ unprocessed do
        S = processed ⋃ {u}
        for g = 0; g ≤ 1; g+ =grid_spacing do
            w = ((g × alpha[i−1]), (1−g))
            f_m = evaluateBSP(S, w)
            if f_m > f_M then
                ū = u
                w̄ = w
        bandseq[i] = ū
        alpha[i] = w̄
        measure[i] = f_M
        unprocessed − = {ū}
        processed + = {ū}
```

Algorithm 1 is adapted for FLF by configuring the dataflow in the BSP subsystem to perform fusion at the feature level, and replacing the grid search to optimize PLF weights with a grid search to optimize the majority threshold ρ (see Equation 3.2). This is a relatively straightforward replacement of one kind of grid search with another grid search having a similar form. We omit the details for brevity. This replacement allows us derive an optimized sequence of band subsets using FLF along with an accuracy-optimizing majority value $rho[i]$ for each subset cardinality i.

LDspectral Band Subset Selection (*LBSS*) operates by first applying both Algorithm 1 and the adapted version of Algorithm 1 that employs FLF instead of PLF. We refer to the resulting band subsets (*bandseq* outputs) as β_{plf} and β_{flf}, respectively. Similarly, the resulting average accuracy results (*measure* outputs) are denoted M_{plf} and M_{flf}, respectively.

Then for each band subset cardinality $i \in \{2,3,\ldots,N_b\}$, LBSS selects either $\beta_{plf}[i]$ (along with the associated weight vector $alpha[i]$) or $\beta_{flf}[i]$ (along with the associated majority value $rho[i]$) depending on whether $M_{plf}[i] \geq M_{flf}[i]$ or $M_{plf}[i] < M_{flf}[i]$, respectively. For $i = 1$, there is no fusion involved so the singleton subset selected by LBSS is simply equal to the common value of $\beta_{plf}[1]$ and $\beta_{flf}[1]$.

3.3 Results

3.3.1 Experimental Setup

Multispectral video sequences used for training and testing in our experiments were obtained from a multispectral dataset published by Benezeth et al. [1]. The parts of this

dataset that we used include foreground truth to enable assessment of background subtraction accuracy.

The video data in this dataset was acquired from a commercial multispectral camera, the FD-1665-MS from FluxData, Inc. The dataset incorporates 7 bands in total, including 6 different channels in the visible spectrum (B_1 through B_6) with wavelengths ranging from 400 nm to 700 nm, and one near-infrared band (B_7) with a wavelength in the range of 700 nm to 1000 nm.

We used 1,102 multispectral images from the dataset described above. We divided this set of images into 735 images (approximately 2/3) for training and 367 images for testing.

Our experiments were performed using a desktop computer equipped with a 3.10GHz Intel i5-2400 CPU, 4GB RAM, and the Ubuntu 15.10 LTS operating system. Results from these experiments are discussed in Section 3.3.3 and Section 3.3.4.

3.3.2 Example Images

Figure 3.3 shows a composite (all-band) scene; 7 different images corresponding to single-band foreground results on the 7 available multispectral bands; and the foreground result that is derived by LDspectral using background subtraction along with PLF across all 7 bands. This scene is selected here from Benezeth's dataset as an example to illustrate techniques for fusion and background subtraction that are employed in LDspectral. The foreground fusion result derived by LDspectral is shown in the image at the bottom right corner of Figure 3.3.

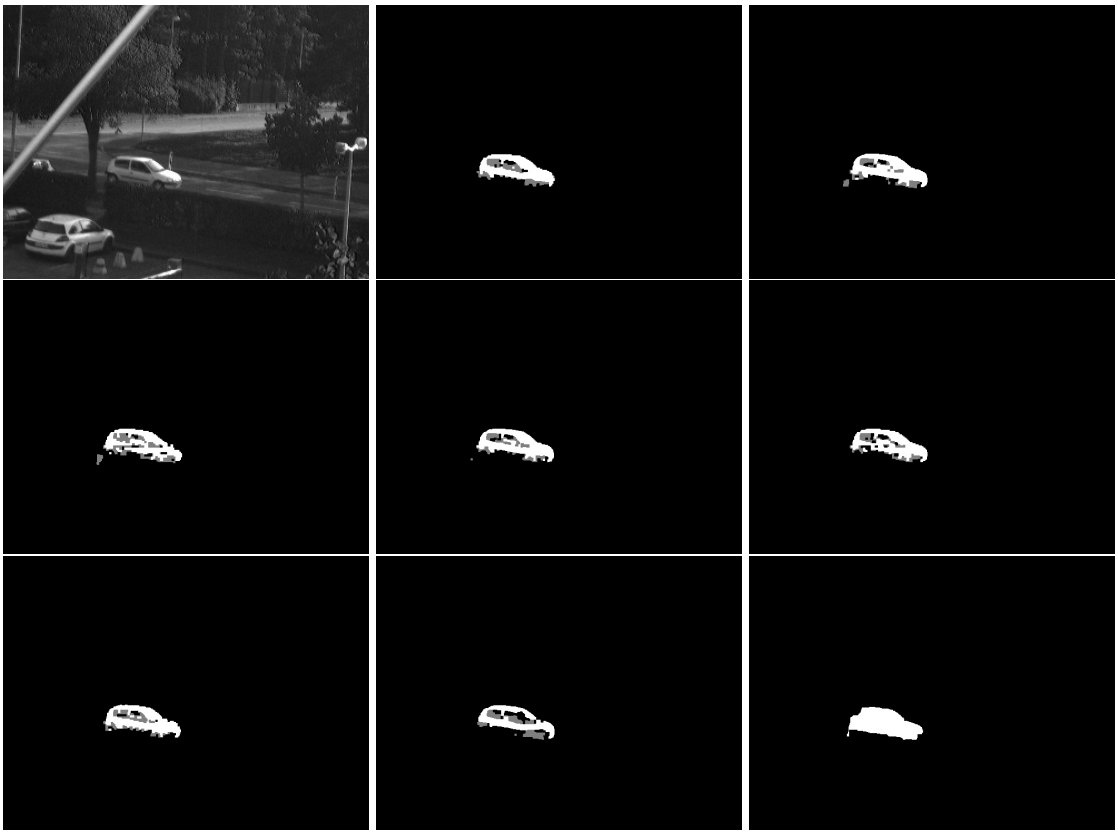

Figure 3.3: An example from Benezeth's dataset that is used to illustrate the techniques for fusion and background subtraction that are employed in LDspectral: the scene, 7 bands, and foreground fused image.

Examination of these images shows that performing background subtraction in conjunction with image fusion yields more accurate results compared to the results of performing background subtraction on individual bands in isolation — for example, the hollow portions in the single band results are largely filled in within the fused result.

While it is intuitively clear and concretely illustrated in the example of Figure 3.3 that fusion can significantly improve accuracy, the overall objective of LDspectral is to enable efficient, dynamic adaptation across video processing configurations that trade-off accuracy and real-time performance subject to operational constraints. The utility of LDspectral for performing such trade-off optimization is demonstrated in the remainder of this section.

3.3.3 Accuracy Evaluation

Table 3.1 shows foreground accuracy results for all possible one- and two-band subsets for both PLF and FLF. The results shown here are derived using search processes within the LBSS algorithm presented in Section 3.2.3. Each off-diagonal table entry in Table 3.1 consists of two values that are stacked vertically — the top value corresponds to PLF and the bottom value to FLF. The entries on the diagonal correspond to singleton (one-band) subsets, while each off-diagonal entry at row i and column j represents the two-band subset $\{B_i, B_j\}$. The values in the table are the average $F_{measure}$ values computed across the training part of our multispectral dataset. The boldface values in Table 3.1 show the accuracy values for FLF in cases where FLF achieves higher accuracy than PLF.

These results show that FLF achieves higher accuracy compared to PLF in only a

Table 3.1: Accuracy results for different one- and two-band subsets using LDspectral with both PLF and FLF. In each off-diagonal table entry, the top value corresponds to PLF, and the bottom value corresponds to FLF.

band	1	2	3	4	5	6	7
1	0.934	0.940 **0.942**	0.943 **0.944**	0.945 0.940	0.943 **0.947**	0.943 **0.946**	0.933 0.933
2		0.931	0.942 0.942	0.936 **0.937**	0.942 0.942	0.937 0.937	0.930 0.926
3			0.939	0.939 0.938	0.943 0.941	0.940 0.937	0.939 0.935
4				0.929	0.940 0.937	0.932 0.929	0.930 0.926
5					0.942	0.938 0.937	0.937 0.933
6						0.919	0.925 0.922
7							0.843

small fraction of the evaluated band subsets. Furthermore, as we show in Section 3.3.4, FLF requires significantly higher execution time on our experimental platform compared to PLF. Thus, for the remaining experiments reported in this section (Section 3.3.3), we "turn off" or disable FLF in LBSS so that only weight-optimized configurations of PLF are considered. However, the option of enabling FLF in LBSS may be useful in general for other target platforms, such as platforms that have more parallelism available to speed up the performance of the FLF-enabled BSP dataflow graph.

From the results in Table 3.1, we also see that when the number of bands increases from one to two, a significant improvement in accuracy results. This helps to confirm and quantify the utility of maintaining progressively larger subsets of spectral bands as alternative configurations for dynamic adaptation in LDspectral.

Table 3.2 shows the results of incremental band subset construction using the LBSS

Table 3.2: Results of incremental band subset construction using the LBSS algorithm in LDspectral.

Band(s)	1-4	1-4-6	1-4-6-3	1-4-6-3-5	1-4-6-3-5-2
1	-	-	-	-	-
2	0.948	0.957	**0.957**	**0.958**	-
3	0.946	**0.957**	-	-	-
4	-	-	-	-	-
5	0.949	0.950	0.957	-	-
6	**0.949**	-	-	-	-
7	0.918	0.935	0.938	0.940	**0.961**

algorithm in LDspectral. The columns correspond to progressively larger subsets of bands that are derived by LBSS, while the rows correspond to individual bands that are incrementally added and evaluated in the search process. For example, the entry corresponding to Row `3` and Column `1-4-6` shows the best accuracy achieved (across all PLF configurations that are evaluated through grid search) for the band subset $\{B_1, B_3, B_4, B_6\}$.

The boldface values in Table 3.2 correspond to the best configurations represented in the corresponding columns. These are the configurations that are "picked up" by the search process in LBSS. For example, band B_6 exhibits the best accuracy when combined with bands B_1, B_4 (the tie here with band B_5 is broken arbitrarily or based on less significant digits that are not shown in the table), and thus, the union $\{B_1, B_4, B_6\}$ of these two subsets is taken as the best 3-element subset. This subset is then represented in the next column of the table (labeled `1-4-6`).

The results in Table 3.2 are shown based on the band subset $\{B_1, B_4\}$ as a starting point — i.e., as the initial two-band sequence that defines the first column of data in the table. This pair of bands is selected because it corresponds to the best two-band PLF results in Table 3.1, and as motivated above, we have disabled FLF in LBSS for this part

Table 3.3: Accuracy improvement compared with results from [1] using the same multispectral dataset.

	Precision	Recall	F_measure
LDspectral	0.969	0.953	0.961
[Benezeth 2014]	0.870	0.925	0.897
Improvement	11.4%	3.0%	7.1%

of the experimental evaluation.

From the results in Table 3.2, we see that, as expected, $F_{measure}$ increases as the cardinality of the set of selected bands increases. The improvement is larger at first (when constructing smaller band subsets), and then becomes smaller when constructing larger subsets. These trends are important as they influence trade-offs between the increased accuracy provided by processing additional bands and the increased computational cost incurred by such processing. The execution time aspects of these trade-offs are investigated in Section 3.3.4.

Table3.3 shows the improvement in accuracy provided by LDspectral compared to related methods reported in the literature that are evaluated on the same multispectral dataset. These results are for the full set of (7) available multispectral bands. The $F_{measure}$ value is improved by 7.1% through methods in LDspectral. This is a relatively large improvement given that an upper bound on the achievable improvement (represented by $F_{measure} = 1$) is $(1-0.897)/0.897 = 11.4\%$. Factors that contribute to this improvement include the integrated use in LDspectral of a GMM model for PLF, and the grid search optimization of the PLF configurations.

Table 3.4: Variation in execution time for different numbers of processed bands and different fusion modes. The units of execution time in this table are milliseconds/frame.

	1	2	3	4	5	6	7
PLF	31.1	34.9	38.8	43.0	47.5	51.6	55.4
FLF	31.1	52.7	81.3	93.7	101.8	121.0	138.5

3.3.4 Execution Time Evaluation

Table 3.4 shows the variation in execution time for different numbers of processed bands and different fusion modes. These results are given in terms of milliseconds per video frame that are required to execute the BSP dataflow graph (Figure 3.2). The columns in the table correspond to different band subset sizes. The first column corresponds to the execution time required for BSP when only one spectral band is involved, and hence no fusion is performed. Thus, the execution times reported in both rows are the same for the first column.

The execution times reported in Table 3.4 are obtained by averaging over ten iterations through the training dataset for a band subset of each given cardinality. These results demonstrate that the execution time of FLF exhibits a significantly more rapid increase compared to PLF as the number of bands increases. For example, from the trends shown in Table 3.4, we see that the execution time required to perform FLF on 2 bands exceeds the time required to perform PLF on 6 bands. This kind of result further highlights the need for careful, joint selection of fusion configurations and band subsets in a system that is geared toward optimizing trade-offs between accuracy and real-time performance.

3.4 Summary

In this chapter, we have developed new methods for integrated band subset selection and video processing parameter optimization in LDspectral, which is a software tool for model-based system design, prototyping, and optimization of data-driven, multispectral video processing systems. LDspectral is developed for optimization in the context of novel video processing design spaces introduced by multispectral image acquisition techniques. The methods developed in this chapter enable experimentation with and optimization of data-driven video processing for DDDAS. The methods are demonstrated in terms of accuracy and execution time using a case study involving background subtraction, and a relevant multispectral data set for this application.

Both pixel level fusion and feature level fusion are implemented in LDspectral. In pixel level fusion, a greedy selection algorithm is designed to fully explore the space of all possible band subsets. The weighting coefficients are optimized as an integral part of the band selection process. Improvements in accuracy at the cost of increased execution time are observed with increases in the number of selected bands.

Chapter 4

Hyperspectral Video Processing on Resource-Constrained Platforms

In this chapter, we extend LDspectral to support hyperspectral image and video processing. Moreover, we optimize the system for efficient implementation on resource-constrained platforms. First, we develop a multithreaded version of LDspectral to enhance the execution speed of the system on resource constrained platforms that are equipped with multicore processors. The number of actors devoted to each thread is carefully configured to distribute the workload in a balanced manner across the available processor cores.

Second, we present a novel adaptive video processing system that is derived from the LDspectral framework. The system exploits the flexibility of band-subset selection to efficiently handle time-varying requirements on the frame rate and video analysis accuracy.

Material in this chapter was published in preliminary form in [37] and [38]

Background on hyperspectral image and video processing systems can be found in a variety of tutorials in the literature. For example, Birk and McCord provide a review of many different airborne hyperspectral sensing systems, and also provide a detailed comparison of their system specifications [39]. Matteoli, Diani, and Corsini present a survey of methods for processing hyperspectral imagery to detect small human-made anomalies that are relevant in defense and surveillance applications [40].

4.1 Introduction

Hyperspectral video processing systems (HVPSs) offer advanced capabilities for scene analytics and knowledge extraction due to their high levels of spectral diversity and spectral resolution compared to conventional video technologies. With the advancement of video acquisition techniques, HVPSs are playing increasingly important roles in video processing applications. Hyperspectral video streams provide high spectral diversity due to their high density of sampling rate in the wavelength dimension, and their capacity to incorporate diverse regions of the spectrum. Major application areas for hyperspectral image and video processing include remote sensing [41], agriculture [42], vehicle tracking [28], and medical diagnostics [43]. However, the high density of bands involved in HVPSs brings challenges in exploiting the potential of hyperspectral imaging technology. These challenges are especially severe in the context of resource-constrained, embedded deployment, where limited memory and computational resources are available due to constraints on size, weight, power or cost.

Figure 4.1 shows a hyperspectral cube of a scene in a dataset that we use in our experiments in this chapter. The dataset is generated by the Digital Imaging and Remote Sensing Image Generation (DIRSIG) system, which contains 110 frames with 110 bands in the range of visible light and infrared (from 400 nm to 1000 nm with 10 nm spectral resolution) with a spatial resolution of 1200x800 [28]. In recent years, the development of hyperspectral photogrammetry has increased requirements on real-time hyperspectral video processing. There are many published works on airborne hyperspectral video processing applications, such as ecological monitoring using drones [44].

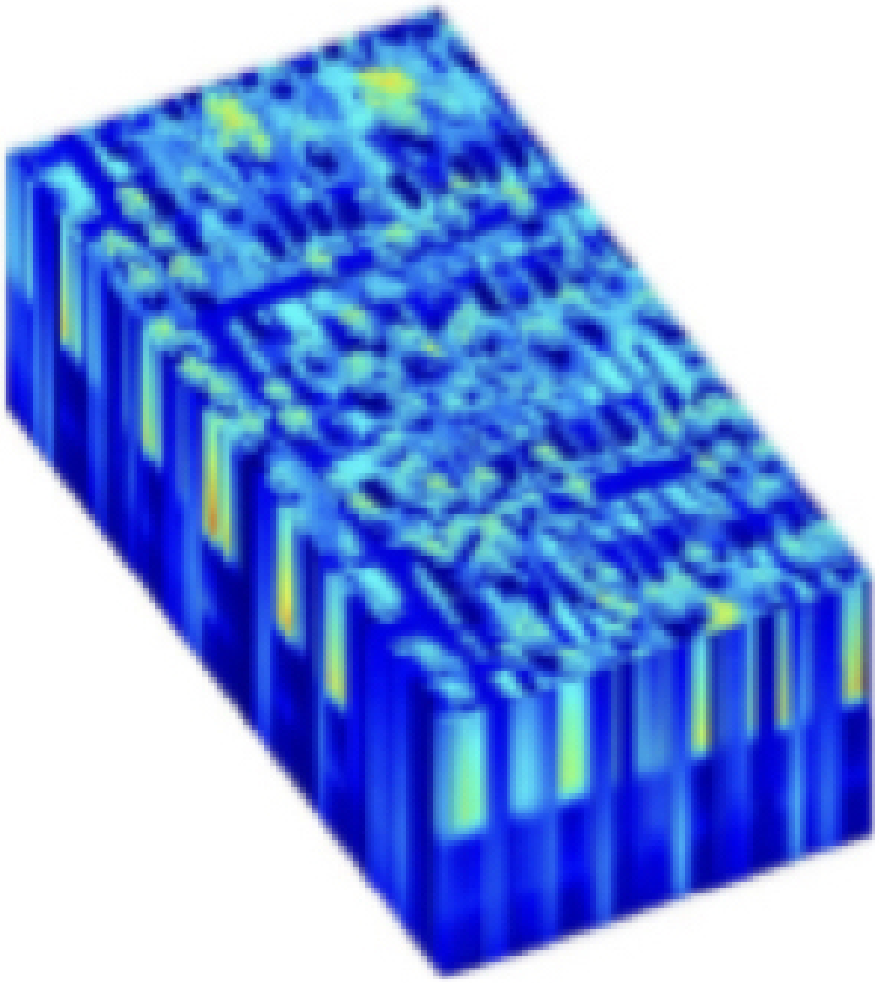

Figure 4.1: A hyperspectral cube of a scene in a dataset that we use in our experiments in this chapter.

We develop systematic methods for optimizing real-time performance subject to stringent resource constraints, and for efficiently trading-off real-time performance and video analysis accuracy. We demonstrate our design methods through an implementation case study involving a background subtraction application that is mapped to a low-cost Android platform.

In this chapter, we develop new system design methods to address these challenges, thereby contributing novel capabilities for deploying HVPS technology in a wider variety

of applications. The design methods include strategic selection of band subsets to reduce processing requirements without major loss in video analysis accuracy. Applying these design methods, we design and implement a prototype HVPS on an Android platform, and conduct experiments using a relevant hyperspectral video dataset. The experimental results demonstrate the capability of the proposed system to provide optimized hyperspectral video processing operation subject to stringent resource constraints, and to efficiently trade off real-time performance and video analysis accuracy.

In recent years, advances in hyperspectral sensor technology have helped to increase the availability of hyperspectral imaging systems, which results in an increasing variety of applications for hyperspectral image and video processing (e.g., see [14]). Generally, hyperspectral imaging systems can involve hundreds, thousands or even more bands for the same scene. In addition to being more numerous, the bands employed in hyperspectral imaging systems have narrower bandwidths, thereby offering greater spectral resolution. Along with this increased resolution, however, comes the increased potential for redundancy across different bands. Thus, a natural mechanism for reducing processing requirements (e.g., to improve real-time performance or energy efficiency) in an HVPS is to select a proper subset of the available bands that provides sufficient accuracy and minimizes the storage and processing of spectral information that is redundant or is otherwise not of high relevance for the required video analysis tasks.

Various methods have been reported in the literature that are relevant to extraction of useful information from the diverse channels provided by hyperspectral imaging sensors. For example, Liu et al. provide a comparative study of different multiresolution algorithms for image fusion [17]. Wei et al. present an image fusion method for multispectral

and hyperspectral images. Their method leads to less spectral error and spectral distortion compared to related fusion techniques [26]. Lin et al. compare four state-of-the-art methods for fusion of hyperspectral images [45]. Chen et al. demonstrate a pan-sharpening approach for fusing low-spatial-resolution hyperspectral images and high-spatial-resolution multispectral images of the same scene [27].

In our previous work, we demonstrated a multispectral video processing system for dynamically reconfigurable band-subset selection [13]. The system optimizes trade-offs between video analysis accuracy and processing speed. This chapter goes beyond the previous work in its focus on the more challenging requirements of hyperspectral video processing, and its targeting of highly resource-constrained devices that enable less costly, more widespread deployment.

Uzkent et al. have developed a framework for controlling hyperspectral data collection [28]. They also introduced a publicly available hyperspectral video dataset for vehicle tracking. Sobral et al. propose a stochastic tensor decomposition algorithm for robust background subtraction. Sobral's results show that red-green-blue (RGB) features are not sufficient to handle color saturation, illumination variations and problems due to shadows, while incorporating six visible spectral bands together with one near-infra-red band helps to address these limitations [29].

The distinguishing aspects of our work include its emphasis on jointly optimizing accuracy and real-time performance in HVPSs under stringent resource constraints, with specific use of an Android smartphone platform to demonstrate the proposed methods. This chapter also introduces a novel adaptive video processing system that exploits the flexibility of band-subset selection to efficiently handle time-varying requirements on the

frame rate and video analysis accuracy.

4.2 Design Methods

A core aspect of our proposed design methods is the emphasis on strategically selecting the subsets of bands to store and process at run-time. We refer to this problem as the *band subset selection* problem for HVPS design. The band subset selection problem captures the problem in resource-constrained design contexts where there is insufficient time (under real-time constraints) or insufficient resources to process all of the available spectral bands in an HVPS.

In the version of the band subset selection problem that we study in this chapter, we assume that the available hyperspectral data comes from a set $Z = \{B_1, B_2, \ldots, B_N\}$ of spectral bands, where N denotes the total number of available bands. The problem is to choose an $S \in 2^Z$ so that processing of this selected subset of bands (while discarding all other bands) maximizes video analysis accuracy subject to a real-time performance constraint C_r, where C_r specifies the maximum allowable latency for processing a single video frame. Other versions of this problem can be formulated, for example, by replacing C_r by a constraint on energy consumption or by a weighted sum of the run-time and energy consumption.

Band subset selection tries to optimize the subset of bands that is to be processed during the interval of video processing. The output of band subset selection is a subset $S \subset Z$ of the bands that are to be stored and processed, while all of the other bands (the bands in $(Z - S)$) are ignored. Band subset selection is performed initially at design time

or during system initialization, and then the selection process can generally be repeated periodically at run-time based on some reconfiguration interval parameter T_r.

In Chapter 2, we introduced a system design framework, called *LDspectral*, for efficient processing of multispectral video streams. Here, "LD" stands for *lightweight dataflow*, which is a design methodology and software tool for model-based design and implementation of signal and information processing systems [10]. LDspectral applies lightweight dataflow and integrates on top of it novel methods for efficient processing of multispectral video streams. In this chapter, we build on LDspectral and extend its capabilities in two major ways. First, we extend it for use on hyperspectral video streams. Second, we develop methods to map the framework into efficient implementations on resource-constrained platforms. We demonstrate the results of our optimized hyperspectral video processing framework on a background subtraction application that is implemented on a low-cost Android framework.

The tools that we used to implement the Android version of LDspectral include the Android Native Development Kit (NDK), Android Debug Bridge (ADB), OpenCV4Android SDK, DICE [46], and LIDE [47]. NDK is a cross-compiler developed for the Android platform that supports the integration of native code (C/C++) into Android applications. ADB allows communication between an Android device and a development machine so that the application can be developed and executed using the Linux command-line environment, which in turn allows the design process to be integrated with the signal processing software development features of the DICE Package. For more details about DICE, we refer the reader to [46] The OpenCV4Android SDK provides a handful of libraries that can be used along with the NDK to build applications or libraries

for Android applications.

In our experimental setup, the LDspectral system is designed and trained off-line on a desktop computer. This is illustrated in Figure 4.2. Then the trained model, including the optimized PBC parameters, is pushed to the Android device through ADB. The real-time execution uses the trained parameters from a look-up table and dynamically reconfigures the system based on considerations of real-time performance, energy consumption, and accuracy.

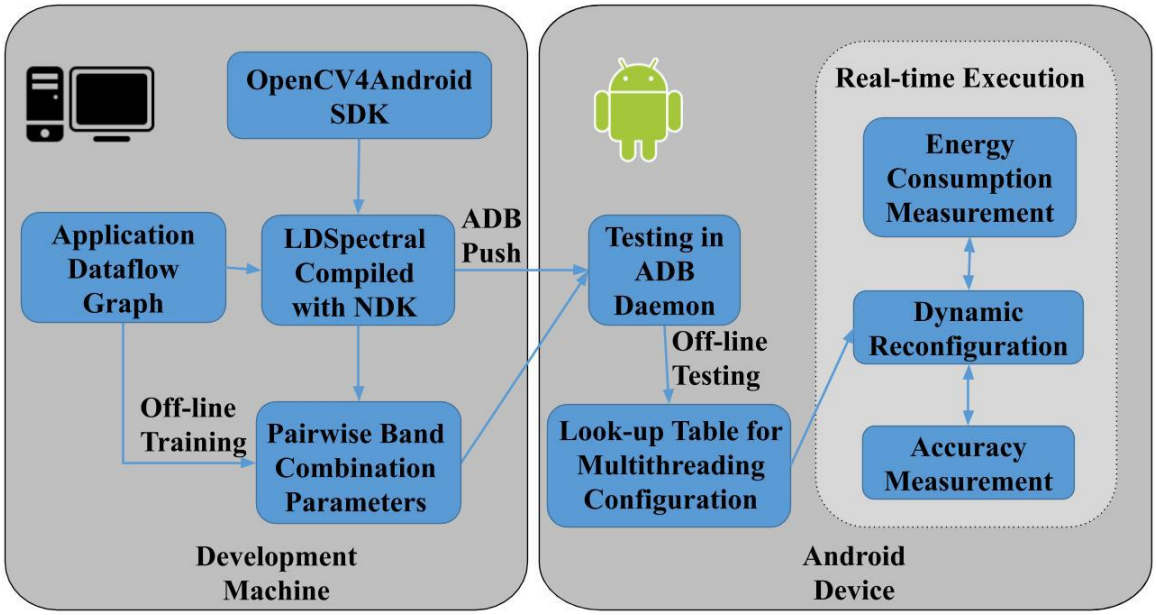

Figure 4.2: Experimental setup for applying LDspectral to Android-based hyperspectral video processing implementation.

The proposed new design flow using LDspectral is summarized as follows. First, a dataflow model of the given HVPS is developed using OpenCV on the development machine (host computer), and functional testing is performed using DICE. Then LDspectral is compiled using the NDK cross-compiler and OpenCV4Android SDK so that it can be ported to an Android device. Finally, ADB is used to transfer the compiled program to the

targeted mobile device, and provide a DICE-integrated environment for in-device testing and measurement of the implemented HVPS.

In this section, we introduce an efficient real-time HVPS that is targeted to an Android platform, and we present the underlying system design methods, which are centered on band-subset selection. The system implements background subtraction as a concrete video analysis application. The background subtraction (back-end) component of the system can readily be replaced or augmented with other video analysis techniques. This capability allows system designers to utilize in different ways the framework's capabilities for adaptive, resource-constrained hyperspectral video processing.

An important feature of the proposed HVPS is its efficiency and configurability for processing streams of hyperspectral image inputs. The proposed HVPS maintains a priority list of spectral bands that is determined through an off-line training process. The priority list is created based on each band's contribution to the overall accuracy when the bands are equally weighted. At run-time, the HVPS accesses the priority list to select N_b bands that have the highest priority, where N_b is determined based on the current real-time constraint and a constraint on video analysis accuracy. These constraints are assumed to be system parameters that can be changed dynamically. The real-time constraint specifies the minimum number of frames per second (fps) at which the system is expected to process its hyperspectral input stream.

Figure 4.3 illustrates the dataflow for a small-scale example configuration of the sequential, fixed-configuration (non-adaptive) first-version HVPS from [37], which we used as a starting point in this work. The dataflow graph shown in Figure 4.3 corresponds to a configuration in which five bands are selected for the enclosing background subtraction

application, while all other available bands are ignored.

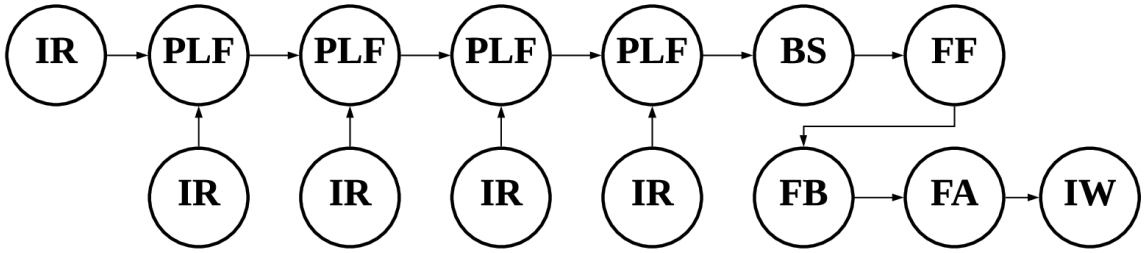

Figure 4.3: Dataflow graph for an example configuration of the first-version HVPS.

Each circle in Figure 4.3 represents an actor (signal processing module) in the dataflow graph. Brief descriptions of the actors are as follows: IR — Image Read, PLF — Pixel-Level Fusion, BS — Background Subtraction, FF — Foreground Filter, FB — Foreground Binarization, FA — Foreground Accuracy computation (for measurement and diagnostic purposes), and IW — Image Write. In each iteration of the dataflow graph, the IR actor reads from a set of files the selected bands of the next input image, and injects the image into the dataflow graph for processing.

Figure 4.4 illustrates the dataflow for a multithreaded version of Figure 4.3, which provides improved processing efficiency on the targeted multicore Android platform. This version allows more bands to be processed under a given real-time constraint, thereby improving background subtraction accuracy. The dataflow graph is composed of two parts, which we refer to as the pixel-level fusion (PLF) section (Threads 1–3) and background subtraction (BS) section (Thread 4). These sections are denoted, respectively, as S_p and S_b.

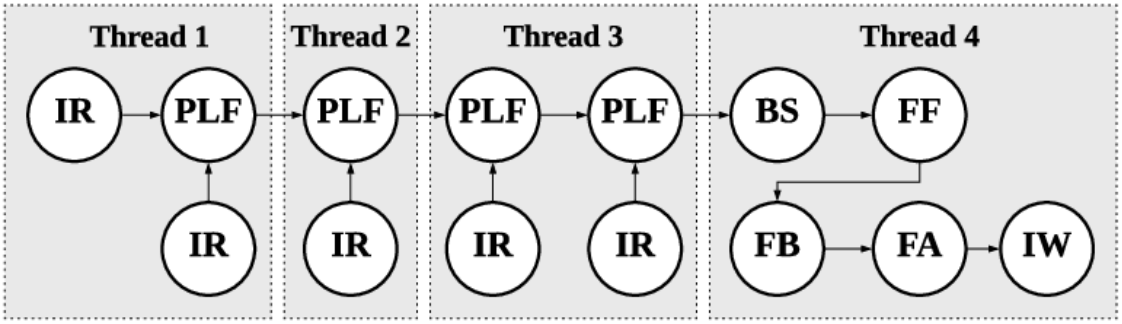

Figure 4.4: Multithreaded version of Fig. 4.3 for mapping onto the targeted Android platform.

The PLF section performs pixel-level fusion to integrate pixel values from different spectral bands into a single image. The BS section then uses a Gaussian Mixture Model to perform background subtraction on the fused image. We pipeline the PLF section, using a simple pipeline of three stages, where each thread corresponds to a single stage. The three stages apply different steps of the PLF section concurrently across three successive frames of the input video stream, thereby helping to improve the achievable frame rate. The BS stage operates as an additional (fourth) pipeline stage, which processes the most recent image frame that has passed through all stages of the PLF section.

The pipelining process used in this design method is generalized naturally to handle arbitrary numbers of threads (e.g., for systems in which smaller or larger numbers of processing cores are available), arbitrary numbers of spectral bands, and the possibility of adding multithreading to the BS section. In the generalized form, let P, Q_p and Q_b respectively denote the total number of available threads, the number of threads allocated to the PLF section S_p, and the number of threads allocated to the BS section S_b. We assume that the available threads are utilized fully in the system design so that $P = Q_p + Q_b$. The

decomposition of the available P threads into Q_p and Q_b is performed through experimentation — e.g., a binary search on Q_p can be used to arrive at a decomposition through a low-complexity experimentation process. More systematic approaches to performing this decomposition represent an interesting direction for future work.

Now if Q_p evenly divides N_b, then the number of bands allocated to each thread in S_p is simply N_b/Q_p. Otherwise, each S_p thread is assigned either $\mathit{flr}(N_b/Q_p)$ or $\mathit{clg}(N_b/Q_p)$ bands with the assignment performed in such a way that the sum of the band-to-thread assignments across S_p equals N_b. Here, flr and clg represent the floor and ceiling functions, respectively. This simple approach to distributing the processing of bands helps to keep the load of the pipeline stages balanced, which is important for throughput optimization. Here, we have assumed that $P < N_b$. The approach can be adapted easily to accommodate the case in which $P \geq N_b$; we omit the details for brevity.

Figure 4.5 illustrates a multithreaded design of the proposed HVPS for $P = 6$, and $Q_p = Q_b = 3$. The multithreaded configuration of the BS section incorporates two additional actors, denoted IP (Image Partitioning) and IS (image stitching). These actors, respectively, partition an image into subframes for processing across multiple threads, and integrate the different results of subframe processing into a single result.

Another important aspect of the proposed HVPS is the capability to dynamically adapt band-subset selection based on changes in real-time processing requirements or requirements in the level of video analysis accuracy (e.g., based on switching between high- and low-criticality modes of operation). Algorithm 2 gives a pseudocode sketch of the algorithm used for top-level configuration control and processing in the proposed HVPS. The while-loop ("infinite loop") in the algorithm simply indicates continuous op-

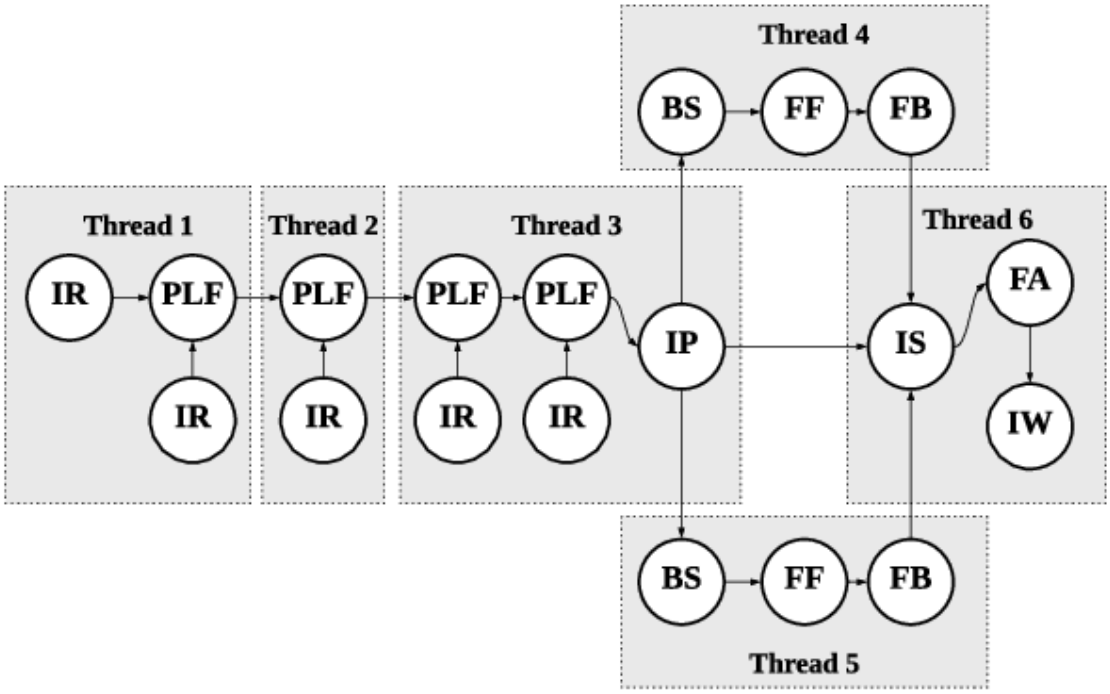

Figure 4.5: Multithreaded version of Fig. 4.3 for $P = 6$, and $Q_p = Q_b = 3$.

eration that keeps processing input until the system is terminated through some sort of external/asynchronous control, such as a power-down operation.

The $changed(p)$ function returns a Boolean value indicating whether or not the dynamic parameter p has changed (by some process external to the procedure) since system initialization (the first time the *changed* function is called on a given parameter) or since the previous call to the *changed* function (for all subsequent calls).

The video analysis accuracy metric used in the proposed HVPS can be defined based on the associated back-end video analysis functionality that is employed. For our background-subtraction-based HVPS prototype, we use the $F_{measure}$ metric, which is a commonly-used metric for assessing results of background subtraction (e.g., see [1]). We use f_m throughout the remainder of this chapter as a shorthand for $F_{measure}$.

Algorithm 2 A pseudocode sketch of the algorithm used for top-level configuration control and processing.

`parameter` C_r: frame rate (throughput) constraint.
`parameter` f_M: accuracy constraint.
`parameter` T: configuration monitoring interval.
`parameter` P: number of available threads.

1: **procedure** HVPS-TOPLEVEL(C_r, f_M, T, P)
2: **while** *true* **do**
3: **if** changed(f_M) **then**
4: $N_{\text{b1}} = lookup1(T_1, f_M)$
5: **if** changed(C_r) **then**
6: $N_{\text{b2}} = lookup2(T_2, C_r)$
7: $N_b = max(N_{\text{b1}}, N_{\text{b2}})$
8: $process\_frames(N_b, T, P)$

The algorithm utilizes two lookup tables, denoted T_1 and T_2. The table T_1 tabulates for different values of the accuracy metric (f_m) an estimate of the minimum numbers of spectral bands (band-subset size) that are required to achieve the specified accuracy levels. The table T_2, on the other hand, tabulates for different throughput (fps) levels, estimates on the maximum values of N_b that can be utilized without having performance fall below the throughput levels. The estimates stored in T_1 and T_2 are determined off-line through experimentation and stored in a sorted form for fast retrieval of the information at run-time. The function $lookup1(x)$ shown in Algorithm 2 returns the smallest value of N_b from lookup table T_1 that can achieve the specified accuracy level x. Similarly, the function $lookup2(x)$ returns the largest value of N_b from lookup table T_2 that can achieve the throughput level specified by x.

The two table-lookups described above result in two candidate values for N_b, which are denoted, respectively by N_{b1} and N_{b2}. Algorithm HVPS-Toplevel then sets N_b by taking the maximum of these two candidate values, which effectively gives priority to

the accuracy criterion. By changing this maximum operation to a different function, the designer can change the way the two criteria are considered in the HVPS configuration process (e.g., by prioritizing the throughput metric or applying a weighted combination to achieve a composite priority function).

After determining N_b, Algorithm HVPS-Toplevel calls *process_frames*, which encapsulates the core processing functionality (dataflow graph) of the HVPS. The function is called by passing the total number P of threads, and the band-subset size N_b that should used for the processing. The function is also called with a parameter T, which specifies the number of frames for which processing should continue before control is returned to the top-level control/configuration process represented by Algorithm HVPS-Toplevel. The parameter T effectively specifies the periodicity with which the system configuration is re-examined for a possible change in system constraints (C_r or f_M) and subsequent adaptation of processing parameters in response to such a change.

4.3 Results

We use an Oppo N3 Android phone as the testing platform for our proposed HVPS. Oppo N3 features a Qualcomm MSM8974AA Snapdragon 801 Quad-core ARM CPU with a maximum frequency of 2.3 GHz, 2GB of RAM, and 32GB internal storage capacity. The Android OS version is 4.4.4 and Linux kernel version is 3.4.0. The hyperspectral dataset we use is generated by the DIRSIG model [28]. The dataset has 110 frames of video, where each frame has 61 spectral bands. More details on the dataset can be found in [37, 28].

Table 4.1: Derived energy consumption, execution time, and CPU usage on Android device for sequential version.

N_b	7	14	28	56
C_e (mAh)	16.20	22.18	33.65	63.94
C_t (s)	94.16	133.88	209.74	397.22
C_u (%)	32.18	28.05	28.41	26.55

Table 4.1 shows results on our background subtraction application for the average energy consumed C_e in mAh, average runtime C_t in seconds, and average CPU usage C_u as a percentage. The results are shown for different values of the number of bands N_b. The results show a clear trend of increasing energy consumption and increasing runtime for increasing values of N_b. On the other hand, there is relatively little variation in CPU utilization seen for the different values of N_b. We anticipate that this is because we limited the number of threads in the CPU implementation to 1 in the experiments reported in Table 4.1.

For the first experiment, we collect average accuracy and frame rate results delivered by the proposed HVPS. The results are presented in terms of f_m across different values of N_b, and different multithreading configurations for each value of N_b. The data is collected for $N_b \in Z_b$, where $Z_b = \{10, 20, \ldots, 60\}$, and summarized in Figure 4.6. Each bar in Figure 4.6 represents the frame rate for a specific multithreaded or sequential configuration (Q_p, Q_b), denoted in the form "$Q_p + Q_b$", and for a specific value of $N_b \in Z_b$ (the bars for each $N_b \in Z_b$ are grouped together in the figure). Each of the six dark-shaded diamonds shows the accuracy for a given value of N_b based on the vertical-axis scale provided on the right side of the figure. The fps values displayed in Figure 4.6 are averaged over 50 executions for each (N_b, Q_b, Q_p) combination, and the accuracy values are

averaged over all five sets of 50 executions for each N_b setting.

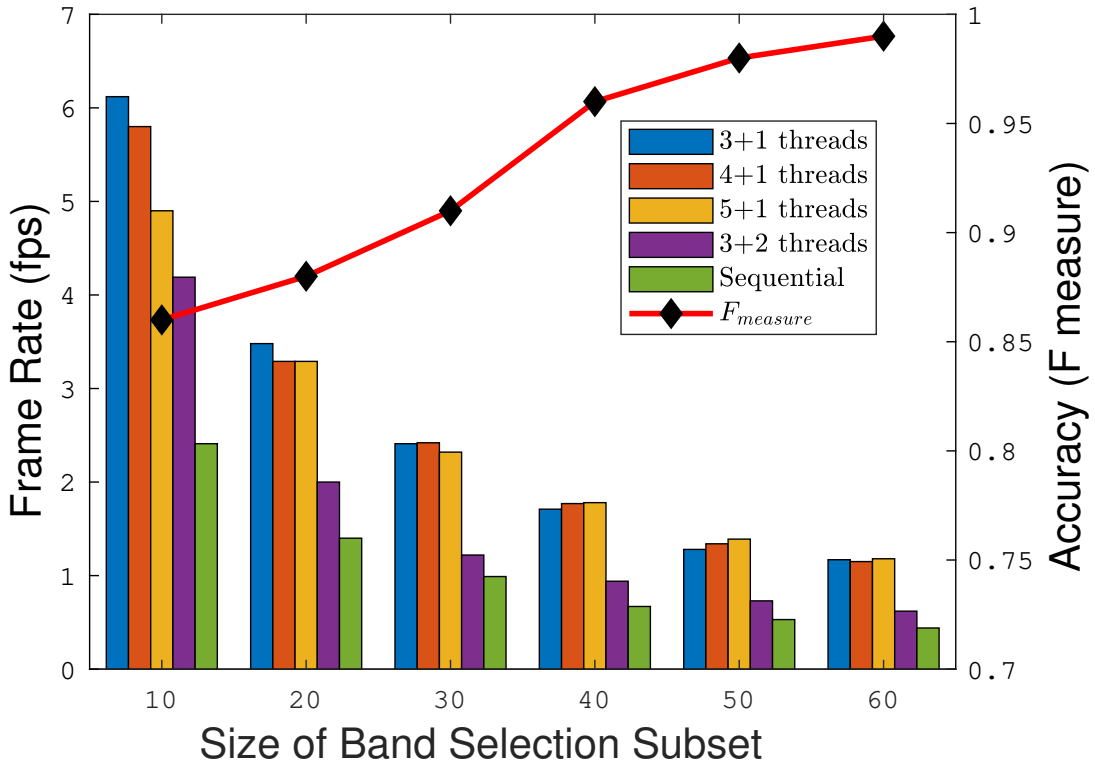

Figure 4.6: Frame rate and accuracy for different N_b and different multithreading configurations.

The figure shows that the accuracy increases monotonically with increasing $N_b \in Z_b$, and quantifies this trend of increasing accuracy. For a given (Q_p, Q_b) configuration, we see that throughput also decreases monotonically with increasing N_b. However, for a given N_b, there is no general monotonic trend of throughput in terms of the number $P = Q_p + Q_b$ of allocated threads. We expect that this is due to nonlinear effects related to thread allocation, such as overhead due to contention and communication across threads.

To provide more detailed insight on video processing throughput in our Android-based HVPS, a tabulation of the experimental results for the frame rate (fps) is shown in Table 4.2. As mentioned previously, each configuration is executed 50 times in our

experiments; each row of the table shows key statistics across the 50 executions associated with a given configuration. In particular, the table shows the maximum, minimum, mean, median, and standard deviation σ of measured fps for each of the configurations represented in Figure 4.6. From the results in Table 4.2, we see that beyond the throughput trends discussed above in relation to Figure 4.6, the data in Table 4.2 demonstrates that variations in the frame rate are typically small for a given configuration (e.g., with relatively low deviation between the minimum and maximum measured values), leading to production of hyperspectral video analysis results at a fairly consistent rate.

In general, for a given value of N_b, all multithreaded versions achieved better frame rates compared to the corresponding sequential versions ($P = 1$). However, multithreading in the BS section resulted a performance degradation. This is observed when comparing the results for $Q_p = 3, Q_b = 1$ to the corresponding results for $Q_p = 3, Q_b = 2$. We anticipate that this is because the calculation for the Gaussian Mixture Model used in the BS actor, which is the core computation of the BS section, is not a bottleneck of the HVPS.

While the achieved frame rate levels, as reported in Figure 4.6 and Table 4.2, are relatively low, they are sufficient for applications of resource-constrained sensing where the scene changes slowly or response time is not critical — for example, scenarios at the network edge in which the resource constrained system is used as a first-level of analysis, and is to be followed by more communication- or resource-intensive analysis at a base station if certain types of events are detected.

Table 4.2: Statistics on the measured frame rates for different operational configurations. The unit for each entry in the table is frames per second (fps).

Configuration	max.	min.	mean	median	σ
N_b = 10, 3+1 threads	6.30	5.92	6.12	6.12	9.42×10^{-2}
N_b = 10, 4+1 threads	5.95	5.52	6.80	5.80	8.71×10^{-2}
N_b = 10, 5+1 threads	5.02	4.80	4.90	4.89	5.38×10^{-2}
N_b = 10, 3+2 threads	4.65	4.04	4.19	4.13	1.84×10^{-1}
N_b = 10, Sequential	2.44	2.34	2.42	2.42	3.14×10^{-2}
N_b = 20, 3+1 threads	3.59	3.29	3.48	3.48	3.18×10^{-2}
N_b = 20, 4+1 threads	3.39	3.09	3.29	3.31	7.55×10^{-2}
N_b = 20, 5+1 threads	3.36	3.06	3.21	3.21	6.94×10^{-2}
N_b = 20, 3+2 threads	2.04	1.98	2.00	1.99	1.77×10^{-2}
N_b = 20, Sequential	1.41	1.40	1.40	1.41	0.91×10^{-2}
N_b = 30, 3+1 threads	2.43	2.39	2.41	2.41	1.23×10^{-2}
N_b = 30, 4+1 threads	2.46	2.35	2.42	2.43	3.37×10^{-2}
N_b = 30, 5+1 threads	2.41	2.25	2.32	2.32	5.32×10^{-2}
N_b = 30, 3+2 threads	1.24	1.21	1.22	1.23	7.10×10^{-3}
N_b = 30, Sequential	1.00	0.97	0.99	0.99	7.90×10^{-3}
N_b = 40, 3+1 threads	1.73	1.69	1.71	1.72	1.61×10^{-2}
N_b = 40, 4+1 threads	1.80	1.72	1.77	1.78	2.34×10^{-2}
N_b = 40, 5+1 threads	1.82	1.73	1.78	1.78	2.47×10^{-2}
N_b = 40, 3+2 threads	0.96	0.93	0.94	0.94	6.60×10^{-3}
N_b = 40, Sequential	0.69	0.66	0.67	0.67	9.60×10^{-3}
N_b = 50, 3+1 threads	1.30	1.23	1.28	1.29	2.02×10^{-2}
N_b = 50, 4+1 threads	1.35	1.33	1.34	1.34	6.60×10^{-3}
N_b = 50, 5+1 threads	1.41	1.37	1.39	1.39	1.46×10^{-2}
N_b = 50, 3+2 threads	0.74	0.72	0.73	0.73	4.59×10^{-3}
N_b = 50, Sequential	0.55	0.53	0.53	0.53	6.19×10^{-3}
N_b = 60, 3+1 threads	1.18	1.16	1.17	1.17	7.93×10^{-3}
N_b = 60, 4+1 threads	1.17	1.13	1.15	1.16	1.06×10^{-2}
N_b = 60, 5+1 threads	1.19	1.17	1.18	1.18	7.90×10^{-3}
N_b = 60, 3+2 threads	0.63	0.62	0.62	0.62	4.28×10^{-3}
N_b = 60, Sequential	0.45	0.44	0.44	0.44	4.06×10^{-3}

4.4 Additional Experiments

In this section, we present additional experiments beyond the main experimental results of this chapter, which were presented in Section 4.3. The results in this section provide additional insight into the methods and tools developed in this chapter.

Figure 4.7 shows measurements of how accuracy varies with variation of the number of selected bands N_b. Similarly, Figure 4.8 shows measurements of how battery consumption and execution time per frame vary with N_b.

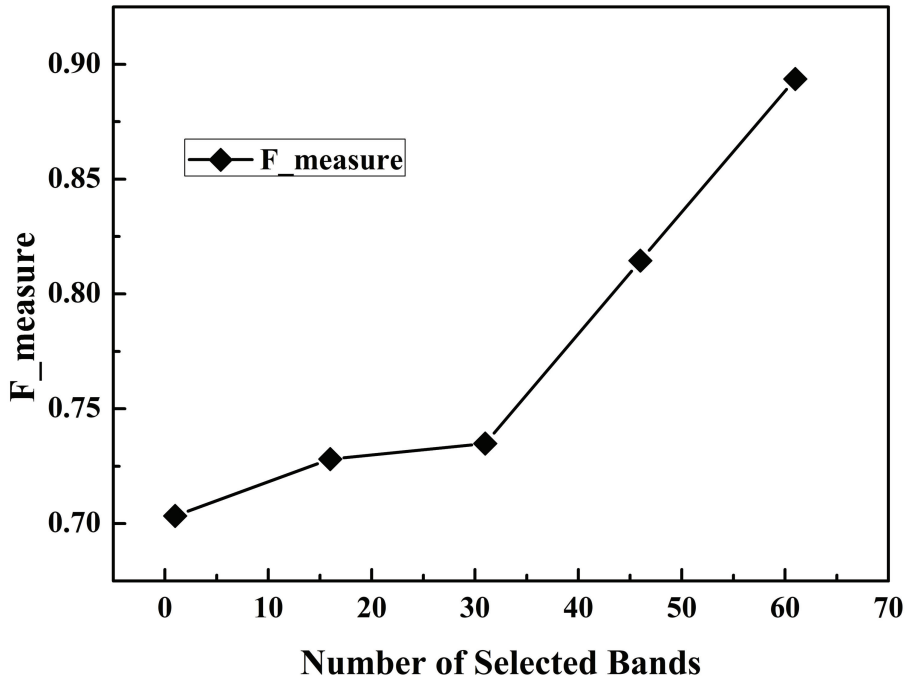

Figure 4.7: Variation in measured accuracy ($F_{measure}$) for different values of N_b.

As shown in Figure 4.7, and as expected, the accuracy increases with the increasing of number of selected bands. The *F_measure* value increases by 27% when the number of bands increases to 61 from 1. From Figure 4.8, we can see that the average battery power consumed C_e and average execution time per frame C_t of our background subtraction application on the Android device is positively correlated to the number of selected bands

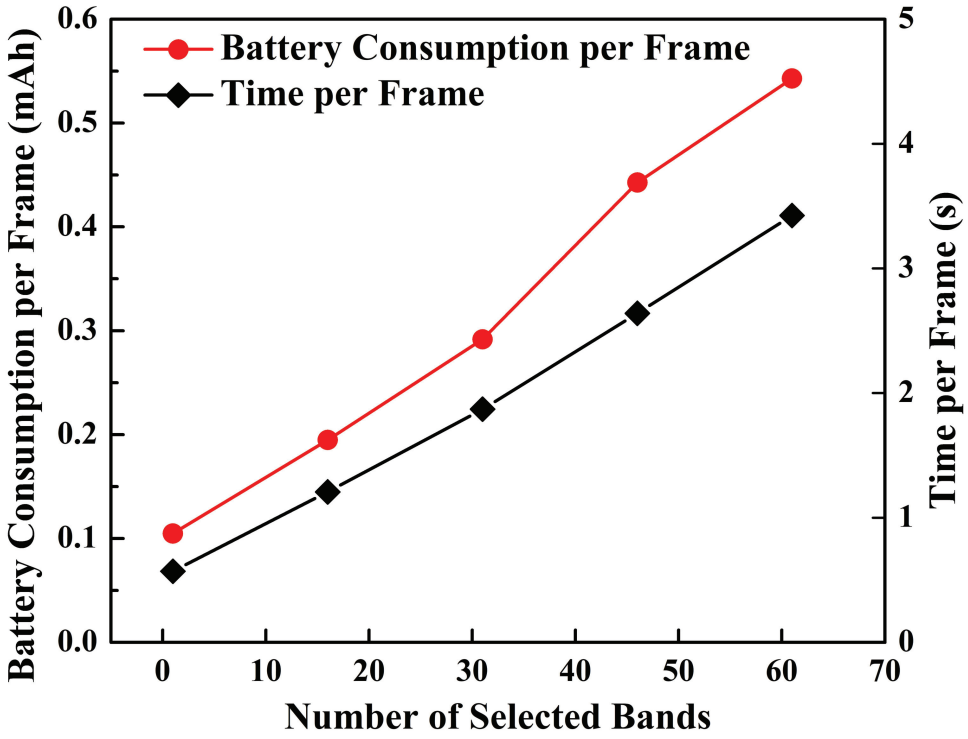

Figure 4.8: Variation in battery consumption and execution time per frame for different values of N_b.

N_b. The energy consumption increases from 11.9 mAh to 59.7 mAh as N_b increases from 1 to 61, while the execution time per frame increases from 0.57 seconds to 3.43 seconds. Both of these figures of merit increase monotonically, as expected, with the increase of N_b.

The system derived from LDspectral is dynamically reconfigurable in that N_b can be adjusted automatically to ensure that the system stays within a predefined constraint on the system power consumption. The reconfiguration is performed by periodically monitoring the real-time current drawn from the battery by using the ADB command `dumpsys batteryproperties` and calculating the average current over non-overlapping windows of 110 video frames. Since the system voltage remains approximately constant, the maximum power consumption constraint can easily be translated to

Table 4.3: Frame rate (frames per second) for different multithreading configurations.

$Q_p + Q_b \backslash N_b$	10	20	30	40	50	60
3+1	6.12	3.48	2.40	1.71	1.28	1.17
4+1	5.80	3.29	2.41	1.77	1.34	1.15
5+1	4.90	3.21	2.32	1.78	1.39	1.18
3+2	4.90	3.21	2.32	1.78	1.39	1.18
Sequential	2.41	1.40	0.98	0.67	0.53	0.44

a constraint I_{max} on the current consumption.

In our experiments to validate the power-constrained reconfiguration capability of LDspectral, we simulate video inputs by repetitively iterating through the DIRSIG dataset and performing background subtraction continuously at run-time. The step size we chose for testing is 15 bands. This means that if the real-time current is above the given current threshold I_{max}, then the system will reduce N_b by 15 in the next reconfiguration round. The reconfiguration process terminates when the real-time current is less than or equal to the threshold I_{max}, which indicates that the system is running within the predefined power consumption level. During our experiments, we observed that the voltage stays at 4.2 V while the current drawn from the battery varies depending on N_b, as expected.

The frame rate generally increases when multithreading is enabled for pixel level fusion (PLF). Measured variations in frame rate across different PLF multithreading configurations are summarized in Table 4.3. Each entry in the table is the measured frame rate (frames per second) for a specific combination of Q_p, Q_b, and N_b. The 3+1 multithreading configuration has higher frame rate compared to other configurations when $N_b < 30$. For $N_b > 30$, the 5+1 configuration performs the best in terms of the frame rate. However, for such larger values of N_b, the frame rate differences between different multithreaded configurations are very small.

We also compared results from applying Algorithm 2 under different operational constraints (constraints on C_r and f_M) and different configurations $Q_P = \{3,4,5\}$ and $Q_b = 1$. Here, f_M denotes the constraint on the minimum acceptable value for f_m. The results are shown in Figure 4.9 and Figure 4.10. The figures show variations in the achieved frame rate with the number of dataflow graph iterations that are executed.

In these two figures, red, blue, and green curves correspond to $Q_P = \{3,4,5\}$. In Figure 4.9, the black line shows the constraint on C_r, while a similar line does not appear in Figure 4.10 because its value falls outside of (above) the vertical axis limit in the figure. In Figure 4.9, all three configurations are able to satisfy the constraint on C_r when $f_M = 0.9$. On the other hand, in Figure 4.10, none of the three configurations meet the $C_r = 5$ requirement, which means that the constraint pair $(C_r, f_M) = (5.0, 0.95)$ cannot be satisfied simultaneously for this Android device. We anticipate that the frame rate decrease over time, which is pronounced in Figure 4.10, is caused by significant temperature increase in the device.

4.5 Summary

In this chapter, we have developed new system design methods for deploying hyperspectral video processing systems (HVPSs) on highly resource-constrained platforms. Using these design methods, we have prototyped an HVPS for background subtraction on an Android platform, and conducted experiments using the prototype. The experimental results validate capabilities in the proposed HVPS framework to enable efficient design space exploration for hyperspectral video processing on resource-constrained platforms.

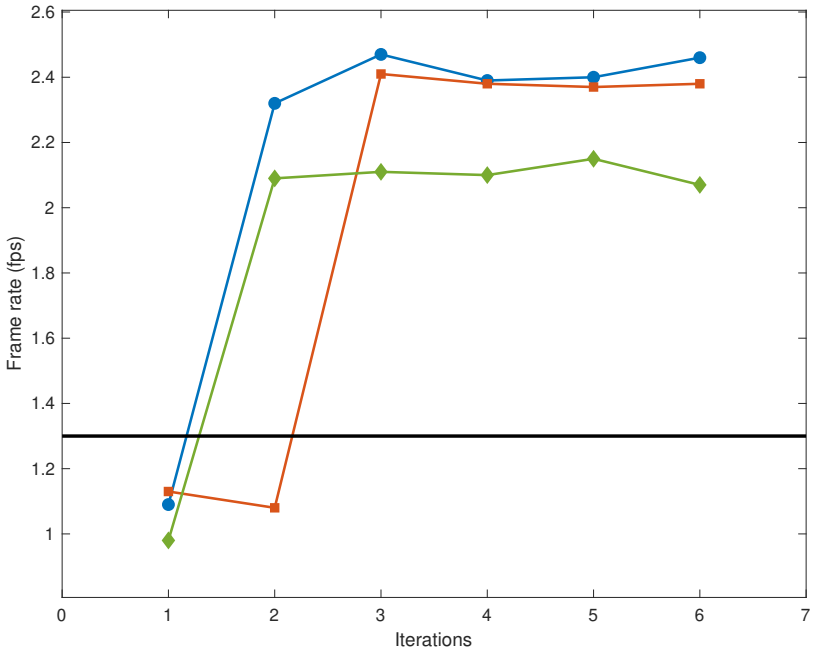

Figure 4.9: Results from applying Algorithm 2 under specific operational constraints: $(C_r, f_M) = (1.3, 0.9)$.

The supported exploration demonstrated in these experiments involves complex factors, including band-subset selection, and multithreading configurations, and their impact on trade-offs between video analysis accuracy and achievable frame rate.

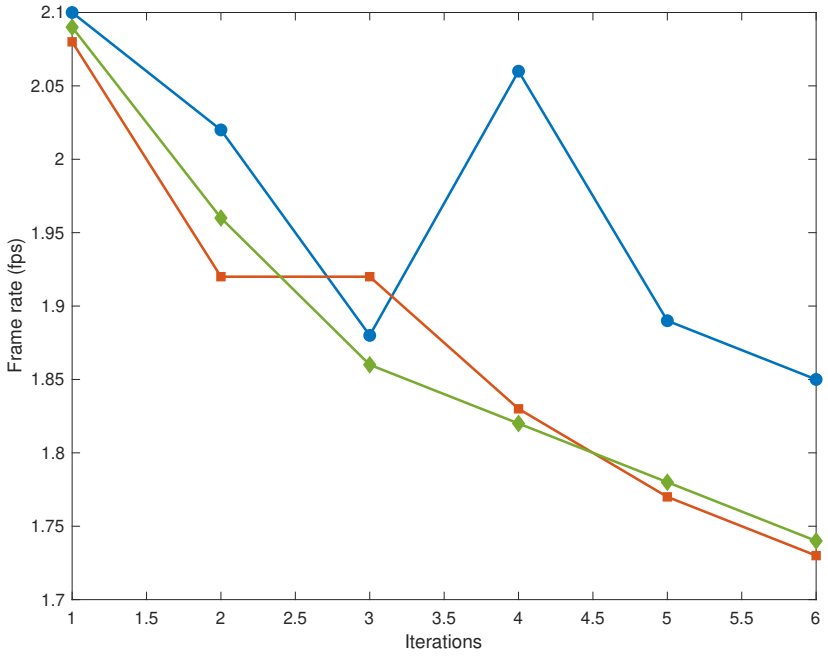

Figure 4.10: Results from applying Algorithm 2 under specific operational constraints: $(C_r, f_M) = (5.0, 0.95)$.

Chapter 5

Design Space Exploration for Wireless-Integrated Factory Automation Systems

In Chapter 2, Chapter 3, and Chapter 4, we developed methods and tools for dynamic data-driven application systems (DDDAS) in the area of multispectral and hyperspectral image and video processing. In this chapter, we examine a different application area that is also ripe for development of DDDAS methods. This is the area of smart factory systems that are equipped with wireless communication capability. We refer to this application domain more concisely as the domain of wireless-integrated factory systems.

The increased use of wireless communication capabilities in manufacturing systems brings challenging new requirements for stable wireless networks that can operate reliably under harsh communication conditions [48]. In this chapter, we develop new methods for modeling, simulating, and analyzing networked embedded systems to aid in system-level optimization of wireless-integrated factory systems.

A major contribution of this chapter is a novel software tool for model-based design space exploration of wireless-integrated factory systems. The tool, called Wireless-Integrated factory System Evaluator (WISE), integrates the design perspectives of physical factory layouts, factory process flows, and wireless communications, including protocol functionality and channel characteristics.

Material in this chapter was published in preliminary form in [49]

5.1 Introduction

Modern factory automation systems are equipped with advanced wireless communications capability. Integration of such capability provides important potential advantages, such as lower cost to deploy and maintain networking capabilities within factories, and the ability to install sensors and monitoring functionality in parts of factories that are not possible to be efficiently instrumented using wired communications (e.g., see [48]).

Along with these potential advantages, integration of wireless communications introduces new challenges and novel constraints in the analysis and design of factory automation systems. A major source of these new challenges and constraints is the complex interaction among the factory layout and configuration. This interaction includes the placement of factory subsystems and their partitioning into nodes of the wireless network (network nodes); the performance of the wireless network that connects network nodes; and overall factory system performance. These factors lead to complex design spaces, which are composed of factory layouts, wireless communication networks, and interactions between them in system configuration and operations. We refer to these design spaces as *wireless-integrated factory system* (*WIFS*) design spaces.

We develop new models and evaluation tools for understanding and experimenting with WIFS design spaces. Since evaluating these design spaces by physically constructing the different layout/networking combinations is in general infeasible, we present a new simulation-based design space exploration tool called *WISE* (*Wireless-Integrated factory System Evaluator*). WISE is designed for model-based simulation of factory automation subsystems that are equipped with wireless communications capability, and rapid

simulation-based evaluation of alternative networked factory system designs.

Here, by *model-based*, we mean that the modeling techniques that underlie the tool are based on formal models of computation rather than on ad-hoc, tool-specific techniques that are difficult to precisely understand or to adapt to other modeling and simulation environments. Model-based design is a useful concept for many areas of cyber-physical systems and signal and information processing (e.g., see [50, 6]). The specific forms of model-based design emphasized in WISE are *dataflow modeling* for factory process-flows, and systematic interfacing of dataflow models with arbitrary network simulators that are based on discrete-event modeling.

The emphasis on dataflow is useful due to the utility of dataflow modeling across the areas of signal processing, control, and machine learning [6], which are all relevant to design and implementation of factory automation systems. This allows not only the high-level process-flow behavior of process networks to be modeled naturally and formally with WISE, but also lower level subsystems of the process-flows. Such a unified, model-based approach across levels of design hierarchy is useful for enhancing design modularity, analysis, and optimization.

Important features of WISE include capabilities for automatically generating (autogenerating) complex lower-level simulation models from compact representations at higher levels of abstraction. WISE also applies a new concept of *cyber-physical flow graphs* (*CPFGs*) as a graph-theoretic model for factory process-flows and other flow-oriented types of cyber-physical systems. We demonstrate WISE through extensive experiments that highlight its utility for exploring complex WIFS design spaces.

5.2 Related Work

A significant body of the existing literature is relevant to modeling and simulation of factory automation systems that are equipped with wireless communication capabilities. Some of these works are based on novel applications of existing simulation frameworks. For example, Liu et al. apply the OMNET++ simulation library to develop an integrated framework for factory process control simulation and wireless network simulation [51]. Marghescu et al. study the simulation of Zigbee-based wireless sensor networks using OPNET to evaluate and optimize the various network parameters [52]. Harding et al. develop a simulator that incorporates mathematical modeling and feedback control by developing an interface between MATLAB and OPNET [53].

Other works emphasize new models or simulation methods. For example, Vogel-Heuser et al. present approaches for modeling real-time requirements and properties of networked automation systems [54]. Schlick discusses advances, such as component-based automation and self-organizing production systems, in cyber-physical systems for factory automation [55]. Kurte et al. introduce a simulator for wireless sensor and actuator networks that allows simulation of heterogeneous systems through a novel interface abstraction for the operation of physical radio hardware [56]. Chaves et al. present a design environment for simulation and testing that is based on a service-oriented software architecture [57].

The novelty of the contribution in this chapter centers on the development and application of WISE to explore complex WIFS design spaces. Compared to related work such as the works summarized above, distinguishing characteristics of WISE include its

model-based architecture, which systematically integrates dataflow-based modeling of factory process-flows with discrete event modeling of wireless communication networks. WISE also provides autogeneration of low-level simulation code from high-level models, and cyber-physical flowgraph modeling, which further enhance the utility of the tool for WIFS design space exploration.

5.3 Design Flow of Cosimulator

Figure 5.1 illustrates the design flow associated with applying WISE for WIFS design space exploration.

Blocks with solid borders in the figure represent designer input, while dashed borders indicated subsystems or autogenerated, lower-level models that are used within the toolset.

As illustrated in Figure 5.1, WISE builds upon a recently-introduced co-simulation tool called Tau Lide Factory Sim (TLFS) [58]. TLFS provides dataflow-based modeling of factory process-flows and systematic integration of the resulting process-flow models with arbitrary discrete event tools for network simulation. As illustrated in Figure 5.1, WISE introduces and integrates with TLFS two new software tools, called the Network Model Generator and the CPFG Generator, and one new intermediate representation (graphical modeling data structure), called the CPFG Model. Additionally, the implementation of TLFS is extended in this work to support details of the CPFG Model. In Figure 5.1, designer input, intermediate representations, and software tools are represented with thin-solid, dashed, and thick-solid borders, respectively.

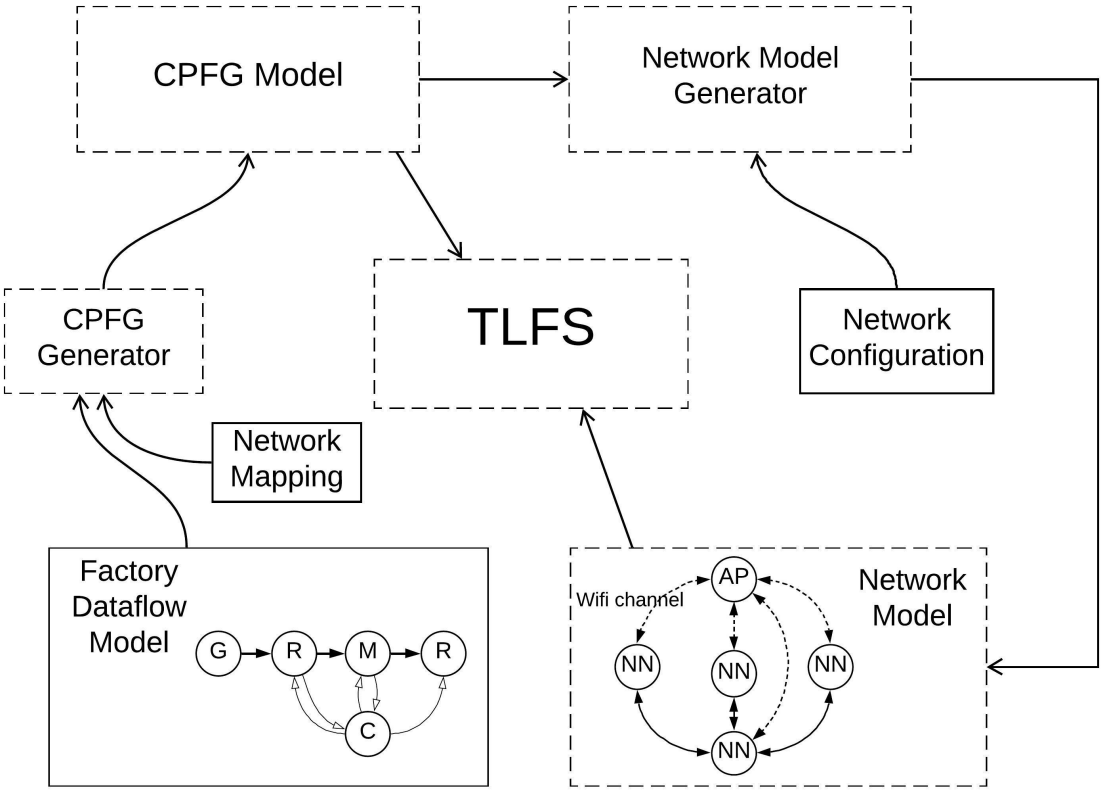

Figure 5.1: An illustration of the new design flow involved in applying WISE for WIFS design space exploration.

5.3.1 Model-Based Architecture

The model-based architectures of WISE and TLFS emphasize dataflow-based modeling of factory process-flows and systematic interfacing between the process-flow models and arbitrary discrete-event simulators for communication architectures. Due to the abstract, model-based architectures of WISE and TLFS, the co-simulation and design space exploration techniques can be adapted readily to different dataflow-based design tools (for the process-flow modeling), and different communication network simulators.

A specific configuration of WISE involves two "plug-in" components for dataflow and communication network simulation. We refer to the two plug-ins as the *dataflow*

simulation plug-in and *network simulation plug-in*, respectively. WISE systematically integrates the given pair of plug-ins into a model-based environment for exploring WIFS design spaces. In our experiments, which we report on in Section 5.5, we utilize two specific dataflow and network simulation tools as plug-ins. These tools are, respectively, (1) the lightweight dataflow environment (LIDE) [59], and (2) the NS3 network simulation tool [60]. However, as described above, the model-based design of the WISE architecture enables retargeting the design space exploration techniques to other tools for dataflow and network simulation. This retargetability is useful because both of these areas for tool development — dataflow and network simulation tool development — are active areas for research and innovation.

5.3.2 Designer Input

The blocks in Figure 5.1 labeled *Factory Dataflow Graph*, *Network Mapping*, and *Network Configuration* refer to simulation model input that is provided by the designer to represent the WIFS that is currently being studied.

The Factory Dataflow Graph models the factory process-flow between factory subsystems as a dataflow graph. The Factory Dataflow Graph is specified in a manner that is independent of the wireless network that is used for communication across distributed subsystems of the factory. Instead, the partitioning of Factory Dataflow Graph components into nodes of a wireless communication network, and the configuration of the network are specified separately. These specifications, represented by the blocks in Figure 5.1 labeled Network Mapping and Network Configuration, are elaborated on in

Section 5.3.4. More details about Factory Dataflow Graph models are discussed in Section 5.3.3.

The separation of concerns among the Factory Dataflow Graph, Network Mapping, and Network Configuration representations improves the efficiency and automation with which the system designer can explore different ways of integrating wireless communication functionality into a given factory process-flow. In particular, the designer does not have to modify the Factory Dataflow Graph when the communication architecture changes; instead, only the relevant parts of the Network Mapping and Network Configuration specifications need to be changed. Then the detailed factory/communication co-simulation model is generated automatically. This separation of concerns and associated autogeneration capability is a major advance of WISE beyond TLFS.

5.3.3 Factory Dataflow Graphs

Formally, a Factory Dataflow Graph is a directed graph $G = (V, E)$, where the vertices (elements of V) represent factory subsystems such as machines, rails, parts generators, and machine/rail controllers. Directed edges (elements of E) in G represent the flow of information or physical entities (such as manufacturing parts) between factory subsystems. In the general terminology of dataflow graphs, the graph vertices are referred to as *actors*. Thus, actors in the Factory Dataflow Graph correspond to factory subsystems.

A dataflow graph executes by repeatedly executing actors that are ready (*enabled*) for execution, where the dataflow model provides a precise formulation for this form of readiness. As actors execute, they exchange packets of information (*tokens*) across the

edges in the graph. These packets can have arbitrary data types associated with them, ranging from primitive types such as integers or floating point values to composite data types that correspond to user-defined objects (in an object-oriented programming sense). In Factory Dataflow Graphs, tokens may, for example, encapsulate information associated with the flow of physical parts, control messages, or instrumentation data.

Execution of a dataflow actor is decomposed into well-defined quanta of execution, called *firings*. Each firing is associated with characterizations of the amount of input data (number of tokens on the input edges) that is consumed by the firing, and the number of output tokens that is produced by the firing. These amounts of input and output data are referred to, respectively, as the consumption and production rates associated with the firing. An actor is said to be enabled for execution when there is a sufficient quantity of tokens buffered on its input edges, and a sufficient amount of empty buffer space available on its output edges to support the firing, as determined by the buffer sizes associated with the edges and the consumption and production rates of the firing.

For more background on the use of dataflow methods to model factory process-flows, we refer the reader to the detailed presentation of TLFS [58]. A notable difference, however, between the Factory Dataflow Graph of WISE and the dataflow graphs employed in TLFS is that Factory Dataflow Graphs do not incorporate any information about the communication architecture. These graphs are therefore simpler for the designer to work with. Furthermore, in conjunction with the separation of concerns described in Section 5.3.2 and the new automated model generation capabilities in WISE, Factory Dataflow Graphs are part of a more efficient approach for WIFS design space exploration. We elaborate on the automation capabilities further in Section 5.4, along with

their utility in supporting design space exploration.

5.3.4 Network Mapping and Configuration

As shown in Figure 5.1, the Network Mapping and Network Configuration are the two designer-provided inputs to specify the communication architecture that is to be integrated with the Factory Dataflow Graph for a given WIFS co-simulation (factory/network co-simulation). The Network Configuration input includes aspects related to factory layout.

Intuitively, the Network Mapping specifies how the given factory process-flow (as represented by the Factory Dataflow Graph) is distributed across different network nodes that communicate through wireless communication. The Network Mapping M for a Factory Dataflow Graph $G = (V, E)$ can therefore be represented as as a partitioning $M = N_1, N_2, \ldots, N_m$ ($m \geq 1$) of V — that is, the N_is are mutually disjoint subsets ($N_i \cap N_j = \emptyset$ for all $i \neq j$), and $N_1 \cup N_2 \cup \ldots \cup N_m = V$. To represent a fully centralized process-flow (with no wireless communication involved), one can simply set $m = 1$ so that the Network Mapping consists of just a single set $N_1 = V$. This type of mapping can be useful, for example, as a baseline to assess basic trade-offs associated with introducing wireless communication into the factory system.

The Network Configuration is another component of designer-provided input to WISE, as illustrated in Figure 5.1. This input includes wireless communication parameter settings, such as the type of protocol and the propagation loss model. The desired Network Configuration settings are provided by the designer in a simple text

file called `net_parameters.txt`. These parameter values are then converted to corresponding settings associated with the network simulation plug-in. To run a family of simulations with varying network parameters, the designer can easily edit the `net_parameters.txt` file or auto-generate a collection of files that can be iterated through for a set of simulation runs.

The parameters that can be specified in the `net_parameters.txt` file include the wireless communication protocol, propagation loss model, antenna transmitter gain, antenna receiver gain, noise figure for the noise signal, and others.

A Network Configuration specification for WISE also includes factory layout settings, which pertain to the spatial layout of factory subsystems, and can have significant impact on communication system performance. Factory layout settings in WISE network configurations are discussed in more detail in Section 5.5.

5.3.5 Lower Level Models and Auto-generation

The input provided by the designer (user of WISE) is at a high-level of abstraction. This facilitates design space exploration because the models are easier to manipulate and reason about. However, to perform complete system simulation, the high level models must be translated into a lower-level form, which includes the simulation input to the network simulation plug-in, and details of interfacing between the dataflow simulation plug-in and the network simulation plug-in. Such details are autogenerated in WISE by the blocks in Figure 5.1 that are labeled Network Model Generator and CPFG Generator, respectively.

The output models that are generated by these two autogeneration subsystems are called the Network Model and CPFG Model, respectively. These two autogenerated models can be simulated together using WISE to achieve WIFS cosimulation between the given factory process-flow model and wireless networking capability that is integrated with the process-flow based on the given Network Mapping.

The structure and format of the generated network model are determined by the network simulation plug-in. As discussed previously, we presently employ NS3 as the network simulation plug-in. Thus, the Network Model Generator frees the designer from having to write NS3 code. The NS3 model is generated automatically from the designer's dataflow-based specification of the factory process-flow together with the Network Mapping and Network Configuration information.

The CPFG model includes special components, called *communication interface actors*, that model sending and communication of data between subsystems in a process-flow model. Communication interface actors model the exchange of data across a wireless communication network, and provide an abstract, modular interface between the dataflow simulation plug-in and the network simulation plug-in [58].

In Section 5.4, we discuss CPFG modeling concepts further, and provide an example of the CPFG model and parameterized network model that are generated from a given Factory Dataflow Graph and Network Mapping.

5.4 Dataflow and Wireless Communication Models

In this section, we introduce details of the CPFG model, and its use as an intermediate representation in WISE. Second, we discuss communication link modeling for wireless channels in WISE. We also present a WIFS modeling example to illustrate the autogeneration of CPFG models and NS3 network models from the higher-level models provided as input to WISE.

5.4.1 Cyber Physical Flow Graph

The CPFG model is a specialized form of dataflow model that is useful for modeling and simulating WIFSs. In addition to its suitability for WIFSs, as we demonstrate in this chapter, the CPFG model is applicable to a broad variety of modeling scenarios in cyber-physical systems. The CPFG model formulated here generalizes and formalizes an integrated, dataflow-based modeling approach for networked factory process-flows that was presented in preliminary form in [58].

Additionally, in this chapter we introduce capabilities in WISE for autogenerating CPFG models from higher level representations. This is an important feature in streamlining the design process so that complex WIFS design spaces can be explored more efficiently, and more accurately.

A CPFG $G_{\text{cp}} = (V_{\text{cp}}, E_{\text{cp}})$ is a dataflow graph whose actors can be partitioned into three subsets V_p, V_c, V_i, which are called the physical, computational, and communication interface actors of G_{cp}, respectively. The computational actors correspond to actors in the usual sense of actors in signal processing oriented dataflow graphs (*dataflow process*

networks) [61]. Such actors represent computational modules that represent discrete units of computation, called *firings*, as described in Section 5.3.3.

Whereas an actor in a conventional signal processing oriented dataflow graph represents a computational module, a physical actor in a CPFG represents a physical subsystem or device, such as a factory machine or rail. A physical actor may encapsulate computational processing within it (e.g., processing that determines when to input a new part into a machine).

What distinguishes physical actors in the CPFG modeling approach is that any given physical actor must consume or produce *physical tokens* on at least one actor input or output, respectively. A physical token in turn models a discrete physical form of output (such as a generated or partially-processed part in a factory) rather than a packet of data, which is what a conventional dataflow token models. If a CPFG edge carries physical tokens, it is is referred to as a *physical edge*, otherwise, we call it a *cyber edge*.

As described in Section 5.3.5, a communication interface actor (i.e., an element of V_i) models the sending or receiving of data across a communication network. In the CPFGs that we are concerned with in this work, the communication interface actors model wireless communication across distributed subsystems within a WIFS.

In WISE, communication interface actors provide a modular, model-based interface between the dataflow simulation plug-in and network simulation plug-in. For example, to retarget a CPFG to a different network simulator, one only has to change the implementations of the communication interface actor types. In WISE, we use only two types of communication actors, called *send interface actors* (*SIAs*) and *receive interface actors* (*RIAs*). Thus, only these two software components need to be retargeted to adapt a CPFG

in WISE to work with a different network simulator.

As their names suggest, SIAs and RIAs model the sending and receiving of data, respectively, between dataflow actors across a communication network. For more background on SIAs and RIAs, we refer the reader to [58].

An example of CPFG modeling and associated use of SIAs and RIAs is presented in Section 5.4.3.

In summary, the CPFG model is distinguished by the partitioning of actors into physical, computational, and communication interface actors, and a dichotomy of edges as physical or cyber edges. A CPFG can apply general dataflow process networks [61] as the underlying dataflow model of computation or any specialized form of signal processing oriented dataflow that is compatible with the modeling requirements of communication interface actors. In this chapter, we employ *core functional dataflow* (*CFDF*) [62] as the underlying dataflow model of computation. Background on CFDF and its utility in modeling factory process-flows is discussed in [58].

5.4.2 Communication Link Modeling

Figure 5.2 illustrates different components of communication link modeling in WISE. Parameters associated with these components are configured by the designer as part of the Network Configuration block in Figure 5.1, as described in Section 5.3.4. Different antenna models are available for reception and transmission; the antenna is modeled as isotropic by default.

For the experiments reported on in this chapter (Section 5.5), signal noise is char-

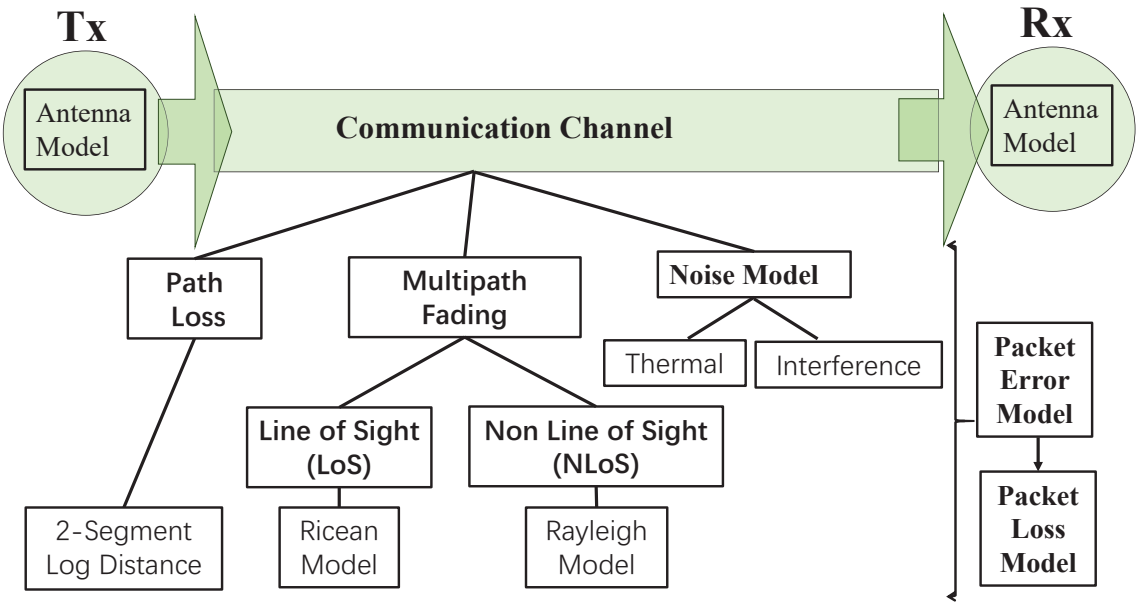

Figure 5.2: Communication link modeling in WISE.

acterized as additive white Gaussian noise (AWGN). For the propagation loss model, a two-segment log distance model is applied. For multipath fading, Ricean and Raleigh models are used. For calculation of packet loss, the error rate is modeled based on a model presented by Miller [63], and subsequently validated by Pei and Henderson [64].

5.4.3 Autogeneration Example

In this section, we illustrate the models and autogeneration capabilities in WISE with a simple WIFS example.

Figure 5.3 illustrates a Factory Dataflow Graph that is used to model a small-scale, pipeline-structured factory process-flow. The actor P represents a *parts generator*, which generates parts that are processed by the factory pipeline. The actors M_1 and M_2 model two machines that process parts, one by one, to add specific features to the parts. Parts are sent to and from each machine through rails, which are represented by the actors R_1

and R_2. The last stage in the pipeline is represented by the actor K. This actor, called the *parts sink*, represents a subsystem that collects and stores the parts after they are fully processed by the pipeline.

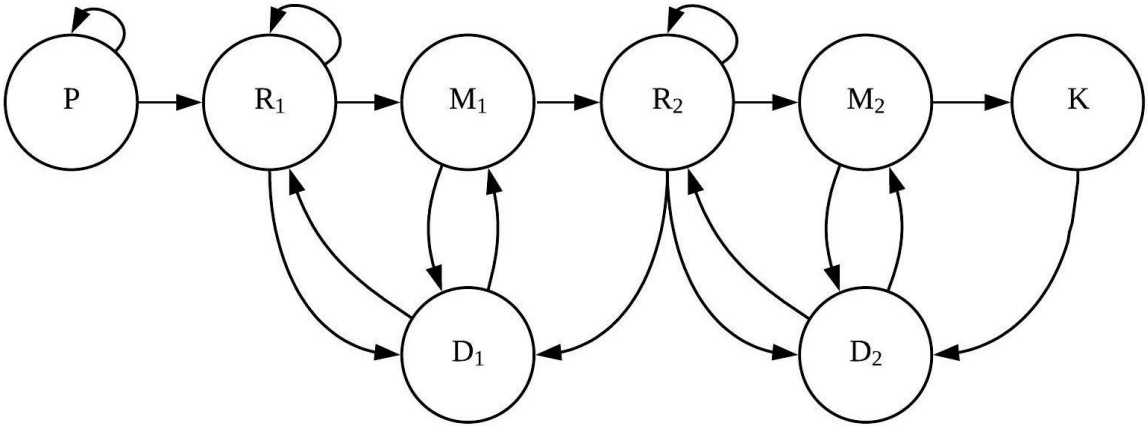

Figure 5.3: An example of a Factory Dataflow Graph.

The actors D_1 and D_2 in Figure 5.3 represent dual-rail, single machine (DRSM) controllers. A DRSM controller is a factory subsystem controller that is designed to interface with a single machine, a rail connected to the input of this machine, and a rail or parts sink that is connected to the machine output. Each DRSM controller sends commands to coordinate the flow of parts through the set of subsystems that it controls. For more details on the operation and modeling of DRSM controllers, we refer the reader to [58].

Figure 5.4 illustrates the CPFG that is autogenerated by WISE for the Factory Dataflow Graph of Figure 5.3 together with an example Network Mapping M. The mapping M involves seven distinct network nodes $N_1, N_2, \ldots, N_7$, and assigns the actors P, R_1, M_1, D_1, R_2, M_2, D_2, K, respectively to network nodes $N_1, N_1, N_2, N_6, N_3, N_4, N_7, N_5$. The solid edges in Figure 5.4 carry physical tokens, while the dashed edges carry conventional

dataflow tokens. In WISE, the determination of whether or not a given CPFG edge is a physical edge can be made automatically from the type of data that is associated with the Factory Dataflow Graph.

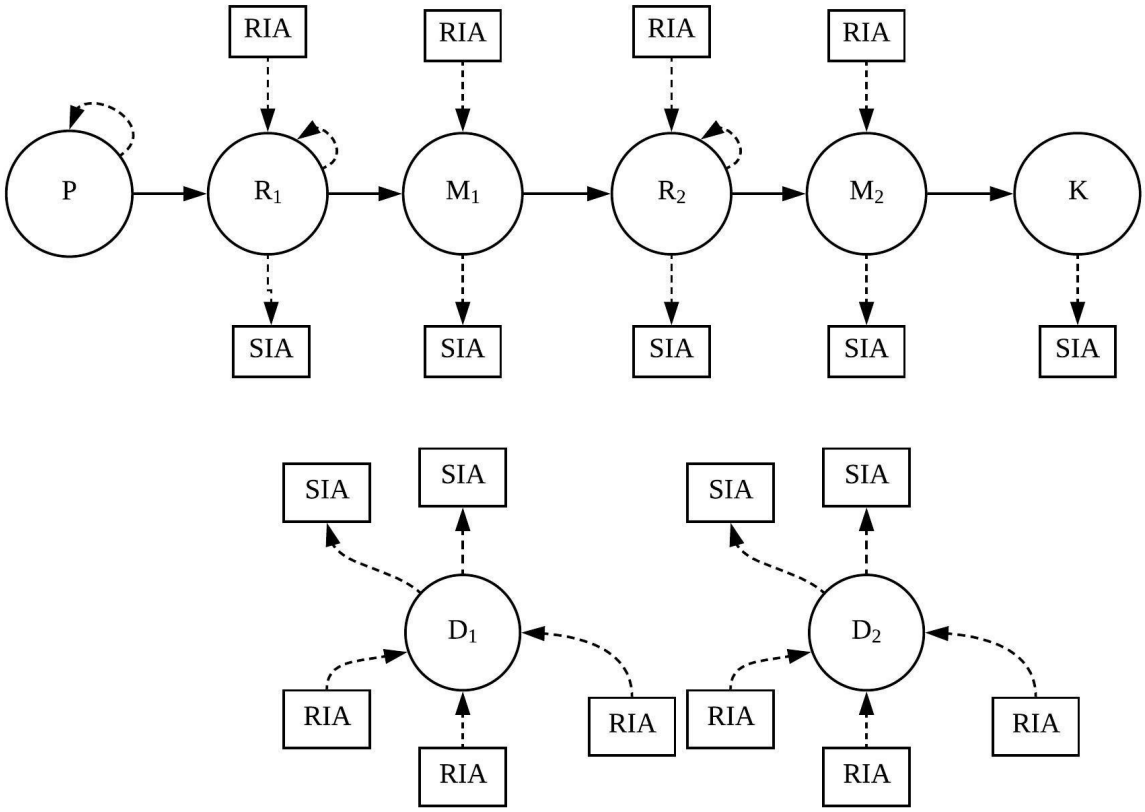

Figure 5.4: Autogenerated CPFG.

The actors labeled SIA and RIA in Figure 5.4 are communication interface actors that are automatically inserted by WISE in the process of autogenerating the CPFG. For each cyber edge whose source and sink actors are mapped to different network nodes, the communication associated with the edge is modeled with a separate (SIA, RIA) pair. For example, R_2 sends data to D_1, as shown by the edge (R_2, D_1) in Figure 5.3, and these actors are mapped by M to distinct network nodes, N_3 and N_6, respectively. Accordingly an SIA S is connected to R_2 to model the sending of data to D_1 through a wireless channel,

and a corresponding RIA is connected to D_1 to model the reception of data that is sent by S.

In WISE, all wireless communication is modeled in the autogenerated CPFGs through SIA-RIA pairs. Thus, all cyber edges in the CPFGs are associated with wired communication. In the current version of WISE, the latency of wired communication is assumed to be negligible compared to the latency of wireless communication and the execution time of machines. However, WIFS can readily be extended to incorporate latency models for wired communication — for example, by adding additional interfaces to the network simulation plug-in or by adding actors in the CPFG that model wired communication delays.

Figure 5.5 illustrates the network model that is autogenerated by WISE for the CPFG in Figure 5.4. This graph shows the structure of the NS3 simulation model that is generated for co-simulation by TLFS with the generated CPFG. Each vertex N_i in Figure 5.5 corresponds to a network node and each edge corresponds to a communication channel. The vertex *Ap* represents a single access point that is associated with the network nodes.

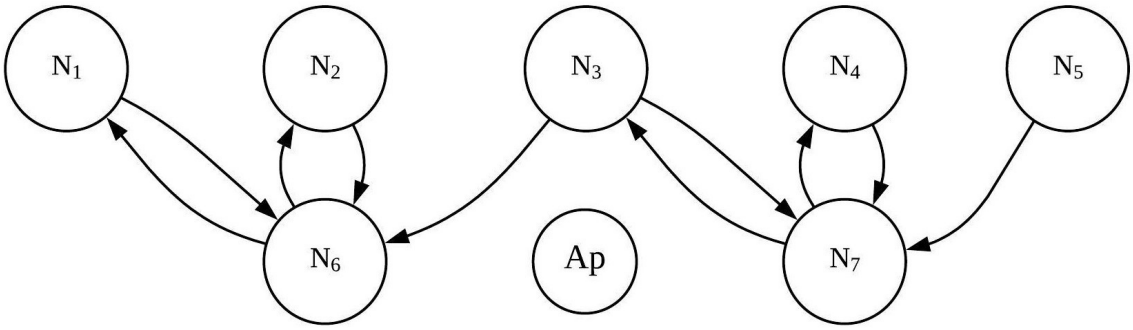

Figure 5.5: The network model that is autogenerated by WISE for the example associated with Figure 5.3 and Figure 5.4.

Even for this simple, small-scale example, we see that the complexity of the CPFG together with the network model is significantly higher than that of the Factory Dataflow Graph, which is the designer's primary interface for working with WISE. This increase in complexity includes larger model sizes (more vertices and edges in the graph), as well as detailed software code that must be provided to correctly specify the lower-level models and ensure their consistency. The new models and autogeneration capabilities in WISE free the designer from the burden of managing this lower level design complexity.

5.5 Results

In this section, we demonstrate the utility of WISE through extensive experiments related to exploration of WIFS design spaces. We apply WISE in experiments with representative factory scenarios. Our experiments are performed using a desktop computer equipped with a 3.10 GHz Intel i5-2400 CPU, 4GB RAM, and the Ubuntu 16.04 LTS operating system.

5.5.1 Factory Layout Parameters

Presently, WISE assumes that a factory layout is in the form of one or more pipelines. Machines that belong to the same pipeline are arranged "horizontally", while different pipelines are arranged "vertically". Factory layout is therefore specified in terms of two distance-related parameters d_x and d_y, which respectively specify uniform (horizontal) spacing between successive subsystems (e.g., machines and rails) of a given pipeline, and uniform (vertical) spacing between successive pipelines in the vertical ar-

rangement. Two additional layout-related parameters, N_p and N_m, specify the number of pipelines, and the number of factory machines within a given pipeline, respectively.

In most experiments in this section, we assume that each pipeline is assumed to have its own access point (AP), with a dedicated wireless channel assigned to each AP. It is assumed that if $N_p > 1$, then all of the dataflow occurs within the individual pipelines; that is, there is no communication across the pipelines. In Section 5.5.7, we experiment with a set of scenarios in which all pipelines share a common access point.

The parameterized model of factory layouts supported in WISE represents a large class of factory systems with which capabilities of WISE can be demonstrated and experimented with. Also, the parameterized structure of the supported class of layouts is useful for demonstrating scalability-related factory performance trends. The extensible architecture of WISE makes it readily generalizable to support larger classes of factory layouts, such as layouts in which different pipelines have different numbers of machines, horizontal or vertical spacing between adjacent subsystems is non-uniform, or the overall layout structure does not necessarily involve horizontally-arranged pipelines. Such generalization is a useful direction for future work in WISE.

Figure 5.6 shows an example of a factory layout of the form currently supported in WISE. In this example, $N_p = 2$, $N_m = 3$, and each pipeline has its own access point. Here, each $M_{i,j}$, $R_{i,j}$, and $D_{i,j}$ represents the jth machine, jth rail, and jth DRSM controller, respectively, for the ith pipeline. Each P_i, K_i, and A_i represents, respectively, the parts generator, parts sink, and access point for the ith pipeline.

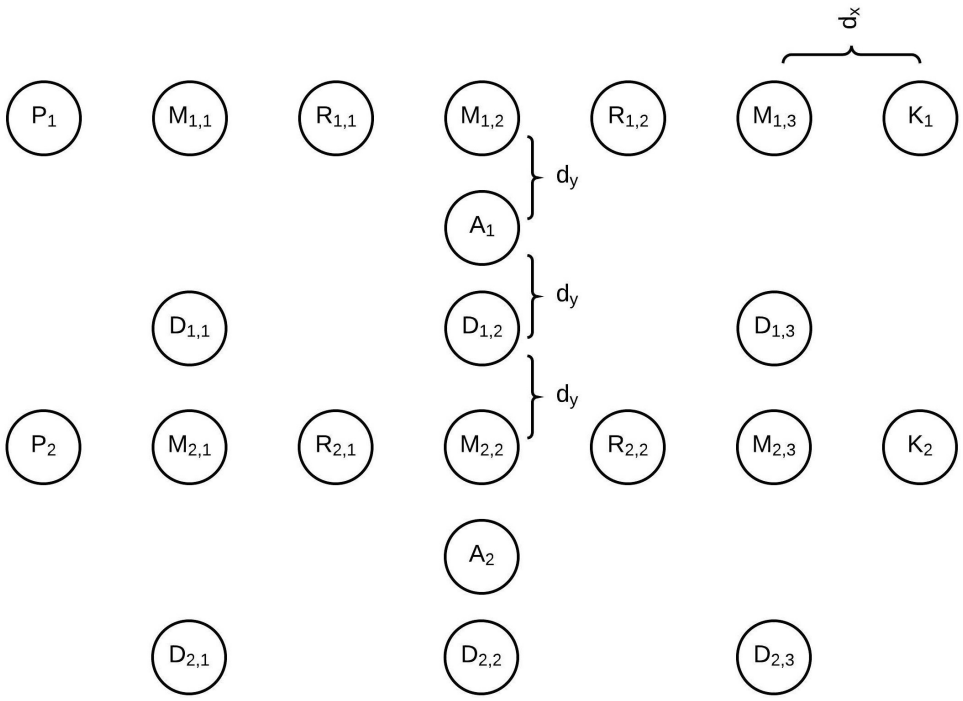

Figure 5.6: Factory layout example.

5.5.2 Experiment Parameters

For each type of factory configuration simulated, we ran 50 WISE simulations independently and averaged the results. In each experiment, the simulation involved the production of 100 parts by each parts generator in each of the N_p pipelines, and the complete processing by the machines in each pipeline of the parts generated by the corresponding parts generator. The working time of each machine (the time required to process a given part) was determined randomly by the simulator using a designer-specified *mean working time* parameter μ. More specifically, the time for a given machine to process a given part was determined from a uniform distribution on $[0.9\mu, 1.1\mu]$. Each simulation terminated after the N_p parts sinks had each received 100 fully-processed parts.

Table 5.1 summarizes the other key simulation parameters used in our experiments. A given data point in the experiments is derived by executing a simulation with the same

Table 5.1: Simulation parameters.

Parameters	Values (Options)
Parts Generated Per Pipeline N_j	100
Number of Simulation Iterations N_s	10
Machine Processing Time t_m	10 sec
Rail Transfer Time t_r	4 sec
Part Generation Interval t_i	10 sec
Channel Frequency	2.4 GHz
Large Scale Path Loss Model	Log-distance
Decay Exponent α	3
Distance Reference d_0	1 m
Loss at Reference L_0	46.6777 dB

settings N_s times, and averaging the results over the N_s executions. Each such simulation involves N_j generated parts for each Parts Generator actor in the factory dataflow graph. The simulation completes when all of the generated parts are fully processed in their respective pipelines. Since there is one Parts Generator actor per pipeline, this means that each simulation involves processing a total of $(N_p \times N_j)$ parts.

The values of t_m and t_r give, respectively, the estimated execution time values used in the simulation models for a machine to process a part, and for a rail r to move a part from one end of r to the other end. Similarly, t_i is the estimated time required to generate a new part after the previous part has been generated. The values of t_m, t_r, and t_i are used in the execution time estimation functions (θs) for the relevant actors.

The parameters α, d_0, and L_0 in Table 5.1 are related to the simulation of propagation path loss. In our simulations, we apply features in NS-3 for using the log-distance path loss model to estimate signal loss in communication channels. The log-distance model is often used to estimate path loss within buildings. In this model, the power loss at the receiver side when transmitting over a distance d is calculated by

$$L = L_0 + 10\alpha log_{10}(\frac{d}{d_0}) + Z, \tag{5.1}$$

where L_0 is the path loss at the reference distance, d_0 is the reference distance, α is the decay exponent, and Z is the log-normal shadowing.

The wireless communication protocol employed in all of the experiments reported on in this section is IEEE 802.11b. Since the protocol can be conveniently configured as part of the Network Configuration input to WISE, the experiments discussed here can be easily adapted to other protocols of interest.

WISE measures the communication delay associated with a packet P as $t_r(P) - t_s(P)$, where $t_s(P)$ is the time when P is sent by the corresponding SIA (see Section 5.4.1), and t_r is the time when P is received by the corresponding RIA. The average communication delay for a given simulation experiment is computed by averaging the difference $t_r(P) - t_s(P)$ over all communication packets.

5.5.3 Variation of Communication Delay with N_p

Figure 5.7 shows how the average communication delay varies with the number of pipelines N_p. In this experiment, the Wi-Fi manager is configured to be the CARA (Collision-Aware Rate Adaptation) algorithm; $d_x = 10$ meters (m); $d_y = 10$ m; and $N_m = 3$.

As shown in Figure 5.7, the results for each N_p value are summarized in the form of a box plot. The endpoints of the vertical line segment for each plot extend from the minimum observed value to the maximum observed value. The three horizontal lines

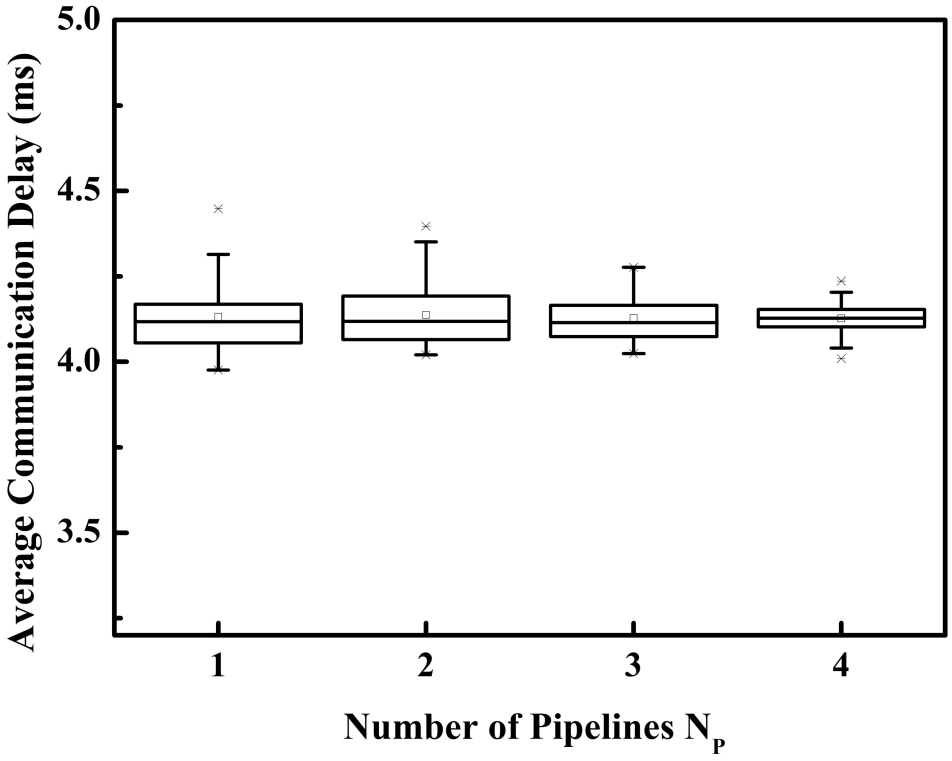

Figure 5.7: Variation in average communication delay with N_p.

in each large box represent, from top to bottom, the 75th percentile, median, and 25th percentile of the corresponding set of 50 measurements. The small box inside each large box represents the mean value.

As shown in Figure 5.7, the number of pipelines N_p has little influence on average communication delay for the class of factory systems considered in this experiment. This is because we allocate an independent access point for each pipeline and there is no communication between different pipelines.

5.5.4 Variation of Communication Delay with Both N_m and N_p

Figure 5.8 shows results from an experiment where we have varied both the number of machines N_m and number of pipelines N_p. The variation is performed such that $N_m =$

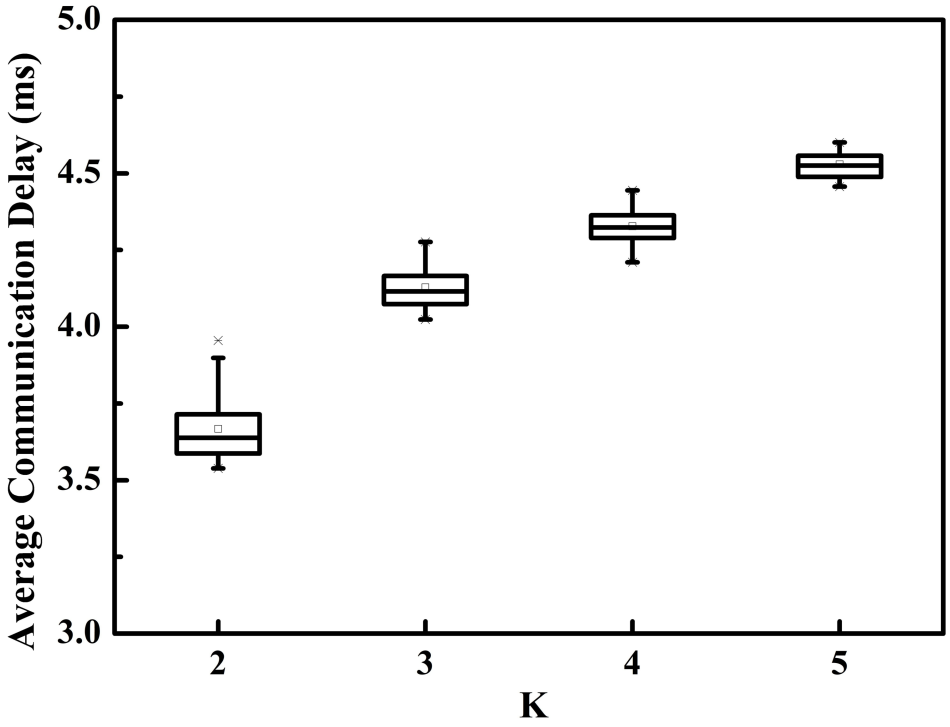

Figure 5.8: Variation in average communication delay with $N_p = N_m = K$.

N_p. This allows us to visualize the effects of layout-complexity scaling in terms of a single parameter K, which is defined as the common value of N_p and N_m. The Wi-Fi manager algorithm is configured to be CARA as in Section 5.5.3, and all other experiment parameters are as specified in Section 5.5.2. The distance parameters are again configured as $d_x = 10$ m and $d_y = 10$ m.

As shown in Figure 5.8, the average communication delay increases with larger K. This trend is largely due to two factors. First, the length of each pipeline increases with K, and correspondingly, the average distance from communication transceivers to the access point in each pipeline increases with K. Second, longer pipelines with more subsystems introduce more contention in the access points. The simulation results in this experiment provide specific insights on how communication delays vary and corresponding real-time performance issues are affected as a function of K, while other factory layout parameters

are fixed.

5.5.5 Varying the Distance Parameters d_x and d_y

Figure 5.9 presents a histogram of average communication delay, as determined by WISE simulation, with varying values of the distance parameters d_x and d_y. Each bar of the histogram is determined by averaging across 50 simulation runs. In this experiment, $N_p = 1$ and $N_m = 3$. The Wi-Fi manager algorithm is again configured to be CARA, and all other experiment parameters are as summarized in Section 5.5.2.

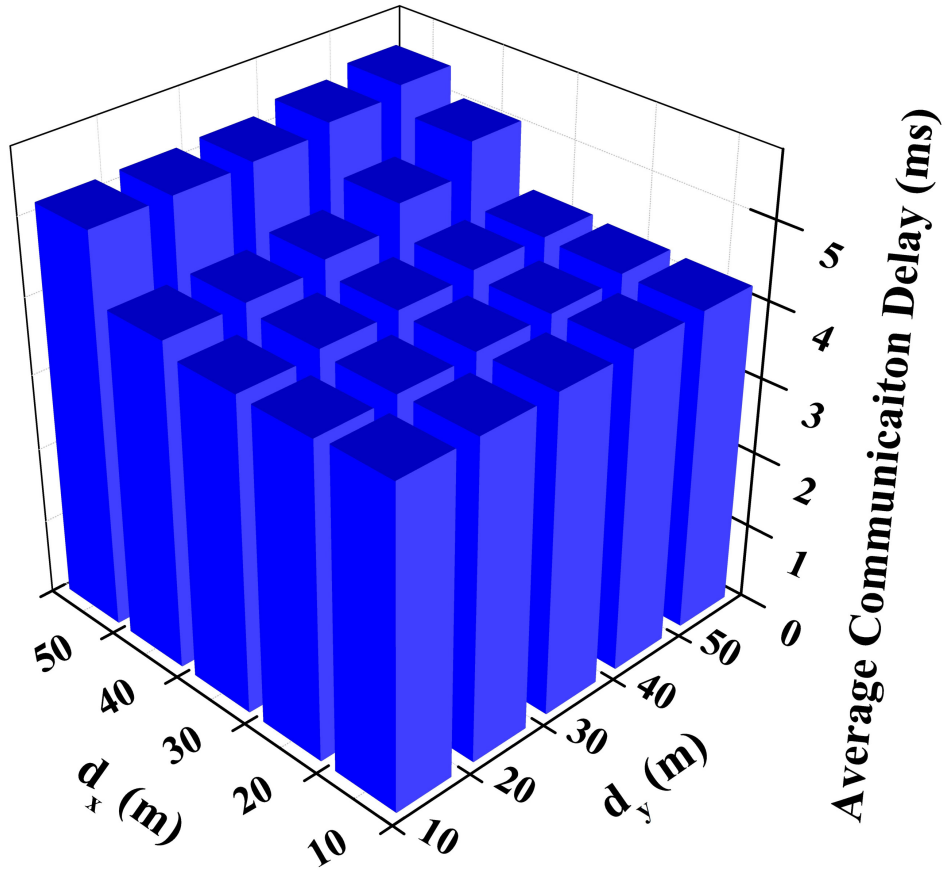

Figure 5.9: Histogram of average communication delay with varying d_x, d_y.

This experiment shows a gradual trend toward increasing communication delay for $d_x, d_y \in \{10\text{ m}, 20\text{ m}, 30\text{ m}\}$, while for values of $d_x, d_y \in \{40\text{ m}, 50\text{ m}\}$, we see steeper

rates of increase. We expect that this accelerated increase arises due to nonlinear effects such as the way in which the Wi-Fi manager downgrades the data rate when a significant frequency of communication failures is encountered.

When $d_x = 60$ m, the simulation does not progress for any value of $d_y \in \{10\text{ m}, 20\text{ m}, \ldots, 50\text{ m}\}$. In other words, wireless communication throughout the simulated factory network fails, and therefore, the factory is not capable of processing parts. Such results are useful in exploring the limits to which factory subsystems can be separated while preserving system functionality.

5.5.6 Varying the Wi-Fi Manager Algorithm

Figure 5.10 shows changes in the average communication delay with changes in the Wi-Fi manager algorithm and number of machines N_m. For these experiments, $N_p = 1$, and $d_x = d_y = 10$ m. All other parameters are set as summarized in Section 5.5.2. The Wi-Fi manager algorithms investigated in this experiment are: Collision-Aware Rate Adaptation (CARA), Adaptive Auto Rate Fallback (AARF), collision detection for adaptive auto rate fallback (AARFCD), and Adaptive Multi Rate Retry (AMRR) [65].

5.5.7 Shared Access Point across Pipelines

In this section, we revisit the experimental setup of Section 5.5.3 with one change: we use a single, shared access point across all pipelines instead of a separate access point for each pipeline. Thus, the total number of access points in a given factory layout is reduced from N_p to 1.

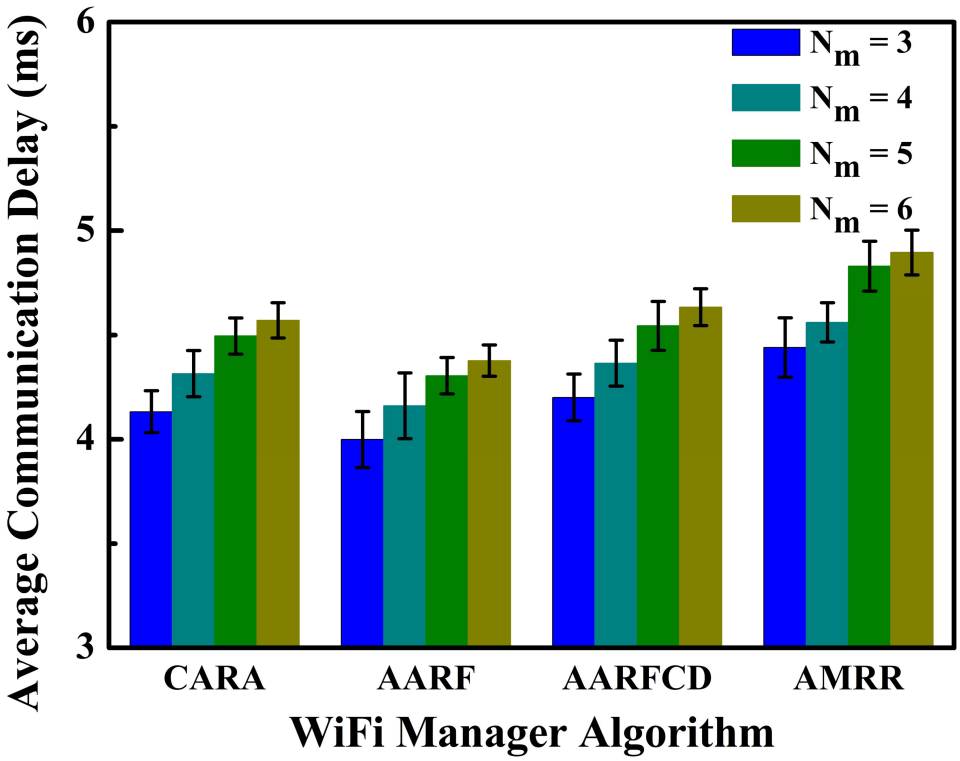

Figure 5.10: Variation in average communication delay with the Wi-Fi manager algorithm and number of machines N_m.

As in Section 5.5.3, the Wi-Fi manager algorithm is configured to be CARA; $d_x = d_y = 10$ m; and $N_m = 3$. All other experiment settings are as described in Section 5.5.2.

Figure 5.11 shows how the average communication delay varies with variation in the number of pipelines N_p under a single, shared access point configuration. We see in Figure 5.11 a clear trend toward increasing average communication delay with increasing N_p. We anticipate that this is because with a single access point across all pipelines, increasing N_p results in more contention in the access point. Moreover, since d_x and d_y are fixed in this experiment, the average distance between communication transceivers and the access point increases with increasing N_p (see Figure 5.6).

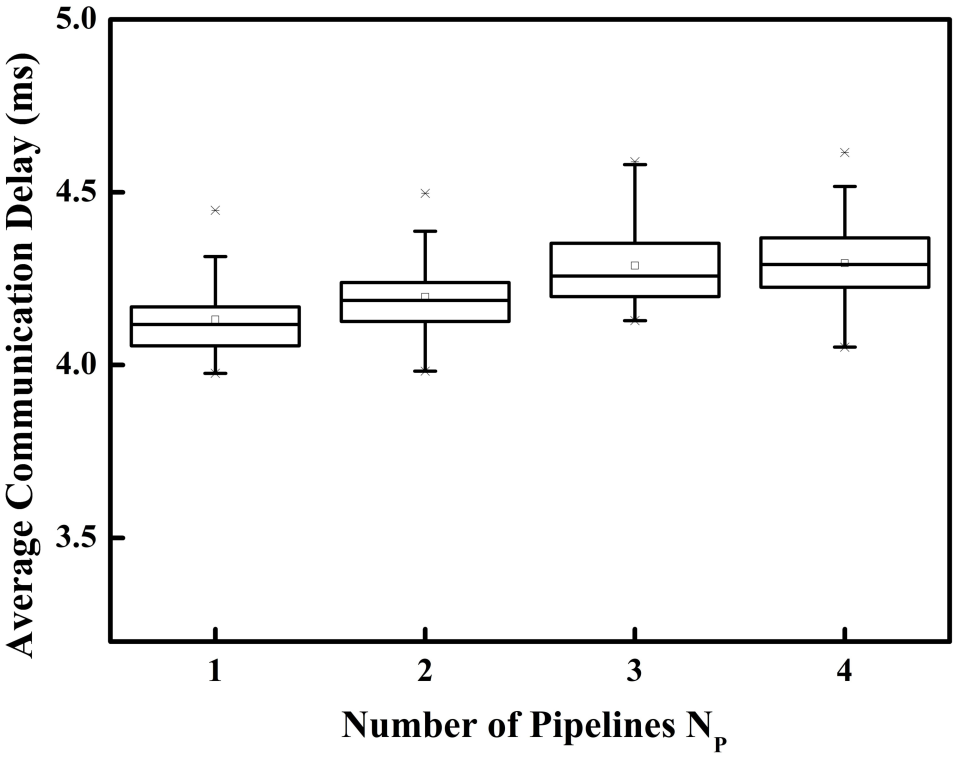

Figure 5.11: Variation in average communication delay with N_p when a single, shared access point is used across all pipelines.

5.6 Additional Experiments

In Section 5.5, we presented the main experimental results from our evaluation of WISE. In this section, we present additional experiments that provide further demonstration of the capabilities of WISE.

Figure 5.12 shows the measured relationship between the average communication delay and the distance between adjacent network nodes for a specific factory configuration. The results are aggregated over 100 experiment repetitions for the same distance. The communication protocol used in the experiments is IEEE802.11b. The factory model is that of a single pipeline with 3 machines and 100 products to be processed. The WiFi Manager is CaraWifiManager, and the average machine working time is 10 seconds.

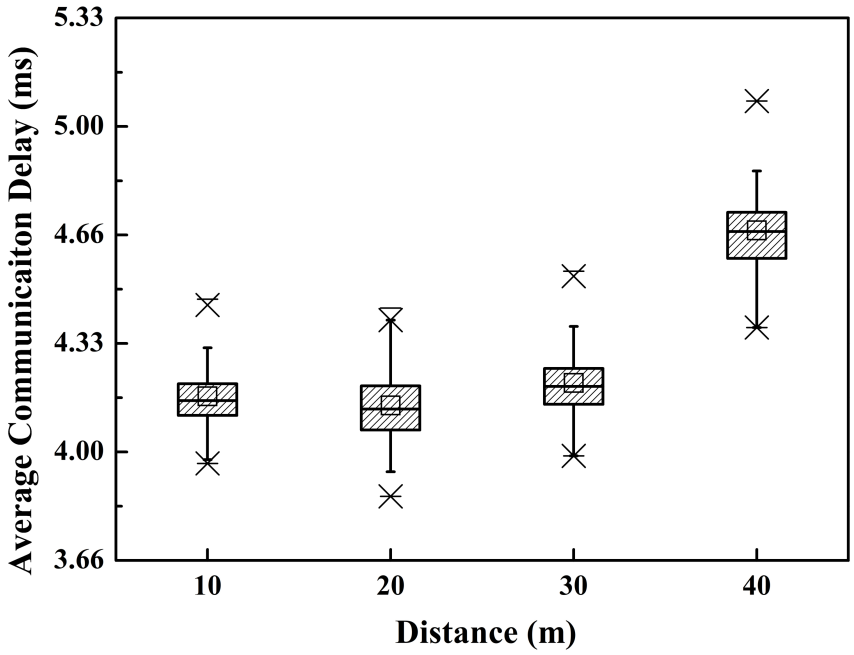

Figure 5.12: Measured relationship between the average communication delay and the distance between adjacent network nodes for a specific factory configuration.

Figure 5.13 shows the relationship between the average packet retransmission rate and the distance between adjacent nodes under the same experimental setup as the experiment associated with Figure 5.12. As shown in Figure 5.13, both the average communication delay and packet retransmission rate increase with increases in the inter-node distance. We anticipate that this trend results from the signal strength decay caused by increased distances.

Keeping the data rate constant while increasing the distance can cause major degradation of communication performance. We conduct experiments to explore this trend and its dependence on the WiFi manger (rate control algorithm) using WISE. Figure 5.14 illustrates the measured relationship between the average communication delay and the selected rate control algorithm for a relatively large inter-node spacing of 20 meters. The

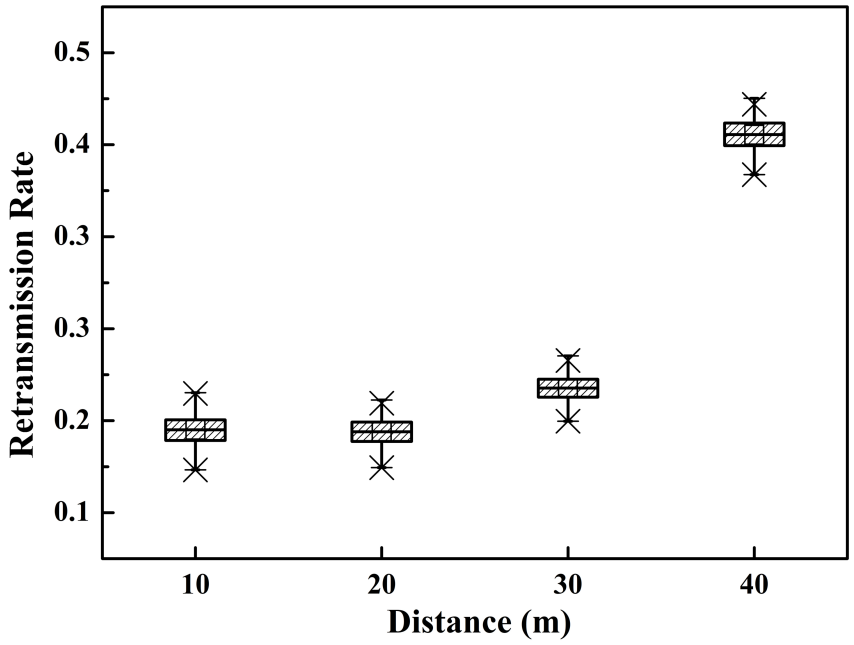

Figure 5.13: Measured relationship between the average packet retransmission rate and the distance between adjacent nodes.

selected rate control algorithms in this experiment are cara, aarf, aarfcd, amrr, and aparf, as shown in the horizontal axis labels of Figure 5.14. The results are aggregated over 100 experiment iterations for each rate control algorithm. The protocol employed in these experiments is IEEE802.11a. The factory model is again a pipeline with 3 machines and 100 products to be processed. The average machine working time is set to be 10 seconds.

Figure 5.15 shows the relationship between the average packet retransmission rate and the rate control manager algorithm under the same experimental setup as that associated with Figure 5.14 . The results in Figure 5.14 and Figure 5.15 show the potential for significant variation in communication performance depending on the rate control algorithm. Quantitative insight on such variation, as provided by WISE, can be useful to designers in selecting the WiFi manager algorithm for a given network setup or configu-

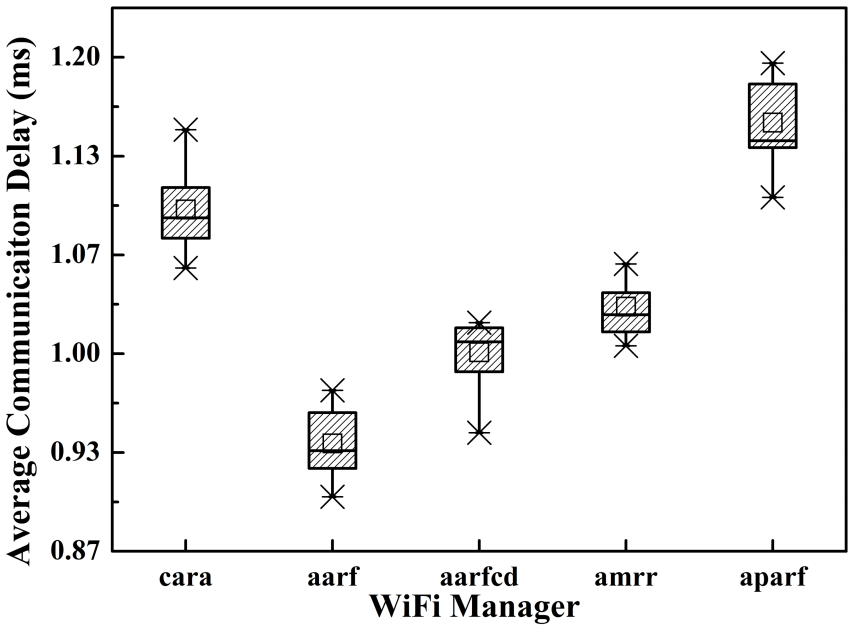

Figure 5.14: Measured relationship between the average communication delay and the selected rate control algorithm for an inter-node spacing of 20 meters.

ration.

5.7 Summary

In this chapter, we have developed new models and computer-aided design tools that help in understanding and experimenting with complex, wireless-integrated factory system (WIFS) design spaces. The models and tools developed in this chapter build upon a recently-introduced co-simulation tool called Tau Lide Factory Sim (TLFS) [58]. The developed tools are integrated into a novel framework called called Wireless-Integrated factory System Evaluator (WISE), which integrates the design perspectives of physical factory layouts, factory process flows, and wireless communications, including protocol functionality and channel characteristics. Important features of WISE include capabili-

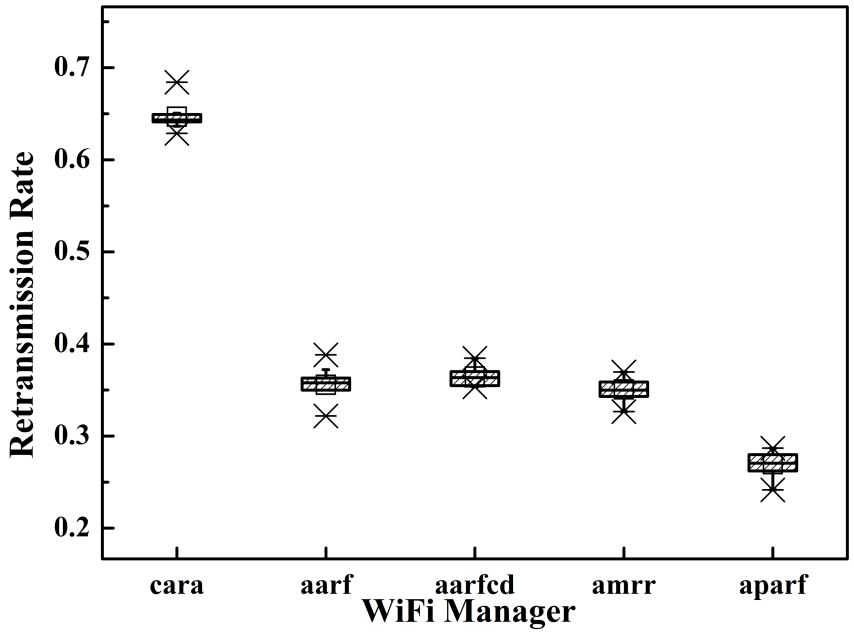

Figure 5.15: Relationship between the average packet retransmission rate and the rate control manager algorithm.

ties for automatically generating (autogenerating) complex lower-level simulation models from compact representations at higher levels of abstraction. Through extensive experiments, we have demonstrated the utility of WISE in exposing insights and performance trends involving multidimensional interactions among factory layout and communication system parameters.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis and discuss directions for future work that are motivated by the thesis.

6.1 Conclusions

In this thesis, we have developed new methods for modeling, simulating, and optimizing dynamic data-driven application systems (DDDAS). We have developed new methods for integrated band subset selection and video processing parameter optimization in LDspectral, which is a software tool for model-based system design of data-driven systems for multispectral and hyperspectral video processing. LDspectral is developed for optimization in the context of novel video processing design spaces introduced by multispectral and hyperspectral image acquisition techniques. The methods developed in this thesis enable experimentation with and optimization of data-driven video processing adaptation for multispectral and hyperspectral video analytics. The methods are demonstrated in terms of accuracy and execution time using relevant multispectral and hyperspectral applications and datasets. The developed LDspectral system provides a prototype software tool for dynamically reconfigurable multispectral and hyperspectral video processing applications.

We have also developed a software tool for applying DDDAS methods to the area of

smart factory systems that are equipped with wireless communication capability. We refer to this application area as the domain of wireless-integrated factory systems. The software tool that we have developed provides novel capabilities for model-based design space exploration of wireless-integrated factory systems. The tool, called Wireless-Integrated factory System Evaluator (WISE), integrates the design perspectives of physical factory layouts, factory process flows, and wireless communications, including protocol functionality and channel characteristics. WISE applies a new concept of cyber-physical flow graphs (CPFGs) as a graph-theoretic model for factory process-flows and other flow-oriented types of cyber-physical systems. We have demonstrated WISE through extensive experiments that highlight its utility for exploring complex design spaces associated with wireless-integrated factory systems.

6.2 Future Work

6.2.1 Generalization of LDspectral

There are a number of useful directions for generalizing the models and methods developed in LDspectral. Such generalization will help to establish design methodologies and system architectures that are applicable across broader classes of image and video processing systems.

First, it useful to integrate capabilities in LDspectral for handling images that are not aligned. The current version of LDspectral is developed for input streams in which the multispectral images are well-aligned across the different bands. The band subset processing subsystem in LDspectral can readily be augmented to incorporate image reg-

istration, which would be useful to extend the capabilities of the overall system to handle images that are not aligned. Such extension together with the integrated optimization of associated operational trade-offs is an interesting direction for future work.

A second interesting direction for investigation is development of extensions to incorporate energy consumption systematically into the design evaluation space considered in LDspectral. Such investigation may include methods to jointly optimize video processing accuracy and energy efficiency subject to constraints on real-time performance. Such investigation may also involve characterizing the energy efficiency of alternative video processing configurations in LDspectral, and integrating the resulting characterizations as part of the design optimization processes within the framework.

Third, extensions to LDspectral may be investigated for design and implementation of networked video processing systems. Such extensions will consider constraints and characteristics of communication channels and network protocols in the design and implementation process. Work in this direction is motivated, for example, by networked video processing applications that monitor a region from multiple viewpoints, and perform communication and distributed analytics to extract knowledge from the captured images. Useful topics include investigation of systematic methods to take into account characteristics of communication channels and processing techniques for multiview image fusion within the framework of LDspectral.

6.2.2 Multiple Objective Optimization Using WIFS

Useful directions for future work involving WIFS include developing optimization strategies, such as those based on randomized search (e.g., evolutionary algorithms or particle swarm optimization), for strategically iterating through families of simulations using our new WIFS-oriented models and tools. Multiobjective optimization strategies could be investigated in conjunction with such an effort to systematically derive Pareto fronts in multi-dimensional design spaces associated with wireless-integrated factory systems. Examples of strategies for multiobjective optimization that may be applied in such an investigation include the strength Pareto evolutionary algorithm (SPEA) and SPEA-2 [66, 67]; multiobjective evolutionary algorithm based on decomposition (MOEA/D) [68]; and simplified multi-objective particle swarm optimization (SMPSO) [69].

Bibliography

[1] Y. Benezeth, D. Sidibé, and J. B. Thomas, "Background subtraction with multispectral video sequences," in *Proceedings of the Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision*, 2014.

[2] F. Darema, "Grid computing and beyond: The context of dynamic data driven applications systems," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 692–697, 2005.

[3] E. P. Blasch, S. Ravela, and A. J. Aved, Eds., *Handbook of Dynamic Data Driven Applications Systems*, Springer, 2018.

[4] L. Ferrato, "Comparing hyperspectral and multispectral imagery for land classification of the lower don river, toronto," M.S. thesis, Ryerson University, 2012.

[5] D. Rufenacht, C. Fredembach, and S. Susstrunk, "Automatic and accurate shadow detection using near-infrared information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1672–1678, 2014.

[6] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, Springer, third edition, 2019.

[7] E. Blasch, "Enhanced air operations using JView for an air-ground fused situation awareness UDOP," in *Proceedings of the IEEE/AIAA Digital Avionics Systems Conference*, 2013, pp. 5A5–1–5A5–11.

[8] N. Nguyen, M. Maifi, and H. Khan, "Context aware data acquisition framework for dynamic data driven applications systems (DDDAS)," in *Proceedings of the Military Communications Conference*, 2013, pp. 334–341.

[9] W. Silva, E. W. Frew, and W. Shaw-Cortez, "Implementing path planning and guidance layers for dynamic soaring and persistence missions," in *Proceedings of the International Conference on Unmanned Aircraft Systems*, 2015, pp. 92–101.

[10] C. Shen, W. Plishker, H. Wu, and S. S. Bhattacharyya, "A lightweight dataflow approach for design and implementation of SDR systems," in *Proceedings of the Wireless Innovation Conference and Product Exposition*, Washington DC, USA, November 2010, pp. 640–645.

[11] C. Shen, W. Plishker, and S. S. Bhattacharyya, "Dataflow-based design and implementation of image processing applications," in *Multimedia Image and Video Processing*, L. Guan, Y. He, and S.-Y. Kung, Eds., pp. 609–629. CRC Press, second edition, 2012, Chapter 24.

[12] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV," *Communications of the ACM*, vol. 55, no. 6, 2012.

[13] H. Li, K. Sudusinghe, Y. Liu, J. Yoon, M. van der Schaar, E. Blasch, and S. S. Bhattacharyya, "Dynamic, data-driven processing of multispectral video streams," *IEEE Aerospace & Electronic Systems Magazine*, vol. 32, no. 7, pp. 50–57, 2017.

[14] L.-J. Ferrato and K. W. Forsythe, "Comparing hyperspectral and multispectral imagery for land classification of the lower Don River, Toronto," *Journal of Geography and Geology*, vol. 5, no. 1, pp. 92–107, 2013.

[15] K. P. Ramesh, S. Gupta, and E. P. Blasch, "Image fusion experiment for information content," in *Proceedings of the International Conference on Information Fusion*, 2007, pp. 1–8.

[16] J. Patrick, R. Brant, and E. Blasch, "Hyperspectral imagery throughput and fusion evaluation over compression and interpolation," in *Proceedings of the International Conference on Information Fusion*, 2008, pp. 1–8.

[17] Z. Liu, E. Blasch, Z. Xue, J. Zhao, R. Laganiere, and W. Wu, "Objective assessment of multiresolution image fusion algorithms for context enhancement in night vision: A comparative study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 94–109, 2012.

[18] B. Liu, Y. Chen, A. Hadiks, E. Blasch, A. Aved, D. Shen, and G. Chen, "Information fusion in a cloud computing era: A systems-level perspective," *IEEE Aerospace & Electronic Systems Magazine*, vol. 29, no. 10, pp. 16–24, 2014.

[19] D. E. Nirmala and V. Vaidehi, "Comparison of pixel-level and feature level image fusion methods," in *Proceedings of the International Conference on Computing for Sustainable Global Development*, 2015, pp. 743–748.

[20] Y. Zhou, A. Mayyas, A. Qattawi, and M. Omar, "Feature-level and pixel-level fusion routines when coupled to infrared night-vision tracking scheme," *Infrared Physics & Technology*, vol. 53, pp. 43–49, 2010.

[21] C. D. Demars, M. C. Roggemann, and T. C. Havens, "Multispectral detection and tracking of multiple moving targets in cluttered urban environments," *Optical Engineering*, vol. 54, no. 12, pp. 123106–1–123106–14, 2015.

[22] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proceedings of the International Conference on Pattern Recognition*, 2004, pp. 28–31.

[23] Z. Zivkovic and F. van der Heijdenb, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern Recognition Letters*, vol. 27, no. 7, pp. 773–780, 2006.

[24] H. Li, Y. Liu, K. Sudusinghe, J. Yoon, E. Blasch, M. van der Schaar, and S. Bhattacharyya, "Design of a dynamic data-driven system for multispectral video processing," in *Handbook of Dynamic Data Driven Applications Systems*, pp. 529–545. Springer, 2018.

[25] V. Bhateja, A. Srivastava, A. Moin, and A. Lay-Ekuakille, "NSCT based multispectral medical image fusion model," in *Proceedings of the IEEE International Symposium on Medical Measurements and Applications*, 2016, pp. 1–5.

[26] Q. Wei, J. Bioucas-Dias, N. Dobigeon, and J.-Y. Tourneret, "Hyperspectral and multispectral image fusion based on a sparse representation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 7, pp. 3658–3668, 2015.

[27] Z. Chen, H. Pu, B. Wang, and G.-M. Jiang, "Fusion of hyperspectral and multispectral images: A novel framework based on generalization of pan-sharpening methods," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 8, pp. 1418–1422, 2014.

[28] B. Uzkent, M. J. Hoffman, and A. Vodacek, "Integrating hyperspectral likelihoods in a multidimensional assignment algorithm for aerial vehicle tracking," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 9, pp. 4325–4333, 2016.

[29] A. Sobral, S. Javed, S. Ki Jung, T. Bouwmans, and E. Zahzah, "Online stochastic tensor decomposition for background subtraction in multispectral video sequences," in *Proceedings of the International Conference on Computer Vision Workshop*, 2015, pp. 946–953.

[30] B. C. S. Reddy, P. Shah, S. N. Merchant, and U. B. Desai, "Visualization of multispectral video with moving background based on background extraction and fusion," in *Proceedings of the International Conference on Information, Communications and Signal Processing*, 2011, pp. 1–5.

[31] A. J. Aved, E. P. Blasch, and J. Peng, "Regularized difference criterion for computing discriminants for dimensionality reduction," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 5, pp. 2372–2384, 2017.

[32] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur, "BPDF: A statically analyzable dataflow model with integer and Boolean parameters," in *Proceedings of the International Workshop on Embedded Software*, 2013, pp. 1–10.

[33] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–2421, October 2001.

[34] Z. Liu, E. Blasch, and V. John, "Statistical comparison of image fusion algorithms: Recommendations," *Information Fusion*, vol. 36, pp. 251–260, 2017.

[35] K. Sudusinghe, S. Won, M. van der Schaar, and S. S. Bhattacharyya, "A novel framework for design and implementation of adaptive stream mining systems," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, San Jose, California, July 2013, pp. 1–6.

[36] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[37] H. Li, L. Pan, Z. Li, M. J. Hoffman, A. Vodacek, and S. S. Bhattacharyya, "Design methods for hyperspectral video processing on resource-constrained platforms," in *Proceedings of the Hyperspectral Imaging & Applications Conference*, Coventry, UK, October 2018, 2 pages in online proceedings.

[38] H. Li, L. Pan, E. J. Lee, Z. Li, M. J. Hoffman, , A. Vodacek, and S. Bhattacharyya, "Hyperspectral video processing on resource-constrained platforms," in *10th IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE, 2019, pp. 1–5.

[39] R. J. Birk and T. B. McCord, "Airborne hyperspectral sensor systems," *IEEE Aerospace & Electronic Systems Magazine*, vol. 9, no. 10, pp. 26–33, 1994.

[40] S. Matteoli, M. Diani, and G. Corsini, "A tutorial overview of anomaly detection in hyperspectral images," *IEEE Aerospace & Electronic Systems Magazine*, vol. 25, no. 7, pp. 5–28, 2010.

[41] S. B. Serpico and L. Bruzzone, "A new search algorithm for feature selection in hyperspectral remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 7, pp. 1360–1367, 2001.

[42] T. Adão, J. Hruska, L. Pádua, J. Bessa, E. Peres, R. Morais, and J. J. Sousa, "Hyperspectral imaging: A review on UAV-based sensors, data processing and applications for agriculture and forestry," *Remote Sensing*, vol. 9, no. 11, pp. 1–30, 2017.

[43] J. Freeman, F. Downs, L. Marcucci, E. N. Lewis, B. Blume, and J. Rish, "Multispectral and hyperspectral imaging: applications for medical and surgical diagnostics," in *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society*, 1997, pp. 700–701.

[44] J. Arroyo-Mora, M. Kalacska, D. Inamdar, R. Soffer, O. Lucanus, J. Gorman, T. Naprstek, E. Schaaf, G. Ifimov, and K. Elmer, "Implementation of a uav–hyperspectral pushbroom imager for ecological monitoring," *Drones*, vol. 3, no. 1, pp. 12, 2019.

[45] H. Lin and J. Chen, "Comparison of several hyperspectral image fusion methods for superresolution," in *Proceedings of the International Conference on Image, Vision and Computing*, 2018.

[46] S. S. Bhattacharyya, W. Plishker, C. Shen, N. Sane, and G. Zaki, "The DSPCAD integrative command line environment: Introduction to DICE version 1.1," Tech. Rep. UMIACS-TR-2011-10, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011.

[47] C. Shen et al., "A lightweight dataflow approach for design and implementation of SDR systems," in *Proceedings of the Wireless Innovation Conference and Product Exposition*, 2010, pp. 640–645.

[48] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen, "An industrial perspective on wireless sensor networks — a survey of requirements, protocols, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1391–1412, 2014.

[49] H. Li, J. Geng, Y. Liu, M. Kashef, R. Candell, and S. Bhattacharyya, "Design space exploration for wireless-integrated factory automation systems," in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, Sundsvall, Sweden, May 2019, 8 pages in online proceedings.

[50] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, Mit Press, 2011, `http://LeeSeshia.org`, ISBN 978-0-557-70857-4.

[51] Y. Liu, R. Candell, K. Lee, and N. Moayeri, "A simulation framework for industrial wireless networks and process control systems," in *Proceedings of the IEEE World Conference on Factory Communication Systems*, 2016, pp. 1–11.

[52] C. Marghescu, M. Pantazica, A. Brodeala, and P. Svasta, "Simulation of a wireless sensor network using OPNET," in *Proceedings of the IEEE International Symposium for Design and Technology in Electronic Packaging*, 2011, pp. 249–252.

[53] C. Harding, A. Griffiths, and H. Yu, "An interface between MATLAB and OPNET to allow simulation of WNCS with MANETs," in *Proceedings of the International Conference on Networking, Sensing and Control*, 2007, pp. 711–716.

[54] B. Vogel-Heuser et al., "Modeling of networked automation systems for simulation and model checking of time behavior," in *Proceedings of the International Multi-Conference on Systems, Signals & Devices*, 2012, pp. 1–5.

[55] J. Schlick, "Cyber-physical systems in factory automation — towards the 4th industrial revolution," in *Proceedings of the IEEE World Conference on Factory Communication Systems*, 2012.

[56] R. Kurte, Z. Salcic, and K. Wang, "A system level simulator for heterogeneous wireless sensor and actuator networks," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2018, pp. 776–783.

[57] A. Chaves et al., "KhronoSim: A platform for complex systems simulation and testing," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2018, pp. 131–138.

[58] J. Geng et al., "Model-based cosimulation for industrial wireless networks," in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, 2018, pp. 1–10.

[59] S. Lin et al., "The DSPCAD framework for modeling and synthesis of signal processing systems," in *Handbook of Hardware/Software Codesign*, S. Ha and J. Teich, Eds., pp. 1–35. Springer, 2017.

[60] ns–3 Project, *ns–3 Tutorial, Release ns–3.25*, 2016.

[61] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, pp. 773–799, May 1995.

[62] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional DIF for rapid prototyping," in *Proceedings of the International Symposium on Rapid System Prototyping*, 2008, pp. 17–23.

[63] L. E. Miller, "Validation of 802.11a/UWB coexistence simulation," Tech. Rep., National Institute of Standards and Technology, 2003.

[64] G. Pei and T. Henderson, "Validation of ns-3 802.11b PHY model," Tech. Rep., The Boeing Company, May 2009.

[65] ns–3 Project, *The ns–3 Wi-Fi Module Documentation*, 2016.

[66] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation, and Control*, pp. 95–100. Citeseer, 2002.

[67] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, November 1999.

[68] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[69] V. Trivedi, P. Varshney, and M. Ramteke, "A simplified multi-objective particle swarm optimization algorithm," *Swarm Intelligence*, vol. 14, pp. 83–116, 2020.