

ABSTRACT

Title of dissertation: **DESIGN OPTIMIZATION OF EMBEDDED
SIGNAL PROCESSING SYSTEMS
FOR TARGET DETECTION**

Kyunghun Lee
Doctor of Philosophy, 2018

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

Sensor networks for automated detection of targets, such as pedestrians and vehicles, are highly relevant in defense and surveillance applications. For this purpose, a variety of target detection algorithms and systems using different types of sensors have been proposed in the literature. Among them, systems based on non-image sensors are of special interest in many practical deployment scenarios because of their power efficiency and low computational loads. In this thesis, we investigate low power sensor systems for detecting people and vehicles using non-image sensors such as acoustic and seismic sensors. Our investigation is focused on design optimization across trade-offs including real-time performance, energy efficiency, and target detection accuracy, which are key design evaluation metrics for this class of systems.

Design and implementation of low power, embedded target detection systems can be decomposed into two major, inter-related subproblems: (a) algorithm development, which encompasses the development or selection of detection algorithms and optimiza-

tion of their parameters, and (b) system development, which involves the mapping of the algorithms derived from (a) into real-time, energy efficient implementations on the targeted embedded platforms. In this thesis, we address both of these subproblems in an integrated manner. That is, we investigate novel algorithmic techniques for improvement of accuracy without excessive computational complexity, and we develop new design methodologies, tools, and implementations for efficient realization of target detection algorithms on embedded platforms.

We focus specifically on target detection systems that employ acoustic and seismic sensing modalities. These selected modalities support the low power design objectives of our work. However, we envision that our developed algorithms and implementation techniques can be extended readily to other types or combinations of relevant sensing modalities.

Throughout this research, we have developed prototypes of our new algorithms and design methods on embedded platforms, and we have experimented with these prototypes to demonstrate our findings, and iteratively improve upon the achieved implementation trade-offs. The main contributions of this thesis are summarized in the following.

- (1). Classification algorithm for acoustic and seismic signals. We have developed a new classification algorithm for discrimination among people, vehicles, and noise. The algorithm is based on a new fusion technique for acoustic and seismic signals. Our new fusion technique was evaluated through experiments using actual measured datasets, which were collected from different sensors installed in different locations and at different times of day. Our proposed classification algorithm was shown to achieve a significant reduction in the number of false alarms compared to a baseline fusion approach.

(2). Joint target localization and classification framework using sensor networks.

We designed a joint framework for target localization and classification using a single generalized model for non-imaging based multi-modal sensor data. For target localization, we exploited both sensor data and estimated dynamics within a local neighborhood. We validated the capabilities of our framework by using an actual multi-modal dataset, which includes ground truth GPS information (e.g., time and position) and data from co-located seismic and acoustic sensors. Experimental results showed that our framework achieves better classification accuracy compared to state of the art fusion algorithms using temporal accumulation and achieves more accurate target localizations than a baseline target localization approach.

(3). Design and optimization of target detection systems on embedded platforms using dataflow methods. We developed a foundation for our system-level design research by introducing a new rapid prototyping methodology and associated software tool. Using this tool, we presented the design and implementation of a novel, multi-mode embedded signal processing system for detection of people and vehicles related to our algorithmic contributions. We applied a strategically-configured suite of single- and dual-modality signal processing techniques together with dataflow-based design optimization for energy-efficient, real-time implementation. Through experiments using a Raspberry Pi platform, we demonstrated the capability of our target detection system to provide efficient operational trade-offs among detection accuracy, energy efficiency, and processing speed.

(4). Software synthesis from dataflow schedule graphs on multicore platforms. We developed new software synthesis methods and tools for design and implementation of

embedded signal processing systems using dataflow schedule graphs (DSGs). DSGs provide formal representations of dataflow schedules, which encapsulate information about the assignment of computational tasks (signal processing modules) to processing resources and the ordering of tasks that are assigned to the same resource. Building on fundamental DSG modeling concepts from the literature, we developed the first algorithms and supporting software synthesis tools for mapping DSG representations into efficient multi-threaded implementations. Our tools replace ad-hoc multicore signal processing system development processes with a structured process that is rooted in dataflow formalisms and supported with a high degree of automation. We evaluated our new DSG methods and tools through a demonstration involving multi-threaded implementation of our proposed classification algorithm and associated fusion technique for acoustic/seismic signals.

DESIGN OPTIMIZATION OF EMBEDDED SIGNAL PROCESSING
SYSTEMS FOR TARGET DETECTION

by

Kyunghun Lee

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
2018

Advisory Committee:
Professor Shuvra S. Bhattacharyya, Chair/Advisor
Professor Jonathan Simon
Professor Gang Qu
Dr. Benjamin S. Riggan
Professor Ki-Yong Kim

© Copyright by
Kyunghun Lee
2018

Dedication

To my Lord Jesus. To My father, mother, brother, and friends

Acknowledgments

I sincerely thank Prof. Shuvra Bhattacharyya for his invaluable support, guidance and encouragement throughout the years of my PhD study. His continuous support from all aspects of my work and life has allowed me to complete this study, and overcome difficulties of all kinds. His generous and insightful guidance is indispensable for my successful completion of my study. I am truly grateful to have him as my advisor.

Besides my advisor, I would like to thank Dr. Riggan for his invaluable guidance throughout my Ph. D. study. His insightful comments and encouragement, but also for the hard question which incited me to widen my research from various perspectives, which are indispensable to my completion of research and dissertation.

I want to thank my committee members, Prof. Simon, Prof. Qu, and Prof. Kim for providing insightful and valuable feedbacks and suggestions to my research.

I am grateful to Prof. Jarmo Takala, Dr. Thyagaraju Damarla, Dr. William Plishker, Dr. Timo Viitanen, Dr. Inkeun Cho, Dr. Shuoxin Lin, Dr. Kishan Sudusinghe, Dr. Yanzhou Liu, Dr. Lai-huei Wang, Dr. Ilya Chukhman, Ms. Haifa Salem, Ms. Lin Li, Mr. Jiahao Wu, Mr. Adrian Sapio, Mr. Honglei Li, Mr. Jing Geng, Mr. Eungjoo Lee, Mr. Abhay Raina, Ms. Yeasop Lee, Mr. Lei Pan, and other colleagues and collaborators for their generous help.

Finally, I give my special thanks to my father and mother, my brother, and my girlfriend for their love and understanding.

All praise, honour and glory to my Lord Jesus Christ for His richest grace and mercy for the accomplishment of this thesis.

The research underlying this thesis was supported in part by the U.S. Army Research Laboratory, and U.S. Air Force Office of Scientific Research

Table of Contents

Dedication	ii
Acknowledgements	iii
List of Tables	viii
List of Figures	ix
List of Abbreviations	x
1 Introduction	1
2 Prototyping Real-Time Tracking Systems on Mobile Devices	6
2.1 Introduction	7
2.2 Background	9
2.2.1 DICE	9
2.2.2 LIDE	9
2.3 Related Work	10
2.4 Background	11
2.4.1 JVMs on Android	12
2.4.2 Android SDK and JNI	13
2.4.3 Android NDK and Android Native Terminal	15
2.5 DICE-based Prototyping Framework for Tracking Systems	16
2.5.1 Cross-Platform Design	18
2.5.2 Sensor Integration	19
2.6 Tracking System Case Study	21
2.7 Experiments	24
2.7.1 Execution Time Performance	25
2.7.2 Lines of Code Efficiency	26
2.7.3 Energy Efficiency	29
2.7.4 Sensor Experimentation	29
2.7.5 Summary of Experimental Results	30
2.8 Summary	31

3	An Accumulative Fusion Architecture for Discriminating People and Vehicles using Acoustic and Seismic Signals	32
3.1	Introduction	33
3.2	Related Work	34
3.3	Fusion Framework	36
3.3.1	Cepstral Analysis and SVM Classification	36
3.3.2	Baseline Fusion Architecture	39
3.3.3	ALFFS	40
3.4	Experiments	43
3.5	Summary	45
4	An Optimized Embedded Target Detection System using Acoustic and Seismic Sensors	46
4.1	Introduction	47
4.2	Related Work	48
4.3	System Design	49
4.3.1	Single-Modality Operation	52
4.3.2	DST-based Fusion Mode	54
4.3.3	ALFFS Mode	56
4.4	Experiments	58
4.5	Conclusion	62
5	A Joint Target Localization and Classification Framework for Sensor Networks	64
5.1	Introduction	64
5.2	Related Works	66
5.3	Joint Localization and Classification Framework	69
5.4	Experiments and Results	75
5.5	Conclusion	78
6	Software Synthesis from Dataflow Schedule Graphs	79
6.1	Concurrent DSGs	80
6.1.1	Inter-SDSG Coordination Actors	82
6.1.2	Delays	85
6.1.3	Loop SCAs	86
6.1.4	Example	87
6.2	Case Study: Real-Time Classification System	89
6.2.1	Application Graph	90
6.2.2	Alternative DSGs	94
6.2.3	Performance Evaluation	94
6.3	Summary	100
7	Conclusion and Future Work	102
7.1	Conclusions	102
7.2	Future Work	104

List of Tables

2.1	Execution time comparison between the ANT-based development approach used in DPTS and conventional SDK-based development.	25
2.2	CPU load comparison between the ANT-based development approach used in DPTS and conventional SDK-based development.	26
2.3	Lines of code (LOC) comparison between SDK-based development and DPTS for tracking system coordination code.	28
2.4	Energy efficiency comparison.	29
2.5	Comparison between sensors.	30
3.1	Accuracy comparison (%).	45
4.1	Parameter values used in our experiments.	59
4.2	Accuracy comparison (%).	60
4.3	Power consumption, run-time, and energy efficiency comparison.	61
5.1	Accuracy comparison (%).	76
5.2	Tracking performance comparison (people)	77
5.3	Tracking performance comparison (vehicle)	77
6.1	Comparison of run-times and peak memory requirements.	98

List of Figures

2.1	Illustration of the DPTS framework.	17
2.2	Integration among the sensing subsystem, embedded signal processing subsystem, and host computer in the DPTS-based prototyping process. . .	22
2.3	Top-level dataflow model for the signal processing core of the DTMSD prototype that we experiment with in this case study.	23
3.1	Fusion architecture using DST.	37
3.2	Fusion architecture using ALFFS.	38
3.3	ROC curves for multiclass classification.	41
4.1	Dataflow graph for single-modality PVD.	52
4.2	Dataflow graph for the dual-modality mode μ_{dst}	54
4.3	Dataflow graph for dual-modality PVD using ALFFS.	56
5.1	Localization using multilateration.	68
5.2	Grid map for estimating the next target location.	72
5.3	Confidence maps for each node are shown on the above. The spatial accumulation of these maps are shown on the below.	74
5.4	Layout of sensor nodes.	75
6.1	An illustration of <code>snd</code> and <code>rec</code> actors in CDSGs.	84
6.2	An example that illustrates the concepts and constructions developed in this section.	88
6.3	Dataflow graph for ALFFS application.	92
6.4	An SDSG for the application graph of Figure 6.3.	95
6.5	CDSG (2 threads) for the application graph of Figure 6.3.	96
6.6	CDSG (3 threads) for the application graph of Figure 6.3.	97

List of Abbreviations

A/D	Analog/Digital
ALFFS	Accumulation of Local Feature-level Fusion Scores
ARM	Advanced RISC Machine
ART	Android RunTime
ANT	Android Native Terminal
API	Application Programming Interface
BDF	Boolean DataFlow
BSCs	Binary SVM Classifiers
CDSG	Concurrent DSG
CLEDM	Classification and Localization using Estimated Dynamics and Multimodal data
COTS	Commodity-Of-The-Shelf
CPU	Central Processing Unit
DC	Direct Current
DDDAS	Dynamic Data Driven Applications Systems
DICE	DSPCAD Integrative Command Line Environment
DIF	Dataflow Interchange Format
DIF-DSG	Dataflow Interchange Format for Dataflow Schedule Graph
DPDC	Distinct Pair of Decision Classes
DPTS	DICE-based Prototyping framework for Tracking Systems
DSD	Dataflow-based System Design
DSG	Dataflow Schedule Graph
DSP	Digital Signal Processing
DSPCAD	Computer-Aided Design and implementation of DSP systems
DST	Dempster-Shafer Theory
DTSMD	DDDAS-enabled Tracking System for Mobile Devices
DTT	DSG Token Tracking
FFT	Fast Fourier transform
FFTW	Fastest Fourier Transform in the West
FIFO	First-In, First-Out
FIR	Finite Impulse Response
GCC	GNU Compiler Collection
GPS	Global Positioning System
GPU	Graphics Processing Unit

ISCAs	Inter-SDSG Coordination Actors
IDS	Input Data Segment
JIT	Just-In-Time
JNI	Java Native Interface
JVM	Java Virtual Machine
LAE	Loop-And-Exit
LIDE	Lightweight Dataflow Environment
LOC	Lines Of Code
MDC	Multi-Dataflow Composer
MIPS	Microprocessor without Interlocked Pipeline Stages
MSRP	Manufacturer's Suggested Retail Price
NDK	Native Development Kit
PREESM	Parallel and Real-time Embedded Executives Scheduling Method
PVD	Person-and-Vehicle Detection
RA	Reference Actor
ReLU	Rectified Linear Unit
RISC	Reduced Instruction Set Computing
ROC	Receiver Operating Characteristic
RSS	Received Signal Strength
SCA	Schedule Control Actor
SDF	Synchronous DataFlow
SDSG	Sequential DSG
SDK	Software Development Kit
SS	Simple Scheduler
STAA	Single Thread per Application Actor
SVM	Support Vector Machine
TDOA	Time-Difference-Of-Arrival

Chapter 1: Introduction

Automated detection of targets, such as people and vehicles, in wilderness environments is of great relevance in border security and other defense- and surveillance-related applications. With advances in sensor technology and algorithms, wireless sensor networks can be designed to deploy such target detection capability.

However, the cost and energy efficiency of such networks is a major bottleneck to their utility. The high cost of nodes in such networks may require personnel to retrieve nodes when they fail or to periodically perform maintenance in the field on the nodes. This can pose great risk to these personnel when the networks are deployed in dangerous areas. Such risk can be eliminated through use of nodes that are of much lower cost and can simply be disposed of when they fail or under-perform. At the same time, the energy of the nodes relates directly to the useful lifetime of such deployments (in disposable network scenarios) or the maintenance interval for networks that are intended for longer term operation.

In this thesis, we propose to develop novel design methods and tools that address these bottlenecks of cost and energy efficiency in development of target detection systems. Our focus is on developing design optimization techniques that integrate algorithm and implementation aspects to derive streamlined embedded implementations of target

detection systems. These design optimization methods take into account key design evaluation metrics, including target detection accuracy, real-time performance and energy consumption.

For concreteness, we focus specifically on target detection systems that employ acoustic and seismic sensing modalities. Image-based target detection can provide significantly better target classification performance over other modalities (e.g., see [1]), but image-based systems involve higher energy consumption for the sensing subsystems, and also involve higher computational costs, including increased energy consumption, memory requirements, and execution time. Compared to image-based sensors, acoustic and seismic sensors are attractive in our design context due to their energy efficiency and their lower requirements in terms of computation and storage for processing and managing the acquired sensor data. However, we envision that the algorithms and implementation techniques developed in this research can be extended readily to other types or combinations of relevant sensing modalities. This is a useful direction for future work that is motivated in this thesis.

Throughout the thesis, we apply dataflow-based methods for system design, analysis and optimization to help address challenges of cost- and energy-constrained implementation of target detection systems. *Dataflow* is a form of model-based design that provides a valuable formal foundation for developing complex signal processing systems [2,3]. In a dataflow graph representation of a signal processing application, vertices, called *actors*, represent computational tasks (signal processing hardware or software modules), and edges represent first in, first out (FIFO) communication of data between actors. Data values that pass through edges are encapsulated in objects that are referred to as

tokens. The execution of actors in a dataflow graph is decomposed into discrete units of execution, which are referred to as *firings* of the associated actors. For more background on dataflow modeling in the context of signal processing systems, we refer the reader to [2, 3].

Synchronous dataflow (SDF) is a specialized form of dataflow that we use extensively in the architecture design and implementation aspects of this thesis. In SDF, the number of tokens produced and consumed by each actor (on the associated input and output edges) is constant across all firings of the actor [4].

The remainder of this thesis is organized as follows. In Chapter 2, we develop a foundation for our system-level design research by introducing a new rapid prototyping methodology and associated software tool. This tool, called the *DICE-based Prototyping framework for Tracking Systems (DPTS)*, applies the DSPCAD Integrative Command Line Environment (DICE) [5], and Lightweight Dataflow Environment (LIDE) [6], and builds on these previously-developed tools in new ways to enable design, prototyping and optimization of power-efficient tracking systems on mobile devices. The contributions summarized in this chapter are published in [7].

In Chapter 3, we introduce new algorithms for sensor fusion in support of multiclass classification among people, vehicles, and noise (the absence of people or vehicles). Our fusion algorithms operate on the two sensing modalities, seismic and acoustic sensing, that are motivated above. We develop and comparatively evaluate two different multiclass algorithms, a score-level fusion algorithm that is based on Dempster-Shafer Theory (DST), and an accumulative algorithm that exploits feature-level fusion. The contributions summarized in this chapter are published in [8].

In Chapter 4, we present the design and implementation of a novel, multi-mode embedded signal processing system for detection of people and vehicles using acoustic and seismic sensors. In this work, we apply a dataflow-based methodology for model-based implementation and design optimization of the proposed multi-mode target detection system. We apply a strategically-configured suite of single- and dual-modality signal processing techniques together with dataflow-based design optimization for energy-efficient, real-time implementation. The contributions summarized in this chapter are published in [9].

In Chapter 5, we provide a framework that can be used for a classification and localization of targets simultaneously using multiple sensor nodes with a singular generalized model, which can be applied to every node in a sensor network. By employing probabilistic maps from acoustic, seismic, and estimated velocities, we improve classification performance compared to our recent work in [8], and we provide better localization (tracking) capability compared to multilateration-based methods. The contributions summarized in this chapter are published in [10].

In Chapter 6, we develop methods to synthesize efficient multithreaded embedded software for implementing schedules from abstract dataflow schedule graph (DSG) representations of the schedules. The DSG abstraction allows designers to model a schedule as a separate dataflow graph, thereby providing a formal, abstract (platform- and language-independent) representation for the schedule [11]. The software synthesis methods that we develop in this chapter can be applied to a broad class of signal processing applications, and are not restricted to applications involving target detection. At the same time, they are of great utility in the efficient implementation of target detection systems, as we

demonstrate through a complex application example and associated experiments in this chapter.

In Chapter 7, we summarize the developments of the thesis, and outline interesting directions for future work that are motivated by these developments.

Chapter 2: Prototyping Real-Time Tracking Systems on Mobile Devices

In this chapter, we address the design and implementation of low power embedded systems for real-time tracking of humans and vehicles. Such systems are important in applications such as activity monitoring and border security. We motivate the utility of mobile devices in prototyping the targeted class of tracking systems, and demonstrate a dataflow-based and cross-platform design methodology that enables efficient experimentation with key aspects of our tracking system design, including real-time operation, experimentation with advanced sensors, and streamlined management of design versions on host and mobile platforms. Our experiments demonstrate the utility of our mobile-device-targeted design methodology in validating tracking algorithm operation; evaluating real-time performance, energy efficiency, and accuracy of tracking system execution; and quantifying trade-offs involving use of advanced sensors, which offer improved sensing accuracy at the expense of increased cost and weight. Additionally, through application of a novel, cross-platform, model-based design approach, our design requires no change in source code when migrating from an initial, host-computer-based functional reference to a fully-functional implementation on the targeted mobile device.

The work presented in this chapter has been published in [7].

2.1 Introduction

Automated detection and tracking of people and vehicles is an important area of low power signal processing with applications in activity monitoring and border security. The development of tracking systems involves complex trade-offs among algorithmic, sensing, and processing considerations (e.g., see [12, 13]). Extensive experimentation with such trade-offs in practical settings is critical before committing resources to development of specialized sensor node implementations, where major design changes are often impractical or costly to make.

To support such experimentation, we develop in this chapter a new rapid prototyping methodology and associated software libraries and tools, called the *DICE-based Prototyping framework for Tracking Systems (DPTS)*. DPTS applies the DSPCAD Integrative Command Line Environment (DICE), which is a software environment for cross-platform and model-based design, implementation, and testing of signal processing systems (e.g., see [5, 14]).

A second distinguishing aspect of DPTS is that it applies commodity-of-the-shelf (COTS), Android-based mobile devices (tablet computers and smartphones) as integrated platforms for sensing, signal processing, and communication. Compared to conventional use of desktop-computer-based algorithm development and prototyping methodologies, this application of mobile devices provides significantly more flexibility, and cost-efficiency in deploying prototype sensor nodes in the field for experimentation and demonstration. At the same time, mobile devices provide extensive and steadily increasing capabilities for embedded processing, sensor interfacing, and communication, which

are all important for effective prototyping of advanced tracking capabilities.

A third distinguishing aspect of this work is the focus on tracking systems that employ acoustic sensors. Such systems offer advantages in terms of energy efficiency and cost compared to visual-sensor-based tracking systems. However, advanced signal processing algorithms are required to achieve acceptable levels of tracking accuracy using acoustic sensors, and significant trade-offs involving sensor cost, signal quality, and tracking system range (the maximum distance of the sensor from the target) are involved. We demonstrate the effectiveness of the DPTS framework in supporting design, implementation and experimentation related to such critical system-level trade-offs.

We would like to emphasize that although we focus in this work on tracking systems that employ acoustic sensors, the DPTS framework can readily be adapted to other sensing modalities, and to multimodal tracking systems. Developing such adaptations is a useful direction for further work.

We validate the capabilities of our prototyping framework by demonstrating a mobile-platform-based tracking system implementation that provides high tracking accuracy under real-time constraints. We demonstrate the efficiency with which trade-offs involving platform cost, peripheral device characteristics, and overall system performance can be explored. We also demonstrate experiments involving an advanced acoustic sensor device and a high-quality, external analog-to-digital (A/D) converter in place of the built-in, low cost acoustic sensing interface within the targeted COTS Android platform. Additionally, we demonstrate the streamlining of the code development and experimentation process that is achieved through our novel application in this work of DICE and the associated cross-platform design and implementation methods.

2.2 Background

In this section, we provide background on DICE and LIDE, which are software tools that the contributions in this chapter build upon. DICE and LIDE are core components within the DSPCAD Framework, which is a framework for model-based design and implementation of signal processing systems [15].

2.2.1 DICE

The DSPCAD Integrative Command Line Environment (DICE) is a package of utilities that facilitates efficient development and testing of embedded software projects, and incorporates special emphasis on support for embedded signal and information processing [5, 14]. DICE provides integrated support for cross-platform development, model-based design methodologies, designs evolving heterogeneous programming languages, and application of different kinds of design and testing methods. DICE provides a useful foundation for developing and maintaining libraries and design tools for optimized implementation of signal and information processing systems. DICE can be used on different operating systems, including Android, Linux, MacOS, and Windows (with Cygwin).

2.2.2 LIDE

The Lightweight Dataflow Environment (LIDE) is a flexible design environment that allows designers to experiment with dataflow-based design and implementation directly on different types of programmable platforms [6, 16]. LIDE is “lightweight” in the sense that it is based on a compact set of APIs that can be retargeted to different platforms

and integrated into different design processes relatively easily. LIDE includes application programming interfaces (APIs) for developing actors and edges in signal processing dataflow graphs. These APIs are defined in terms of fundamental dataflow principles rather than being specific to any particular actor programming language. The APIs can be retargeted readily across a wide variety of specific languages for DSP simulation and implementation, including, for example, C, C++ CUDA, MATLAB, OpenCL, and Verilog/VHDL.

2.3 Related Work

Various algorithms have been developed for detection of people and vehicles using acoustic sensors (e.g., see [17–21]). Multimodal, non-visual sensing systems have also been proposed — in particular, use of acoustic sensor systems for vehicle detection, and seismic sensor systems for people detection (e.g., see [22–24]). In contrast to these works, which focus on algorithm aspects, we focus in this chapter on methods for efficient prototyping of and experimentation with such algorithms so that relevant system design trade-offs can be evaluated. Such rapid prototyping is critical to validate operation and optimize system-level trade-offs before deploying such complex tracking systems.

DPTS applies dataflow-based modeling techniques for signal processing systems (e.g., see [2]). In particular, we apply a dataflow-based design tool called the lightweight dataflow environment (LIDE) [6, 25], which is based on application programming interfaces (APIs) for dataflow design that can be retargeted to arbitrary implementation languages, such as C, CUDA, and Verilog. To support the DPTS approach, we have de-

veloped new capabilities in LIDE to support efficient development and testing of dataflow graphs using Android Terminal (a command line interface for Android platforms). This integration with Android Terminal allows automation of actor (dataflow functional component) and graph testing and design space exploration through application of standard scripting techniques. Our use of retargetable dataflow graph APIs in conjunction with scripting techniques allows the DPTS approach to be applied across different kinds of host and mobile device platforms. Due to their base in retargetable dataflow techniques, we envision that the models and methods in DPTS can be readily adapted for use with other relevant dataflow tools, such as Orcc [26], PREESM [27], and the multi-dataflow composer (MDC) [28]. Exploration of such adaptations is a useful direction for future work.

2.4 Background

In this section, we provide background on concepts and general-purpose tools for Android-based, mobile software development. DPTS applies these concepts and tools, and integrates them into a domain-specific environment for design and implementation of energy-efficient, real-time tracking systems.

Android applications are typically developed using the Java programming language, and executed using a Java virtual machine (JVM) for enhanced portability. In this section, we review JVMs for Android devices and discuss their performance compared to that of C-based implementations. We also discuss the Java Native Interface (JNI), which allows integration of subsystems programmed in other languages — such as C, C++, or

assembly language — with Java [29]. Then we discuss limitations of JNI in relation to our targeted domain of energy-efficient, real-time tracking systems, which motivates our use of Android Native Terminal (ANT). Finally, we elaborate on ANT and Android standalone toolchains, which are important features of Android environments that DPTS applies.

2.4.1 JVMs on Android

Earlier versions of Android used the Dalvik virtual machine. For improved efficiency, Dalvik incorporates trace-based, just-in-time (JIT) compilation, which compiles frequently-executed segments of code at run-time so that they can be executed directly on the target processor rather through interpretation by the virtual machine [30]. However, this JIT compilation process produces penalties in performance and energy consumption since it is carried out at run-time. The performance penalties can be problematic, for example, in real-time application scenarios, where deadlines must be met consistently at all stages of execution.

Android runtime (ART) is a runtime system for Android that was developed as a successor to Dalvik, and includes features to improve upon the performance and energy consumption overheads in Dalvik. For example, ART avoids the penalties of JIT compilation described above by compiling Java-based application bytecode into machine code when an application is installed rather than when it is executed. In addition, ART's compiled code reduces user interface latency and stuttering. It has been demonstrated that through such enhancements, ART provides significantly better performance compared to

Dalvik (e.g., see [31]).

In spite of these improvements, Java-based Android applications — whether they are interpreted, compiled just-in-time, or compiled at installation time — remain slower than efficiently designed native applications that are compiled from C code (*C-native* applications). For example, a performance comparison was reported among Dalvik, ART, and native GCC compilation on a finite impulse response (FIR) filter example [32]. The evaluation was carried out on a Google Nexus 5 device. The results demonstrated significant performance improvement of ART over Dalvik, and further improvement of C-native implementation (using `gcc`) over ART.

2.4.2 Android SDK and JNI

Android applications are typically developed using the Android Software Development Kit (SDK), which allows user-friendly, interactive applications to be developed using Java along with optional use of C-based modules that are integrated using the Java Native Interface (JNI). Because it is centered on use of the Android SDK, we refer to this approach — which is suitable, for example, for a wide variety of consumer-oriented applications — as the *SDK-based* development approach for Android applications.

Use of C-based signal processing libraries and subsystems is important due to efficiency considerations in our targeted domain of tracking system design, especially for mission-critical tracking systems. JNI is one approach for integrating such libraries and subsystems into the framework of Android application development [29]. JNI operates within the framework of SDK-based application development, so it shares both advan-

tages and disadvantages of SDK-based development.

Advantages of SDK-based development include its features for intuitive user interfaces in the developed applications, such as elaborate graphical interfaces and touch screen input. However, these capabilities result in additional code, energy consumption, and computational resource utilization. Such overhead is not problematic for many consumer-oriented applications; however, it is highly undesirable in the implementation of mission-critical tracking systems, which are geared towards reliable, accurate, ultra-low-energy, unattended operation rather than for use in highly interactive, end-user-centric scenarios. Due to resource limitations on mobile devices, the overheads resulting from SDK-based development can lead to significant limitations on the effectiveness of tracking systems from the perspectives of computing speed and power consumption.

Although use of JNI can help to reduce some of the overhead involved in SDK-based implementations, it can also lead to increased program complexity; difficulty in guaranteeing compatibility across different hardware platforms and Android versions; and decreased flexibility in adapting implementations to requirements [33]. For example, C source code requires certain modifications to enable integration into Android applications using JNI. This need for modifications leads to different versions of code that must be maintained between simulation (host PC) and embedded (mobile device) versions. In addition, some standard functions in C are not supported in JNI.

2.4.3 Android NDK and Android Native Terminal

The Android Native Development Kit (NDK) is a set of development tools that includes capabilities for compiling C code for use in Android applications. The tools provided in the NDK include cross-compilers for various processing architectures, including ARM, x86, and MIPS architectures. We employ the NDK extensively as part of DPTS to bypass the overheads described in Section 2.4.2 that are associated with SDK-based development.

Specifically, DPTS employs the NDK along with other features of Android development called standalone toolchains and Android Native Terminal (ANT) as a means for deriving streamlined embedded implementations from complex, C-based signal processing software. We refer to this approach to Android development — based on integrated use of NDK, standalone toolchains, and ANT — as *ANT-based development*. Thus, in summary, DPTS employs ANT-based development of Android software as an alternative to conventional SDK-based development.

The ANT environment allows development of applications that execute native C-based code without the need for specialized modifications. Various studies have applied ANT to help improve the efficiency of Android applications (e.g., see [34–37]). These studies help to validate the relevance of ANT in our context of resource- and energy-constrained implementation. A novel aspect of our work on DPTS in relation to works such as these is the development of a comprehensive model-based and cross-platform design methodology and supporting tools that apply an ANT-based development process.

2.5 DICE-based Prototyping Framework for Tracking Systems

Figure 2.1 illustrates the DPTS framework and its application in design and implementation of tracking systems. The methodology supports rapid iteration and performance assessment of interactions among four key aspects of tracking system prototyping: algorithm design; dataflow-based system design; embedded implementation on the targeted mobile devices; and performance evaluation, which may guide refinements to the other three prototyping aspects in subsequent design iterations.

The translation between the Algorithm Design and Dataflow-based System Design (DSD) blocks is carried out by hand to enable flexibility in applying tools and methods that are best matched to each of the steps. This flexibility is important because design decisions and refinements at both of these levels have major impact on implementation trade-offs, and intensive, iterative experimentation is critical to understanding these trade-offs and optimizing them in relation to the overall application objectives.

Support for model-based design and cross-platform, language agnostic unit testing throughout the framework (provided by DICE) helps designers maintain functional consistency among different aspects in the prototyping process (especially between algorithm design and DSD). Additionally, orthogonalization between dataflow graph scheduling and actor implementation in LIDE [6, 16] helps designers to efficiently and systematically explore interactions between these two critical parts of dataflow-based, signal processing system design. For general background on the importance of orthogonalization in system-level design, we refer the reader to [38].

We demonstrate DPTS in this chapter using a tracking system that was introduced

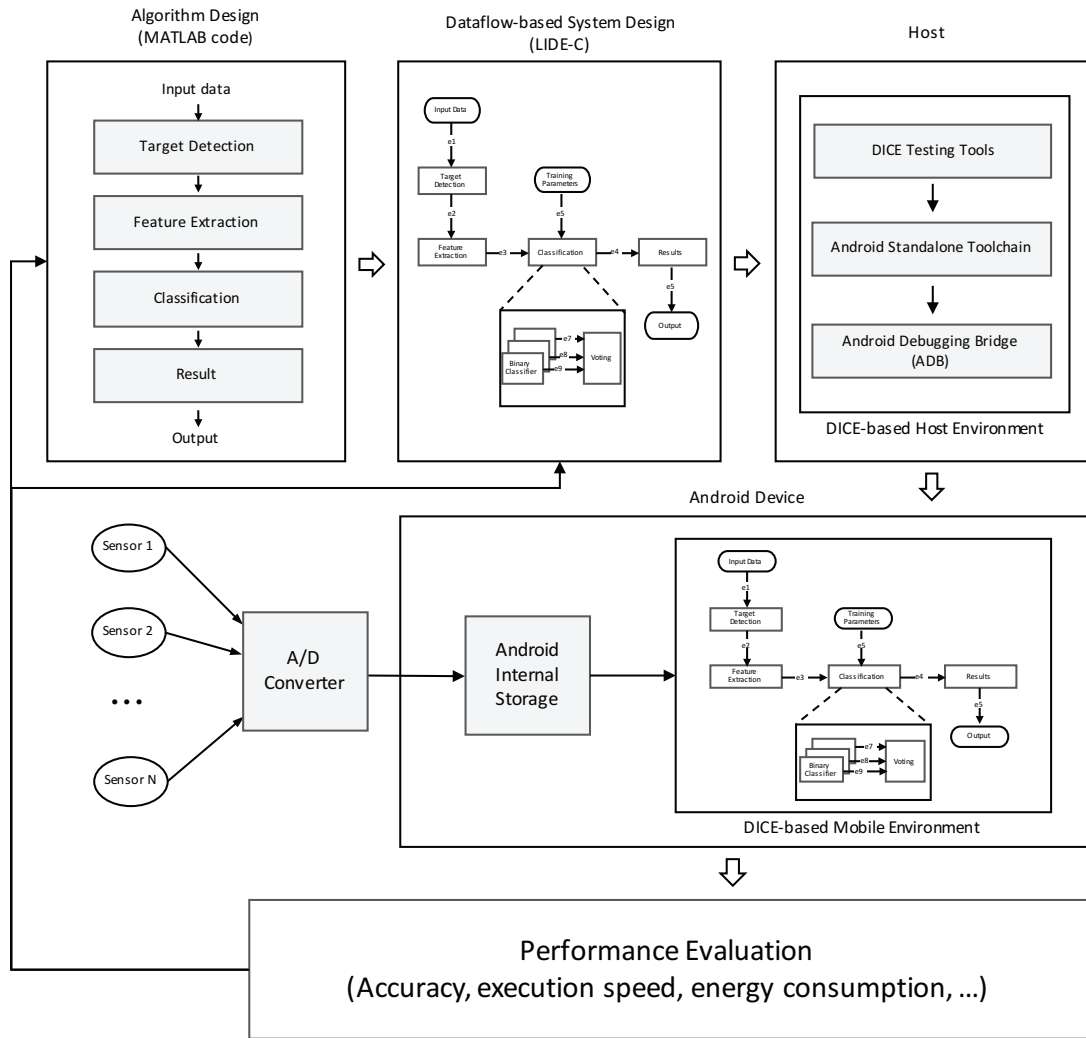


Figure 2.1: Illustration of the DPTS framework.

recently by Ben Salem et al. [39]. Ben Salem’s tracking system is referred to as the *DDDAS-enabled Tracking System for Mobile Devices (DTSMD)*. This name originates from emphasis in this work on integrating principles of Dynamic Data Driven Applications Systems (DDDAS) [40] to the design of adaptive tracking systems that are autonomously reconfigurable across different application scenarios and operational requirements.

In this chapter, we demonstrate the utility of DPTS in carrying out rapid prototyping iterations, and deriving optimized implementations on mobile devices of the adaptive tracking methods introduced in DTSMD. We show in this chapter how DPTS facilitates efficient mobile-device targeted realization of the algorithms employed in DTSMD. Further details on this case study of applying DPTS to DTSMD are presented in Section 2.6 and Section 2.7.

2.5.1 Cross-Platform Design

The cross-platform design capabilities in DPTS, which are derived from features in DICE, operate across stages of development that include design, build, test, and deployment. At the design stage, DPTS provides a uniform prototyping environment across different operating systems. For example, a designer can switch seamlessly between MacOS- and Linux-based development environments for the same system design. Similarly, features in DICE that support DPTS allow for cross-compilation from different host environments to different target environments with easy-to-manage configuration settings for selecting between host/target combinations. The cross platform design capabilities of

DICE and DPTS are geared specifically for design and implementation of signal processing dataflow graphs and libraries, and are interoperable with various platform-specific development environments and general-purpose cross platform tools such as Cmake [41].

In addition to supporting cross-platform design in the host environment, DICE is also embedded in the target (Android) environment as part of the implementation process in DPTS. Thus, features in DICE for testing and native code integration, along with the large set of utilities in DICE (e.g., for managing and navigating through designs) can be applied in the target environment in the same way that DICE features are used in the host environment. Since DICE operates fully within a command-line environment, and requires no graphical user interface support, it can be operated efficiently, with minimal overhead in the target environment, where computational resources are limited and energy efficiency is critical. Additionally, integration of DICE with ANT in DPTS allows designers to develop C-based, native dataflow graph and signal processing library implementations without any need for source code modifications and without incurring overhead associated with SDK-based development and Java virtual machine operation. These capabilities in turn reduce development and maintenance costs, and further improve the efficiency of tracking system implementations on the targeted mobile devices.

2.5.2 Sensor Integration

DPTS allows designers to integrate different types of sensors efficiently into tracking system implementations. Since the particular kinds of sensors used strongly affect the utility of specific signal processing algorithms, and the cost and energy efficiency

of embedded implementations, flexible sensor integration is an important requirement in prototyping environments for our targeted class of tracking systems.

In DPTS, the sensor-interface is controlled by SDK-based tools, which provide flexibility in interfacing to different kinds of sensors, and reading sensor data directly into internal memory on the Android device. Other aspects of the target implementation — including development of the core signal processing processing functionality — are developed using ANT-based development, as motivated in Section 2.4.

Figure 2.2 illustrates the integration among the sensing subsystem, embedded signal processing subsystem, and host device in the DPTS-based prototyping process.

2.6 Tracking System Case Study

In this section, we describe a case study in which we apply the DPTS Framework to mobile-device-based prototyping of a novel tracking system, called the DDDAS-enabled Tracking System for Mobile Devices (DTSMD), which was briefly introduced in Section 2.5. DTSMD is designed for tracking of people and vehicles using acoustic sensors.

By applying DPTS with DTSMD, we develop in this case study a tracking system prototype that uses different algorithms for feature extraction and classification through systematic integration of techniques for algorithm exploration, dataflow-based design, and Android-based mobile implementation. Through this DPTS-enabled prototyping process, the designer is assisted in choosing the appropriate algorithms — in terms of alternative operating modes for relevant actors — based on the environmental operating characteristics (e.g., signal to noise ratio) and resource constraints. Implementing and testing the DTSMD system together with alternative sensors on an Android-based mobile device allows us to investigate different trade-offs among energy consumption, processing time, tracking accuracy, and the cost of the sensing subsystem (e.g., through use of the built-in mobile device sensor versus connection of an external sensor that is designed for mission critical applications).

Figure 2.3 illustrates the top-level dataflow model for the signal processing core of the DTSMD system prototype that we experiment with in this case study. This dataflow model corresponds to the block in Figure 2.1 labeled Dataflow-based System Design and the block in Figure 2.2 labeled Signal Processing Subsystem.

In the target detection actor of Figure 2.3, the input acoustic signal is filtered and

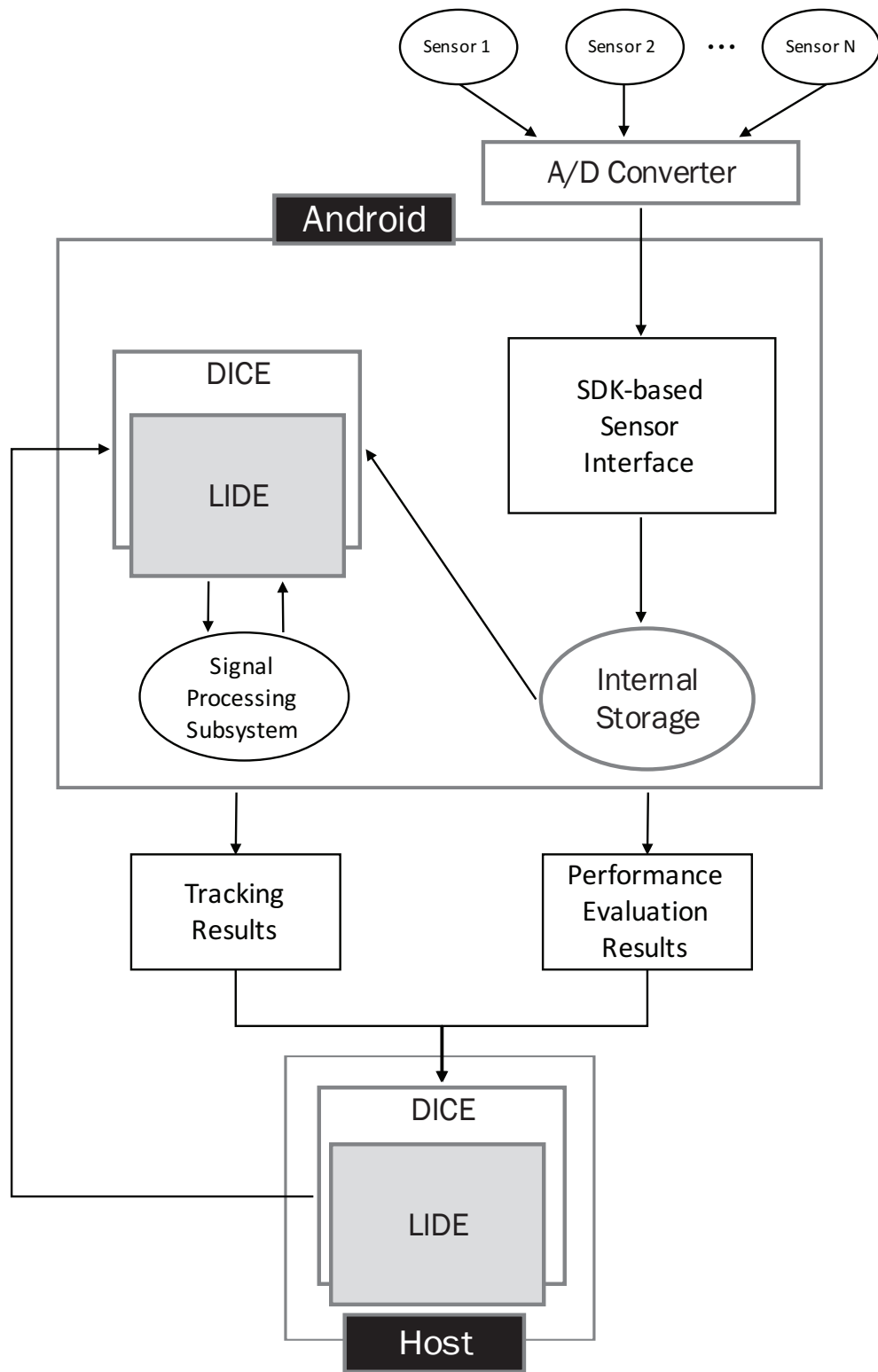


Figure 2.2: Integration among the sensing subsystem, embedded signal processing sub-system, and host computer in the DPTS-based prototyping process.

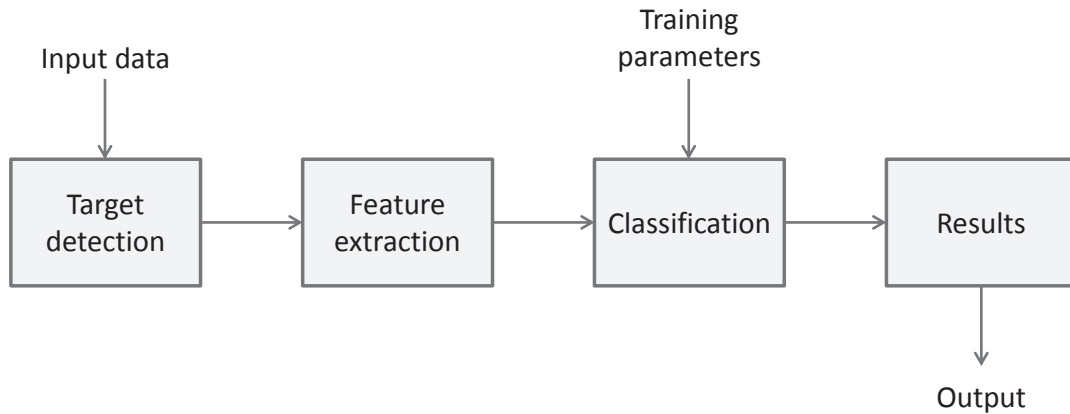


Figure 2.3: Top-level dataflow model for the signal processing core of the DTMSD prototype that we experiment with in this case study.

data segments (selected windows of the input signal) representing potential targets are identified using an adaptive thresholding algorithm. These identified data segments are then processed by a feature extraction stage to provide input to the subsequent classification stage. Two different feature extraction methods, based on frequency domain analysis, are employed. The first method is based on spectral analysis and uses cadence analysis to select the key features to employ for classification [22]. On the other hand, the second method uses cepstral analysis [42].

The classification stage shown in Figure 2.3 is used to process the extracted features to determine if the corresponding signal window identifies a vehicle, a person, or noise (no target). Classification in our DTMSD prototype is performed using alternative support vector machine (SVM) configurations that offer different trade-offs among classification accuracy, real-time performance, and energy consumption. These algorithms include SVMs with linear versus Gaussian kernels.

Since we consider here a 3-class problem, we consider also different approaches

that can be applied on multi-class problems using binary SVM classifiers, including the one-against-all approach, the one- against-one approach [43], and an approach involving a tree-structured combination of binary classifiers [44].

These three multi-class methods were applied in our tracking application, and evaluated as part of the algorithm design stage illustrated in Figure 2.1. We measured the output accuracy in each case for the selected system configurations. In our experiments, the one-against-one approach yielded the best results. Therefore, we chose to implement this approach as part of the Classification block in Figure 2.3.

The SVM classification subsystem employed in the later stages of our prototyping process is composed of two actors — a binary classification actor and a voting actor. These two actors can be viewed as actors that are nested within the Classification block in the hierarchical dataflow graph of Figure 2.3.

For our 3-class problem, we employ a parallel schedule in the Android implementation that involves executing the binary classification actor concurrently 3 times, and then executing the voting actor. This concurrent execution of the binary classification actor is supported by the quad-core processor on the Android device that is targeted in our experiments. Section 2.7 provides more details on the experimental setup that we employed in this study.

2.7 Experiments

In this section, we present results of experiments using the case study involving detection of humans and vehicles described in Section 2.6. The results are presented here

to provide insight into design decisions in our development of the DPTS Framework, and to demonstrate the utility of DPTS in prototyping and demonstrating fully integrated tracking systems on mobile devices.

The mobile device platform used for this experiment is the Motorola Nexus 6. This Android smartphone comes with 3GB RAM and a Qualcomm Snapdragon 805 processor. This processor includes a 2.7 GHz quad-core Krait 450 CPU and Adreno 420 graphics processing unit (GPU). The version of Android OS used in our experiments is 5.0.1.

2.7.1 Execution Time Performance

Table 2.1 shows the performance improvement obtained in our case study by employing the ANT-based development approach used in DPTS over the same application implemented using a conventional SDK-based development approach. The execution time reported here is the average time for processing a single acoustic data set, where the average is taken over 30 data sets, and each data set contains 10,000 samples of acoustic data.

Table 2.1: Execution time comparison between the ANT-based development approach used in DPTS and conventional SDK-based development.

	Total Runtime	Average Runtime	Improvement
SDK-based	2.75×10^2 ms	9.17 ms	baseline
DPTS	1.98×10^2 ms	6.60 ms	28.0%

The results of Table 2.1 show that by incorporating an ANT-based development approach in DPTS, we achieve a 28% improvement in application execution time.

Table 2.2 shows a comparison between the CPU load measured for an SDK-based

Table 2.2: CPU load comparison between the ANT-based development approach used in DPTS and conventional SDK-based development.

	CPU Load				CPU (Average)
	CPU1	CPU2	CPU3	CPU4	
SDK-based	67.5%	41.7%	62.1%	62.7%	58.5%
DPTS	72.4%	2.3%	67.2%	68.0%	52.5%

implementation versus DPTS. The reported values for CPU load are calculated here as the average CPU load for 15 minutes while the mobile device executes the given tracking application. In the column headings of this table, the four cores in the quad-core CPU are referred to as CPU1, CPU2, CPU3, and CPU4. The results in the table show a significant reduction in the average CPU load (across all four cores), and a large variation across different cores of the relative loads resulting from the SDK-based development versus DPTS.

The application used to measure the CPU load values reported in Table 2.2 is the Qualcomm Trepro profiler. The results of Table 2.1 together with Table 2.2 show that DPTS provides a significantly lower overall CPU load, while providing improved execution time as well.

2.7.2 Lines of Code Efficiency

Table 2.3 provides a comparison of the Lines of Code (LOC) between the ANT-based implementation used in DPTS and a corresponding SDK-based implementation for our tracking system case study. LOC, which refers to the number of lines of source code

involved in a given design, is a commonly used metric for evaluating how compact or efficient a given code base is (e.g., see [45,46]). The units of LOC are “lines of code” in the programming language or languages that are used in the given design. LOC is used as a means to estimate costs associated with software system design and maintenance. Based on this interpretation, lower LOC values indicate better designs.

The code base for a tracking system design in DPTS can be partitioned into three parts — algorithm design code (MATLAB-based), actor code, and coordination code. The actor code comes from the LIDE-based signal processing libraries in DPTS. The actor code is designed to be reusable across different applications, and furthermore, actor code is not affected by switching between SDK- and ANT-based development. Similarly, the algorithm design code operates at a high level of abstraction and is unaffected by the choice of mobile development strategy. Thus, we focus in this LOC comparison on the coordination code, which refers to all of the code for a given tracking system prototype that lies outside of the actor implementations and algorithm simulations. This refers to code associated with graph construction, actor scheduling, and parameter management, as well as code (such as makefiles) associated with compiling and building the design.

Table 2.3 provides a comparison of different aspects related to LOC efficiency for the code base associated with our DPTS-based tracking system prototype. As motivated above, the values in Table 2.3 focus only on the coordination code involved in the prototype. Here, *host code* refers to the LIDE-C-based code associated with desktop computer simulation of the tracking system, and *mobile code* refers to tracking code that executes on the targeted Android device. The total code base for the coordination code is the union of the host and mobile code. Thus, lines that need to be changed when migrating a design

Table 2.3: Lines of code (LOC) comparison between SDK-based development and DPTS for tracking system coordination code.

Application	SDK-based	DPTS
Total number of lines (host)	436	436
Number of lines changed	27	0
Number of lines added	55	0
Total LOC (host + mobile)	518	436

from host to mobile code are “counted twice” in the total code base (because two versions of each of these lines must be maintained).

The results in Table 2.3 show a 16% reduction in the code base associated with DPTS compared to a conventional SDK-based design approach. More importantly, the results show that both host and mobile versions of the coordination code are exactly the same, which greatly simplifies the development and maintenance of this code.

Note also that although the coordination code is a relatively small portion of the overall code base (including algorithm design and actor code), the coordination code represents some of the most critical and most frequently changed code that is involved in application fine tuning and exploration across implementation trade-offs. This is because, for example, the coordination code includes code for actor scheduling and buffer management, which have a major affect on implementation metrics in dataflow-based development of signal processing systems [47]. Thus, streamlined management of coordination code is a very useful feature of DPTS.

2.7.3 Energy Efficiency

Table 2.4 shows results of evaluating the energy efficiency of our DPTS-based tracking system prototype, and comparing the results with an SDK-based implementation of the same tracking system functionality. Here, energy efficiency is measured by evaluating the number of acoustic data segments processed for a fixed amount of expended battery capacity. This fixed amount of battery capacity is taken in our experiments to be 322 mAh (10% of the battery capacity on the targeted Android device). We use the same data segment size of 10,000 as reported in Section 2.7.1.

Table 2.4: Energy efficiency comparison.

	SDK-based	DPTS	Improvement
Data Segments Processed	1.34×10^5	1.56×10^5	16.0%
Energy Efficiency (Segments / mAh)	4.19×10^2	4.86×10^2	16.0%

The results in Table 2.4 indicate significant improvements provided by DPTS in the amount of functionality that can be delivered for a given level of battery capacity on a sensor node in the tracking system prototype.

2.7.4 Sensor Experimentation

Another useful feature of DPTS is the capability for efficient interfacing and experimentation with alternative sensors (See Section 2.5.2), which is an important aspect of design space exploration and system-level optimization for tracking systems. Table 2.5 demonstrates experiments performed with two different acoustic sensors — a high quality external sensor subsystem that is connected with an external A/D converter, and

Table 2.5: Comparison between sensors.

Sensor	Built-in Sensor Subsystem	External Sensor Subsystem
Battery Time	3.04×10^4 sec	1.60×10^4 sec
Weight	0g	280g
Price (MSRP)	\$0	\$490
Accuracy (Vehicle)	70.6%	80.4%

the internal acoustic sensor that is built-in to the targeted mobile device. Here, tracking accuracy was measured based on the percentage of vehicles that were detected while 52 vehicles passed. The Battery Time is an extrapolated value that shows the estimated lifetime of the smartphone battery for continuous operation based on the energy drain and elapsed times measured in the experiment. Each weight value shown is the additional weight (in grams) that results in addition to the weight of the smartphone device itself.

2.7.5 Summary of Experimental Results

In summary, the results presented in this section help to validate the capability of DPTS in developing efficient prototypes of tracking systems on mobile devices, and supporting system-level experimentation and design optimization. Additionally, the results provide quantitative insight on the execution time performance, energy efficiency, and LOC efficiency of prototypes developed using DPTS, and its underlying tools, including LIDE and DICE. The results also show significant improvements in multiple dimensions achieved by employing an ANT-based development process as opposed to a more conventional SDK-based development approach within DPTS. These improvements in DPTS come with the limitation that end-user-oriented features (e.g., features that support elaborate user interfaces on the mobile device) and associated libraries that come with

SDK-based development are not available in DPTS. Based on the elaborations given in Section 2.5, this can be viewed as a favorable trade-off for DPTS, where energy efficient, real-time, autonomous execution are more important than supporting features that are optimized for user interaction.

2.8 Summary

In this chapter, we have motivated the use of mobile devices in prototyping autonomous tracking systems for monitoring human and vehicle activity, and we have developed a new rapid prototyping methodology and associated software libraries and tools, called the *DICE-based Prototyping framework for Tracking Systems (DPTS)*. DPTS integrates selected capabilities from the DSPCAD Integrative Command Line Environment (DICE) and Lightweight Dataflow Environment (LIDE), and builds on these capabilities in new ways to enable design, prototyping and optimization of power-efficient tracking systems on mobile devices. Through a case study involving acoustic-sensor-based tracking of humans and vehicles, we demonstrate the utility of the DPTS Framework in validating system functionality; deriving efficient mobile-device-targeted tracking system implementations; experimenting with implementation trade-offs; and integrating different kinds of sensors. Useful directions for future work include exploring the adaptation of DPTS for use with other relevant dataflow tools (in addition to LIDE), and other kinds of sensor network applications.

Chapter 3: An Accumulative Fusion Architecture for Discriminating People and Vehicles using Acoustic and Seismic Signals

In this chapter, we develop new multiclass classification algorithms for detecting people and vehicles by fusing data from a multimodal, unattended ground sensor node. The specific types of sensors that we apply in this work are acoustic and seismic sensors. We investigate two alternative approaches to multiclass classification in this context — the first is based on applying Dempster-Shafer Theory to perform score-level fusion, and the second involves the accumulation of local similarity evidences derived from a feature-level fusion model that combines both modalities. We experiment with the proposed algorithms using different datasets obtained from acoustic and seismic sensors in various outdoor environments, and evaluate the performance of the two algorithms in terms of receiver operating characteristic and classification accuracy. Our results demonstrate overall superiority of the proposed new feature-level fusion approach for multiclass discrimination among people, vehicles and noise.

The work presented in this chapter has been published in [8].

3.1 Introduction

Detection and classification of people and vehicles in outdoor environments is important in various applications related to defense, border patrol, and surveillance. For example, such capabilities help to guard specific regions against enemy intrusion and attack, and to protect borders between countries. In these applications, acoustic and seismic sensors are frequently employed because of their power efficiency and reduced computational requirements compared to other sensing modalities, such as image-based sensing.

Signals from acoustic and seismic sensors have different spectral characteristics in the presence of people and vehicles. This diversity in sensor response provides potential for greater accuracy when signals from both modalities are fused as opposed to solutions that employ only acoustic or only seismic sensors. Voices of people typically generate acoustic signals in the range of 200–800 Hz, while footsteps of people generate seismic signals in the range of 1.9–2.79 Hz [22]. For vehicles, Altmann [48] analyzes spectral characteristics of sets of signals collected from acoustic and seismic sensors. This analysis reveals similarities and differences in signal characteristics between the two modalities along with their influences from factors that include the engine rotation rate, number of engine cylinders, vehicle speed, and track element length (for tracked vehicles).

In this chapter, we investigate fusion algorithms for multiclass classification among people, vehicles, and noise (the absence of people or vehicles) using signals from acoustic and seismic sensors. We develop and comparatively evaluate two different multiclass algorithms, a score-level fusion algorithm that is based on Dempster-Shafer Theory (DST), and an accumulative algorithm that exploits feature-level fusion. Through an ex-

tensive experimental comparison, we demonstrate that our feature-level fusion algorithm achieves significantly better classification performance compared to the DST-based approach.

A distinguishing aspect of our work is our focus on fusion techniques for multiclass classification using both acoustic and seismic signals. This complements related prior work that has investigated binary classification using acoustic and seismic signal processing, but has not addressed multiclass classification problems (e.g., see [12]). Also, previous work on multiclass classifiers for people, vehicles, and noise (e.g., see [7,39] has emphasized use of acoustic signals. In contrast to these works, this chapter contributes fusion techniques for classification using both acoustic and seismic signals.

3.2 Related Work

Various algorithms can be applied naturally to multiclass classification problems. These include k -nearest neighbor [49], decision trees [50,51], neural networks [52], and naive Bayes classifiers [53]. Other algorithms convert a multiclass problem into a set of binary classification problems, which are then solved using more powerful binary classifiers. The techniques that we develop belong to this second class of algorithms. We decompose our targeted multiclass classification problem into three binary classification problems — noise vs. person, noise vs. vehicle, and person vs. vehicle.

A fusion architecture for distinguishing between people and animals using different ultrasonic, seismic and passive infrared sensors is proposed in [12]. In this work, the decisions of different binary classifiers are fused to detect targets (people/animals), and

to distinguish between the people and animal classes whenever a target is detected. Our work differs from this work in that we incorporate a feature-level fusion approach; we address multiclass classification among noise, people, and vehicle classes; and we employ acoustic and seismic sensor types.

Dempster-Shafer Theory (DST) [54] is a common approach used for late fusion, where information from multiple classifiers are combined to produce a single output. For example, Lee et al. [55] apply DST to integrate decisions of classification and detection, and demonstrate that this integration improves the performance of both classification and detection. Wu et al. [56] propose general methods for fusing the signals from multiple sensors to perform binary classification tasks.

Accumulative methods, like the Hough transform [57], have demonstrated excellent performance in a wide range of pattern recognition problems, including image registration [58] and biometrics [59]. The methods used in [58, 59] accumulate local similarity evidences (i.e. probabilities), which are provided by explicitly estimating the probability density function (pdf) over the feature space. The disadvantages of explicitly computing a pdf are efficiency and scalability.

As discussed in Section 3.1, the key distinguishing aspect of our work in this chapter compared to related work in the literature is our joint consideration of seismic signal processing, acoustic signal processing, and multiclass classification for border patrol and related sensor network applications. Additionally, we propose an accumulative fusion framework where the pdf is learned implicitly through an SVM.

3.3 Fusion Framework

In this section, we propose two fusion algorithms for multiclass classification using signals from an acoustic-seismic node. The first is an adaptation to score-level fusion using DST for multiclass classification. We view this approach as a baseline in our experiments to assess our second approach, which is the main fusion approach that is presented in this chapter. This second approach involves the accumulation of similarity evidences derived from a local feature-level fusion model. We refer to this second approach as Accumulation of Local Feature-level Fusion Scores (ALFFS).

3.3.1 Cepstral Analysis and SVM Classification

For both acoustic and seismic signals in the baseline (DST-based) and ALFFS approaches, we employ cepstral analysis for feature extraction [60]. We extract cepstral coefficients using the feature extraction method described in [39, 42]. In cepstral analysis, DC components are removed, low order coefficients characterize the slow spectrum variation, and higher order coefficients characterize the fundamental frequency.

For each sensing modality, we select the first 50 cepstral coefficients for training and testing. We apply SVM classifiers [61, 62] with polynomial kernels for binary classification using the extracted features for each modality. The integration of multimodal features and SVM classifiers in the baseline and ALFFS fusion architectures is illustrated in Figure 3.1 and Figure 3.2, respectively. In Section 3.3.2 and Section 3.3.3, we elaborate on the design of these two alternative fusion architectures.

In Figure 3.1, 3.2 and throughout the remainder of this chapter, we abbreviate

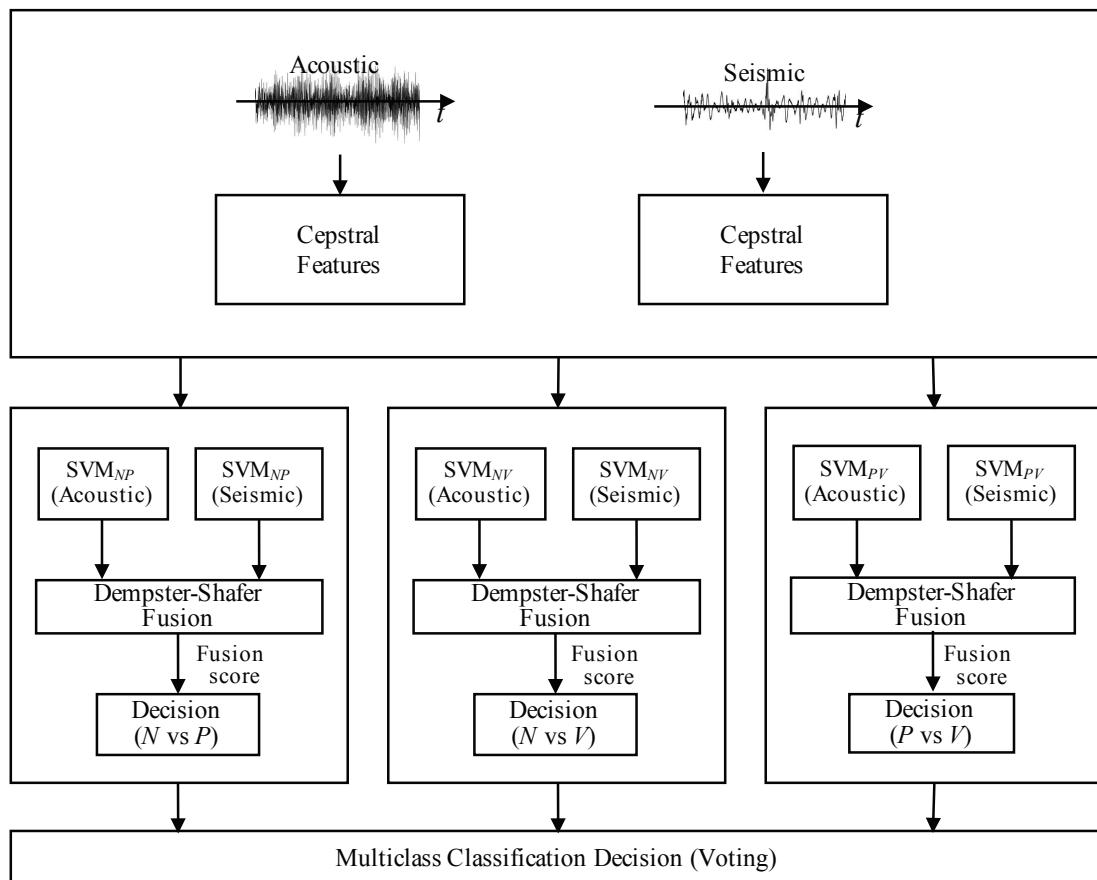


Figure 3.1: Fusion architecture using DST.

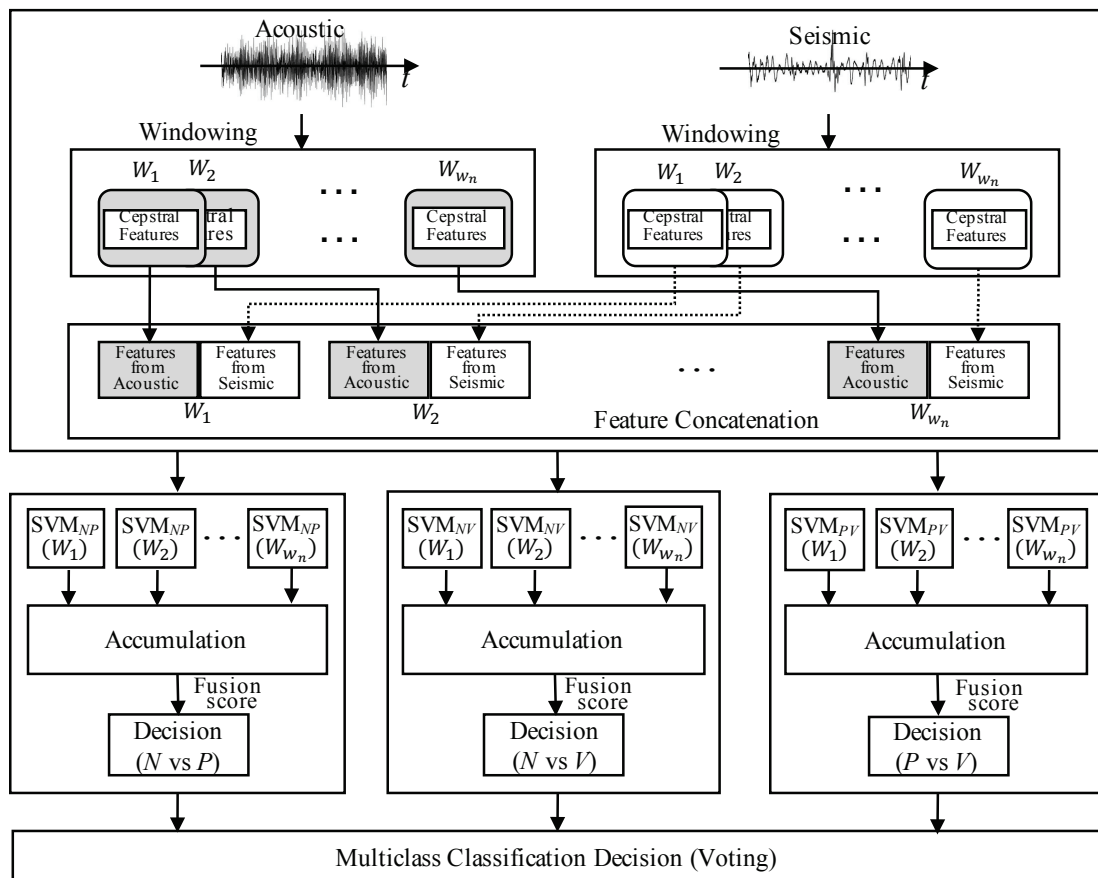


Figure 3.2: Fusion architecture using ALFFS.

“noise”, “person”, and “vehicle” — the three available decision classes — by N , P , and V , respectively. We denote the set of all available decision classes as $\Delta = \{N, P, V\}$.

3.3.2 Baseline Fusion Architecture

In our baseline fusion architecture, we adapt score-level fusion with DST to perform multiclass classification. For each *distinct pair of decision classes (DPDC)*, we employ DST fusion without weights as in [56]. Specifically, suppose that we have a pair of binary SVM classifiers $Z[\rho] = \{C_\alpha[\rho], C_\sigma[\rho]\}$ that discriminate between the two elements of a DPDC $\rho = \{X, Y\} \subset \Delta$ based on signals of type α and σ , where α and σ represent the acoustic and seismic sensing modalities, respectively. Then based on DST, the score $S(Z, A)$ associated with classifier pair Z and decision class $A \in \rho$ can be expressed as

$$S(Z, A) = \frac{\text{Belief}_A}{\text{Belief}_{\neg A}} = \frac{\sum_{E_{\rho,\alpha} \cap E_{\rho,\sigma} = A} E_{\rho,\alpha} E_{\rho,\sigma}}{\sum_{E_{\rho,\alpha} \cap E_{\rho,\sigma} = \neg A} E_{\rho,\alpha} E_{\rho,\sigma}}. \quad (3.1)$$

Here, $\neg A$ is the element of ρ other than A . Additionally, $E_{\rho,x}$ denotes the evidence associated with ρ that is derived from sensing modality $x \in \{\alpha, \sigma\}$. The value of $E_{\rho,x}$ for each modality x can be derived from the scores of the two associated SVM classifiers.

Equation 3.1 can be viewed as a standard DST-based approach to binary classification (for discrimination between A and $\neg A$) using SVM-based binary classifier subsystems. We extend this approach to multiclass classification by instantiating 3 different pairs of SVM classifiers $\{Z[\rho] \mid \rho \in \{\{N, P\}, \{N, V\}, \{P, V\}\}$, where each of these classifier pairs is connected to a fusion subsystem that operates based on Equation 3.1. The results from these 3 fusion subsystems are then combined using voting, as illustrated

in Figure 3.1. Similar to [63, 64], the voting method chooses the class that is classified most frequently by the three SVMs.

3.3.3 ALFFS

Our ALFFS approach is motivated by the significant differences in spectral characteristics between acoustic and seismic signals. To systematically incorporate these different characteristics into the multiclass classification process, ALFFS applies concatenated features that are derived from both acoustic and seismic inputs.

Algorithm 1 presents a pseudocode representation of the ALFFS approach. In the signal processing system represented in Algorithm 1, the subscripts α and σ are used to represent correspondence with acoustic and seismic signals, respectively, as in Section 3.3.2. The input to the system consists of data frames (segments of contiguous signal samples) Γ_α and Γ_σ , and two parameters w_n and w_r , which respectively specify the number of windows and the ratio of inter-window overlap that are to be employed when processing the input frames. The two signals Γ_α and Γ_σ are corresponding acoustic and seismic signals, meaning the two modalities observe the same activity.

In the first two steps of Algorithm 1, a windowing function *Window* decomposes the input data frames into overlapping windows consisting of w_n samples each, where the ratio of overlap is determined by the parameter w_r . The function $f_{cepstral}$ is a function that returns cepstral features for a given window of signal samples. The concatenation of acoustic and seismic features for each window is performed by the function f_{concat} .

The outer `for` loop (line 8) iterates through all relevant DPDCs. For each DPDC

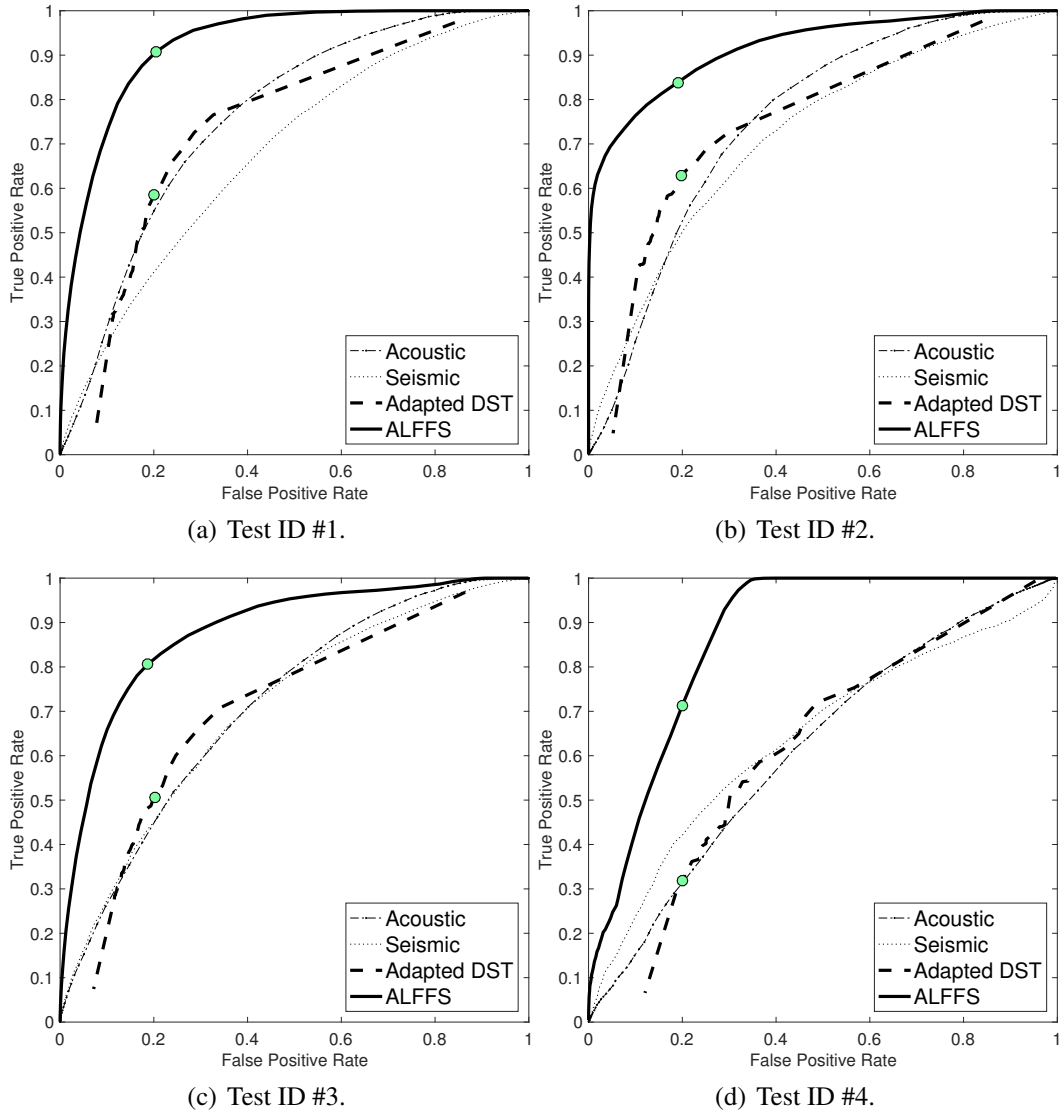


Figure 3.3: ROC curves for multiclass classification.

Algorithm 1: A pseudocode representation of the ALFFS approach.

Input : $\Gamma_\alpha, \Gamma_\sigma, w_n, w_r$

Output: *Class*

```

1  $D_\alpha(1), D_\alpha(2), \dots, D_\alpha(w_n) \leftarrow \text{Window}(\Gamma_\alpha, w_n, w_r)$ 
2  $D_\sigma(1), D_\sigma(2), \dots, D_\sigma(w_n) \leftarrow \text{Window}(\Gamma_\sigma, w_n, w_r)$ 
3 for  $i = 1$  to  $w_n$  do
4    $F_\alpha(i) \leftarrow f_{\text{cepstral}}(D_\alpha(i))$ 
5    $F_\sigma(i) \leftarrow f_{\text{cepstral}}(D_\sigma(i))$ 
6    $F_{\text{concat}}(i) \leftarrow f_{\text{concat}}(F_\alpha(i), F_\sigma(i))$ 
7 end
8 for  $p \in \{\{N, P\}, \{N, V\}, \{P, V\}\}$  do
9   for  $j = 1$  to  $w_n$  do
10     $\text{Score}_p(j) \leftarrow \text{SVM}_p(F_{\text{concat}}(j))$ 
11   end
12    $\kappa(p) \leftarrow \text{Score}_p(1) + \text{Score}_p(2) + \dots + \text{Score}_p(w_n)$ 
13    $R(p) \leftarrow f_{\text{dec},p}(\kappa(p))$ 
14 end
15  $\text{Class} \leftarrow f_{\text{voting}}(R(\{N, P\}), R(\{N, V\}), R(\{P, V\}))$ 

```

p and window index j , the algorithm computes a binary classification score $Score_p(j)$ by applying an SVM classifier SVM_p that is trained specifically for DPDC p . Line 12 then accumulates all of the scores for the given DPDC p to provide a single composite score $\kappa(p)$ across all windows and both sensing modalities. This composite score is then thresholded by the decision function $f_{dec,p}$ to produce the decision $R(p)$ associated with DPDC p . In our experiments, we use a common threshold of 0 for all three decision functions $\{f_{dec,p}\}$.

The three decisions $\{R(x)\}$ are then operated on using a voting process, represented by the function f_{voting} , to produce the final multiclass classification result $Class$. We use the same voting process here as in the adapted DST approach of Section 3.3.2.

3.4 Experiments

In this section, we present an experimental evaluation of the Adapted DST and ALFFS approaches, which were introduced in Section 3.3.2 and Section 3.3.3, respectively. In our evaluation, we employ 4 different datasets, which we refer to as Datasets #1–#4. Datasets #1–#3 were collected on Spesutie Island at the Aberdeen Proving Grounds in Maryland, USA during July 28–30, 2015. These three datasets were collected from different sensors installed in different locations and at different times of day. Further details about Datasets #1–#3 can be found in [65]. Dataset #4 was collected at the US Army Research Laboratory, Adelphi, Maryland, USA on September 16, 2013. Datasets #1–#3 were collected from soil, while Dataset #4 was collected from asphalt. Each dataset contains 1000 data frames, where each frame contains 6 seconds of acoustic and seismic data

sampled at 4096Hz.

For training and testing, we used input data segments (IDSs) that each consist of 500 contiguous data frames from one of the four datasets. For training, we randomly extracted 50 different IDSs from Dataset #1 using the MATLAB `crossvalind` function. Similarly, for testing, we used `crossvalind` to extract 50 different IDS from each of the four available datasets. Thus, we employed 50 IDSs for training, and 200 IDSs for testing. We refer to the set of 50 IDS used for testing that we extracted from each Dataset #X as “Test ID #X”. For ALFFS, we used $w_n = 50$ and $w_r = 0.4$.

To evaluate classification performance, we compared the Adapted DST and ALFFS approaches in terms of their measured ROC curves and accuracy levels. Among the different ways to compute ROC curves for multiclass problems, we employed the method discussed in [66], which is suitable for multiclass classifiers that are composed of binary classifiers. In this method, the multiclass ROC curve is computed by averaging the ROC curves across the corresponding set of pairwise (1-to-1) classifiers. Figure 3.3 and Table 3.1 show the measured ROC curves and accuracy levels, respectively. From these results, we see that the Adapted DST approach shows no significant performance improvement compared to the single-modality classifiers. In contrast, ALFFS exhibits significant improvements compared to the single-modality classifiers, as well as the Adapted DST approach. Specifically, ALFFS achieves 0.908, 0.839, 0.806, and 0.712 true positive rate when operating at 0.2 false positive rate for Test ID #1-#4, respectively. Whereas, the baseline approach achieves 0.586, 0.628, 0.507, and 0.319 at the same false positive rate. Thus, ALFFS achieves fewer false alarms, even when only using a single seismic and single acoustic source. The results in Table 3.1 show that ALFFS achieves an abso-

Test ID	Acoustic	Seismic	DST	ALFFS
#1	64.6	59.3	66.3	86.1
#2	61.4	57.8	60.3	73.8
#3	56.9	51.7	57.2	73.4
#4	53.1	67.2	61.3	76.9
avg.	59.0	59.0	61.3	77.6

Table 3.1: Accuracy comparison (%).

lute improvement of 16.3% (relative improvement of 26.6%) in accuracy compared to the baseline fusion on average.

3.5 Summary

In this chapter, we have introduced an algorithm, called Accumulation of Local Feature-level Fusion Scores (ALFFS), for multiclass classification among people, vehicles, and noise using a single unattended ground sensor node. ALFFS operates by extracting cepstral features, applying feature-level fusion, and applying a bank of support vector machines across sets of concatenated features that are extracted from overlapping windows of the multimodal input signals. We have also introduced an adaptation to our targeted multiclass classification problem of sensor fusion based on Dempster-Shafer Theory (DST). Through extensive experiments, we have demonstrated that ALFFS achieves an average of 16.3% (26.6%) absolute (relative) improvement over the adapted DST approach. Moreover, ALFFS achieves a significant reduction in the number of false alarms compared to the adapted DST approach (and the individual modalities).

Chapter 4: An Optimized Embedded Target Detection System using Acoustic and Seismic Sensors

In this chapter, we build on recent algorithmic advances in sensor fusion, and present the design and implementation of a novel, multi-mode embedded signal processing system for detection of people and vehicles using acoustic and seismic sensors. Here, by “multi-mode”, we mean that the system has available a complementary set of configurations that are optimized for different trade-offs. The multi-mode capability delivered by the proposed system is useful to supporting long lifetime (long term, energy-efficient “standby” operation), while also supporting optimized accuracy during critical time periods (e.g., when a potential threat is detected). In our target detection system, we apply a strategically-configured suite of single- and dual-modality signal processing techniques together with dataflow-based design optimization for energy-efficient, real-time implementation. Through experiments using a Raspberry Pi platform, we demonstrate the capability of our target detection system to provide efficient operational trade-offs among detection accuracy, energy efficiency, and processing speed.

The work presented in this chapter has been published in [8].

4.1 Introduction

Sensor networks for detection of targets such as people and vehicles are of great relevance in defense and security applications. In such networks, use of non-image sensors, such as acoustic and seismic sensors, are of interest in part because of their power efficiency compared to image sensors. Various studies have been focused on development and enhancement of acoustic and seismic signal processing algorithms for high target detection accuracy.

For large-scale deployment of such networks, it is critical to provide methods for their cost- and energy-efficient realization, while providing high detection accuracy and low false alarm rate. In support of these objectives, a significant body of research has focused on the development of novel algorithms for fusion, target detection, and classification from acoustic and seismic signals (e.g., see [12, 22, 67]). In this chapter, we develop design optimization methods that are complementary to this body of prior algorithm-oriented work. In particular, we focus on system design and implementation issues that are important for delivering the accuracy offered by relevant fusion/detection algorithms along with energy-efficient and resource-constrained execution capability on low cost sensor node platforms.

To balance objectives of low average energy consumption (streamlined standby operation) and optimized accuracy during times of critical operation (e.g., when potential threats are actively being monitored), we develop a novel, multi-mode system design that provides alternative configurations to support optimized trade-offs for these standby and critical operation scenarios. Transitions between these modes can then be triggered based

on specific application requirements — for example, transitions may be manually-driven by personnel operating a monitoring station or they may be triggered automatically using some kind of finite state machine logic.

In this chapter, we apply a dataflow-based methodology for model-based implementation and design optimization of the proposed multi-mode target detection system. Dataflow methods are widely used in many areas of signal processing system design (e.g., see [3]). In addition to supporting design optimization, our application of dataflow methods helps to promote reliability and efficiency of the developed implementation, as well support the retargetability of the system to other types of sensor node platforms. We provide extensive experimental results to motivate the use of alternative modes in our dataflow-based target detection system design, and to quantify the useful range of operational trade-offs provided by the different modes.

4.2 Related Work

Various algorithms have been proposed that are relevant to *person-and-vehicle detection (PVD)* using energy-efficient sensing modalities, including acoustic and seismic modalities. For example, Dibazar et al. develop neural networks that operate on seismic signals from footsteps and vehicles [67]. Damarla and Kaplan develop a decision-level fusion architecture for tracking groups of people using acoustic and seismic signal processing [12]. Ben Salem et al. present an adaptive target detection system that employs mobile devices as sensor node platforms, and applies different acoustic signal processing techniques for different signal-to-noise ratio conditions, and energy consumption con-

straints [68]. Our work in this chapter differs from these prior works in that we simultaneously handle (1) multiple sensing modalities (acoustic and seismic); (2) both decision and feature level fusion for improved accuracy; and (3) design optimization for energy- and resource-constrained embedded implementation.

In this chapter, we build on our recent algorithmic investigation of PVD using acoustic and seismic signals [8]. In this investigation, we introduced an adaptation to PVD of sensor fusion based on Dempster-Shafer Theory (DST) [54, 56], and we also introduced a PVD algorithm, called Accumulation of Local Feature-level Fusion Scores (ALFFS). ALFFS extracts cepstral features, and applies an accumulative, feature-level fusion approach. Our work in this chapter employs the algorithms developed in [8] as a starting point, and addresses system design and implementation challenges that are critical to practical deployment of the algorithms. The system design and implementation-oriented focus of this work is significantly different from that of [8], which is focused on algorithmic aspects.

4.3 System Design

The input to our PVD system consists of multi-modal sensor streams that arrive from a pair of sensors — one acoustic sensor and one seismic sensor. Here, we use the term *multi-modal* to represent multiple sensing modalities, while *multi-mode* refers to the incorporation of alternative operational modes with complementary trade-offs. The PVD system operates on fixed-length frames of signal samples. The number of samples per frame is determined by two system parameters — the frame duration t_f , and sample rate

N_r .

The output of our PVD system is a sequence (y_1, y_2, \dots) of target classification results, where each $y_i \in \{P, V, N\}$ provides the derived detection result for the i th frame. Here, P , V , and N correspond, respectively, to detection of a person, a vehicle, or noise (the absence of any person or vehicle), respectively. This is referred to as a *multi-class* classification system since the system must discriminate across more than two classes.

Both the DST-based and ALFFS approaches employed in our PVD system employ support vector machine (SVM) subsystems as core building blocks for the classification process. SVMs are widely used in machine learning applications due to their robustness and classification performance (e.g., see [69]). In each mode of our PVD system, we apply multiple, *Binary SVM Classifiers (BSCs)* along with voting logic to perform the targeted multiclass classification task. In particular, we employ BSCs that are configured to perform P vs. N , P vs. V , and V vs. N classification. These BSCs are embedded in different ways into different PVD architectures that are associated with the alternative modes. These different embeddings provide an efficient range of operational trade-offs, which we will demonstrate quantitatively in Section 4.4.

Our PVD system involves four modes of operation, which we refer to as the *acoustic* mode, *seismic* mode, *DST fusion* mode, and *ALFFS* mode. These modes are denoted as μ_{ac} , μ_{sei} , μ_{dst} , and μ_{alf} , respectively. The modes provide progressively higher levels of accuracy, while the latter two modes — which involve dual-modality processing — consume higher levels of energy and require longer run-time. The first mode μ_{ac} provides lower accuracy compared to all of the other modes, while providing no significant benefit, as evaluated on our target platform, in terms of the run-time or energy efficiency.

Thus, μ_{ac} is disabled in the final implementation. However, the mode is useful to have available for experimentation purposes, and for enhanced configurability. For example, the accuracy of μ_{ac} may improve significantly if the design is (1) retargeted to a platform that employs a higher quality acoustic sensor or (2) adapted to an application in which acoustic signals provide better discrimination potential compared to seismic signals (e.g., speech detection or recognition).

In Section 4.3.1 through Section 4.3.3, we present dataflow graph specifications for the different modes in our PVD system. The presentation here is focused on highlighting relevant aspects of the embedded software architecture. For details of the underlying algorithms, we refer the reader to [8].

Each dataflow graph, including all of its encapsulated actors (dataflow graph vertices) and connections (graph edges), is implemented using LIDE which is introduced in Chapter 2. More specifically, our dataflow graph implementations employ LIDE-C, which is the integration of LIDE with the C programming language. We have used LIDE-C for design and implementation of the entire PVD system, including the dataflow graphs for the different modes.

Actors in LIDE-C, as in other dataflow tools, execute in terms of discrete units of execution, which we refer to as *firings*. As the enclosing signal processing application operates on successive samples or frames of data, each dataflow actor in general executes iteratively through a sequence of successive firings. The operation of an actor is often explained in terms of the computation it performs in a single firing.

4.3.1 Single-Modality Operation

Figure 4.1 illustrates the dataflow graph employed in our design and implementation of the two single-modality modes, μ_{ac} and μ_{sei} . These two modes use exactly the same actors and edges. The key difference in the dataflow graph configurations between the two modes is that different sets of parameters are employed by the SVMs within the SVM Bank actor.

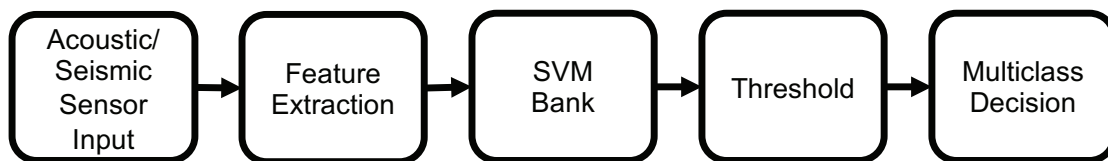


Figure 4.1: Dataflow graph for single-modality PVD.

The Acoustic/Seismic Sensor Input (“Sensor Input”) actor injects digitized samples that are produced by an analog/digital (A/D) converter that is interfaced to the relevant sensor device — that is, interfaced to the acoustic sensor in μ_{ac} and the seismic sensor in μ_{sei} . In our implementation of this sensor input actor, we employ C-based Linux libraries that are developed for GNU-Linux platforms, including x86/64 Linux and ARM-based Linux platforms such as the Odroid and Raspberry PI platforms [70].

The Feature Extraction actor in Figure 4.1 applies cepstral analysis to extract features from the digitized samples arriving from the sensor source. For FFT computation, we employ a LIDE-C wrapper around an optimized module from the FFTW library [71]. On each firing, the Feature Extraction actor consumes N_s samples, corresponding to a single frame of sensor data, and produces N_c cepstral coefficients, where N_c is a parameter of the actor, and N_s can be derived as the product of the system frame duration t_f

and sample rate N_τ . The generated cepstral coefficients are subsequently employed (in the downstream portion of the dataflow graph) as the features of the input frame.

The cepstral features extracted from the Feature Extraction actor are sent as input to the SVM Bank actor, as illustrated in Figure 4.1. The SVM Bank actor, like the Feature Extraction actor, is an important actor in all of the different modes of our PVD system. The SVM Bank actor applies three different BSCs, which we denote by β_{pv} , β_{pn} , β_{vn} . These BSCs are trained, at design time (offline), to discriminate respectively between P vs. V , P vs. N , and V vs. N . Recall that P , V , and N , respectively represent the decision classes “person”, “vehicle”, and “noise”. In a given firing of the SVM Bank actor, each of the three encapsulated BSCs produces a real-valued score ϕ . The sign (negative or positive) of ϕ indicates the predicted decision class (between the two candidate classes), and the absolute value of ϕ provides an indicator of the strength or “confidence” of the prediction. On each firing, the SVM Bank actor produces as output three real-valued, scalar outputs, which correspond to the scores generated by the three encapsulated BSCs.

The Threshold actor in the dataflow graph consumes a block of real values x_1, x_2, \dots, x_M , and simply applies a threshold τ to each one to produce a binary output. In all of our applications of the Threshold actor in the PVD system, the block size M is equal to 3, and each input block corresponds to scalar scores associated with P vs. V , P vs. N , and V vs. N discrimination. The threshold τ , however, is not identical in all PVD modes. In the case of Figure 4.1, we apply $\tau = 0$ for both modalities. The output of the Threshold actor is in general a block z_1, z_2, \dots, z_M of binary values, where for each i , $z_i = 0$ if $x_i < \tau$, and $z_i = 1$ if $x_i \geq \tau$. In the case of μ_{ac} and μ_{sei} , these binary values correspond to prediction results for each of the BSCs employed within the SVM Bank.

The Multiclass Decision actor in Figure 4.1 is used to combine blocks of binary prediction results into a corresponding stream of multiclass decision results. Again, we use block size $M = 3$. The input block consists of a triplet of binary decisions, (z_1, z_2, z_3) , corresponding to β_{pv} , β_{pn} , and β_{vn} , respectively. For example, $z_1 = 0$ if the BSC β_{pv} has generated a prediction of P for the most recent signal frame, and $z_1 = 1$ if β_{pv} has predicted V . The Multiclass Decision actor applies a simple voting rule to generate a single classification result from within the set $\{P, V, N\}$. In case of a tie (all three input predictions are different), the actor produces N as the classification result.

4.3.2 DST-based Fusion Mode

Figure 4.2 shows the dataflow graph for the dual-modality mode μ_{dst} . In this graph, the Acoustic Sensor Input and Seismic Sensor Input actors can be viewed as multiple concurrent instantiations of the single sensor input actor in Figure 4.1. These actors inject digitized data acquired from both sensing modalities for processing and fusion in the downstream portion of the dataflow graph.

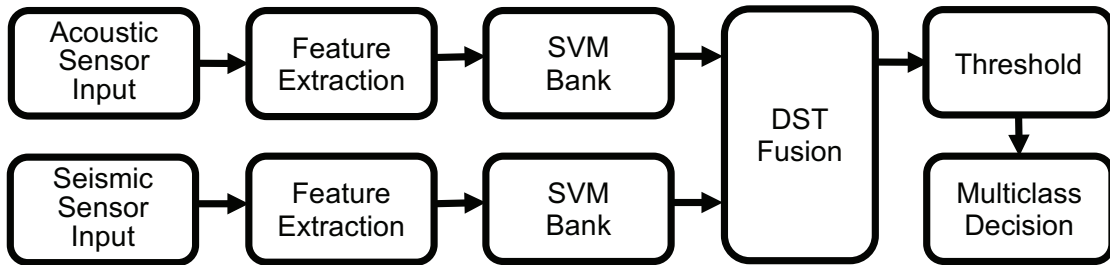


Figure 4.2: Dataflow graph for the dual-modality mode μ_{dst} .

The Feature Extraction actors in the μ_{dst} dataflow graph represent multiple instantiations — one for each sensing modality — of the actor with the same name in Figure 4.1.

The cepstral coefficients extracted by each of these two Feature Extraction actors are processed by separate SVM Bank actors. The SVM Bank actors in Figure 4.2 are identical to the corresponding actor in Figure 4.1; however, they are configured differently at design time. Each of the BSCs encapsulated within the SVM Bank in the upper (acoustic) branch is trained for the associated binary classification task based on acoustic data, and similarly, the training for the lower SVM Bank actor is based on seismic data. In other words, both trained versions of the SVM Bank actor in Figure 4.1 are instantiated concurrently in Figure 4.2.

The DST Fusion actor in Figure 4.2 is the only “new” actor in this dual-modality dataflow graph compared to Figure 4.1. This actor applies a dual-modality fusion algorithm based on Dempster-Shafer Theory, as mentioned in Section 4.2. For complete details on this algorithm, we refer the reader to [8]. In a given firing, the DST Fusion actor takes as input two frames of cepstral coefficients $a(1), a(2), \dots, a(N_c)$ and $s(1), s(2), \dots, s(N_c)$, which are extracted as features from the corresponding acoustic and seismic input signal frames. From the results of its underlying fusion algorithm, the actor then produces (similar to the SVM Bank actor in Figure 4.1) three real-valued, scalar outputs, which represent binary classification scores for discrimination between P vs. V , P vs. N , and V vs. N , respectively. In the case of the DST Fusion actor, each output score σ is a non-negative real number with $\sigma < 1$ corresponding to one decision class and $\sigma > 1$ corresponding to the other.

The Threshold actor in Figure 4.2 applies block size $M = 3$ and threshold $\tau = 1$ to produce, on each firing, a triple of binary prediction results. This triple is then processed by the Multiclass Decision actor to produce a single PVD classification result from the

set $\{P, V, N\}$.

4.3.3 ALFFS Mode

Figure 4.3 shows the dataflow graph for the second dual-modality mode, which is μ_{alf} . As with the μ_{dst} subsystem, input samples are injected into this graph using the Acoustic Sensor Input and Seismic Sensor Input actors. Overlapping windows of samples from each input frame are then processed using the two instances of the Feature Extraction actor shown in Figure 4.3.

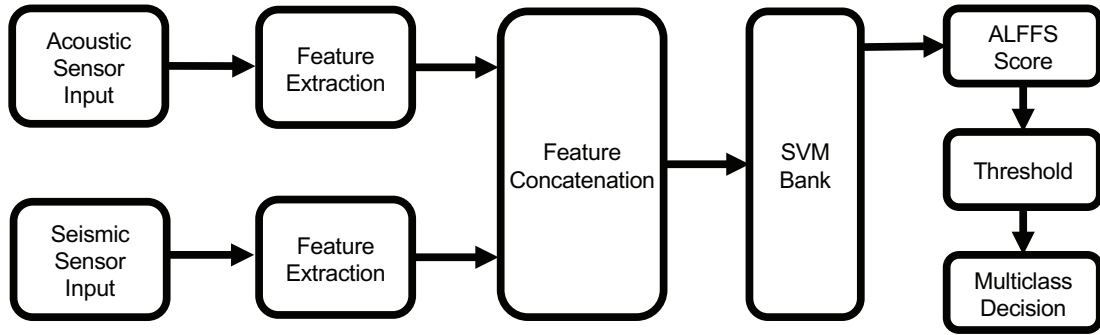


Figure 4.3: Dataflow graph for dual-modality PVD using ALFFS.

In μ_{alf} , each Feature Extraction actor is configured to process overlapping windows of input data through appropriate setting of two actor parameters, called the *threshold parameter* and *consumption parameter*. These parameters, denoted respectively by *thr* and *cons*, control the flow of data from the first-in, first-out (FIFO) buffer that corresponds to the input edge e_{in} of the actor. The threshold parameter specifies the number of samples that must be present on e_{in} before the actor can be fired, and the *cons* parameter specifies how many tokens are consumed (removed) from e_{in} on each firing. This results in a processing pattern whereby successive firings of each Feature Extraction actor process

overlapping windows of data, where each window contains thr samples, and adjacent windows contain $(thr - cns)$ samples of overlapping data.

This use of distinct threshold and consumption parameters is closely related to a similar distinction that is part of the computation graph model [72]. The parameters can be implemented efficiently through a straightforward adaptation of LIDE where the *enable function* (used to determine whether a LIDE actor can be fired) is controlled by the threshold parameter and the *invoke function* (which executes an actor firing) consumes samples based on the consumption parameter. This type of threshold- and consumption-parameterized feature extraction provides efficient sliding window operation.

In the Feature Extraction configurations used in Figure 4.1 and Figure 4.2, $thr = cns$, and the window size in the enclosing dataflow graphs is effectively equal to the input frame size (i.e., multi-window processing is not employed). The μ_{alf} mode is the only PVD system mode that requires $thr \neq cns$, due to the windowed behavior of the underlying ALFFS algorithm [8].

In particular, in ALFFS, feature extraction is performed on N_w overlapping windows of a given input frame. The N_c cepstral coefficients extracted from each window are processed by the SVM Bank actor, as shown in Figure 4.3. The coefficients extracted from each pair of corresponding windows associated with the two sensing modalities are concatenated by the Feature Concatenation actor before arriving as input to the SVM Bank actor. SVM Bank then fires N_w times, and processes a $(2 \times N_c)$ -element feature vector on each firing. This results in a total of $3 \times N_w$ values that arrive at the input of the ALFFS Score actor. These values represent the collection of binary prediction triples (P vs. V , P vs. N , and V vs. N) for all of the windows. Corresponding elements of the

triples are added in the ALFFS actor, and the resulting sums are analyzed, as illustrated in Figure 4.3, to derive the final multiclass classification result for the multi-modal input frame. For further details on the algorithm that underlies the ALFFS mode, we refer the reader to [8].

4.4 Experiments

In this section, we present an experimental evaluation of the multi-mode PVD system design presented in Section 4.3. We compare the run-time and energy consumption performance of the different system modes using the same input data. For this purpose, pre-collected input frames (frames of acoustic and seismic signals) are stored within flash memory on the targeted embedded platform. The “sensor input” actors in Figure 4.1, Figure 4.2 and Figure 4.3 are configured in these experiments to read the pre-collected data from flash memory and inject it into the associated dataflow graphs for processing. The sensor input actors in practice obtain the data directly from the sensors. However, the purpose in these experiments is to demonstrate how the proposed methods provide optimized trade-offs for improving processing capabilities at the network edge.

The pre-collected data used in these experiments is obtained from datasets that were collected from acoustic and seismic sensors on Spesutie Island at the Aberdeen Proving Grounds in Maryland, USA. Further details about these datasets can be found in [65]. We employed a dataset that consists of 1000 data frames, where each frame contains 6 seconds of acoustic and seismic data. 500 of these frames were used in our experiments for training, and the other 500 frames were used for testing.

The target platform that we used in our experiments is the Raspberry Pi 3 Model B, which is equipped with 1GB RAM, a 4x ARM Cortex A53 CPU, and a Broadcom VideoCore IV GPU. The operating system used was Raspbian 4.4. The device we used for measuring power consumption is the Tektronix Keithley Series 2280 Precision Measurement DC Power Supply.

The energy consumption of the sensors is not included in the values reported in this section. This is because separate energy sources may be used for the sensors, and our intent is to focus in the chapter on trade-offs between *processing efficiency* (energy and speed) versus *accuracy* for alternative signal processing techniques. However, the design methodology applied in this chapter can be readily adapted to develop a multi-mode system whose modes are selected in a manner that takes into account the energy efficiency of the sensors. Although developing such adaptations may be useful, it is beyond the scope of this work but may be addressed in the future with large scale sensor networks.

Various parameter values employed in our experiments are summarized in Table 4.1.

Table 4.1: Parameter values used in our experiments.

Description	symbol	value	units
Frame duration	t_f	6	seconds
Sample rate	N_r	4096	Hz
Number of cepstral coefficients	N_c	50	
Number of windows (ALFFS)	N_w	50	
Window overlap ratio (ALFFS)	w_r	0.4	

Table 4.2 shows the measured accuracy of the four different modes in our PVD system. The accuracy is measured as (z_c/F) , where z_c is the number of correct classifica-

tions, and F is the number of frames of input data (i.e., the total number of classification events). From Table 4.2, we see significant variation in accuracy among the modes, with a significant gap from each lower accuracy mode to the next higher accuracy mode. As expected, the dual-modality modes have higher accuracy compared to the single-modality ones.

Table 4.2: Accuracy comparison (%).

	Acoustic	Seismic	DST Fusion	ALFFS
Accuracy (%)	67.9	76.6	81.6	98.4

Table 4.3 summarizes measurements of power consumption P (Watts), run-time R (seconds per data frame), and energy consumption $E = R \times P$ (Joules per data frame). Here, a “data frame” corresponds to a single frame of acoustic or seismic input data for the single-modality modes. For the dual-modality modes, a data frame encapsulates an acoustic input frame together with its corresponding seismic input frame. The reported power values are derived by measuring the power consumption in the associated modes, and subtracting from these measurements the baseline power consumption (i.e., the power consumed when the processing platform is idle). This gives an estimate of the power consumption that is attributable to the processing requirements of each mode.

While the ALFFS mode provides superior accuracy, this enhanced discrimination capability comes at the expense of higher power consumption, and longer execution time. These costs in turn combine to increase energy consumption, leading to faster battery drain. Conversely, the single-modality modes are more energy efficient, but are not as ac-

Table 4.3: Power consumption, run-time, and energy efficiency comparison.

	Current (A)	Power (W)	Run-time (sec)	Energy per frame (J)
Acoustic	1.44×10^{-1}	7.18×10^{-1}	1.43×10^{-1}	1.03×10^{-1}
Seismic	1.41×10^{-1}	7.07×10^{-1}	1.47×10^{-1}	1.04×10^{-1}
DST Fusion	1.57×10^{-1}	7.83×10^{-1}	1.70×10^{-1}	1.33×10^{-1}
ALFFS	1.51×10^{-1}	7.53×10^{-1}	2.50×10^{-1}	1.89×10^{-1}

curate as the modes that employ fusion techniques. This loss in accuracy means that there will be a higher rate of missed events or false event detections. The run-time, accuracy, and energy consumption for DST Fusion provide an intermediate trade-off between the single-modality modes and ALFFS.

These results demonstrate that each of the four modes provides a distinct, Pareto-optimal (non-dominated) design point among the four design points represented by the PVD system modes. Given a multidimensional design evaluation space involving $N > 2$ metrics, a design point p_1 is said to *dominate* another point p_2 if p_1 is better than or equal to p_2 in terms of all of the N relevant metrics, and p_1 is better than p_2 in terms of at least one metric. For example, from the data shown in Table 4.2 and Table 4.3, the acoustic mode (design point) is not dominated by any of the other modes (e.g., because it is better than all other modes in terms of energy consumption per processed input frame). On the other hand, if accuracy and power consumption were the only metrics considered in the design evaluation space, then the acoustic mode would be dominated by the seismic mode. Intuitively, a dominated mode is redundant or “expendable” in the context of the associated design evaluation space.

Despite its status as a non-dominated mode, it can be argued that for this applica-

tion, the acoustic mode incurs an excessive loss in accuracy in exchange for relatively small improvements in run-time and energy consumption compared to the seismic mode. However, the remaining three modes — the seismic, DST fusion, and ALFFS modes — represent diverse operational alternatives that offer significantly different trade-offs among the three design evaluation metrics E , R , P . For example, during critical states of operation, such as when a potential threat is detected, ALFFS may be used, while the seismic mode may be used for standby operation, and the DST fusion mode may be used during times of anticipated transition between standby and critical states.

4.5 Conclusion

In this chapter, we have introduced an optimized, multi-mode embedded target detection system that employs acoustic and seismic sensors for detection of people and vehicles. The system provides two complementary modes of operation that include single-modality processing using seismic and acoustic sensors, respectively. The system also provides two dual-modality modes that incorporate sensor fusion, using methods based on Dempster Shafer Theory (DST) and a recently-introduced algorithm called Accumulation of Local Feature-level Fusion Scores (ALFFS). Experimental results demonstrate that the seismic, DST Fusion, and ALFFS modes provide flexibility in dynamically reconfiguring system execution across a range of useful operational trade-offs. Additionally, the acoustic mode is included within the system to enhance adaptability to other target detection applications and sensing devices that are more amenable to acoustic signal processing. Useful directions for future work include investigating trade-offs involving alternative

windowing configurations in ALFFS, and developing low power hardware accelerators to further improve system trade-offs.

Chapter 5: A Joint Target Localization and Classification Framework for Sensor Networks

In this chapter, we introduce a joint framework for target localization and classification using a single generalized model for non-imaging based multi-modal sensor data. For target localization, we exploit both sensor data and estimated dynamics within a local neighborhood. We validate the capabilities of our framework by using a multi-modal dataset, which includes ground truth GPS information (e.g., time and position) and data from co-located seismic and acoustic sensors. Experimental results show that our framework achieves better classification accuracy compared to recent fusion algorithms using temporal accumulation and achieves more accurate target localizations than multilateration.

The work presented in this chapter has been published in [10].

5.1 Introduction

Automatic target localization and discrimination is critical for border protection and surveillance settings, especially in remote locations where it can be costly or logistically difficult to employ human enforced security. A number of robust target classification and localization algorithms using cameras (e.g., see [73]) However, computing both loca-

tion and class information from these types of devices can be challenging. For example, image-based localization and tracking solutions have several challenges to consider, such as occlusions, fog, lighting variations, limited field of view, and processing/power requirements.

Alternatively, we can consider using non-imaging sensors, such as seismic and acoustic sensors, to perform both target localization and classification. The Doppler effect causes faster objects to generate signals with different signatures compared to those of slower or stationary objects. The Doppler effect in acoustic and seismic signals is significant because acoustic and seismic signals have a slower wave propagation speed compared to electromagnetic signals [74]. Dragoset showed that seismic signals can have phase dispersion caused by the Doppler effect [75].

In this chapter, we introduce a joint framework for tracking and classifying targets using acoustic and seismic signals from multiple, locally distributed sensor nodes. This framework is based on probabilistic confidence maps based on a spatial accumulative framework from acoustic and seismic signals to locate and classify target. Through extensive experiments, we demonstrate that our proposed framework provides better localization than that of baseline multilateration-based location estimation (e.g., beamforming). At the same time, we show that our framework achieves better classification performance than recent seismic and acoustic fusion approaches in [8].

A distinguishing aspect of our work is that we provide a framework that can be used for a classification and localization of targets simultaneously using multiple sensor nodes with a singular generalized model, which can be applied to every node in a sensor network. By employing probabilistic maps from acoustic, seismic, and estimated velocities,

we improve classification performance compared to our recent work [8], and we provide a better localization (tracking) capability compared to multilateration-based methods.

Another distinguishing aspect of our work is that we validate it by using actual data collected in an outdoor setting, mimicking common operating environments and location information from GPS. For experiments, we employ acoustic and seismic data that are synchronized with GPS signals collected from the field. In contrast, many previous studies for localization of acoustic and seismic signals employ simulated signals to validate the work (e.g., see [74]), which can have different characteristics compared to real-time datasets.

5.2 Related Works

Various algorithms have been proposed for target localization using image-based or other modalities. These include vehicle detection and tracking using acoustic and video sensors [76]; location estimation using video, image, and audio signals [77]; location estimation based on Received Signal Strength (RSS) [78–81]; confidence-based iterative localization [82]; and target location estimation from detection of dense sensor networks [83]. Our work differs by jointly performing both classification and localization, and validating our approach under more challenging conditions using sparsely distributed sensor nodes (e.g., 0.0025 sensors per square meter).

Multilateration (see Figure 5.1) is a common approach used for localization using wireless sensor networks (e.g., see [84, 85]). By using measured (or estimated) distances from multiple sensors, the target position can be estimated. The location of an unknown

node (or target) can be estimated based on the intersection of the distance from multiple nodes. For example, Hefeeda et al. [86] provide an approach for early detection of forest fires using multilateration. Damarla et al. [87] provide a sniper localization method using the time-difference-of-arrival (TDOA) between the muzzle blast and the shock wave using multiple single-acoustic-sensor nodes. Our work differs from these works in that we employ a probabilistic score model instead of using estimated distances from nodes, and as emphasized previously, we provide a joint framework for both localization and classification.

Several works estimate the motion of a target (mainly for vehicles) from acoustic signals (e.g., see [88–90]) based on the Doppler effect. While we employ estimated dynamics, our work differs from this earlier work in that we use both acoustic and seismic signals, and we provide not only localization but also classification. Moreover, we consider both people and vehicles. Incorporating detection of people makes the problem significantly more challenging. This is because humans generate very small acoustic and seismic signatures compared to vehicles.

We would like to emphasize that while the methods in this chapter are inspired by the Doppler effect (e.g., faster objects generate signals with different signatures), the methods do not directly derive velocities from the Doppler effect. Instead, velocities are estimated based on the assumption that acoustic and seismic signals carry information about target velocities.

In [8], an accumulative model is proposed to combine multiple evidences over time to improve discriminability. However, this introduces an unnecessary latency. Instead, we propose an accumulative method using spatially distributed sensors for improving

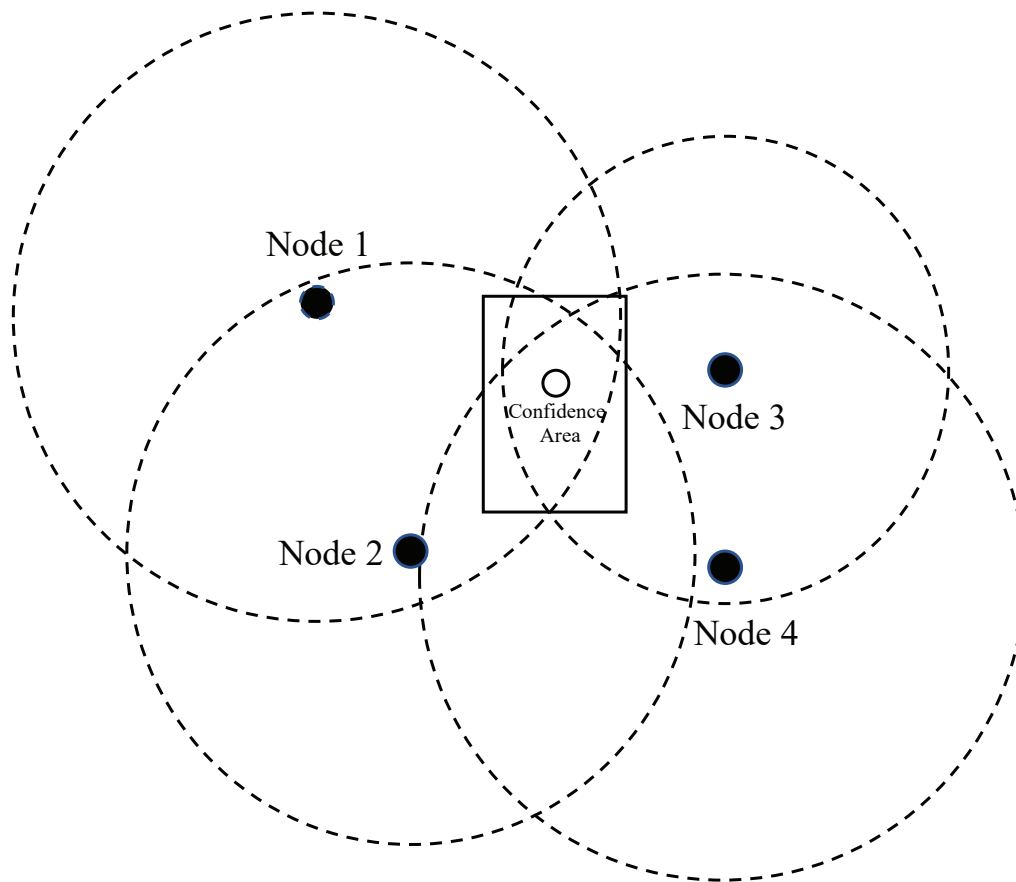


Figure 5.1: Localization using multilateration.

discriminability.

5.3 Joint Localization and Classification Framework

In this section, we describe a joint target localization and classification framework using estimated dynamics (velocities) and multimodal data. We refer to this approach as Classification and Localization using Estimated Dynamics and Multimodal data (CLEDM). In this chapter, we use acoustic and seismic sensing modalities. However, the CLEDM framework is not dependent on these modalities and we envision that it can readily be adapted to other ones. Investigating such adaptations is a useful direction for future work.

CLEDM is motivated by the Doppler effect, which enables us to effectively estimate the velocity of the target that is being tracked. Using a probabilistic confidence map to assess the movement of the target using estimated velocities, we estimate the next target location. We also apply the estimated velocities to improve the performance of classification.

When training, modalities that exploit the Doppler effect are required. We need the ground truth class and location of the target corresponding to each signal segment in the training dataset.

CLEDM decomposes time into windows (*segments*) of some fixed duration t_s . We use $t_s = 1\text{sec}$ in our experiments. For $i = 1, 2, \dots$, we denote the starting time ($t_s \times (i-1)$) of the i th segment by t_i . Let $D_{\alpha,i}(\tau)$ and $D_{\sigma,i}(\tau)$ denote the acoustic and seismic signal, respectively, for the i th time segment ($0 \leq \tau < t_s$).

During the training process, since ground truth target location information is given, we are able to calculate two types of dynamics: a relative speed v_r and an absolute speed v_a . We use an $x - y$ coordinate system to model the spatial layout of the region of interest that is monitored by the given sensor network. We assume that the origin in this coordinate system is the location of an active sensor node ν that acquires the signals D_α and D_σ . If we denote the target location relative to ν at t_k by $\vec{r}_k = (x_k, y_k)$, then the following expressions can be used to determine v_r and v_a :

$$v_r = \frac{|\vec{r}_{i+1}| - |\vec{r}_i|}{t_s}, \text{ and} \quad (5.1)$$

$$v_a = \frac{|\vec{r}_{i+1} - \vec{r}_i|}{t_s}. \quad (5.2)$$

Intuitively, v_r is the estimated rate of change of the distance between the target and the sensor node ν . This rate of change can be positive or negative. Similarly, v_a represents the absolute speed of the target, which is independent of individual sensor nodes.

Now suppose that $F_{\alpha,i}$ and $F_{\sigma,i}$ represent extracted features, such as cepstral features, from $D_{\alpha,i}$ and $D_{\sigma,i}$, respectively, and let $F_{fs,i}$ represent the concatenation $[F_{\alpha,i}, F_{\sigma,i}]$ of these feature vectors. In this chapter, we used 50 cepstral features extracted from acoustic and seismic signals for $F_{\alpha,i}$ and $F_{\sigma,i}$.

Then we formulate the following composite feature vector X_i for time t_i :

$$X_i = [F_{fs,i}, v_a, v_r]. \quad (5.3)$$

During the training process, we compute X_i for each time segment i of available training data. We assume that a ground truth class label Y_i is available as part of the training data for each time segment i . Y_i indicates whether the target is of class person (A) or vehicle (B). After computing the X_i values, we train a model H to classify between classes A and B . In our experiments, we use a support vector machine (SVM) as the model H , but the CLEDM methodology is not restricted to SVMs and can readily be adapted to use other types of models.

Based on the trained model H , a real-valued classification score $\Gamma(X_i)$ can be calculated if F_{fs} , v_a , and v_r are given. The score is formulated such that $sgn(\Gamma)$ represents the classification decision between classes A and B , and $abs(\Gamma)$ represents the classification “confidence” of the associated prediction. Here, sgn and abs represent the sign and absolute value functions, respectively.

After training H , we assume that the initial target location and the neighborhood \mathcal{N} of sensors are known to initialize the tracking component of our framework. From the current target position at time t_i , we can extract $F_{\alpha,i}$, $F_{\sigma,i}$ from $D_{\alpha,i}$, $D_{\sigma,i}$.

Around the current target position, which is located at a distance of \vec{r}_i from a particular sensor, we define a grid G discrete locations that represent potential target locations at time t_{i+1} . This grid map is illustrated in Figure 5.2. For a given candidate target location $p = (p_x, p_y)$ at time t_{i+1} , we set $\vec{r}_{i+1} = (p_x, p_y)$, and then estimate v_a and v_b from Eq. 5.2 and 5.1, respectively.

Next, using Eq. 5.3 and our trained model, we calculate the feature vector $X_i(p; \nu)$ for the candidate next point p and sensor ν .

Then, we calculate the classification score:

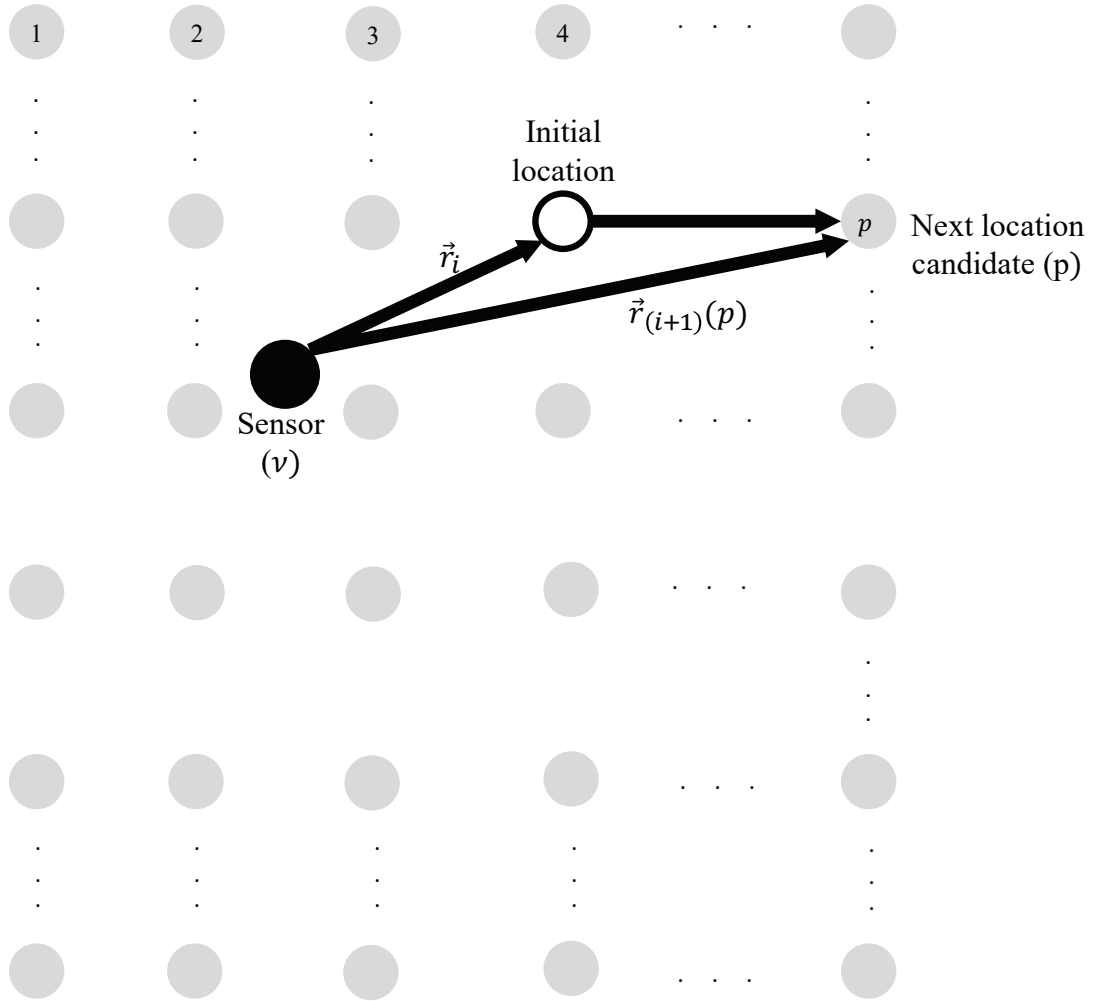


Figure 5.2: Grid map for estimating the next target location.

$$\gamma(p; \nu) = \Gamma(X_i(p; \nu)). \quad (5.4)$$

We repeat this process to determine γ for all points $p \in G$. For a sensor ν , we define the class-specific score functions δ_A and δ_B over the domain G as follows:

$$\delta_A(p; \nu) = \begin{cases} |\gamma(p; \nu)| & \text{if } \gamma(p; \nu) \geq 0 \\ 0 & \text{if } \gamma(p; \nu) < 0 \end{cases} \quad (5.5)$$

$$\delta_B(p; \nu) = \begin{cases} 0 & \text{if } \gamma(p; \nu) \geq 0 \\ |\gamma(p; \nu)| & \text{if } \gamma(p; \nu) < 0 \end{cases} \quad (5.6)$$

Eq. 5.5 and 5.6 are derived based on a specific sensor node, whose position is taken to be the origin of the coordinate system in the associated derivations. When multiple sensor nodes are present, these class- and node-specific score functions can be summed across all of the nodes to yield probabilistic confidence maps $M_A = \sum_{\nu \in \mathcal{N}} \delta_A(p; \nu)$ and $M_B = \sum_{\nu \in \mathcal{N}} \delta_B(p; \nu)$ for the two classes A and B , respectively.

This is a form of spatial accumulation (across the available sensor nodes), which is different from the temporal accumulation applied in [8]. Figure 5.3 illustrates an example of a probabilistic confidence map that is derived using this form of spatial accumulation.

To predict the class of the target, we first determine for each $c \in \{A, B\}$ the maximum absolute value Z_c within M_c over all points in the grid G . Then if $Z_A \geq Z_B$, the predicted class is A ; otherwise, it is B . Here, we arbitrarily select A as the predicted class in case of a tie ($Z_A = Z_B$).

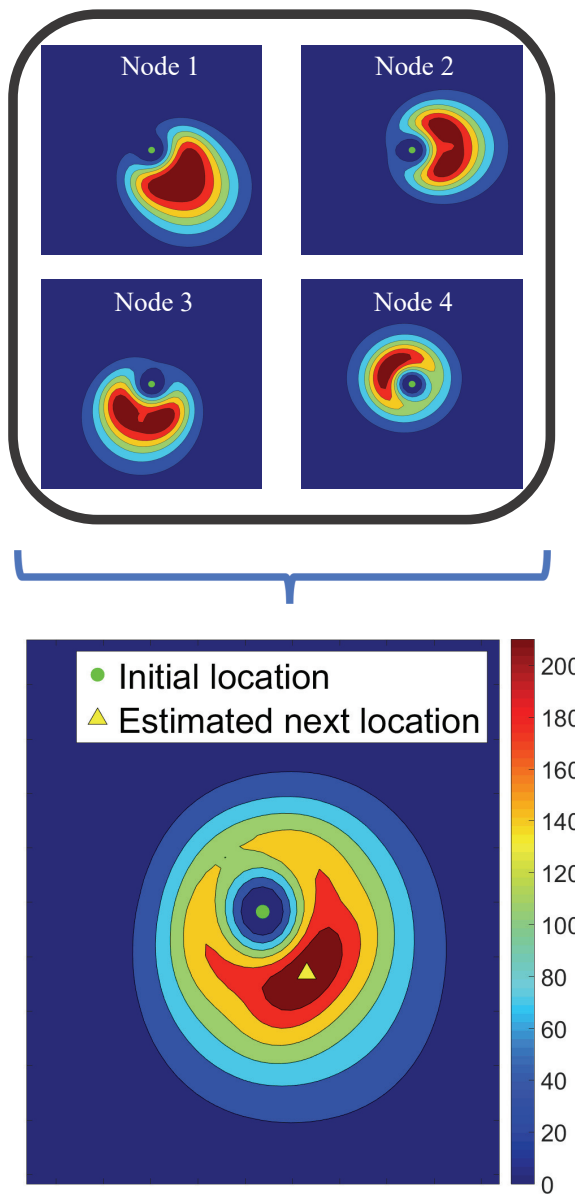


Figure 5.3: Confidence maps for each node are shown on the above. The spatial accumulation of these maps are shown on the below.

After classification, we calculate the centroid of the score map M_κ that is associated with the predicted class κ . This centroid is the next estimated location.

The whole process of joint classification and localization described above is repeated iteratively to provide continuous tracking.

5.4 Experiments and Results

In this section, we present an experimental evaluation of the proposed CLEDM framework. In our evaluation, we employ 16 multimodal sensor nodes that each collect acoustic and seismic data. The nodes are grouped into 4 sets of 4 nodes each and placed in a $20\text{m} \times 20\text{m}$ square. The sets are denoted as Set 1 through Set 4. The placement of the nodes is illustrated in Figure 5.4.

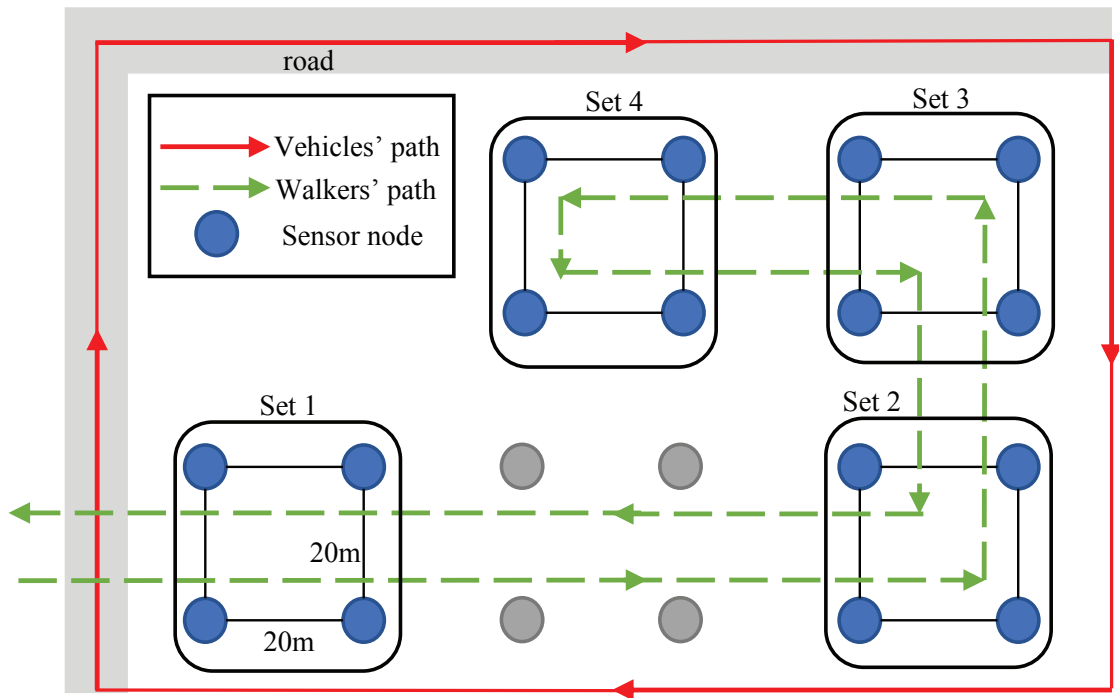


Figure 5.4: Layout of sensor nodes.

Set 1 is used for training, and contains 1523 frames of data. Sets 2–4 are used for testing, and contain a total of 1620 frames. Each frame contains 1 second of acoustic and seismic data corresponding to a single target (person or vehicle). The data within each frame is sampled at 4096Hz. Each frame also contains GPS location data of the associated target. More details about this dataset can be found in [65].

To evaluate classification performance between people and vehicles, we compared classification accuracy among single-modality, SVM-based classification; a state-of-the-art multi-modal classification architecture called Accumulation of Local Feature-level Fusion Scores (ALFFS) for acoustic and seismic signals [8]; and our proposed CLEDM framework. Table 5.1 shows the results of this comparison. The columns labeled Acoustic and Seismic correspond to the single-modality results. The results in Table 5.1 show that the CLEDM framework provides superior accuracy.

Table 5.1: Accuracy comparison (%).

	Acoustic	Seismic	ALFFS [8]	CLEDM
Accuracy (%)	77.7	77.2	79.8	81.9

To evaluate the tracking performance for people and vehicles, we compared tracking using multilateration to our proposed CLEDM approach. For this comparison, we used 44 tracks composed of 1620 data frames in total. For both algorithms, only the first point is synchronized with ground truth, so error increases as time goes on. Table 5.2 and 5.3 summarize the results from our experiments on tracking performance.

For the multilateration-based baseline that we used in these experiments, we employed a convolutional neural network for the regression model. We used this model to

estimate the distances required for localization. We employed a 2-D convolutional layer with 20 filters of size 25 each followed by a ReLU layer. We also employed a fully-connected output layer of size 1, and a regression layer. In this network model, the input data are formed by the concatenation of acoustic and seismic data segments (1 seconds each in duration), and the output is the estimated distance.

We compared the average error between ground truth and estimated location, and the average maximum errors from all 44 tracks. We also compared the percentage of data segments that have less than a certain amount of error: in Table 5.2 and 5.3, $\text{err} < X\text{m}$ gives the percentage of tracks for which the error was less than X meters.

The results in the two tables show that CLEDM has significantly better tracking performance compared to the baseline localization approach overall, and especially favorable performance for tracking people.

Table 5.2: Tracking performance comparison (people)

	avg. error (m)	avg. maximum error (m)	err <3m (%)	err <5m (%)	err <10m (%)
multilateration	6.98	9.39	15.6	28.9	76.6
CLEDM	2.90	4.79	58.6	86.5	99.9

Table 5.3: Tracking performance comparison (vehicle)

	avg. error (m)	avg. maximum error (m)	err <5m (%)	err <10m (%)	err <20m (%)
multilateration	7.06	12.9	41.2	72.5	97.7
CLEDM	6.24	10.3	43.8	72.0	100

5.5 Conclusion

In this chapter, we have introduced a spatial accumulative framework for both target classification and localization. We leveraged signal phenomenology to estimate dynamics and extract discriminative information, which is accumulated across multiple nodes within a local neighborhood. Experimental results have shown that our algorithm provides better localization performance compared to a baseline localization algorithm based on multilateration, while our algorithm also achieves better classification performance compared to relevant prior work. Specifically, CLEDM achieved an absolute improvement of 2.10% in accuracy compared to the baseline ALFFS approach on average. Also, CLEDM achieved 2.90 and 6.24 average error (meter) for people and vehicles. Whereas, the baseline approach achieved 6.98 and 7.06 average error (meter) for people and vehicles. Therefore, accumulating multiple evidences (e.g., dynamics, latent information) across multiple sensors enhances target discrimination and tracking capabilities.

Chapter 6: Software Synthesis from Dataflow Schedule Graphs

In dataflow-based design processes for signal and information processing, scheduling is a critical task that affects practical measures of performance, including latency, throughput, energy consumption, and memory requirements. *Scheduling* in this context refers to the assignment of actors to processors or threads, and the ordering of actors that share the same processor/thread.

Dataflow schedule graphs (DSGs) [11] provide a formal abstraction for representing schedules in dataflow-based design processes. DIF-DSG [91] is a recently developed software synthesis tool for automatically deriving efficient implementations DSGs. This tool is built using the dataflow interchange format (DIF), which provides a design language and utilities for prototyping and experimenting with new dataflow-based design methods and tools [15]. For detailed background on the DSG model and the DIF-DSG software synthesis tool, we refer the reader to [11, 91]. For broader background on model-based schedule representations for dataflow graphs, we refer the reader to [92].

In this chapter, we introduce major new extensions to the DIF-DSG tool that we have developed for multicore platforms. The original DSG framework was developed to support a restricted class of DSGs called sequential DSGs (SDSGs). SDSGs are suited for representing schedules that are targeted to a single processor or single-thread. We

have made significant extensions to DIF-DSG that enable the tool to synthesize software for concurrent DSGs (CDSGs) as well as SDSGs. A concurrent DSG can be viewed as a DSG that consists of multiple SDSGs, along with graph edges that represent communication between the SDSGs. A CDSG provides a formal representation for a multi-threaded schedule, where each SDSG corresponds to the schedule for each thread. Use of CDSGs along with the DIF-DSG tool provides a structured approach to developing implementations of multicore signal processing systems. This approach inherits important properties of dataflow modeling, and avoids pitfalls associated with conventional thread-based development processes [93].

In the remainder of this chapter, by “DIF-DSG” we refer to our extended version of the software synthesis tool, which incorporates support for CDSGs, unless otherwise stated. We demonstrate the capabilities of DIF-DSG by using it to implement a multicore signal processing system for real-time detection of people and vehicles that is based on the ALFSS algorithm presented in Chapter 3.

6.1 Concurrent DSGs

Mapping of dataflow applications onto multicore platforms involves significant complexity, which can greatly increase development time. A major part of this complexity comes from scheduling actors across the available cores, and managing communication and synchronization across the cores. To help manage this complexity, concurrent DSGs (CDSGs) raise the level of abstraction for working with dataflow schedules. In this section, we discuss support for CDSGs in DIF-DSG.

The CDSG model was introduced in [11]. However, the development of the model in this prior work was largely abstract. The implementation and demonstration of DSG concepts in that earlier work was limited to SDSGs. Software synthesis capabilities for SDSGs was subsequently introduced in [91]. In this chapter, we present new methods for implementation of CDSGs, as well as for synthesizing CDSG implementations automatically from high level representations.

DSGs are dataflow graphs for representing schedules of dataflow-based application models. A DSG is either an SDSG or a CDSG. If G_a represents a dataflow model of an application, and G_s is a DSG representation of a schedule for G_a , then G_a is referred to as the *application graph* associated with G_s , and G_s is referred to as the *schedule graph* associated with G_a . DSGs contain two different types of actors, called *reference actors (RAs)* and *schedule control actors (SCAs)*. An RA a represents a conditional firing (“guarded execution”) of an application graph actor, which is referred to as the *referenced actor* associated with the RA. The firing is conditioned on whether or not the input buffers of the associated application graph actor have sufficient data and the output buffers have sufficient empty space to support a firing of the application graph actor [11]. In contrast to RAs, SCAs are used to direct the sequencing of actors within a DSGs, thereby also controlling the order in which application graph actors are executed (through their associated RAs). Specific examples of SCAs are the `if`, `fi`, `snd`, and `rec` SCAs. For more details on these SCAs and on other DSG modeling concepts and notations, we refer the reader to [11].

In DIF-DSG, and in the remainder of this chapter, we assume that each actor of a given application graph is mapped to a single thread. We refer to this as the *Single Thread*

per Application Actor (STAA) assumption. Generalizing the methods of this chapter and the features in DIF-DSG to relax the STAA assumption is a useful direction for future work.

6.1.1 Inter-SDSG Coordination Actors

A CDSG is composed of multiple sequential DSGs, where communication between different sequential DSGs is carried out through a special class of DSG actors that are devoted to communication and synchronization between different SDSGs. Specifically, multiple SDSGs can be integrated into a CDSG through a class of actors called *Inter-SDSG Coordination Actors (ISCAs)*. ISCAs can be viewed as a generalization of the interprocessor communication actors that were introduced in [11]. For example, ISCAs also include actors for inter-thread communication, where SDSGs are mapped to separate threads, and the threads involved can be assigned to the same processor or to different processors.

Except for the edges that are directed between ISCAs, each graph element (actor or edge) in a CDSG is contained within one of the SDSGs that make up the CDSG. Given a CDSG G_c and an actor a in G_c , the unique SDSG that contains a is denoted as $sds(a)$.

The set of SDSGs contained within a given CDSG G_c is referred to as the *SDSG set* of G_c , and denoted as $\sigma(G_c)$. Given a CDSG for an application graph G , parallel execution of actors in G is achieved when multiple RAs in multiple members of $\sigma(G_c)$ are executed at the same time.

CDSG-based software synthesis in DIF-DSG is currently targeted to sequential and

multicore implementations that employ C/Pthreads programs as the output of software synthesis. Thus, DIF-DSG currently supports C/Pthreads-based implementation of IS-CAs. However, due to their model-based orientation in terms of DSG semantics, these actors can be retargeted in different ways for different platforms and target languages.

In the remainder of this section, we describe the implementation of two IS-CAs, the `snd` and `rec` actors, in DIF-DSG. These actors are used to send and receive data across different SDSGs.

In the current version of DIF-DSG, we assume that each `snd` actor a_s has a corresponding `rec` actor a_r , where a_r corresponds uniquely to a_s . This assumption may be relaxed in future versions of DIF-DSG and other implementations of CDSGs (e.g., a single `snd` actor may send data to multiple `rec` actors to achieve broadcasting functionality). In the enclosing CDSGs, an edge is inserted from each `snd` actor a_s to its corresponding `rec` actor to model the associated inter-thread data transfer.

CDSG actors a_s and a_r are used to communicate data from one application graph actor X_s to another application graph actor X_r , where the actors X_s and X_r are contained in different SDSGs $sdsg(a_s)$ and $sdsg(a_r)$, respectively. The actors X_s and X_r communicate through a single FIFO buffer b_{sr} , which can be accessed directly by these actors using Pthreads APIs.

Figure 6.1 illustrates the input and output edges of the `snd` and `rec` actors. The detailed structure and operation of these actors is described as follows.

- `rec`. The actor a_r has one input edge e_{ri} , and one output edge e_{ro} , which are both contained within $sdsg(a_r)$. There is a second input edge to a_r , which is directed

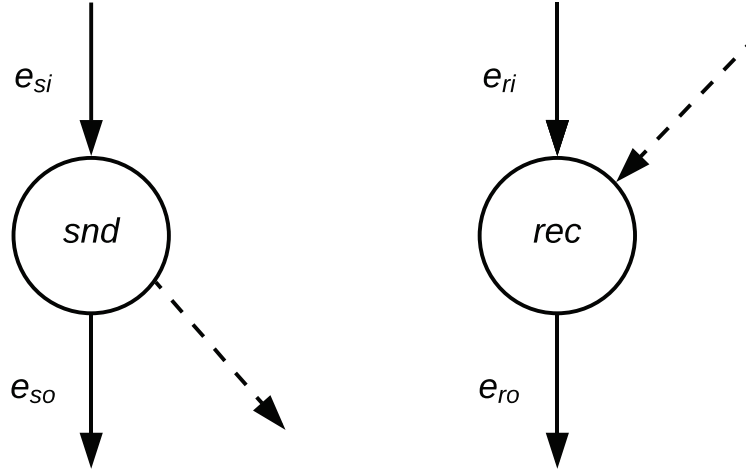


Figure 6.1: An illustration of `snd` and `rec` actors in CDSGs.

from a_s , and which we typically depict in drawings as a dashed edge. We refer to this type of edge as an *inter-thread communication edge* in the SDSG. This dashed edge represents the coupling between a_r and a_s as corresponding ISCA. Consideration of this type of edge can be relevant to certain kinds of dataflow analysis, such as analysis of synchronization structure, buffer memory requirements, and throughput (e.g., see [94]).

On each firing, a `rec` actor a_r waits, using the blocking API function of Pthreads called `pthread_cond_wait`, until the population of b_{sr} exceeds 0. At this point, the `rec` actor finishes execution, and a DSG token is produced on its output to enable the actor in $sdsg(a_r)$ at the sink of e_{ro} . In general, one or more `rec` actors can be used to ensure that each firing of an application graph actor A has sufficient data from other threads that produce input data for A .

- `snd`. Like a_r , the actor a_s has one input edge and one output edge that reside within the same SDSG. We denote these edges, respectively, as e_{si} , and e_{so} . As explained

above, there is a second output edge of a_s , which is directed to a_r , and is called an inter-thread communication edge. The delay on the dashed edge is set to the delay of the corresponding application graph edge (X_s, X_r) . On each firing, a `snd` actor a_s waits, using `pthread_cond_wait`, until the population of the buffer b_{sr} is less than its buffer capacity. Once at least one unit of free space has been validated in the buffer, a_s finishes execution, and a DSG token is produced on its output to enable the actor in $sdsd(a_s)$ at the sink of e_{so} . In general, one or more `snd` actors can be used to ensure that each firing of an application graph actor B has sufficient empty space on its output ports to produce any data needed from B by actors that are assigned to other threads.

6.1.2 Delays

In dataflow representations of signal processing applications, *delays* on edges generally correspond to initial tokens on buffers associated with the edges (e.g., see [3]). Given a dataflow graph edge e , we denote the magnitude of the delay (number of initial tokens) on e as $delay(e)$. We distinguish between two different types of delays when working with CDSGs.

1. Application delays. Delays on an application graph edge e_{app} are implemented as initial tokens on the application graph buffer associated with e_{app} . Since intra-thread communication is not modeled with edges in DSGs, an application graph delay does not show up in the CDSG when the source and sink vertices of e_{app} are assigned to the same thread. If the source and sink of e_{app} are mapped to different

threads, then in general there will be a set S of inter-thread communication edges in the CDSG corresponding to e_{app} .

From the STAA assumption, we are guaranteed that the set S always has exactly one element e_{itc} (when the source and sink of e_{app} are mapped to separate threads).

The delay on this DSG edge is set to the delay of the corresponding application graph edge: $delay(e_{itc}) = delay(e_{app})$.

2. Intra-thread delays. In a CDSG, an *intra-thread edge* is an edge whose source and sink vertices are contained in the same SDSG. All of the intra-thread edges in a given SDSG are assigned zero delay, except for a single intra-thread edge e , called the *starting edge* of the SDSG. The starting edge e is assigned $delay(e) = 1$. When a CDSG starts executing, the execution on each SDSG V commences with the actor at the sink of the starting edge of V .

Note that this approach to assigning inter-thread delays is consistent with the global token population property of SDSGs, which was discussed in [91].

6.1.3 Loop SCAs

Loop SCAs represent an important type of SCA that we apply in our demonstrations and experiments in this chapter. There are two types of loop SCAs in DIF — static and dynamic loop SCAs. Loop SCAs are used to direct DSG tokens through a specific output port, called the *body port*, for some number of successive iterations before the next DSG token output by the actor is directed through a second output port. Tokens on the body port can be viewed as enabling the body of a loop, while the second output port can be

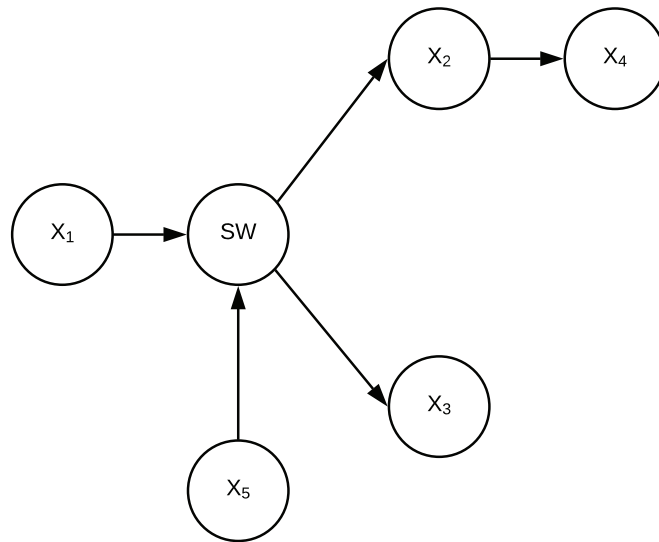
viewed as being associated with the part of the DSG that follows the loop. For further details on this type of SCA, we refer the reader to [11].

A *static loop* SCA is provided an iteration count as a static (compile time) parameter. On the other hand, a dynamic loop SCA receives iteration counts from DSG tokens that it consumes on one of its input ports. Thus, the iteration counts of dynamic loop SCAs can vary at run-time, based on manipulations to the values of the DSG tokens that are provided to them.

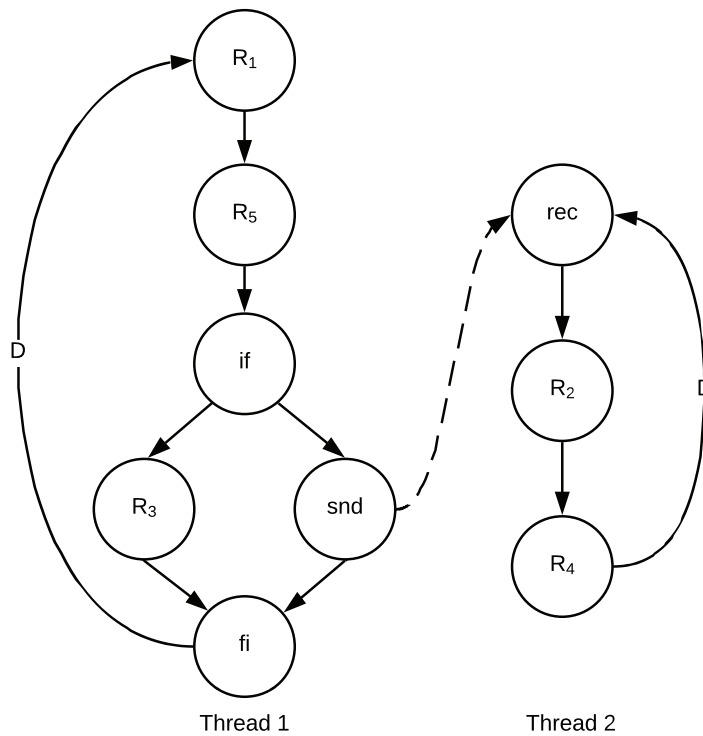
A *loop-and-exit (LAE)* SCA is a variation of the static loop SCA. This actor has only one output port, which is its body port. The actor produces outputs on its body port for a pre-defined number of iterations. After its iteration count is exhausted, the SCA jumps out of (exits) the SDSG to any remaining “wrapup” or continuation code at the end of the enclosing thread. The LAE SCA can be viewed as a special SCA that incorporates control of the overall enclosing SDSG.

6.1.4 Example

To illustrate the concepts and constructions developed in this section, Figure 6.2 shows a simple example of an application graph and an associated CDSG for a 2-thread schedule. The application graph, shown in Figure 6.2(a), consists only of homogeneous SDF actors, except for the actor labeled SW, which is a Boolean dataflow (BDF) *switch* actor. The switch actor is a fundamental BDF actor. For details on its functionality and its modeling in SDSGs, we refer the reader to the works by Buck [95] and Wu [11], respectively. Here, by *homogeneous SDF*, we refer to a special case of SDF in which the



(a) Application graph.



(b) CDSG for a 2-thread schedule.

Figure 6.2: An example that illustrates the concepts and constructions developed in this section.

production and consumption rates on actors ports are identically equal to 1 [4].

In Figure 6.2(b), each actor labeled R_k represents a reference actor $ref(X_k)$ for application graph actor X_k . In addition to these reference actors, the CDSG in Figure 6.2(b) also contains four SCAs — one each of the `if`, `fi`, `snd`, and `rec` SCAs. Actor X_5 provides the control input and actor X_1 provides the data input to the switch actor in the application graph. The actors mapped to Thread 2 are executed when a `true`-valued control token is consumed by the switch actor, whereas `false`-valued control tokens drive the execution of X_3 through its reference actor R_3 .

In Figure 6.2(b), the starting edges for the two SDSGs are (fi, R_1) and (R_4, rec) . Each of the “D” annotations on these edges represents a unit delay, which corresponds to the initial position of the DSG token for the associated SDSG.

6.2 Case Study: Real-Time Classification System

In this section, we demonstrate the utility of DIF-DSG and its underlying modeling techniques through a case study involving a real time classification system. The classification system is designed to discriminate among people, vehicles, and noise using acoustic and seismic signals. Here, by “noise”, we mean the absence of any person or vehicle.

For details on the algorithms and applications associated with this classification system, we refer the reader to [8,9]. The case study in this chapter goes beyond the simulation and prototyping experiments reported in [8,9] in its use of DSGs and DIF-DSG as central parts of the design and implementation process. The experiments described in [8,9] are carried using hand-implemented dataflow graphs that do not employ DSG

modeling nor the DIF-DSG software synthesis tool. However, these earlier studies provide useful LIDE-C actor implementations, which we apply in this case study.

The embedded processing platform used in our experiments is the Raspberry Pi 3 Model B, which is equipped with 1GB RAM, a 4x ARM Cortex A53 CPU, and a Broadcom VideoCore IV GPU. In the experiments described in this chapter, we do not use the GPU.

The input to our classifier application consists of a stream of acoustic data and a stream of seismic data that are delivered from two sensors through an A/D converter. In practice, the data is obtained directly from the sensors in this way. However, we compare the run-time and memory requirements for different implementations using the same input data. For this purpose of reproducibility, pre-collected input frames (frames of acoustic and seismic signals) are used in our experiments. Details about these datasets can be found in [65].

6.2.1 Application Graph

In the classifier application presented in [9], four different modes of operation are provided for people/vehicle/noise classification. Among these modes, we employ in our experiments only the ALFFS (Accumulation of Local Feature-level Fusion Scores) mode. The ALFFS approach provides superior accuracy, although this enhanced discrimination capability comes at the expense of longer execution time.

The application graph for our ALFFS application is illustrated in Figure 6.3. The production and consumption rates associated with actor ports are annotated next to the

ports. These rates are expressed in terms of three static application parameters — D , W , and F . These parameters, respectively, represent the frame size, number of windows per frame, and number of features extracted per window. For more details about these parameters along with the associated notions of frames and windows in the ALFFS application, we refer the reader to [8, 9]. In our experiments, we use $D = 4096$ samples per frame, $W = 50$ windows, and $F = 50$ features.

Descriptions of the actors in Figure 6.3 are summarized briefly as follows.

- *File Source and File Sink*. These actors represent interfaces for reading and writing, respectively, pre-collected sensor data that is stored on a microSD card, which is attached to the targeted Raspberry Pi platform.
- *Feature Extraction*. There are two instances of this actor, one for handling acoustic data and the other for seismic data. These actors apply cepstral analysis to extract features from digitized sensor data. For FFT computation, which is an important part of cepstral analysis, we employ a LIDE-C wrapper around an optimized module from the FFTW library [71]. The generated cepstral coefficients are subsequently employed (in the downstream portion of the dataflow graph) as the features of the input frame.
- *Feature Concatenation*. The coefficients extracted from each pair of corresponding windows associated with the two sensing modalities are concatenated by this actor before arriving as input to the SVM (support vector machine) Bank actor.
- *SVM Bank*. The cepstral features extracted from the Feature Extraction actors are sent as input to the SVM bank actor. In a given firing of the SVM Bank actor, the

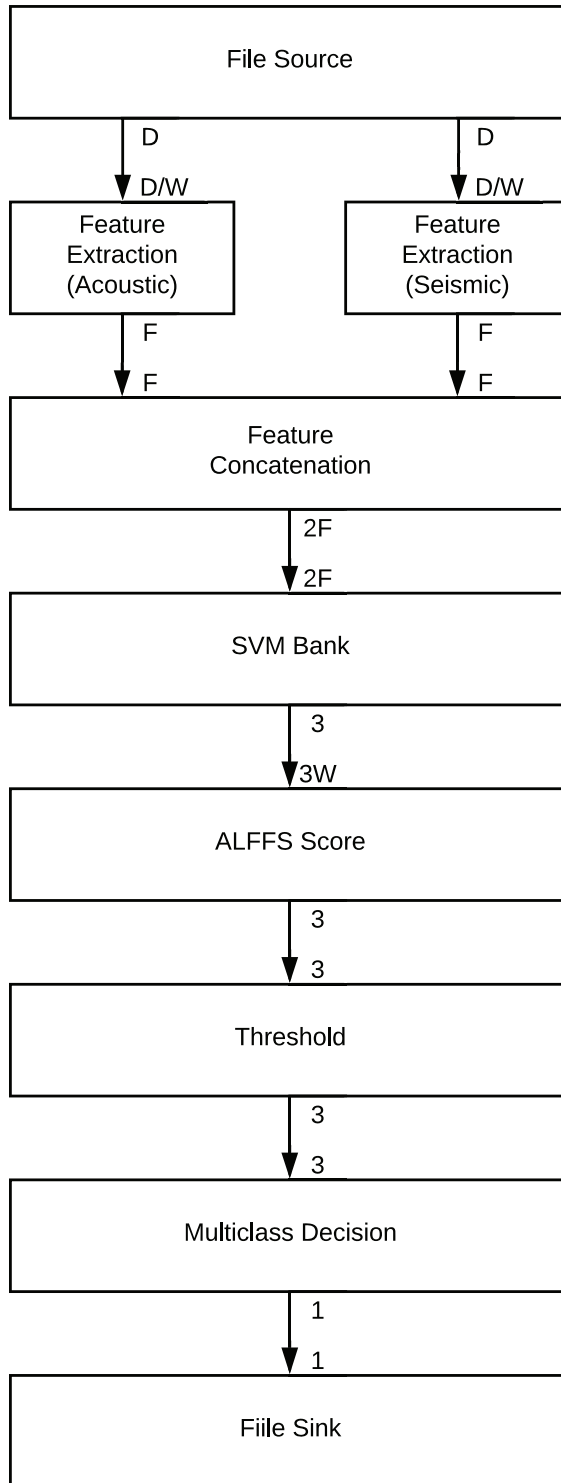


Figure 6.3: Dataflow graph for ALFFS application.

actor accesses trained parameters that are stored in memory as static parameters of the actor. Three classification scores are generated by the SVM bank actor, one for each type of pairwise discrimination a versus b , where $a, b \in \{N, P, V\}$, and N, P, V here represent, respectively, the noise, person, and vehicle classes. For more details on the algorithm underlying the SVM Bank actor, we refer the reader to [8].

- *ALFFS Score*. The ALFFS Score actor takes as input W blocks of M values each, where each block corresponds to the scores derived from a specific window in the current input frame. Corresponding elements of these blocks (triples in our case, since $M = 3$) are added in the ALFFS Score actor, which results in a single, accumulated score for each class.
- *Threshold Actor*. This actor consumes a block of real values x_1, x_2, \dots, x_M , and simply applies a threshold τ to each one to produce a sequence of binary numbers y_1, y_2, \dots, y_M . Each binary number represents a pairwise classification result, which is used as an intermediate result in the overall multiclass classification operation that is performed by each dataflow graph iteration. In this application, the block size and threshold are configured as $M = 3$ and $\tau = 0$.
- *Multiclass Decision Actor*. Each firing of this actor combines a block of M pairwise classification results into a single multiclass classification result. The actor applies a simple voting rule to generate a single classification result from within the set $\{N, P, V\}$. In case of a tie (all three input predictions are different), the actor produces N (noise) as the classification result.

Further details about these actors and the overall classification application can be found in [9].

6.2.2 Alternative DSGs

For the application graph shown in Figure 6.3, we provide experimental results using three different DSGs — an SDSG, a CDSG using 2 threads, and a CDSG using 3 threads. These three DSGs are all derived by hand using the structure of the application graph and knowledge of profiled actor execution times to guide the design of the schedules. The three schedule graphs are shown in Figure 6.4, Figure 6.5, and Figure 6.6, respectively. The actors labeled `loop1`, `loop2`, ..., `loop9` in the figures are static loop SCAs, and the actors labeled `LAE1`, `LAE2`, ..., `LAE6` are loop-and-exit actors (see Section 6.1.3).

Figure 6.5 and Figure 6.3 correspond to two- and three-stage pipelined schedules, respectively, where different iterations of the application graph execute concurrently.

6.2.3 Performance Evaluation

For each of the three schedule graphs illustrated in Figure 6.4, Figure 6.5, and Figure 6.6, we used DIF-DSG to synthesize an implementation. We then executed each of the resulting implementations using two different schedulers as the DSG schedulers — the simple scheduler and the DTT scheduler. For details on these schedulers, we refer the reader to [91]. This resulted in six different sets of measurements, which are represented in the lower six rows of Table 6.1. As a baseline for these measurements, we

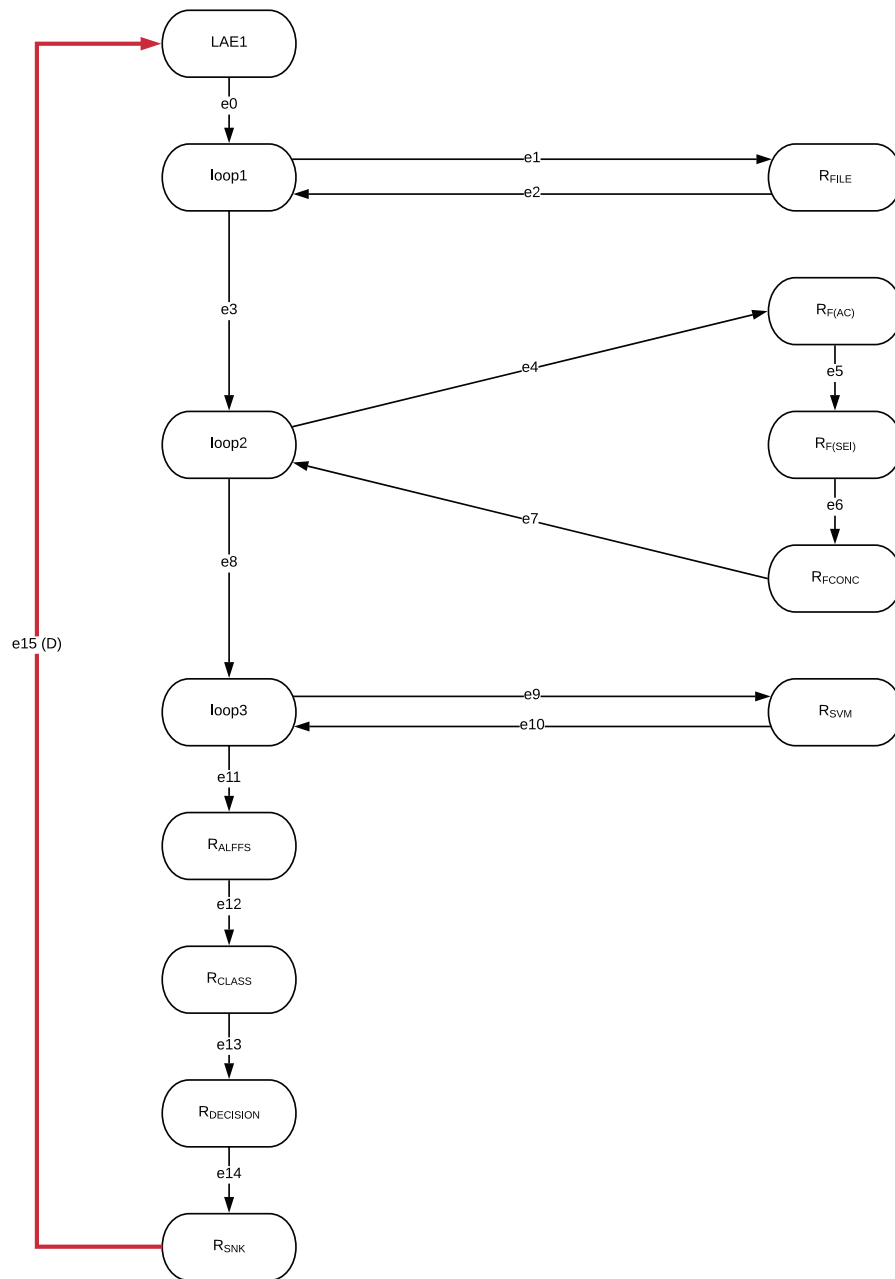


Figure 6.4: An SDSG for the application graph of Figure 6.3.

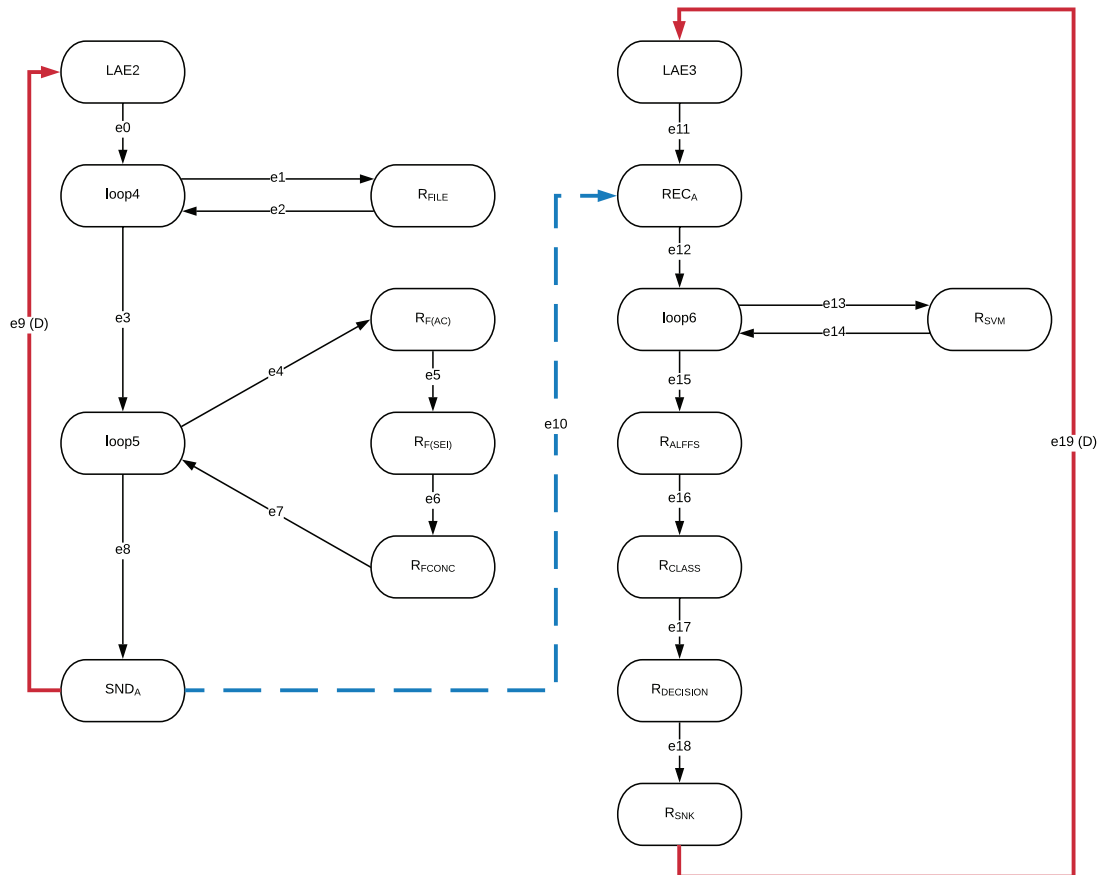


Figure 6.5: CDSG (2 threads) for the application graph of Figure 6.3.

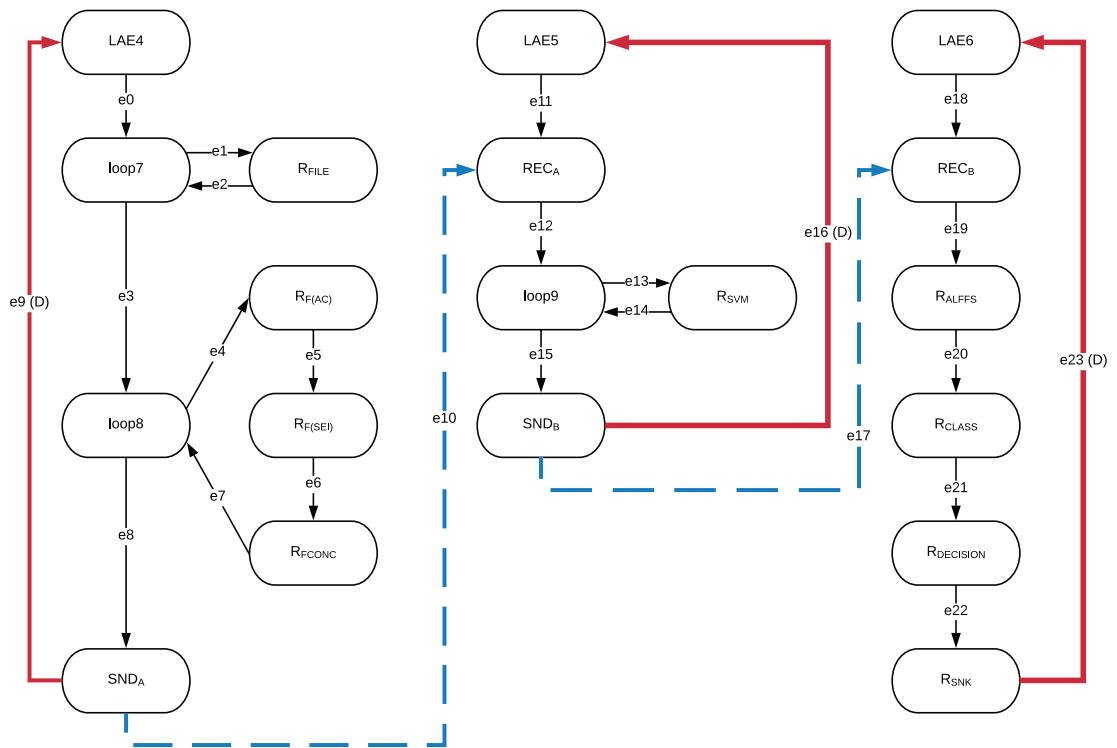


Figure 6.6: CDSG (3 threads) for the application graph of Figure 6.3.

applied the simple scheduler directly to the application graph, without use of any DSG. This configuration is represented by the first row of data (labeled “SS”) in Table 6.1.

For each of the 7 scheduler/DSG configurations corresponding to the 7 rows of Table 6.1, we validated functional correctness by comparing to the output of MATLAB reference code for the underlying algorithm using a given set of test inputs.

Table 6.1: Comparison of run-times and peak memory requirements.

	Run-time (sec)	Peak memory usage
SS	10.8	7.78 MBytes
SDSG-SS	10.0	7.86 MBytes
CDSG-SS (2thread)	9.43	7.94 MBytes
CDSG-SS (3thread)	6.89	8.00 MBytes
SDSG-DTT	8.62	7.88 MBytes
CDSG-DTT (2thread)	7.73	7.96 MBytes
CDSG-DTT (3thread)	6.50	8.02 MBytes

Table 6.1 summarizes measurements of run-times and peak memory usage for the 7 different configurations that we experimented with. The run-times reported here are the average times for processing 100 frames of bimodal data, where each frame contains 1 second of acoustic and seismic data sampled at 4096 Hz.

As expected, the two sequential (single-thread) configurations that employ the simple scheduler are the slowest, since they involve repeatedly visiting actors and checking enable conditions. For both of these configurations, the traversal orders for the simple scheduler are constructed carefully (by hand) to help minimize the rate at which enable checks are found to fail at run-time. This is a useful design-time optimization (for some applications) that can be performed heuristically using topological sort analysis before

deploying an implementation that uses the simple scheduler. The SS-based scheduling process is different for the SS and SDSG configurations in that in the former, the enable conditions for application graph actors are checked at run-time, whereas in the latter, the enable conditions for DSG actors are checked.

As the results in Table 6.1 show, the CDSG-based configurations generally improve performance by making use of multiple threads. This performance improvement is achieved in a structured way based on dataflow semantics, which avoids pitfalls associated with unstructured use of threads (e.g., see [93]).

The performance improvement due to use of multiple threads is significant for both the SS and DTT configurations of SDSGs and CDSGs. We expect that the improvement is less than the ideal 2X and 3X speedups (for two and three threads) (a) due to overheads that are incurred due to inter-thread communication and synchronization, and (b) because the SVM Bank actor is a bottleneck actor in the application design, which limits the available parallelism. Future work on the application graph to decompose this bottleneck (into multiple actors) is a useful direction to expose more parallelism in the design.

Use of the DTT scheduler brings significant improvement in all cases — SDSG-DTT, CDSG (2 thread)-DTT, and CDSG (3 thread)-DTT have 13.9%, 18.0%, and 5.74% less run-time compared to SDSG-SS, CDSG (2 thread)-SS, and CDSG (3 thread)-SS, respectively. The fastest configuration among all 7 studied here is CDSG-DTT using 3 threads, which operates at an average rate of 65.0 milliseconds per frame.

The experiments reported here help to validate the correctness of the DIF-DSG software synthesis process; the capability of CDSGs to provide efficient, model-based representations for implementing multi-threaded signal processing systems; and the util-

ity of DSG token tracking as a first approach to optimized design of DSG schedulers.

6.3 Summary

This chapter has presented new methods and tools for synthesis of software for signal processing systems. The novelty of the methods centers on their support for the dataflow schedule graph (DSG) as a formal model of multi-threaded schedules for dataflow-based application representations.

In conventional dataflow-based software synthesis techniques, schedules are represented using formal representations that are very restricted in their applicability or using general representations that are constructed in ad-hoc ways, without any formal connection to dataflow. This chapter has developed software synthesis techniques that operate on the DSG model, which is both general in its applicability to a broad class of static and dynamic dataflow representations, and formal in its underpinnings in terms of dataflow semantics.

The chapter has presented the first development of multicore implementation techniques using DSGs, and integrated these techniques into a previously developed software synthesis tool, called DIF-DSG. Building on the previous version of DIF-DSG, which was designed for sequential DSGs, we have developed major new extensions that provide support in DIF-DSG for concurrent DSGs. Our new version of DIF-DSG generates multithreaded code automatically from coupled dataflow representations of applications and schedules, and incorporates an optimized run-time system that exploits special characteristics of DSGs.

Our new software synthesis techniques are demonstrated through experiments involving a state-of-the-art signal processing system that is based on the ALFFS algorithm we presented in [Chapter 3](#).

Chapter 7: Conclusion and Future Work

In this chapter, we first briefly summarize our work and contributions, as presented in the previous chapters of this thesis. Then we outline interesting directions for future research that are motivated by the developments in the thesis.

7.1 Conclusions

In this thesis, we have developed novel design methods and tools that address these bottlenecks of cost and energy efficiency in development of target detection systems using acoustic and seismic signals. Our contributions are grouped into three major parts, which encompass algorithms, system design, and software tools.

Firstly, we have contributed new algorithms for target classification and localization using data acquired from acoustic and seismic sensors. As part of this algorithm development, we introduced new algorithms for sensor fusion in support of multiclass classification among people, vehicles, and noise (the absence of people or vehicles). We have developed and comparatively evaluated two different multiclass algorithms, a score-level fusion algorithm that is based on Dempster-Shafer Theory (DST), and an accumulative algorithm that exploits feature-level fusion. Our new fusion algorithm was evaluated through experiments using actual measured datasets, which were collected from differ-

ent sensors installed in different locations and at different times of day. Our proposed classification algorithm achieved high accuracy with low false alarm rate.

Also, we have designed a joint framework for target localization and classification using a single generalized model for non-imaging based multi-modal sensor data. For target localization, we exploited both sensor data and estimated dynamics within a local neighborhood. We validated the capabilities of our framework by using an actual multi-modal dataset, which includes ground truth GPS information (e.g., time and position) and data from co-located seismic and acoustic sensors. Experimental results showed that our framework achieves better classification accuracy compared to state of the art fusion algorithms using temporal accumulation and achieves more accurate target localization than a baseline target localization approach.

Secondly, we have presented system design methods for efficiently mapping our new algorithms into real-time, energy efficient implementations on state of the art embedded platforms. As a foundation for our system-level design research, we introduced a new rapid prototyping methodology and associated software tool. This tool, called DICE-based Prototyping framework for Tracking Systems (DPTS), enables integrated design, prototyping and optimization of power-efficient tracking systems on mobile devices.

Using DPTS, we have presented the design and implementation of a novel, multi-mode embedded signal processing system for detection of people and vehicles using acoustic and seismic sensors. The system provides multiple modes of operation that are matched to different operating scenarios. In this work, we applied a dataflow-based methodology for model-based implementation and design optimization of the proposed multi-mode target detection system. We applied a strategically-configured suite of single-

and dual-modality signal processing techniques together with dataflow-based design optimization for energy-efficient, real-time implementation. Through experiments using a Raspberry Pi platform, we demonstrated the capability of our target detection system to provide efficient operational trade-offs among detection accuracy, energy efficiency, and processing speed.

Thirdly, we have developed new software tools that synthesize efficient multi-threaded software for implementing schedules from abstract representations of the schedules. These abstract representations are based on the dataflow schedule graph (DSG) model, which is a recently-introduced approach for improving the reliability, efficiency, and retargetability of dataflow schedule implementation. We demonstrated our new tools for DSG software synthesis through multi-threaded implementations of target classification algorithms presented in earlier parts of the thesis.

7.2 Future Work

Various useful directions for future work have been motivated from the developments of this thesis. In this section, we summarize some of the key directions for future work.

The algorithms for classification and localization proposed in this thesis have involved support vector machine (SVM) models. SVM models have been used in this work for a number of reasons. First, the models are relatively efficient when the available data for training is small and the model is used for classification. Second, the application development process involves less complexity compared to various other machine learn-

ing models, such as neural networks. Third, the models can be applied naturally within dataflow design processes (as can neural networks [96]).

Although we have used SVM models throughout the thesis to demonstrate the proposed classification and localization frameworks, the core of these frameworks is not dependent on SVM models. Demonstrating the generality of the frameworks by applying them in conjunction with other types of models, including neural networks, is a useful direction for future work. Another useful direction is the development of multi-mode frameworks that employ multiple models and switch among the models using principles of Dynamic Data Driven Applications Systems [40, 97].

On the system design and software tools side, there are also a number of useful directions for future work. These include the following.

- Adapting the models and methods in DPTS for use with other relevant dataflow tools, such as Orcc [26], PREESM [27], and the multi-dataflow composer (MDC) [28].
- Investigating trade-offs involving alternative windowing configurations in the Accumulation of Local Feature-level Fusion (ALFFS) algorithm.
- Developing low power hardware accelerators to further improve operational trade-offs in systems that apply the classification and localization methods proposed in the thesis.
- Generalizing the DIF-DSG software tool to relax the Single Thread per Application Actor (STAA) assumption, and thereby enlarge the supported design space.
- Decomposing bottleneck actors in the proposed systems, such as the SVM Bank

actor in Figure 6.3, into multiple actors is a useful direction to expose more parallelism in the designs, and potentially provide further performance improvements.

Bibliography

- [1] D. Shepherd and S. Kumar, “Microsensor applications,” in *Distributed Sensor Networks Image and Sensor Signal Processing*, S. S. Iyengar and R. B. Richard, Eds. CRC Press, 2012, pp. 3–19.
- [2] E. A. Lee and T. M. Parks, “Dataflow process networks,” *Proceedings of the IEEE*, pp. 773–799, May 1995.
- [3] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, 2nd ed. Springer, 2013.
- [4] E. A. Lee and D. G. Messerschmitt, “Synchronous dataflow,” *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.
- [5] S. S. Bhattacharyya, W. Plishker, C. Shen, N. Sane, and G. Zaki, “The DSPCAD integrative command line environment: Introduction to DICE version 1.1,” Institute for Advanced Computer Studies, University of Maryland at College Park, Tech. Rep. UMIACS-TR-2011-10, 2011, <http://drum.lib.umd.edu/handle/1903/11422>.
- [6] 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.
- [7] K. Lee, H. Ben Salem, T. Damarla, W. Stechele, and S. S. Bhattacharyya, “Prototyping real-time tracking systems on mobile devices,” in *Proceedings of the ACM International Conference on Computing Frontiers*, Como, Italy, May 2016, pp. 301–308, invited paper.
- [8] K. Lee, B. S. Riggan, and S. S. Bhattacharyya, “An accumulative fusion architecture for discriminating people and vehicles using acoustic and seismic signals,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, New Orleans, Louisiana, March 2017. [Online]. Available: <http://www.ece.umd.edu/DSPCAD/papers/lee2017x3.pdf>

- [9] ———, “An optimized embedded target detection system using acoustic and seismic sensors,” in *Proceedings of the European Signal Processing Conference*, Kos Island, Greece, August 2017, pp. 1021–1025.
- [10] ———, “A joint target localization and classification framework for sensor networks,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Calgary, Canada, April 2018, pp. 3076–3080.
- [11] H. Wu, C. Shen, N. Sane, W. Plishker, and S. S. Bhattacharyya, “A model-based schedule representation for heterogeneous mapping of dataflow graphs,” in *Proceedings of the International Heterogeneity in Computing Workshop*, Anchorage, Alaska, May 2011, pp. 66–77.
- [12] T. Damarla and L. M. Kaplan, “A fusion architecture for tracking a group of people using a distributed sensor network,” in *Proceedings of the International Conference on Information Fusion*, 2013, pp. 1776–1783.
- [13] A. Benavoli and L. Chisci, “Towards optimal energy-quality tradeoff in tracking via sensor networks,” in *Proceedings of the European Control Conference*, 2007, pp. 1523–1529.
- [14] S. Kedilaya, W. Plishker, A. Purkovic, B. Johnson, and S. S. Bhattacharyya, “Model-based precision analysis and optimization for digital signal processors,” in *Proceedings of the European Signal Processing Conference*, Barcelona, Spain, August 2011, pp. 506–510.
- [15] S. Lin, Y. Liu, K. Lee, L. Li, W. Plishker, and S. S. Bhattacharyya, “The DSPCAD framework for modeling and synthesis of signal processing systems,” in *Handbook of Hardware/Software Codesign*, S. Ha and J. Teich, Eds. Springer, 2017, pp. 1–35.
- [16] C. Shen, W. Plishker, and S. S. Bhattacharyya, “Dataflow-based design and implementation of image processing applications,” in *Multimedia Image and Video Processing*, 2nd ed., L. Guan, Y. He, and S. Kung, Eds. CRC Press, 2012, pp. 609–629, chapter 24. [Online]. Available: <http://www.crcpress.com/product/isbn/9781439830864>
- [17] M. E. Munich, “Bayesian subspace methods for acoustic signature recognition of vehicles,” in *Proceedings of the European Signal Processing Conference*, 2004, pp. 2107–2110.
- [18] M. F. Duarte and Y. H. Hu, “Vehicle classification in distributed sensor networks,” *Journal of Parallel and Distributed Computing*, vol. 64, no. 7, pp. 826–838, 2004.
- [19] B. Guo, M. S. Nixon, and T. R. Damarla, “Acoustic information fusion for ground vehicle classification,” in *Proceedings of the International Conference on Information Fusion*, 2008, pp. 1–7.

- [20] P. Huang, T. Damarla, and M. Hasegawa-Johnson, “Multi-sensory features for personnel detection at border crossings,” in *Proceedings of the International Conference on Information Fusion*, 2011, pp. 1–8.
- [21] S. G. Iyengar, P. K. Varshney, and T. Damarla, “On the detection of footsteps based on acoustic and seismic sensing,” in *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, 2007, pp. 2248–2252.
- [22] T. Damarla, A. Mehmood, and J. Sabatier, “Detection of people and animals using non-imaging sensors,” in *Proceedings of the International Conference on Information Fusion*, 2011, pp. 1–8.
- [23] H. O. Park, A. A. Dibazar, and T. W. Berger, “Cadence analysis of temporal gait patterns for seismic discrimination between human and quadruped footsteps,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2009, pp. 1749–1752.
- [24] A. Mehmood, V. M. Patel, and T. Damarla, “Discrimination of bipeds from quadrupeds using seismic footstep signatures,” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, 2012, pp. 6920–6923.
- [25] C. Shen, L. Wang, I. Cho, S. Kim, S. Won, W. Plishker, and S. S. Bhattacharyya, “The DSPCAD lightweight dataflow environment: Introduction to LIDE version 0.1,” Institute for Advanced Computer Studies, University of Maryland at College Park, Tech. Rep. UMIACS-TR-2011-17, 2011, <http://hdl.handle.net/1903/12147>.
- [26] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, “Orcc: multimedia development made easy,” in *Proceedings of the ACM International Conference on Multimedia*, 2013, pp. 863–866.
- [27] M. Pelcat, J. Piat, M. Wipliez, S. Aridhi, and J.-F. Nezan, “An open framework for rapid prototyping of signal processing applications,” *EURASIP Journal on Embedded Systems*, vol. 2009, January 2009, article No. 11.
- [28] F. Palumbo, C. Sau, and L. Raffo, “Coarse-grained reconfiguration: dataflow-based power management,” *IET Computers and Digital Techniques*, vol. 9, no. 1, pp. 36–48, 2015.
- [29] S. Liang, *The Java Native Interface: Programmer’s Guide and Specification*. Addison-Wesley, 1999.
- [30] B. Cheng and B. Buzbee, “A JIT compiler for Android’s Dalvik VM,” 2010, pPDF presentation slides, Presented at Google I/O.
- [31] R. Yadav and R. S. Bhadoria, “Performance analysis for Android runtime environment,” in *Proceedings of the International Conference on Communication Systems and Network Technologies*, 2015, pp. 1076–1079.

- [32] “A performance comparison between Java and C on the Nexus 5,” 2016, <http://www.learnopengles.com/>, Visited on March 10, 2016.
- [33] I. Krajci and D. Cummings, *Android on x86: An Introduction to Optimizing for Intel Architecture*. Apress, 2013, ch. Creating and Porting NDK-Based Android Applications, pp. 75–130.
- [34] Y. Lu, X. J. Tang, N. Liu, Y. Y. Mao, M. X. Li, P. Xiao, and H. W. Wang, “Application and research of mobile terminal with Android in the equipment monitoring system,” in *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Programming*, 2014, pp. 293–296.
- [35] Y. Mori, H. Kojima, E. Kohno, S. Inoue, T. Ohta, Y. Kakuda, and A. Ito, “A self-configurable new generation children tracking system based on mobile ad hoc networks consisting of Android mobile terminals,” in *Proceedings of the International Symposium on Autonomous Decentralized Systems*, 2011, pp. 339–342.
- [36] Z. Wei, Q. Shi, and D. Jia, “Design and implementation of an intelligent information integration system based on Android mobile terminals,” in *Proceedings of the International Conference on Computer Science and Electronics Engineering*, 2012, pp. 161–165.
- [37] K. Nagata, S. Yamaguchi, and H. Ogawa, “A power saving method with consideration of performance in Android terminals,” in *Proceedings of the International Conference on Ubiquitous Intelligence & Computing and International Conference on Autonomic & Trusted Computing*, 2012, pp. 578–585.
- [38] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: orthogonalization of concerns and platform-based design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, December 2000.
- [39] H. Ben Salem, T. Damarla, K. Sudusinghe, W. Stechele, and S. S. Bhattacharyya, “Adaptive tracking of people and vehicles using mobile platforms,” *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 65, pp. 1–12, 2016. [Online]. Available: <http://dx.doi.org/10.1186/s13634-016-0356-9>
- [40] 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.
- [41] K. Martin and B. Hoffman, *Mastering CMake*. Kitware, Incorporated, 2015.
- [42] B. M. Smith, P. Chattopadhyay, A. Ray, S. Phoha, and T. Damarla, “Performance robustness of feature extraction for target detection & classification,” in *Proceedings of the American Control Conference*, 2014, pp. 3814–3819.
- [43] C. Hsu and C. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

- [44] X. Jin, S. Sarkar, A. Ray, S. Gupta, and T. Damarla, “Target detection and classification using seismic and PIR sensors,” *IEEE Sensors Journal*, vol. 12, no. 6, pp. 1709–1718, 2012.
- [45] B. Boehm *et al.*, “Cocoma II model definition manual,” University of Southern California, Tech. Rep., 1998.
- [46] L. H. Putnam and W. Myers, *Five Core Metrics: The Intelligence Behind Successful Software Management*. Dorset House, 2003.
- [47] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, 2nd ed. Springer, 2013, iISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online). [Online]. Available: <http://dx.doi.org/10.1007/978-1-4614-6859-2>
- [48] J. Altmann, “Acoustic and seismic signals of heavy military vehicles for cooperative verification,” *Journal of Sound and Vibration*, vol. 273, no. 4–5, pp. 713–740, 2004.
- [49] S. D. Bay, “Combining nearest neighbor classifiers through multiple feature subsets,” in *Proceedings of the International Conference on Machine Learning*, 1998, pp. 37–45.
- [50] C. J. S. R. A. O. L. Breiman, J. Friedman, *Classification and regression trees*. CRC press, 1984.
- [51] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.
- [52] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.
- [53] I. Rish, “An empirical study of the naive Bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.
- [54] A. P. Dempster, “Upper and lower probabilities induced by a multivalued mapping,” *The Annals of Mathematical Statistics*, vol. 38, no. 2, pp. 325–339, 1967.
- [55] H. Lee, H. Kwon, R. M. Robinson, W. D. Nothwang, and A. M. Marathe, “Dynamic belief fusion for object detection,” in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–9.
- [56] H. Wu, M. Siegel, R. Stiefelhagen, and J. Yang, “Sensor fusion using Dempster-Shafer theory [for context-aware HCI],” in *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, 2002, pp. 7–12.
- [57] D. H. Ballard, “Generalizing the Hough transform to detect arbitrary shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.

- [58] B. S. Riggan, W. E. Snyder, X. Wang, and J. Feng, “A human factors study of graphical passwords using biometrics,” in *Proceedings of the German Conference on Pattern Recognition*, 2014, pp. 464–475.
- [59] K. Krish, S. Heinrich, W. E. Snyder, H. Cakir, and S. Khorram, “Global registration of overlapping images using accumulative image features,” *Pattern Recognition Letters*, vol. 31, no. 2, pp. 112–118, 2010.
- [60] A. Martin, D. Charlet, and L. Mauuary, “Robust speech/non-speech detection using LDA applied to MFCC,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2001, pp. 237–240.
- [61] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [62] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” *Knowledge Discovery and Data Mining*, vol. 2, no. 2, 1998.
- [63] S. Knerr, L. Personnaz, and G. Dreyfus, “Single-layer learning revisited: a step-wise procedure for building and training a neural network,” in *Neurocomputing*. Springer, 1990, pp. 41–50.
- [64] J. H. Friedman, “Another approach to polychotomous classification,” Stanford University, Tech. Rep., October 1996.
- [65] S. M. Nabritt, T. Damarla, and G. Chatters, “Personnel and vehicle data collection at Aberdeen proving ground (APG) and its distribution for research,” US Army Research Laboratory, Tech. Rep. ARL-MR-0909, October 2015.
- [66] D. J. Hand and R. J. Till, “A simple generalisation of the area under the ROC curve for multiple class classification problems,” *Machine Learning*, vol. 45, no. 2, pp. 171–186, 2001.
- [67] A. A. Dibazar, H. O. Park, and T. W. Berger, “The application of dynamic synapse neural networks on footstep and vehicle recognition,” in *Proceedings of the International Joint Conference on Neural Networks*, 2007, pp. 1842–1846.
- [68] H. Ben Salem, T. Damarla, K. Sudusinghe, W. Stechele, and S. S. Bhattacharyya, “Adaptive tracking of people and vehicles using mobile platforms,” *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 65, pp. 1–12, 2016.
- [69] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
- [70] “Measurement computing (MCC) Linux drivers,” https://github.com/wjasper/Linux_Drivers/, visited on March 3, 2017.

- [71] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
- [72] R. M. Karp and R. E. Miller, "Properties of a model for parallel computations: Determinacy, termination, queuing," *SIAM Journal of Applied Math*, vol. 14, no. 6, November 1966.
- [73] L. Meng and J. P. Kerekes, "Object tracking using high resolution satellite imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 1, pp. 146–152, February 2012.
- [74] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed, "Detection, classification, and tracking of targets," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 17–29, March 2002.
- [75] W. H. Dragoset, "Marine vibrators and the Doppler effect," *GEOPHYSICS*, vol. 53, no. 11, pp. 1388–1398, 1988.
- [76] R. Chellappa, G. Qian, and Q. Zheng, "Vehicle detection and tracking using acoustic and video sensors," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, May 2004, pp. iii–793–6 vol.3.
- [77] G. Friedland, O. Vinyals, and T. Darrell, "Multimodal location estimation," in *Proceedings of the ACM International Conference on Multimedia*, 2010, pp. 1245–1252.
- [78] B. Ferris, D. Hähnel, and D. Fox, "Gaussian processes for signal strength-based location estimation," in *Proceeding of Robotics: Science and Systems*, vol. 2, 2007, pp. 303–310.
- [79] Y. Y. Cheng and Y. Y. Lin, "A new received signal strength based location estimation scheme for wireless sensor network," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1295–1299, August 2009.
- [80] R. Niu and P. K. Varshney, "Target location estimation in sensor networks with quantized data," *IEEE Transactions on Signal Processing*, vol. 54, no. 12, pp. 4519–4528, December 2006.
- [81] C. Feng, W. S. A. Au, S. Valaee, and Z. Tan, "Received-signal-strength-based indoor positioning using compressive sensing," *IEEE Transactions on Mobile Computing*, vol. 11, no. 12, pp. 1983–1993, December 2012.
- [82] Z. Yang and Y. Liu, "Quality of trilateration: Confidence-based iterative localization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 631–640, May 2010.
- [83] A. Artes-Rodriguez, M. Lazaro, and L. Tong, "Target location estimation in sensor networks using range information," in *Processing of the IEEE Workshop on Sensor Array and Multichannel Signal Processing*, July 2004, pp. 608–612.

- [84] K. Langendoen and N. Reijers, “Distributed localization in wireless sensor networks: a quantitative comparison,” *Computer Networks*, vol. 43, no. 4, pp. 499 – 518, 2003.
- [85] A. Awad, T. Frunzke, and F. Dressler, “Adaptive distance estimation and localization in wsn using rssi measures,” in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, August 2007, pp. 471–478.
- [86] M. Hefeeda and M. Bagheri, “Wireless sensor networks for early detection of forest fires,” in *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, October 2007, pp. 1–6.
- [87] T. Damarla, L. M. Kaplan, and G. T. Whipps, “Sniper localization using acoustic asynchronous sensors,” *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1469–1478, September 2010.
- [88] B. G. Quinn, “Doppler speed and range estimation using frequency and amplitude estimates,” *The Journal of the Acoustical Society of America*, vol. 98, no. 5, pp. 2560–2566, 1995.
- [89] V. Cevher, R. Chellappa, and J. H. McClellan, “Vehicle speed estimation using acoustic wave patterns,” *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 30–47, January 2009.
- [90] C. Couvreur and Y. Bresler, “Doppler-based motion estimation for wide-band sources from single passive sensor measurements,” in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, April 1997, pp. 3537–3540 vol.5.
- [91] A. Raina, “Synthesis of embedded software using dataflow schedule graphs,” Master’s thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, March 2017.
- [92] S. S. Bhattacharyya and J. Lilius, “Model-based representations for dataflow schedules,” in *Principles of Modeling*. Springer, 2018, pp. 88–105.
- [93] E. A. Lee, “The problem with threads,” *Computer*, vol. 39, no. 5, May 2006.
- [94] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. CRC Press, 2009, iSBN:1420048015.
- [95] J. T. Buck and E. A. Lee, “Scheduling dynamic dataflow graphs using the token flow model,” in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, April 1993.
- [96] R. Xie, H. Huttunen, S. Lin, S. S. Bhattacharyya, and J. Takala, “Resource-constrained implementation and optimization of a deep neural network for vehicle classification,” in *Proceedings of the European Signal Processing Conference*, Budapest, Hungary, August 2016, pp. 1862–1866.

- [97] F. Darema, “Dynamic data driven applications systems: A new paradigm for application simulations and measurements,” in *Proceedings of the International Conference on Computational Science*, 2004, pp. 662–669.