Prototyping Real-Time Tracking Systems on Mobile Devices

– Invited Paper –

Kyunghun Lee* Haifa Ben Salem* Thyagaraju Damarla†
Walter Stechele§ Shuvra S. Bhattacharyya*‡
*Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA
†U.S. Army Research Laboratory, Adelphi, MD, USA
§Institute for Integrated Systems, Technical University of Munich, Munich, Germany
‡Department of Pervasive Computing, Tampere University of Technology, Tampere, Finland
{leekh3, hbensale, ssb}@umd.edu {thyagaraju.damarla.civ}@mail.mil
{walter.stechele}@tum.de

ABSTRACT

In this paper, we address the design an 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.

Keywords

Dataflow, low power design, mobile platforms, model-based design, signal processing systems, target tracking.

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 [3, 8]). 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 paper 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, 17]).

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 in-

CF'16, May 16 - 19, 2016, Como, Italy

ACM ISBN 978-1-4503-4128-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2903150.2903471

volving 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. RELATED WORK

Various algorithms have been developed for detection of people and vehicles using acoustic sensors (e.g., see [11, 12, 14, 15, 26]). 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 [9, 24, 29]). In contrast to these works, which focus on algorithm aspects, we focus in this paper 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 [20]). In particular, we apply a dataflow-based design tool called the lightweight dataflow environment (LIDE) [33, 34], 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 developed 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 [38], PREESM [30], and the multi-dataflow composer (MDC) [28]. Exploration of such adaptations is a useful direction for future work.

3. 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 [21]. 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.

3.1 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 [7]. 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 [37]).

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 [1]. 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.

3.2 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 [21]. JNI operates within the framework of SDK-based application development, so it shares both advantages 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 [19]. 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.

3.3 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 3.2 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 [22, 25, 27, 36]). 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.

3.4 DICE

DPTS applies the The DSPCAD Integrative Command Line Environment (DICE), which 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, 17]. 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).

3.5 LIDE

DPTS also applies the The Lightweight Dataflow Environment (LIDE), which is a flexible design environment that allows designers to experiment with dataflow-based design and implementation directly on different types of programmable platforms [32, 33]. 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, such as our targeted form of ANT-based development, relatively easily. LIDE includes 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 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.

4. DICE-BASED PROTOTYPING FRAMEWORK FOR TRACKING SYSTEMS

Figure 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 [32, 33] 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 [18].

We demonstrate DPTS in this paper using a tracking system that was introduced recently by Ben Salem et al. [2]. 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) [10] to the design of adaptive tracking systems that are autonomously reconfigurable across different application scenarios and operational requirements.

In this paper, 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 paper 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 5 and Section 6.

4.1 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

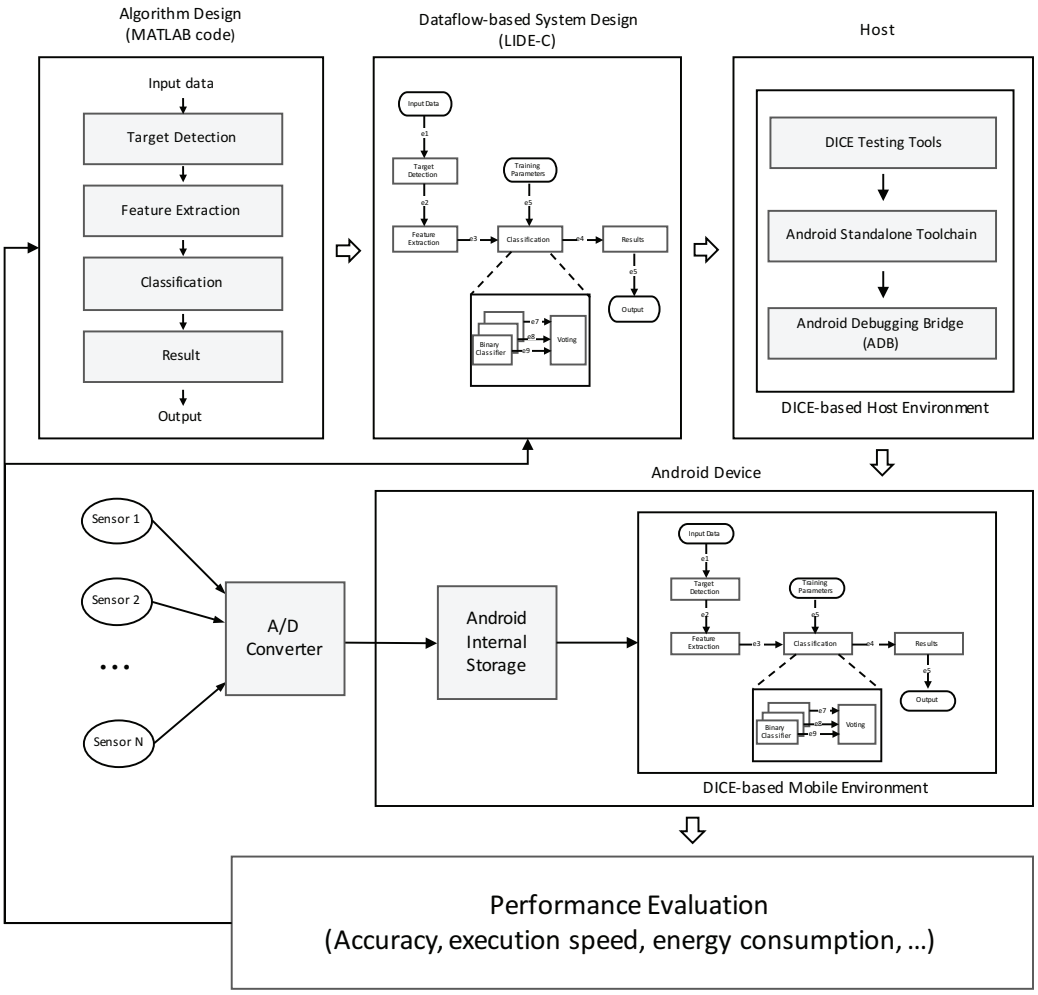

Figure 1: Illustration of the DPTS framework.

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 [23].

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.

4.2 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 3.

Figure 2 illustrates the integration among the sensing subsystem, embedded signal processing subsystem, and host device in the DPTS-based prototyping process.

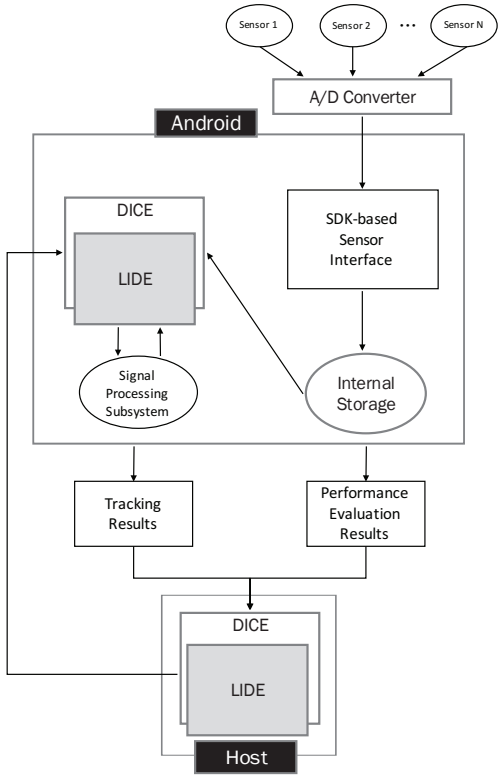

Figure 2: Integration among the sensing subsystem, embedded signal processing subsystem, and host computer in the DPTS-based prototyping process.

5. 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 4. 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 3 illustrates the top-level dataflow model for the signal processing core of the DTMSD system prototype that we experiment with in this case study. This dataflow model corresponds to the block in Figure 1 labeled Dataflow-based System Design and the block in Figure 2 labeled Signal Processing Subsystem.

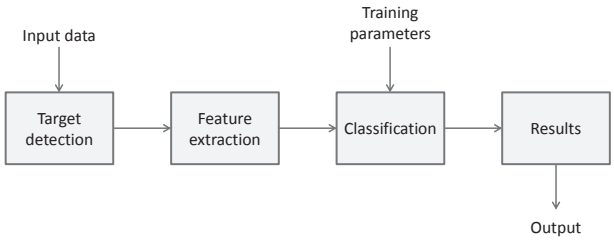

Figure 3: Top-level dataflow model for the signal processing core of the DTMSD prototype that we experiment with in this case study.

In the target detection actor of Figure 3, the input acoustic signal is filtered and 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 [9]. On the other hand, the second method uses cepstral analysis [35].

The classification stage shown in Figure 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 [13], and an approach involving a tree-structured combination of binary classifiers [16].

These three multi-class methods were applied in our tracking application, and evaluated as part of the algorithm design stage illustrated in Figure 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 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 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 6 provides more details on the experimental setup that we employed in this study.

Table 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	275 ms	9.17 ms	baseline
DPTS	198 ms	6.60 ms	28.0%

Table 2: CPU load comparison between the ANT-based development approach used in DPTS and conventional SDK-based development.

	CPU Load				
	CPU1	CPU2	CPU3	CPU4	CPU (Average)
SDK-based	67.499%	41.692%	62.092%	62.694%	58.494%
DPTS	72.397%	2.275%	67.232%	67.962%	52.467%

6. EXPERIMENTS

In this section, we present results of experiments using the case study involving detection of humans and vehicles described in Section 5. 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.

6.1 Execution Time Performance

Table 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.

The results of Table 1 show that by incorporating an ANT-based development approach in DPTS, we achieve a 28% improvement in application execution time.

Table 2 shows a comparison between the CPU load measured for an SDK-based 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 is the Qualcomm Trepn profiler. The results of Table 1 together with Table 2 show that DPTS provides a significantly lower overall CPU load, while providing improved execution time as well.

6.2 Lines of Code Efficiency

Table 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 [6,31]). 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 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 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 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 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 [4]. Thus, streamlined management of coordination code is a very useful feature of DPTS.

6.3 Energy Efficiency

Table 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

Table 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

segment size of 10,000 as reported in Section 6.1.

Table 4: Energy efficiency comparison.

	SDK-based	DPTS	Improvement
Data Segments Processed	134,860.9	156,405.6	16.0%
Energy Efficiency (Segments / mAh)	418.8	485.7	16.0%

The results in Table 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.

6.4 Sensor Experimentation

Another useful feature of DPTS is the capability for efficient interfacing and experimentation with alternative sensors (See Section 4.2), which is an important aspect of design space exploration and system-level optimization for tracking systems. Table 5 demonstrates experiments performed with two different acoustic sensors — a high quality external sensor sensor subsystem that is connected with an external A/D converter, and 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.

6.5 Summary

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 4, 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.

7. CONCLUSIONS

Table 5: Comparison between sensors.

Sensor	Built-in Sensor Subsystem	External Sensor Subsystem
Battery Time	30,409.0 sec	16,012.5 sec
Weight	0g	280g
Price (MSRP)	$0	$490
Accuracy (Vehicle)	70.59%	80.39%

In this paper, 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.

8. ACKNOWLEDGMENTS

This research was supported in part by the U.S. Army Research Laboratory, and by the U.S. Air Force Office of Scientific Research under the DDDAS Program.

9. REFERENCES

[1] A performance comparison between Java and C on the Nexus 5. http://www.learnopengles.com/, Visited on March 10, 2016.

[2] 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*, 2016. To appear.

[3] A. Benavoli and L. Chisci. Towards optimal energy-quality tradeoff in tracking via sensor networks. In *Proceedings of the European Control Conference*, pages 1523–1529, 2007.

[4] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, editors. *Handbook of Signal Processing Systems*. Springer, second edition, 2013. ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[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. Technical Report UMIACS-TR-2011-10, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. http://drum.lib.umd.edu/handle/1903/11422.

[6] B. Boehm et al. Cocomo II model definition manual. Technical report, University of Southern California, 1998.

[7] B. Cheng and B. Buzbee. A JIT compiler for Android's Dalvik VM, 2010. PDF presentation slides, Presented at Google I/O.

[8] 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*, pages 1776–1783, 2013.

[9] 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*, pages 1–8, 2011.

[10] F. Darema. Grid computing and beyond: The context of dynamic data driven applications systems. *Proceedings of the IEEE*, 93(2):692–697, 2005.

[11] M. F. Duarte and Y. H. Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, 2004.

[12] 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*, pages 1–7, 2008.

[13] C. Hsu and C. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[14] 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*, pages 1–8, 2011.

[15] 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*, pages 2248–2252, 2007.

[16] X. Jin, S. Sarkar, A. Ray, S. Gupta, and T. Damarla. Target detection and classification using seismic and PIR sensors. *IEEE Sensors Journal*, 12(6):1709–1718, 2012.

[17] 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*, pages 506–510, Barcelona, Spain, August 2011.

[18] 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*, 19, December 2000.

[19] I. Krajci and D. Cummings. *Android on x86: An Introduction to Optimizing for Intel Architecture*, chapter Creating and Porting NDK-Based Android Applications, pages 75–130. Apress, 2013.

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

[21] S. Liang. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley, 1999.

[22] 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*, pages 293–296, 2014.

[23] K. Martin and B. Hoffman. *Mastering CMake*. Kitware, Incorporated, 2015.

[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*, pages 6920–6923, 2012.

[25] 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*, pages 339–342, 2011.

[26] M. E. Munich. Bayesian subspace methods for acoustic signature recognition of vehicles. In *Proceedings of the European Signal Processing Conference*, pages 2107–2110, 2004.

[27] 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*, pages 578–585, 2012.

[28] F. Palumbo, C. Sau, and L. Raffo. Coarse-grained reconfiguration: dataflow-based power management. *IET Computers and Digital Techniques*, 9(1):36–48, 2015.

[29] 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*, pages 1749–1752, 2009.

[30] 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*, 2009, January 2009. Article No. 11.

[31] L. H. Putnam and W. Myers. *Five Core Metrics: The Intelligence Behind Successful Software Management*. Dorset House, 2003.

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

[33] 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*, pages 640–645, Washington DC, USA, November 2010.

[34] 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. Technical Report UMIACS-TR-2011-17, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. http://hdl.handle.net/1903/12147.

[35] 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*, pages 3814–3819, 2014.

[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*, pages 161–165, 2012.

[37] R. Yadav and R. S. Bhadoria. Performance analysis for Android runtime environment. In *Proceedings of the International Conference on Communication Systems and Network Technologies*, pages 1076–1079, 2015.

[38] 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*, pages 863–866, 2013.