ACCURACY- AND RESOURCE-AWARE FRAMEWORK FOR RESOURCE-CONSTRAINED MOBILE COMPUTING

by

PARUL PANDEY

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Dario Pompili

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

MAY, 2019

ABSTRACT OF THE DISSERTATION

Accuracy- and Resource-aware Framework for Resource-constrained Mobile Computing

By PARUL PANDEY

Dissertation Director:
Dario Pompili

Mobile computing is one of the largest untapped reservoirs in today's pervasive computing world as it has the potential to enable a variety of in-situ, real-time applications. However, the domain of mobile computing suffers from the constraints of limited resources such as device battery, CPU, and memory while at the same time users' expectations in terms of response times, accuracy, and data rates are increasing at a fast pace. As a result, achieving high energy efficiency while maintaining a high quality of service is a crucial challenge. Many of the mobile applications that are pervasive in our lives–such as localization, object/activity recognition, and mobile gaming to name a few–are expected to perform seamlessly with near-instantaneous responses, but are also affected by the same constraints. Current solutions based on offloading computationally-intensive applications from resource-constrained mobile devices to powerful remote computing platforms (such as the Cloud) or nearby mobile devices, suffer from uncertainty in wireless network connectivity or availability of devices in proximity, respectively.

To overcome the limitation of current works, the paradigm of approximate computing emerges as a solution to enable resource-intensive mobile applications in resource-constrained environment. Approximate computing reduces the amount of computation that an application is expected to perform, as a result of which the execution time reduces, which in turn reduces the energy consumption of the application. The gain achieved via reduction in energy consumption, however, comes with a

potential loss in the accuracy of the results (within acceptable limits). By leveraging approximate computing, we achieve dynamically a tradeoff between accuracy (or optimality of the results produced by an application) and utilization of the available resources (such as battery, CPU cycles, memory, and I/O data rate).

The goal of this thesis is to design new techniques so as to enable real-time computation-intensive mobile applications in resource-limited and uncertain environments. In order to achieve this goal, we leverage the paradigm of approximate computing and propose the following three solutions. First, approximation at the application level is introduced by joint optimization of algorithm and parameter space of different tasks in the application and a light-weight algorithm is developed that selects the approximated tasks that should be executed to meet the application deadline under uncertainties encountered at run-time. Second, temporal correlation between the continuous stream of frames obtained from the camera sensors is exploited to learn the application parameters that give acceptable accuracy in each frame of the video with significant savings in time and energy. The problem of selecting the algorithm and input parameters for a video is cast as a Markov Decision Process. Third, to reduce the energy consumption of data-intensive applications in distributed camera networks a novel protocol is proposed to identify the camera nodes in the network with correlated multimedia data. Low-computational-complexity metrics are used to quantify the correlation across cameras nodes by using only local knowledge of the network available to the camera nodes. Furthermore, the effectiveness of the proposed approaches is validated through extensive simulations on publicly available datasets and data collected by building multiple end-to-end computationally-intensive applications from the computer vision domain. The proposed innovations in this research will provide novel solutions to the issue of limited resource availability in mobile devices and will foster the development of mobile research community.

Acknowledgements

I would like to express my sincere gratitude towards my PhD advisor, Dr. Dario Pompili, for having had faith in my abilities throughout the course of my PhD. I would like to thank him for pushing me to achieve things that I did not think would have been possible. During our many discussions, he has advised me on discipline and on how to do quality work, and also provided useful feedback on my work. He has always held his students to the highest standard, which has been instrumental in improving the quality of my work. His ability to do multi-disciplinary research is remarkable and motivates me to expand my knowledge, to always strive to improve, and to never give up. Thank you Prof. Pompili, for encouraging me to set bigger goals and supporting me in achieving them.

I would like to extend my gratitude to Drs. Manish Parashar, Ivan Marsic, and Roberto Tron for accepting to be the committee members in my dissertation defense, and I thank them for the valuable comments during the course of my PhD. I would like to especially thank Prof. Tron for his valuable suggestions on my work and for guiding me in improving the quality of my experiments. Thank you Prof. Tron, for always being available to answer my questions and for providing time and encouragement via many Skype calls and emails.

I would also like to thank CPS Lab members, Abolfazl Hajisami, Mehdi Rahmati, Vidyasagar Sadhu, Tuyen Tran, Ayman Younis, and Xueyuan Zhao, with whom I have brainstormed on several different occasions that have helped me immensely. They have also encouraged me several times over the course of my PhD. I especially want to thank Mehdi for always being there to listen to my technical and personal problems and for helping me work through my problems no matter how long it took. I would like to also thank all my wonderful friends, Divya Raju, Eden Buenaventura, Azita Nouri, Chryssalenia Koumpouzi, Priya Govindan, and Parneet Kaur for their support in this journey. I especially want to thank Priya for motivating me and always pushing me to try harder.

This thesis could not have reached its fruition without the presence of Dr. Diane Simmons in my life. Thank you for always finding time to meet with me even with your busy schedule and

discussing at length all the difficulties I was facing in graduate school. I also cannot fathom this thesis without my guardian angel, Russi Soffer. Thank you Russi, for always finding time to meet me, suggesting innovating ways to handle stress in graduate school, and always reminding me of my accomplishments whenever I was unhappy about my progress in graduate school.

Finally, I would like to thank my parents, Mrs. Kalpana Pandey and Mr. Kamal Kishore Pandey, for always believing in me and supporting me via countless Skype calls. I would like to especially thank my father who always stood by me and never questioned my dreams no matter how impossible they seemed. I would also like to thank my husband, Aditya Desai, for always being there to enthusiastically listen to how my day was. Thank you Aditya, for being my best friend and cheerleader and for helping me find my strengths when I could not find any.

Dedication

To

Mama, Papa, and Aditya

Table of Contents

List of Tables

List of Figures

List of Acronyms

AUC Area Under Curve.

AUV Autonomous Underwater Vehicle.

CBIR Content Based Image Retrieval.

DAG Directed Acyclic Graph.

DFS Depth First Search.

DP Data Provider.

FoV Field of View.

FPS Frames Per Second.

HoG Histogram of Gradients.

IoU Intersection over Union.

JPEG Joint Photographic Experts Group.

KLT Kanade Lucas Tomasi.

KUL Katholieke Universiteit Leuven.

LAN Local Area Network.

LiDAR Light Detection and Ranging.

LTE Long-Term Evolution.

MCC Mobile Cloud Computing.

MDC Mobile Device Cloud.

MDP Markov Decision Process.

MSE Mean Squared Error.

NS Network Simulator.

OC Opportunistic Computing.

QoS Quality of Service.

ROC Receiver Operating Characteristic.

ROS Robot Operating System.

RP Resource Provider.

RPC Remote Procedure Call.

RSP Restricted Shortest Path.

SIFT Scale Invariant Feature Transform.

SP Service Provider.

SVM Support Vector Machine.

VM Virtual Machine.

Chapter 1

Introduction

1.1 Resource-constrained Mobile Computing

Nature provides miraculous ways of sustaining life in resource-constrained environments. In a situation of limited availability of food, for example, snakes lower their metabolic rates up to 72% and sand gazelles shrink their heart and liver size in order to survive. Humans, in a situation of starvation, change their source of energy to fatty acids when their primary source of energy, glycogen, has exhausted. These bio-inspired examples show how Nature provides a *'graceful' degradation* of operation rather than shutting down completely when resources are limited. Similarly, in computation, compression and lossy encoding methods are routinely used to transmit smaller and faster data so to meet real-time demands, and heuristics are applied to find (sub-optimal) solutions to otherwise computationally infeasible problems.

Technology too has the power to adapt to the limitations in human perceptions. With high-speed and time-lapse photography, we can appreciate and understand processes not visible to human eye (as either happening too fast or too slowly); with the creation of overlays from multiple, spatially separated data sources on Google Earth we can visualize information not naturally visible to human senses; with deep-learning techniques we can achieve leaps of improvement in mature domains such as speech recognition [11]. All these technologies, which help us understand phenomena unimaginable otherwise, have *computation as their core infrastructure*. We envision mobile computing to become pervasive and bring all these technologies anywhere and everywhere.

The state of the art in mobile computing falls short in achieving this vision on hand-held devices. This computing paradigm, in fact, suffers when the available resources – such as device battery, CPU cycles, memory, I/O data rate – are limited. In spite of these limitations, many computation-intensive applications from a variety of domains such as computer vision (e.g., object recognition, panorama stitching), machine learning (e.g., natural language translator, speech recognizer), and

artificial intelligence (e.g., gaming applications, online learning) are expected to work seamlessly on smart hand-held devices and give results in *real time*. In fact, in an event of failure of the device (on which the computation is being performed) due to exhaustion of battery, no result for the application will be obtained. Major breakthrough in battery capacity came around the turn of the century when due to the transition from nickel to lithium, the battery energy density nearly doubled [12]. The impact of this advancement is witnessed by the fact that most of the mobile devices have lithium-ion batteries. However, the improvement in battery since then has been slow and it is unlikely that the fundamental problems limiting a faster trend will be solved in the near future. We have seen new technologies aimed at replacing lithium-ion batteries such as super-capacitors, graphene-based batteries, and solid state batteries, however, all these technologies are still at their infancy [13, 14].

To overcome the constraints of limited computational power of mobile devices a large body of work on mobile cloud computing moves the application execution from the resource-constrained mobile devices to powerful and centralized remote computing platforms such as the Cloud or static idle device in the neighborhood. This concept has been named as cyber-foraging. Various offloading methods based on Remote Procedure Call (RPC) procedure calls, Virtual Machine (VM) migration, and mobile code and at different levels of granularity such as coarse partitioning (entire application is offloaded) or fine-grained offloading (only resource-intensive tasks are offloaded) have been proposed. However, offloading faces many challenges, for example, good connectivity from the device to a WiFi network may not always be possible. Although 3G has a near-ubiquitous coverage, recent studies have shown that round-trip times are often long and that communication links are bandwidth limited; the former have been shown to be consistently on the order of hundreds of milliseconds and in some cases even reaching seconds [15]. This is unacceptable in real-time and interactive applications, which require low response times.

Since physical resources appear to remain constrained for the foreseeable future and the software-based approaches proposed in literature are not suitable for applications that require response in real-time, we tackle the problem from another angle. We present an “energy-" and “accuracy-aware" framework that exploits the new paradigm of *approximate computing* to enable near real-time mobile applications in resource-constrained environments. Approximate computing reduces the amount of computation that an application is expected to perform, as a result of which the execution time, i.e., the makespan, as well as the energy consumption, reduce. The gain achieved via

reduction in makespan and energy expenditure, however, comes with a potential loss in the accuracy of the results (within acceptable limits). In a situation of resource scarcity a sudden breakdown of an application due to the exact implementation of tasks (with higher resource demand), may eventually lead to wastage of resources without generating meaningful results. To circumvent this problem we provide an application-layer solution to provide a graceful degradation of service in resource-constrained environments.

1.2 Research Challenges

One of the challenges of applying approximate computing to an application is to determine if it is a good candidate for approximation, i.e., will there be significant gains in energy for small loss in accuracy. An application is made up of number of tasks, hence, we have to determine the tasks that can be approximated and which tasks are not approximable. We have to make sure that when different tasks of an application are approximated together, the overall accuracy loss of the application is lower than the maximum acceptable accuracy loss.

One of the foremost question that a run-time execution of an approximate framework has to handle if an application requires approximation given the available resources. Second, if approximation is required, when the number of resources (CPU cycles or battery) is not enough to perform an exact execution of the application then how much accuracy loss should be incurred to execute the applications with the given resources. We can prevent jeopardizing the execution of an application at run-time due to limited resource availability by introducing approximate computing for the mobile cloud computing domain. However, the execution of tasks at run-time faces uncertainty when the execution time of the application during run-time does not mirror the behavior observed during the offline profiling. Execution time of tasks depends on its implementation along with input parameters, size of input data, input value, and architecture of the device. For a given implementation of a task and input parameter value, the task execution time can vary significantly with input data; in certain situations, it can lead to missing the application deadline. In order to enable approximate computation at run-time and get results in near real time, we should be able to answer the following questions, (i) how much accuracy loss should be incurred to provide meaningful results within the application deadline, (ii) which tasks should be executed to deliver results within the acceptable

accuracy loss while simultaneously meeting such deadline, and (iii) how does the uncertainty in the mobile distributed environment impact the performance gain of approximate computing. The first item identifies how much accuracy loss an application should incur to meet the application deadline and if this is within the acceptable accuracy loss range set up the application developer. If we are able to get acceptable accuracy loss then the second item identifies all the approximate implementations of the application, which give result within the acceptable accuracy range and executes the instance with minimum execution time.

1.3 Research Objectives and Contribution

In order to address the research challenges associated with ensuring quality of service in mobile computing via approximation, we make the following contributions in this dissertation.

1.3.1 Mobile Distributed Computing Framework via Approximation

We introduce a new paradigm of *approximate computing* to enable real-time computation-intensive mobile applications in resource-limited and uncertain environments. An energy- and accuracy-aware approximate-computing framework to support real-time mobile applications in limited resource environments [16, 17]. We present an offline phase that 1.) determines if the application is a good candidate for approximation, 2.) which tasks in the application are approximable, and 3.) statistical guarantees on the speed-up achieved for a certain loss in accuracy when the tasks in the application are approximated. An online algorithm that selects the approximated tasks that should be executed to meet the application deadline under uncertainties encountered at run-time is also presented. Finally, our approach is validated through simulation and testbed experiments comparing the performance of approximate versus exact computing. We motivate our results via two different algorithms for interactive perceptive object recognition, and observed that on our testbed their approximate implementations performed better than their exact counterpart.

1.3.2 Light-weight Object Detection and Decision Making via Approximate Computing in Resource-constrained Mobile Robots

In the earlier research task we looked at the resources available at a mobile device to determine the parameters of the application that should be executed. We now look at the problem of running computationally applications on resource-constrained devices from another angle. Specifically, we see how we can tailor the parameters of a computationally intensive algorithm to the incoming input data stream and thereby achieve savings in time and energy. This approach leverages the offline phase explained in the previous research task to identify the parameter space. We propose a decision framework that selects the parameters of computationally intensive algorithm that are "good-enough", i.e., parameters that give acceptable accuracy in each frame of the video with significant savings in time and energy. To this end, we use object detection and tracking for drone navigation as a case study where we apply our proposed solution and study the effectiveness of our approach. Most of the current solutions for autonomous flights in indoor environments rely on purely geometric maps (e.g., point clouds). There has been, however, a growing interest in supplementing such maps semantic information (e.g., object detections) using computer vision algorithms. Unfortunately, there is a disconnect between the relatively heavy computational requirements of these computer vision solutions, and the limited capacities available on mobile autonomous platforms. In this chapter we propose to bridge this gap with a novel Markov Decision Process (MDP) that adapts the parameters of the vision algorithms to the incoming video data rather than fixing them apriori [18, 19]. As a concrete example, we test our framework on a object detection and tracking task, showing significant benefits in terms of energy consumption without considerable loss in accuracy on a combination of publicly available and novel datasets. We identify via offline experiments the algorithm and parameters of object detection application that can be approximated and show that we can achieve an increase in speed. Finally, we validate the proposed solution on publicly available datasets, and on a new dataset of videos collected with a Bebop drone.

1.3.3 Robust Distributed Dictionary Learning for In-network Image Compression

In this chapter we move from a single camera system such as a mobile device, Google glass or drones to a distributed camera system. Camera networks are resource-constrained distributed systems that communicate over (wireless) networks to make decisions collaboratively. Multiple cameras enable redundancy and hence prevent against single point of failure and object occlusion. For surveillance applications, the camera nodes in a network take decisions about an object of interest within incoming videos by coordinating with neighboring nodes, which is a costly process in terms of both time and energy. Data-compression methods can bring significant energy savings in camera nodes while transmitting or storing data in the network. Signal representation using sparse approximations and overcomplete dictionaries have received considerable attention in recent years and have been shown to outperform traditional compression methods. However, distributed dictionary learning itself relies on consensus-building algorithms, which involve communicating with neighboring nodes until convergence is achieved. To this end, we design a novel protocol to enable energy-efficient and robust dictionary learning in distributed camera networks by leveraging the spatial correlation of collected multimedia data [20]. We employ low-computational-complexity metrics to quantify the correlation across cameras nodes. We also present a feasibility study of the parameters of the network that impact the performance of distributed dictionary learning and consensus process in terms of accuracy of the algorithm and energy consumed by the camera nodes. The performance of the proposed approach is validated through extensive simulations using a network simulator and public datasets as well as via real-world experiments on a testbed of Raspberry Pi nodes.

1.4 Dissertation Organization

This rest of this dissertation is organized as follows.

Chapter 2 reviews the related and prior work in the fields of exact mobile computing that includes mobile cloud computing and mobile device clouds. The proposed approaches are discussed in detail along with their pros and cons. We also present via experiments the limits of exact mobile computing in uncertain mobile environment and motivate the need of approximate computing. We quantify the gain achieved by Mobile Device Cloud (MDC) in comparison to a centralized execution and the effects of various factors that introduce uncertainty in MDCs. We also study self-healing

based fault tolerance techniques for distributed mobile environment and finally, motivate the need for approximate computing in MDCs.

Chapter 3 describes the entities of our approximate-computing framework. We present details of an exhaustive offline phase which determines the that helps us identify promising applications whose tasks can be approximated so to gain significant benefits in energy at the cost of marginal loss in accuracy. We also provide a novel solution applying approximate computing to time-critical applications under run-time uncertainties. We also study the performance of approximate versus exact computing. We consider applications from computer vision domain. We provide details of our experimental setup, input data sets, and software used to quantify this gain.

Chapter 4 describes a novel MDP framework that adapts the parameters of the vision algorithms to the incoming video data rather than fixing them apriori. For this, we test our framework on a object detection and tracking task, showing significant benefits in terms of energy consumption without considerable loss in accuracy on a combination of publicly available and novel datasets. We validate the proposed solution on publicly available datasets, and on a new dataset of videos collected with a Bebop drone, achieving up to 30 fps for 480 X 270 video frames with an accuracy drop of less than 2% with respect to a fixed choice of parameters.

Chapter 5 describes energy-efficient dictionary-learning techniques for distributed camera networks. First a feasibility study of the parameters of the network that impact the performance distributed dictionary learning in terms of accuracy and energy is presented. Next, a novel design for energy efficient and robust dictionary learning framework specific to distributed camera networks is presented by considering the spatial correlation of cameras, so that the consumed energy of the nodes does not drop below a predefined threshold. The effectiveness of the proposed approach is validated on image compression as an application through extensive simulation using network simulator and public datasets.

Chapter 6 summarizes our main contributions and provides suggestions on future research directions that will push the state-of-the-art in the area of enable computationally intensive applications on resource constrained devices.

1.5 Summary

In this chapter we presented different works in the area of mobile computing to handle the issue of limited resource availability to run computationally intensive applications. We introduced the concept of approximate computing to solve this issue and studied the research challenges associated with applying approximate computing. Finally, we presented the contributions introduced in different chapters of this thesis.

Chapter 2

Exact Mobile Computing: Background and Limits

Cyber foraging was first introduced in [21] to overcome the challenges faced by resource-constrained mobile devices. The term covers the opportunistic use of available computing resources by resource-constrained mobile devices that surrogate computers, i.e., the stronger computers offering up their services, may be in possession of. These resources could be battery capacity, CPU cycles, storage, network connectivity, display capabilities, printers etc. Traditional mobile computing realized the goal of cyber foraging by offloading resource-intensive tasks to nearby compute servers. The advancements in different areas helped in making the vision of cyber foraging a reality. We now explain the advancements that have made cyber foraging possible:

Advancements in wireless technology: Advancements in wireless networking have made it possible to offload tasks from mobile robots to remote resource-rich entities with low communication delay and high reliability. Current standard IEEE 802.11n interfaces provide a net data rate ranging from 54 to 600 Mbps, which is an increase of up to six times with respect to legacy 802.11a/b/g network performance.

Availability of public Clouds: A datacenter hardware and software constitutes a Cloud. Public Cloud services open a large computation infrastructure (in terms of VMs, operating system, storage space, RAM, etc.) to the scientific community as well as to the general public at a nominal price (e.g., VMs with maximum number of CPU cores are sold at 0.8 $\$/\mathrm{hr}$) [22]. Public Clouds come under the domain of utility computing, and their services are sold by many different companies such as RackSpace, Amazon Web Services, and Microsoft Azure.

Advancements in virtualization technology: Virtualization provides the necessary abstraction such that the underlying resources (e.g., raw compute, memory, storage, network interfaces) can be unified as a *pool of resources*, and resource/service overlays (e.g., data storage services, Web hosting environments) can be built on top of them. This technology improves scalability and overall

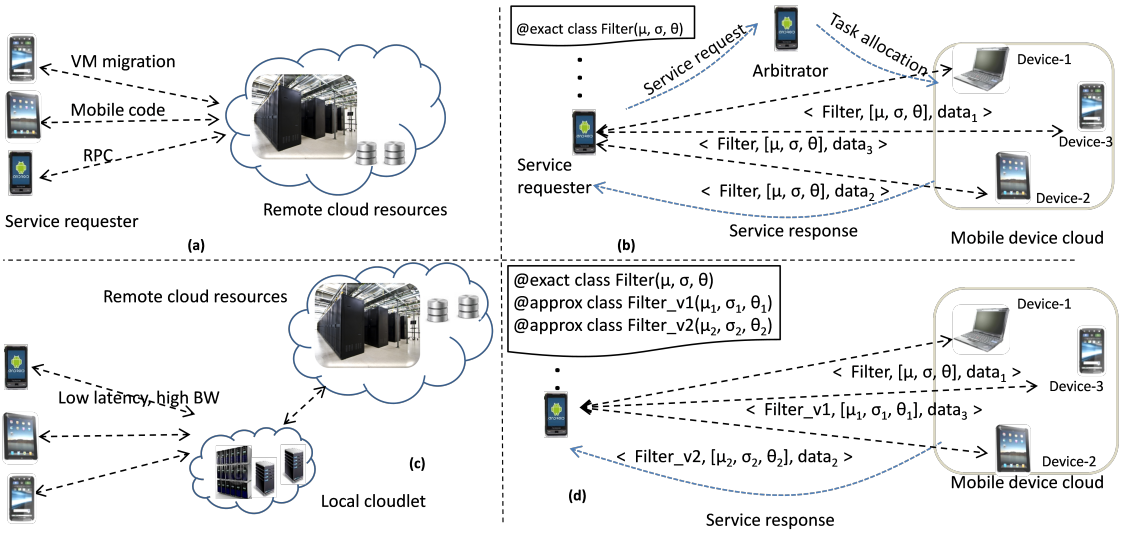


Figure 2.1: Different concepts introduced in the mobile computing domain to overcome the resource-constraints offered by limited battery capacity and CPU cycles: (a) Offloading to the cloud via Virtual Machine (VM) migration or Mobile code (offloading executable code) or Remote procedure calls (RPC); (b) Mobile Device Clouds (MDCs); (c) Cloudlets [1]; (d) Approximate Computing.

hardware-resource utilization, provides enormous cost benefits to the datacenter owners, and is one of the main enabling technologies of Cloud Computing.

Classification of cyber foraging techniques: Research efforts towards realizing the vision of cyber foraging can be broadly classified into works in the fields of Mobile Cloud Computing (MCC), Opportunistic Computing (OC), and MDC as shown in Fig. 2.1. MCC offloads application from resource-constrained mobile devices to powerful and centralized remote computing platforms such as the Cloud. Therefore, many works in MCC have primarily focused on offloading expensive (compute and energy-intensive) tasks to *dedicated* and *trusted* computing resources, either situated remotely (in the *cloud* [15, 23–26] or proximally (in *cloudlets* [27]) in a transparent manner. However, these approaches are not suitable for enabling data-intensive applications in real time due to prohibitive communication cost and response time, significant energy footprint, and the curse of extreme centralization. The works in OC and MDC explored the feasibility of leveraging the computing and communication capabilities of other mobile devices *in the field* to enable innovative mobile applications. Recent research in the areas of MDC and OC have explored the potential of code offloading to proximal devices by following two entirely different approaches. Solutions for MDCs advocate a structured and robust approach to workflow and resource management; whereas

Table 2.1: Comparison of various techniques in mobile computing domain to handle the issue of limited resource availability to run computationally intensive applications.

Technique	Pros	Cons
Remote procedure call [28]	Stable APIs	Need prior installation
Virtual machine migration [15, 29]	No code rewriting	Time consuming VM synthesis
Mobile code [30]	Dynamic deployment	Security constraints
Mobile device cloud [31, 32]	Low latency	Scarcity of nearby devices
Approximate computing [17, 33]	Speed up in execution	Nominal loss in accuracy

OC depends entirely on direct encounters and is highly unstructured with little or no performance guarantees. However, both these works do not take into account the inherent uncertain nature of distributed mobile computing due to wireless network connectivity and mobility of mobile devices. We will now discuss each of these techniques in detail. Interested readers can refer to Table 2.1 to study the pros and cons of different techniques in mobile computing domain to handle the issue of limited resource availability to run computationally intensive applications.

2.1 Mobile Cloud Computing

As mentioned earlier MCC offloads application from a resource-constrained mobile device to powerful and centralized remote computing platforms such as the Cloud. In [1] the authors presented “cloudlets" as an alternative to remote datacenters to provide crisp application response to the user for interactive applications. The cloudlet is a collection of multi-core computers, with gigabit internal connectivity and high bandwidth wireless LAN. The mobile devices connect to the cloudlet for low latency interactive applications and the cloudlet connects to the cloud. However, this approach requires existence of a cloudlet in a proximity which might not always be possible. The MCC have been implemented in the literature via different techniques. These techniques differ in type of offloading methods used (Pre-installed RPCs, VM Migration, Mobile code), surrogate type (stationary or mobile), granularity of offloading (entire application or certain tasks), and objective of offloading (reduction in execution time or energy). We will now cover how each of these techniques can be implemented.

In offloading via RPCs [34, 35] static offloading was used, i.e., the program was partitioned before the execution to indicate which methods or sub-routines of the program will be offloaded.

In [15], the application developers annotated the programs which can be offloaded to gain savings in energy and during run-time based on network connectivity decisions are taken on where the functions should be executed, remote server or locally. RPC based approach requires that applications are pre-installed in both the service-requester and the surrogate. This leads to small overhead and application execution can start at the surrogate as soon as the application input data is received. Application code need not be sent. Also, RPC based approach is language agnostic. Application can be written in any language that supports RPC standard. The second method of offloading was VM migration. VM migration refers to the transferring the memory image of a VM from a source server (here, service requesting mobile device) to the destination server without stopping its execution. This gives an impression of seamless migration to the user and also does not require any changes to the application code. The third method was via mobile code, which requires installation of the application on the surrogate devices. In [30] the entire application is offloaded via mobile code. This is advantageous as it does not require any preparation and installation time will be short as in the RPC approach. The decision to offload, including which tasks to offload and to which device is determined via a cost-benefit analysis. This analysis estimates the cost of offloading to determine if there is any gain from offloading to the remote resource, where gain is defined as a reduction in execution time and energy expenditure achieved upon offloading.

The surrogate selection is restricted to devices that have the application installed. In terms of surrogate type it could be offloading to a nearby mobile device or stationary entity such as a remote server or nearby desktop. Granularity of offloading involves identifying how to partition the application to identify the parts of the application that will be executed locally and in the cloud. Finally, objective of offloading depends on the type of application. For interactive applications the application is expected to give results in near real-time and hence the objective is to reduce time. However, for applications that are not latency sensitive reduction in energy is used as the objective.

Limitations of MCC: As mentioned MCC offloads application from resource-constrained mobile devices to powerful and centralized remote computing platforms such as the Cloud to save battery capacity of local devices. However, this approach has its own drawbacks: For example, good connectivity from the device to a WiFi network may not always be possible, leading to frequent disconnections. Offloading to nearby static idle surrogates or remote cloud servers via mobile

broadband technologies such as 3G or 4G Long-Term Evolution (LTE) have been studied [15]. According to this work although 3G has a near-ubiquitous coverage, recent studies have shown that round-trip times are often long and that communication links are bandwidth limited in comparison to wireless Local Area Network (LAN); the former have been shown to be consistently on the order of hundreds of milliseconds and in some cases even reaching seconds [15]. This is unacceptable in real-time/interactive applications, which require low response times. Higher round-trip times also lead to higher energy consumption at the devices. Although the penetration of mobile broadband internet is very high in developed countries (86.7%), it is much lower in the developing countries, such as India, at around (39.1%) [36]. On the other hand, the roaming data rates offered by leading service providers to tourist is very high, for example, Verizon offers a $\$40$ per month international travel plan that comes with only 100 MB of data. This fee will rise quickly if the user is relying heavily on mobile data [37]. Other options for travelers or in developing countries is to search for free WiFi hotspots.

2.2 Mobile Device Clouds

To address the limitation of MCC mentioned above, the concept of offloading computation in cloud computing can be used to address the problem of limited resource availability in mobile computing by using resource providers (mobile devices) in the *vicinity* other than the mobile device itself to host the execution of mobile applications. A pool of proximal mobile devices that offer their resources to a weak mobile device (i.e., a device with insufficient battery capacity to execute an application) form a MDC. A weak mobile device can offload its tasks to a MDC and the devices forming the MDC collaborate to execute tasks on behalf of the weak device. This paradigm aims to overcome the problems experienced by traditional mobile computing by relying on nearby devices and hence have lower latency and also relying only on existing infrastructure of mobile devices instead of setting up new infrastructure for cloudlets.

The paradigm of MDC is based on splitting the tasks in an application and executing them in parallel on nearby mobile devices. Many of the works on MDCs have focused on developing task solutions to allocate tasks to nearby devices based on different objective such as conserving energy of the device requesting service in an MDC, fairness (conserving energy of the whole device pool)

[31, 38], and achieving speed-up [32]. A variety of MDC environment have also been evaluated such as single-hop, multi-hop, highly collaborative, medium collaborative, opportunistic along with various communication technologies such as WiFi and Bluetooth [39]. We present briefly our work on energy-aware mobile device clouds [31].

Framework entities: The entities (i.e., the mobile devices in the vicinity) of the distributed computing framework may at any time play one or more of the following three logical roles: i) *service requester*, which places requests for workloads that require additional data and/or computing resources from other devices, ii) *Service Provider (SP)*, which can be a data provider, resource provider, or both, and iii) an *arbitrator* (usually, the base station), which processes the requests from the requesters, determines the set of service providers that will provide or process data, and distributes the workload tasks among them. The service requester offloads the task of executing compute-intensive algorithms to the SPs by submitting service requests to one of the arbitrators. Resource providers lend their computational (CPU cycles), storage (volatile and non-volatile memory), and communication (i.e., network interface capacity) resources for processing data.

Energy-aware resource allocation engine: The arbitrator is aided with an energy-aware resource allocation engine that distributes the workload tasks optimally among the service providers by taking into account the different resource capabilities (in terms of residual battery capacity) in the vicinity. To make this decision the arbitrator needs information about the resources available at each SP. This is achieved via service advertisements from SPs. Service advertisements will include information about the current position, amount of computing, in terms of normalized CPU cycles), memory, and communication resources, the start and end times of the availability of those resources, and the available battery capacity at each service provider. The arbitrator is aware of the instantaneous power drawn by the workload tasks of a specific application when running on a specific class of CPU and memory as well as network resources at each service provider as the information about the different types of devices is known in advance. Our framework applies to applications exhibiting *data parallelism* as well as to applications exhibiting *task parallelism*. Also, our framework is endowed with several *autonomic* capabilities such as self organization, self optimization, and self healing. The self-organization capability (for handling service discovery and service request arrivals as well as for task distribution and management) is imparted by the role-based architectural framework. It also facilitates interactions among the mobile entities for coordination and seamless

Table 2.2: Characteristics of the computing devices in our testbed.

Devices	**Samsung Galaxy Tab**	**ZTE Avid N9120**	**Huawei M931**	**Toshiba Satellite**	**Dell Insp-iron**	**Acer Asprire**
Type of devices	Tablet	Smart phone	Smart phone	Laptop	Netbook	Netbook
No. of devices	2	3	1	1	1	1
CPU	1 GHz Dual-core ARM	1.2 GHz Dual-core	1.5 GHz Dual-core	2.13 GHz i3 Intel	1.66 GHz N450 Intel	1.60 GHz N270 Intel
OS	Android v4.0	Android v4.0	Android v4.0	Windows 7	Windows 7	Windows XP
RAM [GB]	1	0.512	1	4	1	2
Battery [mAh]/[V]	7,000/4	1,730/5	1,650/10.8	4,200/10.8	5,200/11.1	4,840/11.1

switching among the three logical roles, namely, service requesters, service providers, and arbitrators. Interested readers can read more about the framework on energy-aware resource allocation engine developed for MDCs in [31].

Limitations of MDCs: The MDC might suffer from scarcity of devices to which computation can be outsourced. If the number of devices forming the MDC is lower than the number of tasks that can be executed in parallel, then the computation gain by distributed computing reduces. The number of devices in MDC should be such that the computation gain by parallel execution of tasks is higher than the communication gain. In cases of data-intensive applications, the computation gain obtained by parallel execution of sub-tasks might not be able to surpass the communication cost incurred by distributing tasks to nearby devices. The uncertain network connectivity might lead to multiple transmissions by the SPs and service requester. Inherent uncertainty in mobile computing due to device mobility may lead to unavailability of SPs and consequently unavailability of results from SPs. This may result in wastage of resources of service requester. Also, it might not be possible to split tasks in an application to independent sub-tasks to take advantage of their parallel execution or the computation gain obtained by parallel execution of sub-tasks might not be able to surpass the communication cost incurred by distributing tasks to nearby devices.

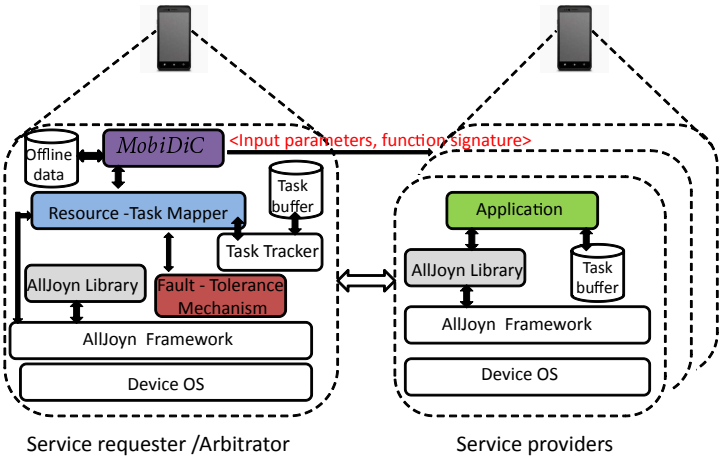


Figure 2.2: Architecture of our mobile device cloud testbed to evaluate the performance of exact mobile computing. This app communicates with our mobile distributed computing framework ($MobiDiC$) which in turn communicates with the resource-task mapper to identify how tasks should be distributed to nearby resources in the MDC.

2.2.1 Performance of MDCs

We now take an experimental approach to study the limits of exact mobile computing in MDCs. Uncertainty in a MDC may arise due to device mobility, which determines the availability duration of devices, and network connectivity, which determines the communication cost of offloading tasks to nearby devices. We compared the performance of centralized execution with Round robin (RR) based task allocation. We now give details of our experimental setup and then quantify the performance obtained.

Experiment testbed: We present the various devices of our experimental testbed, which form the MDC as shown in Table 2.2. Our testbed comprises of state-of-the-art, heterogeneous computing devices (tablets, smartphones, laptops, and notebooks) that vary by type of device, platform, RAM, and processing power. Figure 2.2 shows the architecture of our experimental testbed. The device serving as the service requester contains the resource task mapper, which is responsible to allocate task to different service providers. Our framework also includes self-healing [31] as the fault-tolerance mechanism. We assume a fair, simple, and robust round-robin-based technique to distribute tasks in an MDC.

AllJoyn framework: The communication between arbitrator and the service provider is achieved

via AllJoyn framework [40]. The AllJoyn framework is an open-source platform-independent, software system that provides an environment for distributed applications running across different device classes with an emphasis on mobility, security, and dynamic configuration. The AllJoyn advertisement and discovery mechanism takes care of seamlessly discovering these peers independent of the underlying transport being used. An AllJoyn thin app is designed for energy-, memory-, and CPU-constrained devices. The thin app is designed to have a very small memory footprint and is typically single-threaded service running in the background of the application. This app communicates with our mobile distributed computing framework $MobiDiC$ as shown in Fig 2.2 which in turn communicates with the resource-task mapper to identify how tasks should be distributed to nearby resources in the MDC. In Fig. 2.3 we show the timeline of communication between the arbitrator and SP.

Fault-tolerance in MDC: Our testbed achieves fault-tolerance via Self-Healing. The goal of self-healing is to reallocate the failed task to a new service provider. The arbitrator keeps track of the tasks allocated to different SPs and if the device does not return tasks within a pre-allocated amount of time the self-healing mechanism at the arbitrator aborts the thread responsible for collecting tasks from that device and reallocates it to a new service provider.

Impact of uncertainty: Uncertainty here is used to include the effects arising due to mobility of devices, i.e., the devices have moved out of the network or are no longer reachable by the arbitrator and hence, cannot receive a new task from the arbitrator or return the results back to the arbitrator. We do not consider the effect of poor network availability and consider a stable network connection between service providers and arbitrator. We consider various parameters that impact the gain achieved via distributed implementation. Specifically, we consider mean and standard deviation of availability duration of devices, inter-arrival times, task sizes, buffer size (number of task allocated to each device in RR), battery capacity, and processing power. We model the mobility patterns of devices in the proximity as a normal distribution with mean availability duration of devices varying with $\mu = \{5, 100, 200\}$ s and σ=5 s. Our first result shows the gain obtained by the execution in a MDC in comparison to a centralized computation. In this scenario, we implement the Canny edge-detection application on devices via exact computation. In Fig. 2.4, we plot the time taken to execute an application as the mean availability duration of the devices in the MDC is varied. Interestingly, as the arrival duration of devices increases, the rate at which tasks are completed increases. Also,

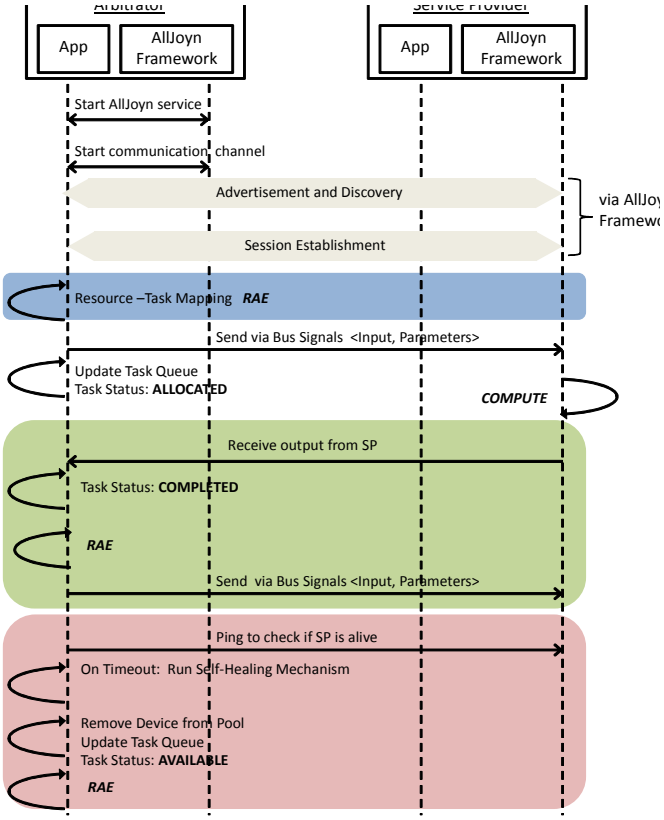


Figure 2.3: Communication between Arbitrator and Service Provider. The blue block indicates the execution of resource-task mapper, the green block indicates successful execution of a task, and the red block indicates the failure to receive results from a SP and steps taken to reallocate a failed task.

in spite of the offloading cost, MDCs finish the execution faster than centralized exact computing. However, as the uncertainty with the availability of mobile device decreases as identified by μ decreasing from 200 s to 100 s, the application takes longer to execute. This is because as the availability duration decreases, the SPs are leaving the network before they finish the task as a result of which the arbitrator has to reallocate the tasks. The performance also gets worse as the task size doubles (in our application it means image resolution doubles) because the availability duration of SPs in the MDC is not enough to complete the tasks assigned to them. These experiments indicate that performance of an MDC is sensitive to task size and the availability duration of SPs. We now discuss the paradigm of approximate computing and how it can overcome the limitations in MDCs.

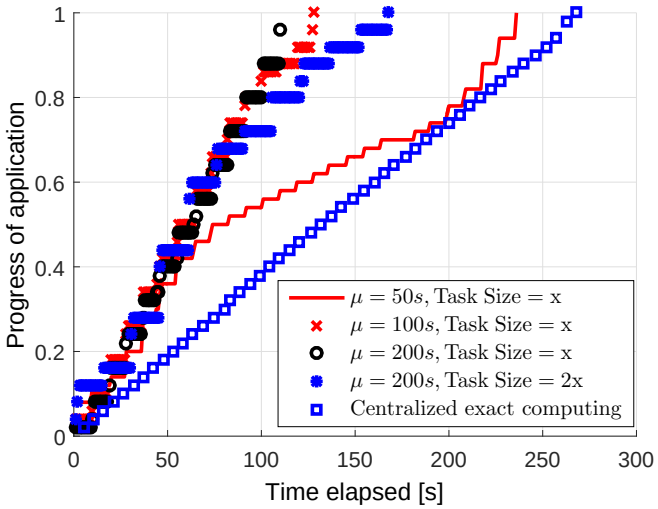


Figure 2.4: Experiments: Comparison of performance of exact computing in a centralized implementation vs. in a mobile device cloud in the presence of uncertainty in i) device availability due to network disconnections and device mobility and ii) variable task sizes.

2.2.2 Approximate Computing

As explained in the previous section, the MDCs suffer from various limitations such as, the local resource pool might suffer from scarcity of devices to which computation can be outsourced or uncertain network connectivity and device availability. Also, it might not be possible to split tasks in an application to independent sub-tasks to take advantage of their parallel execution or the computation gain obtained by parallel execution of sub-tasks might not be able to surpass the communication cost incurred by distributing tasks to nearby devices.

Researchers have developed energy-aware programming languages by introducing approximation at different levels such as mathematical operations and storage of data structures (in the form of unreliable register, data cache, and main memory). One such language is EnerJ [41], which allows the programmer to annotate data as ‘approximate’ or ‘precise’. The system then automatically maps approximate variables to low-power storage, uses low-power operations, and applies more energy-efficient algorithms provided by the programmer. In [33,42,43], the authors employ various approximation techniques such as loop perforation and multiple implementations of tasks.

Earlier work in the field on AI involved work in imprecise computations [44] that focused on offering a tradeoff between execution time and accuracy of an application. However, this work assume a very rigid structure of task for the applications on which imprecise computations can be applied.

Each task can be expressed to have an optional and mandatory part. They solve a scheduling problem to minimize the average time of optional tasks of the application subject to the constraints the total error is less than a acceptable value. Authors in [45] have worked on estimating the resources available for an application in terms of CPU cycles. In this work if the available computational power is not enough to meet the accuracy requirements then an implementation of the application which requires fewer resources and achieves lower accuracy is selected. This work does not explain any technique on how to identify different implementations of the application with possibly lower accuracy. In the next chapter we will give details on our novel approximate computing based solution that will dynamically achieve a tradeoff between accuracy of the application and utilization of the available resources on the device. We will also validate the effectiveness of the proposed approach through extensive simulations on publicly available datasets and data collected by building multiple end-to-end computationally-intensive applications from the computer vision domain.

2.3 Summary

In this chapter we introduced the current work in the area mobile computing, specifically, in the area of mobile cloud computing, mobile device clouds, and opportunistic computing and discussed the pros and cons of each approach. We also presented experimental results to show the limitations of mobile device clouds. Finally, we gave details of the work done in the area of approximate computing and the limitations of those works.

Chapter 3

Exploiting the Untapped Potential of Mobile Distributed Computing via Approximation

In this chapter, the new paradigm of *approximate computing* is proposed to harness such potential and to enable real-time computation-intensive mobile applications in resource-limited and uncertain environments. A reduction in time and energy consumed by an application is obtained via approximate computing by decreasing the amount of computation needed; such improvement, however, comes with the potential loss in accuracy. Hence, a Mobile Distributed Computing framework, is introduced to determine *offline* the 'approximable' tasks in an application and a light-weight *online* algorithm is devised to select the approximate version of the tasks in an application during run time. The effectiveness of the proposed approach is validated through extensive simulation and testbed experiments by comparing approximate versus exact-computation performance.

3.1 Introduction

Our goal is to achieve dynamically a *tradeoff* between *accuracy* (or optimality of the results produced by an application) and *utilization* of the available resources (such as battery, CPU cycles, memory, and I/O data rate). We first discuss a structural approach to approximation in mobile computing. Then, we present the approximation techniques that can be applied to different tasks in an application. We first define an offline phase that helps us identify promising applications whose tasks can be approximated so to gain significant benefits in energy at the cost of marginal loss in accuracy. Once we are able to identify the tasks within an application that can be approximated and the accuracy loss that is incurred by the approximation, we are able to remove the dependence on annotations from the programmer or manually-coded optimization. This is possible because our framework allows the selection of the appropriate type of approximation at run-time based on (1) the real-time available resources (e.g., available energy, CPU cycles, and memory) and (2)

the application requirements (application deadline and accuracy). We present an offline algorithm, Optimized Rich Workflow, that determines the tasks in an application that can be approximated. These candidates provide speed up in execution time with a potential decrease in accuracy. Our next step is to determine the speed up and accuracy achievable in an application when multiple tasks in an application are approximated together. To this end, we extend a generalized workflow representation to a 'rich' workflow representation with approximation transformation applied to tasks. We further reduce the approximable task space via a Pareto-optimality tests to generate 'optimized' workflow. Exact execution of a time-critical application is severely limited by resource availability and resource uncertainty. To prevent such breakdown of an application due to the lack of resources we propose a checkpointing mechanism to keep track of the resources available during online execution of the application. We will now explain the online phase of our proposed solution which is executed at run-time. Next, we determine the acceptable loss acceptable at runtime and present a light-weight algorithm (Heuristic MP – SP) that selects the approximable tasks that will be executed at run-time given the application deadline, acceptable accuracy loss, and computational resources available on the device.

3.2 Approximate Computing Framework

We now present an ontology of approximation that we have developed for mobile computing applications. We also present in detail the two types of approximation techniques that can be applied to different tasks in an application. An offline phase is presented that helps us to identify promising applications whose tasks can be approximated so to gain significant benefits in energy at the cost of marginal loss in accuracy.

3.2.1 Ontology of Approximation

Types of tasks: An application consists of the execution of a set of tasks to obtain the required result. We consider a task in an application to be "elementary" if it cannot be split further into sub-tasks. Each task is represented by an executable code/function (to represent a functionality that cannot be split further) and a set of input parameters. We divide tasks into two different categories, namely, *approximable* and *non-approximable*. We assume that identifying to which category a task

belongs can be achieved via offline profiling (discussed later) or identified by an application developer. The two categories of task are:

Approximable: Tasks that can be approximated to achieve significant savings in energy and/or execution time, with however a potential loss of accuracy in the result.

Non-approximable: Tasks whose execution without any approximation is necessary for the success of the application, i.e., if any approximation technique were applied on these tasks, the application would not generate meaningful results.

Types of approximations: We introduce approximation through two transformations, namely *substitution* and *discarding*, which are applied to different tasks (both at function and input parameter) of the application. Specifically, the former transforms the task(s) in exact computation with those with lower degree of complexity; whereas the latter involves removing certain task(s) of an application used for exact computation. We now briefly explain these transformations.

Substitution: This transformation requires substitution of a computation task (its execution code or input parameter) by a simpler task. At the *function level*, this operation refers to the substitution of a task in exact computation by a computationally less-demanding task with potential loss in accuracy. This requires the availability of multiple implementations of a task, each with a different degree of complexity (e.g., 2D Gaussian function serves as a filtering kernel in image processing; however, it can be replaced with recursive Gaussian or box filters, which are both computationally much less demanding albeit they provide lower accuracy [46]). Substitution transformation requires domain knowledge.

At the *parameter level*, it refers to the scaling up or down of the exact implementation value of a task parameter. A *substitution factor* f determines the factor by which the value of the approximate parameter varies with respect to the exact parameter; for example, if the value of the parameter in the case of exact computation is p, the new value via substitution will be $p * f$. For example, in Content Based Image Retrieval (CBIR) applications, whose aim is to retrieve image features via histogram analysis, the number of bins can be decreased (here, $f < 1$) in such a way as to reduce the computational cost at the cost of a decreased output accuracy. We introduce examples of various applications where substitution transformation can be applied in Table 3.1.

Discarding: Applications consist of tasks that successively improve upon the results obtained from previously executed tasks. Discarding transformation involves not executing these tasks so to

reduce energy consumption at the cost, however, of reduced accuracy. At the *function level*, if the user-specified accuracy is achieved by a subset of the tasks, the application can choose to skip the remaining task and terminate early; hence, discarding certain redundant tasks can lead to significant benefits in terms of energy and/or execution time.

At the *parameter level*, it refers to early termination or skipping of number of iterations in a task. Skipping parameter space was introduced in [33], where only one every n scheduled iterations was in fact executed, as a result of which the systems performs fewer computations than its exact-implementation counterpart. Discarding transformation can be applied to traditional Fast Fourier Transform-based algorithms to get suboptimal results with reduced computation cost [47]. We introduce examples of various applications where discarding transformation can be applied in Table 3.2 of this chapter.

Accuracy metric: In our framework we compare the accuracy or quality of output of an application by executing the application via exact computation and by applying the aforementioned approximate techniques. Different metrics such as F_1 *Score* (i.e., $2\frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$), *peak-signal-to-noise ratio* or any other application-domain metrics can be used to measure the output accuracy. An exact-computation implementation gives the highest accuracy achievable for that application. The percentage loss in accuracy of the output when applying approximation w.r.t. exact computation is calculated as $T = \frac{Q-\hat{Q}}{Q} \cdot 100$, where Q is the accuracy of the output obtained by exact implementation of the application and $\hat{Q}$ is the accuracy of the output obtained by the approximate implementation of the application. We now present our novel solution on transformation of workflows to introduce approximation in mobile applications.

3.2.2 Transformation of Workflows

The order of execution of multiple tasks in an application can be specified by a *workflow*. Here, we first explain our workflow representation for an exact computation implementation, and then show how such workflow is transformed for an approximate-computation implementation. Transformation of workflows is accomplished offline and is leveraged at run-time to make decisions when the application is executed.

Exact-workflow representation: Let the exact workflow $G(V, E)$ be presented by a Directed Acyclic Graph (DAG), as shown in Fig. 3.1 (Left). The workflow is composed of multiple stages

Table 3.1: Various tasks and parameters in different applications that can be substituted to simpler versions.

	Algorithm	**Function**	**Function Version**	**Parameter**
Substitution	Canny/HoG	Smoothing Filter	Box filter Recursive filter Median filter	Kernel size
		Gradient	LoG Roberts operator Sobel operator Prewitt operator	
	HoG/SIFT/CBIR	Histogram		Number of orientation bins
	HoG	HoG descriptor		Cell and Block Size
		Normalization		Type of norm
	Canny	Thinning		Threshold
	SIFT			Number of octaves Number of spatial bins Number of levels

Table 3.2: Various tasks and parameters in different applications that can be discarded.

	Algorithm	**Function**	**Parameter**
Discarding	Canny/HoG	Smoothing Filter	
	Canny	Thinning	
	HoG	Normalization	
	Query Matching		Number of matches
	FFT		Number of iteration

with a set of tasks to be performed at each stage. It is a graphical representation of the set of tasks, $V = \{k_{i,j}\}$, where $k_{i,j}$ is the j^{th} task in the i^{th} stage. The edges in the workflow indicate the dependencies between tasks. Tasks at a stage cannot be executed unless all the tasks in the previous stage have been completed as tasks at a stage accept data from the previous stages. In the workflow representation, square nodes (□) represent the input data whereas circular nodes (◯) represent the computation tasks.

Determining approximable tasks: We explain now how to identify approximable tasks in an application. For example, to determine if Task $k_{1,1}$ is approximable, we first apply discarding transformation separately to each of its input parameter and alternate functions available. We repeat the same by using substitution transformation. Such procedure results in multiple approximate versions of the task. Then, we replace Task $k_{1,1}$ with one of its approximate versions while all other tasks in the exact workflow are left unchanged. After such replacement, we calculate the resulting

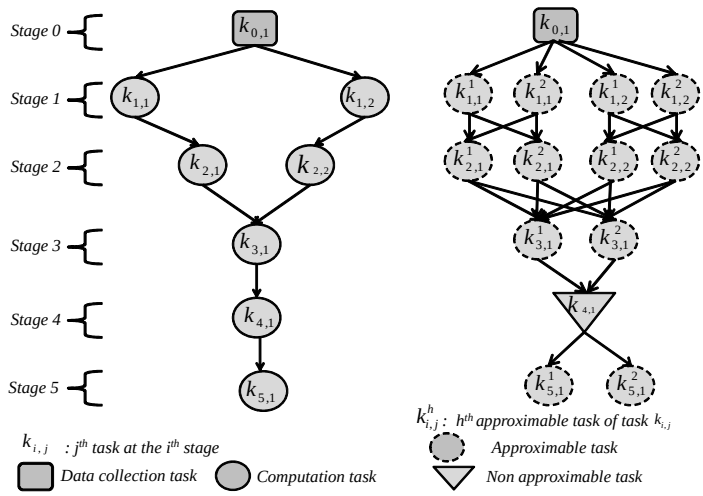


Figure 3.1: (Left) Exact workflow representation; (Right) Rich workflow constructed by extending exact workflow to represent approximation transformations. *Substitution* transformation is represented by multiple (alternate) tasks in a stage (e.g., tasks $k^1_{1,1}$, $k^2_{1,1}$, are approximate substitute tasks for task $k_{1,1}$ in Stage 1); *Discarding* transformation is shown by skipping a task in a certain stage (e.g., Task $k_{2,2}$ in the exact workflow is skipped in Stage 2 and $k^2_{3,1}$ is executed immediately after $k^2_{1,2}$).

makespan and accuracy (Q) of the workflow. The exact-computation implementation gives the highest accuracy results for the application.

The speed-up (*sp*) obtained from one of the approximate versions is calculated by dividing the makespan of the approximate version by the makespan associated with its exact implementation. This is done for a large number of input data so to get the average speed up ($\overline{sp}$) and average accuracy ($\overline{Q}$).

If any approximate version of Task $k_{1,1}$ provides $\overline{sp} > 1$ along with accuracy loss less than the acceptable loss T_A, then $k_{1,1}$ is considered an approximable task. The approximate versions that do not satisfy these constraints are discarded. If none of the approximate versions of a task satisfies these constraints, then that task is deemed non-approximable. If multiple implementations of a task are available, then substitution transformation can be applied; otherwise, only discarding transformation is performed.

Rich-workflow representation: An *approximate instance* of an exact workflow is the one whose tasks satisfy the constraints mentioned earlier. The collection of all the approximate instances of an application forms a rich-workflow, $G^R(V^R, E^R)$. In Fig. 3.1 (Right), we can see that each approximable task ($k_{i,j}$) in the exact workflow has a corresponding approximate version ($k^l_{i,j}$).

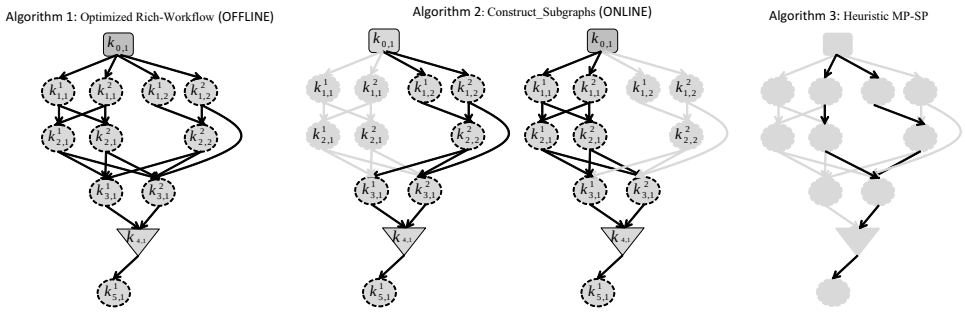


Figure 3.2: Illustration of (a) *Optimized Rich Workflow* constructed from Rich Workflow (Fig. 3.1 (Right)) by reducing the task space; (b) *Subgraphs* formed for multiple independent tasks starting at Stage-1 of the exact workflow (Fig. 3.1 (Left)). The subgraphs are created by Algorithm 2: Construct_Subgraphs and are executed only when the exact workflow is task-parallel with multiple independent tasks at different stages of the workflow; (c) *Approximate Workflow* extracted from Optimized Rich Workflow at run-time.

Each approximate version ($k^l_{i,j}$) in the exact workflow is selected via Algorithm 1 when $sp > 1$ and $\overline{Q} < T_A$. The edge of the rich-workflow is represented as $e^{m,n,l}_{i,j,h} = \{<k^h_{i,j}, k^l_{m,n}> \in E^R\}$. Note that, in Fig. 3.1(Right), non-approximable tasks are represented by triangular nodes (∇).

Reducing approximation space: We discard the approximate instances in the rich workflow that give accuracy loss less than T_A. To further reduce the approximation space in the rich workflow, we select only those approximate instances of the application that are *Pareto Optimal*. An approximate instance is Pareto-optimal if there is no other approximate version of that task that provides *both* better speed up *and* accuracy, i.e., t_1 is a Pareto-optimal approximate instance iff it there is not any other approximate instance t_2 s.t. $\overline{\hat{Q}}(t_1) \leq \overline{\hat{Q}}(t_2) \wedge \overline{sp}(t_1) \leq \overline{sp}(t_2)$, where *at least one* of the inequality is *strict*. The collection of these approximate instances, which consist of Pareto-optimal approximate instances that give percentage accuracy loss w.r.t. exact computation less than T_A, form an *optimized rich workflow*, i.e., $G^O(V^O, E^O)$. Figure 3.2(a) is an example of optimized workflow formed by applying Pareto-optimal test on Fig. 3.1(Right).

Need for an offline phase: The offline tools mentioned above help the programmer identify the subroutines and input parameters of the application that can benefit from various approximation techniques. However, these tools are too heavy to be used during run-time, as the cost of executing these tools at run-time may paradoxically be greater than the savings in time and energy obtained from approximation of the application. As a result, these tools are implemented only offline. Selection of Pareto-optimal tasks reduces the complexity of online mechanisms as it reduces

Algorithm 1: Optimized Rich Workflow (Offline)

Input: A-Application, $\mathcal{B}$-set of approximate versions of all tasks in A, $\mathcal{I}$-Test data set, T_A-acceptable accuracy loss of A
Output: $G^O(V^O, E^O)$-Optimized rich-workflow

$\hat{\mathcal{B}} = \emptyset$;
for $b \in |\mathcal{B}|$ **do**
  for $i \in \mathcal{I}$ **do**
    Replace an exact task in A with b;
    Execute A with input i to get $\hat{Q}_i$ and sp_i ;
  end
  $\overline{\hat{Q}} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \hat{Q}_i$, $\overline{sp} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} sp_i$;
  if $\frac{|\overline{Q} - \overline{\hat{Q}}|}{\overline{Q}} \cdot 100 < T_A \wedge \overline{sp} > 1$ **then**
    $\hat{\mathcal{B}} = \hat{\mathcal{B}} \cup b$;
  end
end
Construct Rich-workflow using tasks in $\hat{\mathcal{B}}$;
Select Pareto-optimal approximate instances to form the Optimized Rich-workflow;

the approximation space and helps the application select optimal approximated tasks from a much smaller space as well as meet the deadline constraints.

3.2.3 Approximation via Input Data

So far we have focused on introducing approximation in the application at the algorithm level by manipulating tasks in the applications. Specifically, we have focused on approximation via joint optimization of function and parameters of the tasks in an application. We will focus now on how approximation of input data can bring additional benefits to compute-intensive applications. Subsampling of images to reduce the image resolution/size has been implemented in literature via a variety of resampling filters such as point filters, box filter, and median filters of input data. To prevent aliasing arising due to subsampling, the image needs to be pre-filtered (e.g., with Gaussian filter) before applying resampling filters [48]. In point filters, one pixel value within a local neighborhood is chosen (perhaps randomly) to be representative of its surroundings. This method is computationally simple but can lead to poor results if the sampling neighborhoods are too large. The second method interpolates among pixel values within a neighborhood by taking a statistical sample (such as the mean or median) of the local intensity values. In this chapter we use nearest-neighbor interpolation method for subsampling.

Our goal is to reduce the amount of input data processed by the application such that the overall execution time of the application reduces. To this end, we reduce the spatial resolution of input

data via various subsampling techniques and study how reduction in spatial resolution impacts the accuracy of the application while bringing simultaneous gains by reducing the makespan. The reduction of input data required for processing can be done at various levels of the input data, i.e., at the raw data (pixel level) and information level (after extracting semantic knowledge from the image), and can be divided into three main classes:

- Data-dependent approximation;
- Information-dependent approximation;
- Hybrid approximation.

In this work we consider the data-dependent approximation and will leave the last two classes for future work. The data-dependent approximation or identifying the sampling rate is done independent of the task and parameter based approximation explained in earlier. At run-time, however, a combination of data-dependent approximation and optimized rich-workflow (Pareto-optimal approximate instances) can be used. In Sect. 3.4 we will present the gains achieved by combining these two approaches. We now give details of how approximation can be applied at run-time in mobile applications.

3.3 Real-time Approximate Computing

Uncertainty at run-time arises when the execution time of the application during run-time does not mirror the behavior observed during the offline profiling. Execution time of tasks depends on its implementation along with input parameters, size of input data, input value, and architecture of the execution location. For a given implementation of a task and input parameter value, the task execution time can vary significantly with input data; in certain situations, it can lead to missing the application deadline. In order to enable approximate computation at run-time and get results in near real time, we should be able to answer the following questions:

- Given the resources available, how much accuracy loss should be incurred to provide meaningful results within the application deadline?
- Which tasks should be executed to deliver results within the acceptable accuracy loss while simultaneously meeting such deadline?

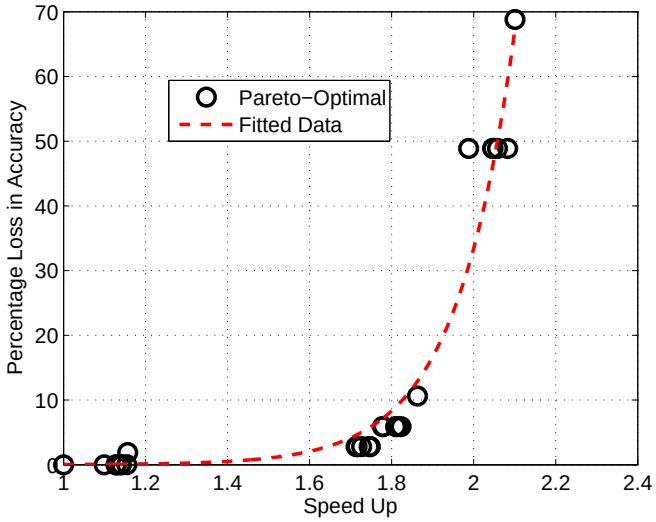


Figure 3.3: Fitting offline profiling data with a non-linear model, $\mathrm{A} \cdot \exp(\frac{\mathrm{sp}}{\mathrm{B}})$, to estimate at run-time the loss in accuracy that should be incurred to achieve a certain speed up.

- How does the uncertainty in the mobile distributed environment impact the performance gain of approximate computing?

In this section we provide novel mechanisms to enable approximate computing in resource-limited environment. An *approximate workflow* is constructed that consists of an approximate version of each task in the exact computation workflow, which is selected from the optimized workflow. We leverage the information obtained from extensive profiling of the application to construct the approximate workflow. This approximate workflow is estimated at run-time based on the resources available with the mobile device and has to be determined only once "per-session", i.e., only once before the application is to be executed.

Determination of accuracy loss: Let sp be the amount of speed up required to complete the execution of the application within its specified deadline. Our goal is to specify to the user at run-time how much accuracy loss needs to be incurred in order to achieve this speed up, given the available computational resources. For this we fit the offline profiling data of the Canny edge-detection application (black circles) with a non-linear model, $\mathrm{A} \cdot \exp(\frac{\mathrm{sp}}{\mathrm{B}})$, which is shown by red-dotted line in Fig 3.3. Goodness of fit statistics such as root mean square error are used to estimate the coefficients A and B. In this work we have learned this model and its parameters specifically for the Canny edge detection application.

Construction of approximate workflow: Our next goal is to determine the approximate instance of the optimized workflow that meets both the makespan and the estimated accuracy loss

bound. Such approximate instance is called an *approximate workflow*. We now present a light-weight solution to determine such approximate workflow at run-time by leveraging the results from offline profiling.

Each edge, $e_{i,j,h}^{m,n,l}$, of the optimized rich-workflow gives the value of execution time of task $k_{m,n}^{l}$, denoted as $d(e_{i,j,h}^{m,n,l})$, after task $k_{i,j}^{h}$ has been executed. For a particular device, the offline profiling provides us with the execution time for running a task of an application with different input data, resulting in varying execution times for the task. Hence, the execution time of an edge can be defined as a real-valued random variable in (0, $+\infty$) varying with the input data set. Theoretically, the distribution of $d(e)$ for any edge e can be captured by a Probability Density Function (PDF); however, in reality, the PDF, $f_d(e)$, of $d(e)$ is often unknown. Instead, a set of samples $\hat{d}(e) = [d_1, d_2, \ldots d_W]$, which are obtained from offline profiling, are used to approximate the distribution of $d(e)$, where W is the number of trials in the offline profiling. Each sample of $\hat{d}(e)$ has a $\Pr\{\hat{d}(e) = d_w\} \in (0, 1]$, where $\sum_{w=1}^{W} \Pr\{\hat{d}(e) = d_w\} = 1$. For sake of compactness, we simply denote $\hat{d}(e)$ as $d(e)$.

A path is a set of consecutive edges that connect the source (first task in the workflow) to the destination node (terminal task in the workflow). The execution time (or makespan) of an application is the sum of execution times of all the edges in a path p and is given by $D(p) = \sum_{e_{i,j,h}^{m,n,l} \in p} d(e_{i,j,h}^{m,n,l})$.

As $d(e_{i,j,h}^{m,n,l})$ is a random variable, $D(p)$ is also a random variable. The delay of a sample path, w, of $D(p)$, associated with a single trial (i.e., an input data) is given by $\sum_{\substack{e_{i,j,h}^{m,n,l} \in p \\ w \in \mathrm{W}}} d_w(e_{i,j,h}^{m,n,l})$.

Each edge is associated with W instances and there are multiple path edges in an application. Our goal is to create a light-weight run-time algorithm; hence, we reduce the complexity of the problem by transforming each edge $d(e)$. We find the edge sample w that has the highest probability, i.e., $w := \max p_w(e)$, $\forall e$, and substitute $d(e)$ with $d_w(e)$.

Now, given an application deadline M, our goal is to find a path $p^* = \arg\max_p \Pr\{D(p) \leq M\}$, i.e., that path for which, for every other path p, it holds,

$$\Pr\{D(p^*) \leq M\} \geq \Pr\{D(p) \leq M\}, \forall p. \tag{3.1}$$

The probability of a path is given as the product of the probability of edges on that path. To solve

Algorithm 2: Construct_Subgraphs (Online)

Input: $G^O(V^O, E^O)$
Output: G_{sub}- Subgraphs
Child set: *Child* $\leftarrow \emptyset$;
$\mathcal{I}$ contains all i^{th} stages, where, $j > 1$;
for $i' \in \mathcal{I}$ **do**
  $J = \max j$ for i'^{th} stage;
  while $j' > J$ **do**
    $i\_temp = i'$;
    $G_{sub}(i', j') = G_{sub}(i', j') \cap k^h_{i\_temp, j'} \forall h$;
    $i\_temp = i\_temp + 1$;
    if $i\_temp \notin \mathcal{I}$ **then**
      $G_{sub}(i', j') = G_{sub}(i', j') \cap$ *Child*$(k^h_{i\_temp,1})$;
    end
    else
      break ;
    end
  end
end

Algorithm 3: Heuristic MP $-$ SP (Online)

Input: $G^O(V^O, E^O)$, k, M , S, $child\_val(v)$-number of children of node v, $d_h(e)$, $p_h(e) \forall \{e, h\}$
Output: $G^A(V^A, E^A)$- Approximate workflow
$count \leftarrow 1$;
$d(e) \leftarrow d_w(e)$ & $p(e) \leftarrow p_w(e)$, where $w := \max p_w(e)\ \forall\ e$;
while $G^O(V^O, E^O) \neq \emptyset \cup$ *count* $< k$ **do**
  $[P, D] = Dijkstra(G^O(V^O, E^O))$;
  for $v \in P$ **do**
    $child\_val(v) = child\_val(v) - 1$;
  end
  if $D > M$ **then**
    break ;
  end
  else
    $G^A(V^A, E^A) \leftarrow G^A(V^A, E^A) \cup P$;
    Remove tasks from $G^O(V^O, E^O)$ with $child\_val = 0$;
    $count = count + 1$;
  end
end

this problem, we transform each probability function as a cost function $c() = -\log(f(d(e)))$, which makes the components additive. Now, as the probability associated with an edge increases, the cost decreases; hence, *our goal is to find the path, with the lowest cost function, that simultaneously meets the makespan constraints*. We formulate this problem as a *Restricted Shortest Path (RSP) Problem*. Given a network $G^O(V^O, E^O)$, execution time and cost associated with each edge in E, and application deadline M, our goal is to find a path (p^*) that solves the following problem,

$$\min \sum_{e^{m,n,l}_{i,j,h} \in p^*} c(e^{m,n,l}_{i,j,h}), \quad s.t. \sum_{e^{m,n,l}_{i,j,h} \in p^*} d(e^{m,n,l}_{i,j,h}) \leq M. \tag{3.2}$$

If our application is task parallel with independent parallel tasks at certain stages, then we first construct subgraphs within the optimized workflow such that, in each subgraphs, there is only one task per stage. Algorithm 2, illustrated in Fig. 3.2(b), shows our proposed algorithm to construct subgraphs with one independent task per stage for task-parallel workflows. Algorithm 3, illustrated in Fig. 3.2(c), shows our proposed heuristic to solve the restricted shortest path problem presented above and extracts the approximate workflow. Complexity of the proposed algorithm is given as $\mathcal{O}(|V|+|E|+k(|V|^2+|V|))$. Here. the first term the second term indicates time taken for statement (2) and (3) in Algorithm 3. The third term indicates the time taken to run Dijkstra Algorithm and statement (8) of Algorithm 3, k times. The time taken for Dijkstra algorithm can be reduced from $\mathcal{O}(|V|^2)$ to $\mathcal{O}(|E| + |V| \log |V|)$ by implementation based on a min-priority queue implemented by a Fibonacci heap. Figure 3.2(c) shows the approximate workflow extracted from optimized workflow Fig. 3.2(a). Problem (3.2) can solved using a variety of commercial mixed integer linear optimization problem (MILP) solving softwares such as CPLEX [49].

3.4 Performance Evaluation

This section is geared towards quantifying the gain of approximate computing over exact computing to support various computer-vision algorithms. We first present the results from offline profiling giving statistical bounds on the speed up achieved via approximate computing along with the accuracy loss incurred. We employ an open-source software system, AllJoyn [40], to implement a distributed computing environment.

Experimental testbed: We present the various elements of our experimental testbed, which is shown in Table 2.2. Our testbed comprises of state-of-the-art, heterogeneous computing devices (tablets, smartphones, laptops, and notebooks) that vary by type of device, platform, RAM, and processing power.

Applications implemented: We motivate and study the performance of approximate computing via three well-known and broadly-applied recognition algorithms, namely, Canny edge detection [50], Scale Invariant Feature Transform (SIFT) [51], and Histogram of Gradients (HoG) [52]. Figure 3.4 shows the different tasks and their functions and input parameters that are approximated for the three aforementioned algorithms. All these exemplified applications extract different features

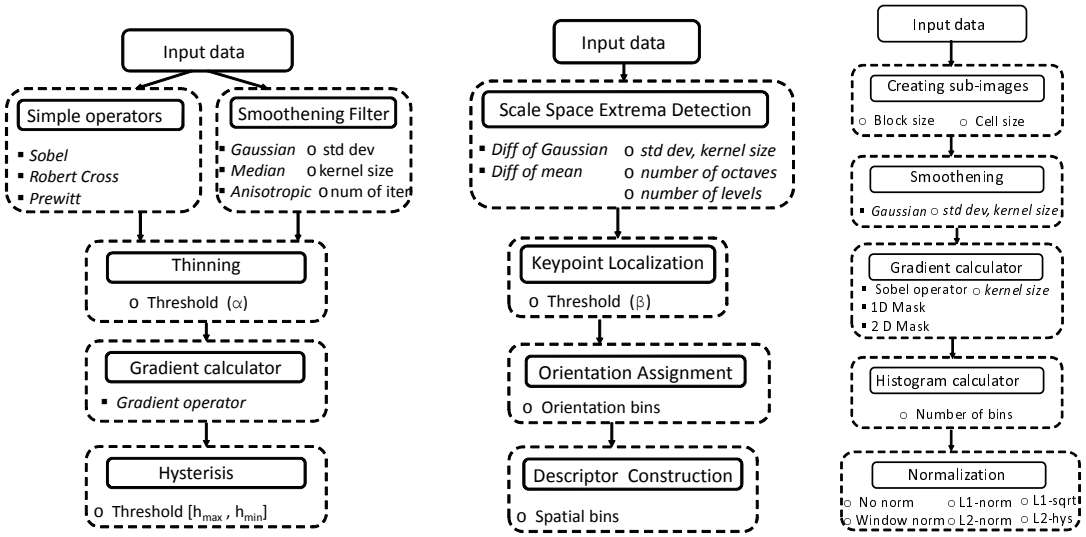


Figure 3.4: Block diagram showing different tasks and parameters for object recognition using (a) *Canny edge detection*, (b) *Scale Invariant Feature Transform (SIFT)*, and (c) *Histogram of Gradients (HoG)*. Each dashed block contains multiple implementations of approximable task types (with varying degree of complexities) and parameters.

from input data for evaluation. We implemented both the applications on computing devices in our testbed using the OpenCV library. We implemented both the applications on computing devices in our testbed using the OpenCV library. We estimate the energy consumption by an application via Power tutor app [53]. It gives the power consumption in Watt at an interval of one second for all the applications running on the mobile device.

Other applications: Our approach can be applied to a variety of other applications such as content-based image retrieval from a database, lossy audio and video decoding such as x264 media application that performs H.264 encoding on a video stream, delivering dynamic application accuracy in large datacenters depending on different factors (power, operating temperature), and other computer-vision/robotic applications (such as panorama stitching and body/object tracking).

Input data set: We execute our application by using input data from the Berkeley image segmentation and benchmark dataset [54]. For offline profiling, we used 200 grayscale images from the training data set; to determine performance benefits at run-time we used 100 images from the test data set, both available in [54]. Resolution of each image is 481×321 pixels. To study the performance gains obtained via approximation on HoG algorithm we used the INRIA Person dataset [55].

Light-weight run-time algorithms: We implement in Android the $\mathrm{Heuristic\ MP-SP}$ algorithm

Table 3.3: Number of approximate instances selected in various stages of the offline profiling for Canny edge-detection algorithm.

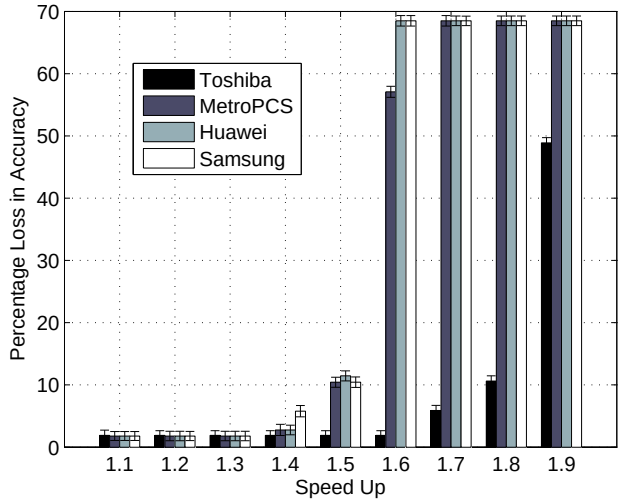
Accuracy Loss [%]	**All Work-flows**	**Rich Work-flows**	**Optimized Work-flows**	**Discarded**
20	150	55	18	130
40	150	59	18	126
65	150	82	23	98
80	150	97	24	82
100	150	97	24	82

Table 3.4: Gain achieved by approximating tasks of Canny edge detection and SIFT.

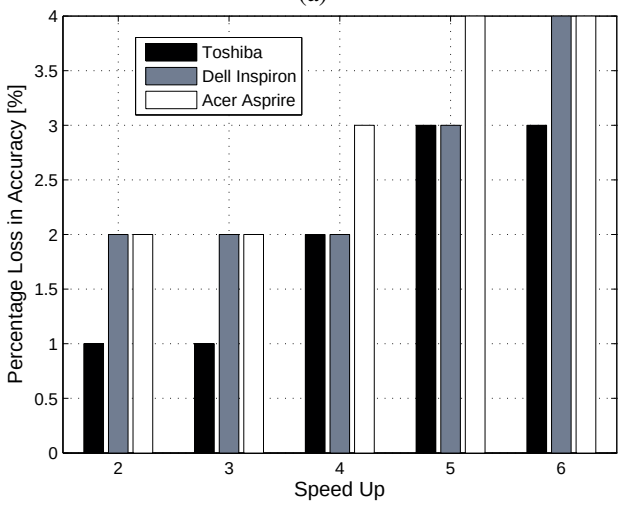
	Function/Parameter	**Range**	**Accuracy Loss [%]**	**Speed Up**
Canny	Threshold	[0,1)	2.76	1.75 ± 0.01
	Sigma	[0,1]	0.01	1.14 ± 0.01
	Kernel Size	[3:11]	7.04	1.13 ± 0.012
	Prewitt Operator	[3]	4.6	1.25 ± 0.90
	Without Smoothing	-	2.8	2.33 ± 0.45
	Without Threshold	-	2.1	1.65 ± 0.52
	Without Thinning	-	7.8	1.33 ± 0.19
SIFT	No. of Octaves	[1,10]	0.7	1.5 ± 0.022
	No. of Spatial Bins	[1,10]	0.8	2.0 ± 0.025
	No. of Orientation	[1, 2^3]	0.6	1.5 ± 0.034
	No. of Level	[1,10]	0.85	2.0 ± 0.025
HoG	No. of Cells	[2,4,8]	0.8	3.0 ± 0.075
	No. of Blocks	[2,4]	1	3.1 ± 0.01
	No. of Orientation	[3:2:11]	1.4	2.6 ± 0.076
	Kernel Size	[3:2:11]	1.3	3.0 ± 0.01

to select the approximation tasks. The algorithms to construct the approximate workflow need to be of low complexity because the gain in reduction in makespan obtained from approximate computing should not be eclipsed by the execution time of algorithms to select the approximate workflow at run time, which would result in a paradox.

Offline profiling: The framework performs offline profiling (as explained in Algorithm 1) of an application. In the profiling phase, we execute the algorithms on the computing devices in our testbed and use as input data the images from the training dataset in [54]. In Table 3.3, we observe how the number of approximate instances in rich workflow and optimized workflow will vary as the acceptable accuracy loss is increased. The number of discarded workflows decreases as the percentage of acceptable accuracy loss is increased; this is because the approximate instances that achieve speed up, although at higher accuracy loss are also included. Table 3.4 quantifies the gain



(a)



(b)

Figure 3.5: Experiments: Percentage loss in accuracy versus speed-ups achieved by applying approximate computing techniques on (a) Canny edge detection and (b) SIFT algorithm; (c) (Top) Pareto-optimal instances for Canny edge detection algorithm, (Bottom) Pareto-optimal instances for SIFT algorithm.

of applying approximation transformations to various tasks and parameters of the aforementioned computer vision applications.

Performance of approximate vs. exact computation: Figs. 3.5(a) and (b) show the results of percentage loss in accuracy obtained when different levels of speed up are achieved by applying approximation transformation to the application. In Fig. 3.5(a), we see that for Toshiba we achieve a speed up of 1.5 for 5% accuracy loss while for the other devices we get around 10% of accuracy loss. The speed up of Toshiba continues up to 1.9 but it saturates at 1.7 as for the other devices.

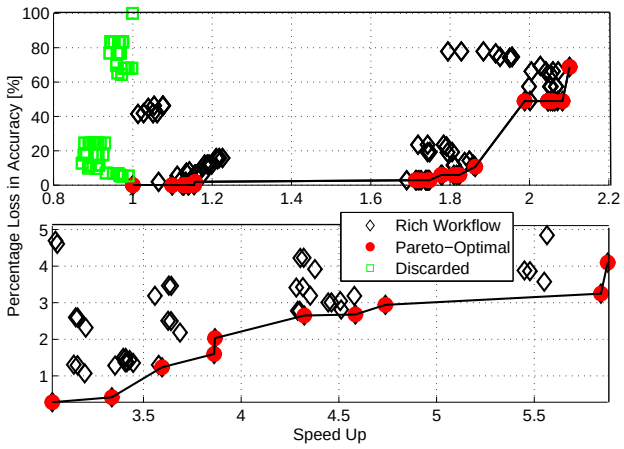


Figure 3.6: Experiments: (Top) Pareto-optimal instances for Canny edge detection algorithm, (Bottom) Pareto-optimal instances for SIFT algorithm.

Similarly, in Fig. 3.5(b) we see that the speed up is 5 times when the percentage accuracy loss is 3% for Toshiba. Similar trend is observed for the other devices. We can notice that, although the makespan has decreased with different user-specified accuracy bounds, it does not come at the cost of significant loss in accuracy. From the approximate instances, we can determine the Pareto-optimal instances to reduce the approximable task space, as shown in Fig. 3.6. The red dots in the figure indicate the *Pareto Front* for different applications. In Fig. 3.8(a), we see that for the devices we achieve energy savings of 30% for around 5% loss in accuracy.

Performance of the online algorithm: We compare the performance of our solution "Optimized-WF Probabilistic" against Optimized-WF Average technique. In the Optimized-WF Probabilistic technique the delay value of an edge, $d(e)$ in Algorithm 3 is substituted with the most probable delay and in the Optimized-WF Average technique it is substituted with the mean delay (i.e., the most frequent value obtained from different trials executed during the offline phase). We also compare the performance when Algorithm 3 is applied to a rich workflow instead of the optimized workflow. We call this approach "Rich-WF Probabilistic". To implement "Optimized-WF Probabilistic" we use our proposed algorithm $\mathrm{Heuristic\ MP-SP}$, which is required to construct an approximate workflow given the run-time application deadline and accuracy loss. We assume the value of $count$, i.e., the number of shortest paths considered in Algorithm 3 to be 3.

In Fig. 3.7(a) we see that all the techniques are able to give the output within the requested deadline. However, the difference in performance of the techniques is evident in Fig. 3.7(b) where

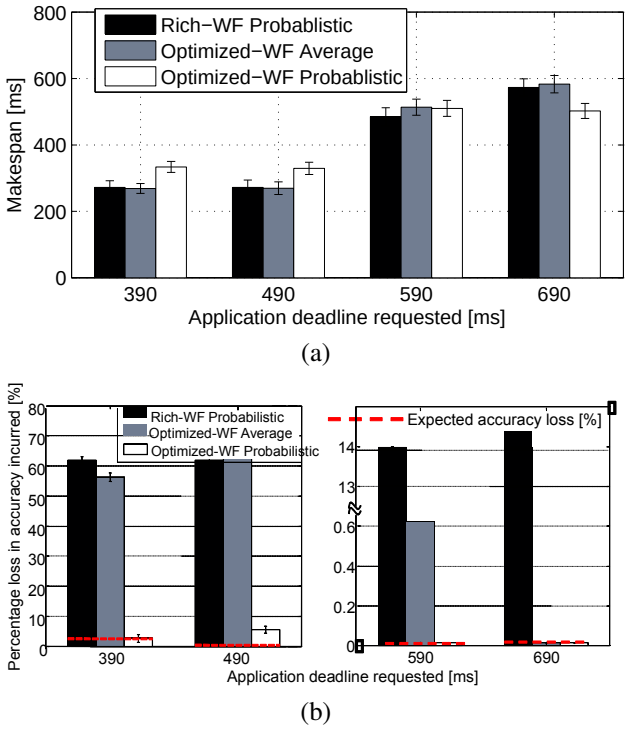
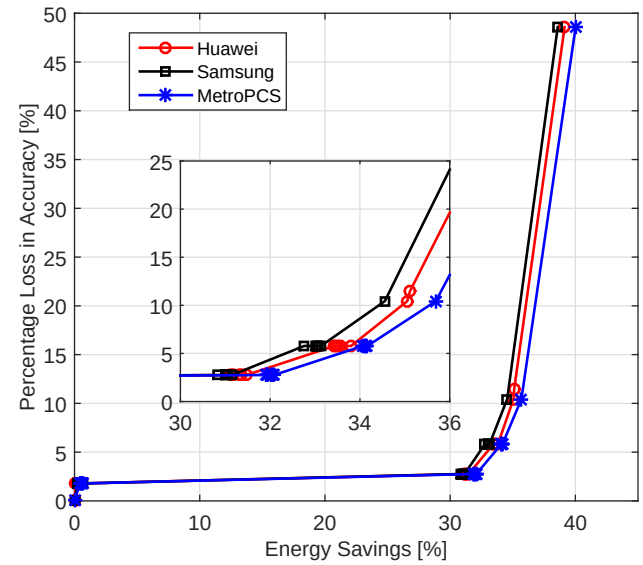


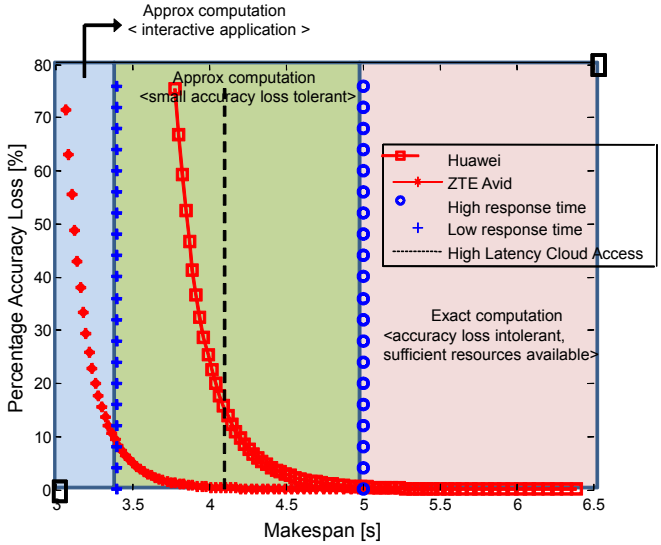
Figure 3.7: Experiments: Comparison of probabilistic and deterministic framework in terms of (a) makespan and (b) accuracy loss incurred.

we see that our Optimized-WF Probabilistic meets the percentage accuracy loss as estimated by the non-linear model (shown in dotted red line). Conversely, for the other two techniques a much higher accuracy loss is incurred in comparison to the accuracy loss as estimated by the non-linear model (Sect. 3.3). This is because Rich-WF Probabilistic considers all the approximate instances to select the approximate workflow and misses selection of Pareto-optimal instances. The instances selected by Rich-WF Probabilistic have slightly lower makespan at the cost of higher accuracy loss in comparison to Pareto optimal instances. Also, Optimized-WF Average considers average performance (time and accuracy) and not the most probable ones so although those instances meet the average time they which leads to higher accuracy loss. Now, although there is not large deviation in average time taken and the most probable time, the average accuracy for a particular parameter (accuracy averaged by executing the algorithm with the parameter over multiple data values) falls below the expected accuracy level. This effect is seen more when the application deadline is low and this deviation in accuracy reduces as the application deadline increases.

Overhead of the online algorithm: The overhead of online Algorithm 3 to select approximate workflow at run-time is of the order of 6 ms. This is much lower than the reduction in gain in



(a)



(b)

Figure 3.8: (a) Energy savings obtained by applying the approximation techniques to various tasks in the algorithms; (b) Scenarios where approximate computing can be beneficial in comparison to local exact computing and exact computing in the Cloud. The type of computation depends on several factors such as type of application (interactive or non-interactive), accuracy requirements of the application, resources available, and network latency.

makespan achieved by approximate computing, which is in the order of hundreds of milliseconds to seconds, as depicted in Fig. 3.5(a). Hence, our online approximation algorithm incurs a very small penalty, i.e., its overhead is almost negligible compared against the substantial performance benefits it brings in terms of speed up.

Applicability of approximate computing: In Fig. 3.8(b), we plot different regions of applicability of approximate computing. If the user cannot tolerate any accuracy loss, e.g., face or fingerprint recognition application to unlock a device or financial website, then the user is ready to wait for a longer duration and utilize higher resources without any sacrifice of quality. In such a situation, exact computing is applied (seen in the rightmost pink region). Conversely, interactive applications such as gaming or object recognition, where the user requires quick response and accuracy loss can be incurred without any perceivable degradation of Quality of Service (QoS) for the user, are good candidates for approximate computing (seen in the leftmost blue region). Also, the situations where the mobile device is limited by battery or does not have enough CPU cycles to give a crisp response to the user, approximate computation is beneficial. Response from cloud applications depend on the network latency; hence, in situations with high cloud latency or with intermittent network connectivity, approximate computing can be applied to give low response time (as can be seen in the middle green region).

Data-dependent approximation: In Fig. 3.9(a) we see the performance of the HoG algorithm when the image resolution is reduced by half, and we study the execution time and accuracy when the approximate parameters (different cell sizes) are applied in conjunction with the reduction of the input size. We see significant energy gains for all the cell sizes. In Fig. 3.9(b) we see the performance of HoG algorithm when the spatial resolution of input data has been reduced by different amounts. We see a reduction in the spatial resolution to be up to 80% for up to 25% loss in accuracy.

In Fig. 3.10 we see the comparison of various approximation techniques that are introduced in this chapter. In Fig. 3.10 we first show the loss in accuracy obtained and then present the speed up obtained by implementing various scenarios. For each scenario we vary the threshold parameter (α) of the Canny-edge detection application, downsample the input data by a certain value (s). The size of the resulting image after downsampling is $\frac{1}{s}$ times the original size of input image. We study the effect of task-based approximation, input data approximation, and both combined together. Each scenario reflects specific values of s and α. For Scenario-1 the value of threshold parameter for task-based approximation (when input data is not downsampled) and sampling factor for input-data based approximation (when tasks are not approximated) are 0.1 and 16, respectively; Scenario-2: 0.3 and 8; Scenario-3: 0.5 and 4 ; Scenario-4: 0.5 and 2. In Fig. 3.10(a) we see that, as expected, the performance of the combination of approximation of both task and data is worse than that of other

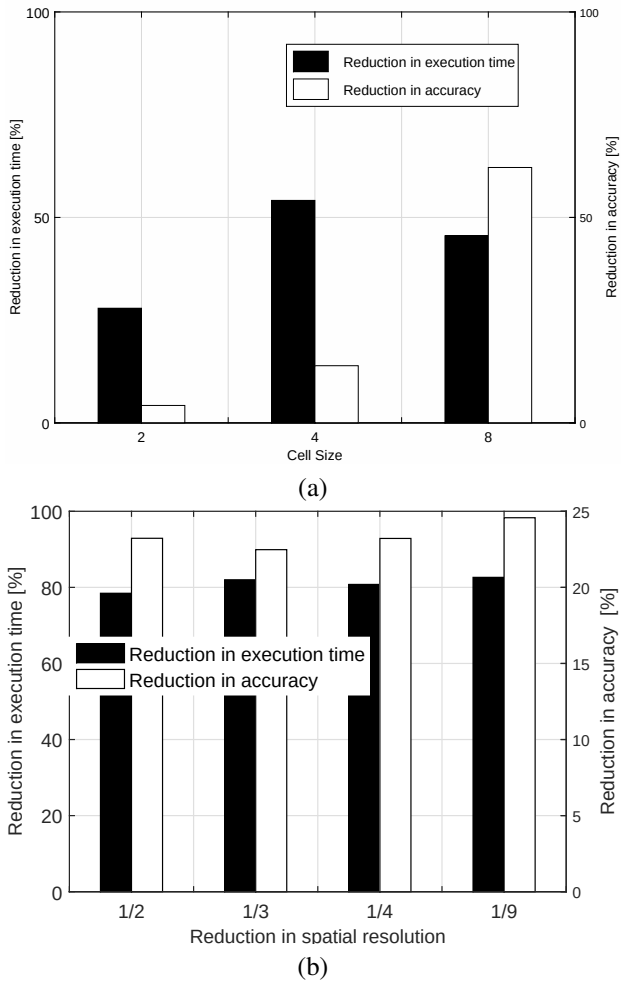
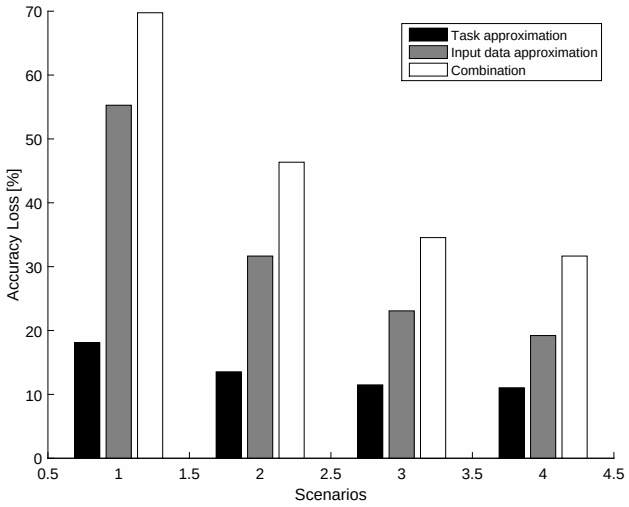
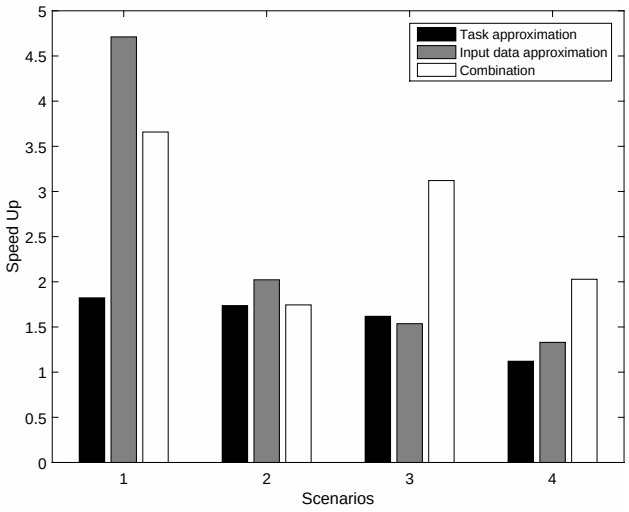


Figure 3.9: (a) Performance gains obtained by reducing the spatial resolution by half of the original input image and executing HoG algorithm on them; (b) Varying the reduction in spatial resolution of the input image to study the performance gains.

techniques; this is because when the two types of approximations are applied together, although they give higher speed up, they also cause a higher loss in accuracy. Such increase in speed up for loss in accuracy can be seen in Scenario 3 and 4 of Fig. 3.10(b). However, in Scenario 1 and 2 we see that we get a reduction in speed up for the combination of approximation of task and data as compared to when only data is approximated, although the accuracy continues to decrease as expected. This result is specific to how Canny-edge-detection algorithm works. In this simulation result we have introduced task approximation in the combination case by varying only the threshold parameter (α) for task approximation and approximating data. The algorithm calculates a value called $level$, which is a function of the pixel values of the image and of the approximated task parameter α. The value of $level$ along with the pixel values of the image impacts the number of iterations incurred in



(a)



(b)

Figure 3.10: (a) Loss in accuracy and (b) Speed-up obtained in Canny-edge detection algorithm by applying various approximation techniques, namely, task approximation, approximation of the input data and combination of both techniques.

the next task “thinning" of the Canny-edge-detection algorithm, as seen in Fig. 3.4(a). There is a possibility in the combination case (data and task approximation) that based on the value of $level$ and on the pixel values, the number of iterations is now higher (as in Scenario A) or comparable (as in Scenario B). The gain achieved via approximation of input data is reduced due to an increase in the number of iterations in the later stages of the algorithm. This is noticeable when the image pixel values do not vary significantly. Such increase in the number of iterations leads to an increase in execution time and a loss in speed up. In Fig. 3.11(a-b) we see similar results for SIFT algorithm. For SIFT algorithm we vary the value of sampling factor (input data approximation) and the number

of levels (task approximation) to study the performance gains. In Scenario-1 the value of number of levels parameter and sampling factor is 2 and 16, Scenario-2: 4 and 8; Scenario-3: 6 and 4; Scenario-4: 8 and 2.

Benefits of approximation to MDC: We study how much benefit can be achieved in MDC by applying approximation. We consider mean and standard deviation of availability duration of devices, inter-arrival times, task sizes, buffer size (number of task allocated to each device in Round Robin), battery capacity, and processing power. We model the mobility patterns of devices in the proximity as a normal distribution with mean availability duration of devices varying with $\mu = \{5, 100, 200\}$ s and σ=5 s. We compare the performance of exact versus approximate execution of an application in a MDC. In Fig. 3.12, we see that by applying approximation in a MDC we are able to achieve a much higher Frames Per Second (FPS) rate. This is beneficial in case of interactive applications that have to meet a time-critical application deadline.

Use cases of approximate computing in a mobile device cloud: We present example scenarios where approximate computing can be applied to enable applications on devices limited by CPU cycles and battery capacity.

Surveillance: Consider a team of mobile robots stationed in a restricted area to detect intruders. Each of these robots have a different spatial view of the area, hence, they collaborate to detect intruders. In existing intruder detection systems compute-intensive exact algorithms are implemented. In situations when the battery capacity of the robots is critically low running exact algorithms is not constructive as the robots can exhaust their battery without generating any meaningful results. These robots can include energy-aware approximate face detection algorithm that are able to tune their tasks and parameters based on the available battery capacity of the device. As a result, the devices can generate less accurate yet meaningful results that can help to alarm the authorities in case of any potential danger.

Disaster damage assessment: During any disaster such as hurricane or earthquake, the goal of the relief team is to assess the damage and provide prompt relief to the affected areas. In the recent past rescue teams have collected Light Detection and Ranging (LiDAR) data by sensors mounted on top of rescue vans [56]. These rescue vans are equipped with few computational device like laptops and workstations which have limited computational and battery capabilities. Currently the data collected by these vans is shipped to the datacenters where compute-intensive computer vision

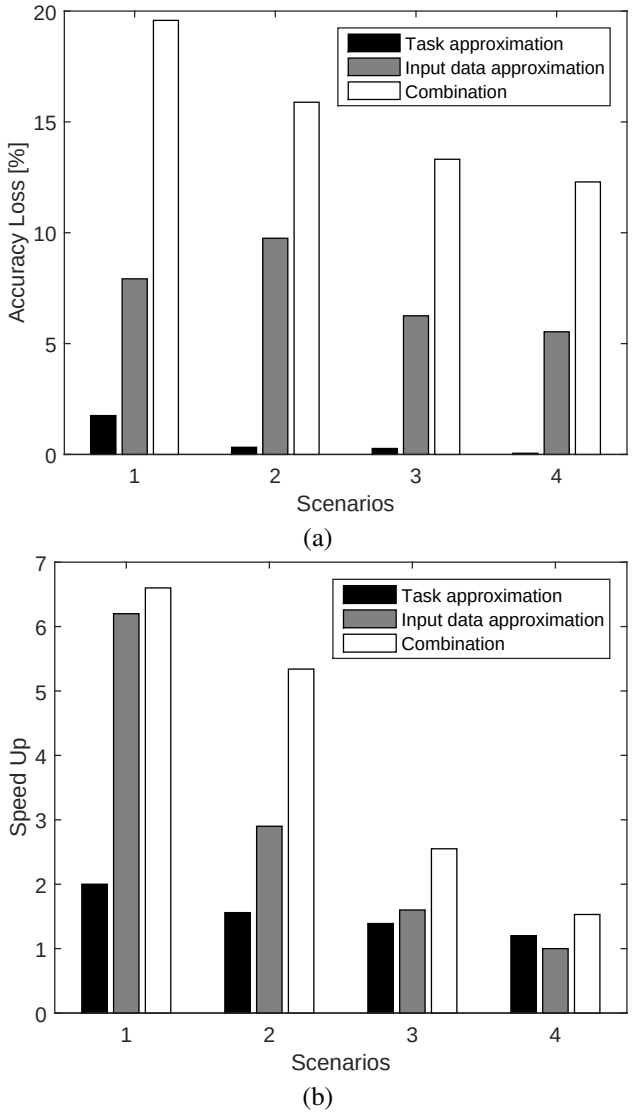


Figure 3.11: (a) Loss in accuracy and (b) Speed-up obtained in SIFT algorithm by applying various approximation techniques, namely, task approximation, approximation of the input data and combination of both techniques.

algorithms are run on the data to determine the structural damage in the affected region. However, because of the large scale of devastation caused by these natural calamities it takes a significant time to run the algorithms to assess the structural damage caused which leads to delay in rescue process. To reduce the execution time of these algorithms the compute nodes in the van can collaborate and run approximate versions of these algorithm to bring down the reconstruction time significantly but at the same time provide high-level information to the authorities. This information can include determining areas with maximum damage which can help in channeling the resources.

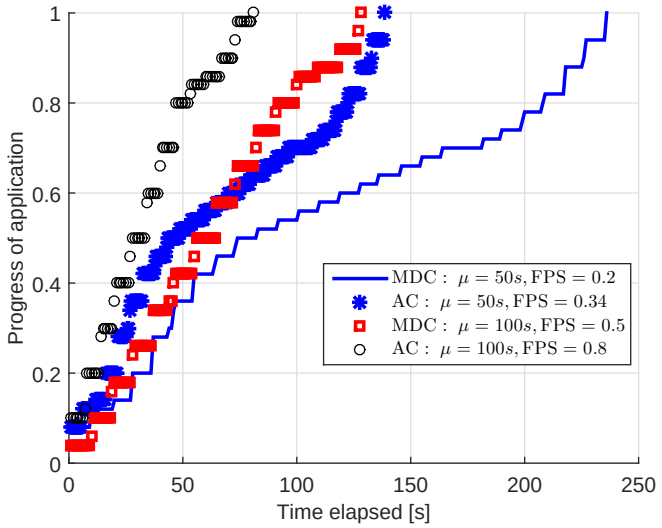


Figure 3.12: Experiments: Comparison of performance of approximate computing vs. exact computing in a mobile device cloud in the presence device mobility

3.5 Summary

We considered the new paradigm of approximate computing to enable accuracy- and deadline-aware applications in a resource-constrained mobile device cloud. We introduced an approximate computing framework that determines the approximable tasks in an application via a powerful workflow representation scheme. We validated the effectiveness of our approach through extensive simulations and testbed experiments taking as motivating example three key algorithms for interactive perceptive object recognition, and observed that—under limited available resources—their approximate implementations perform better than their exact counterpart. While so far we have studied isolated applications running on a mobile device to which approximation can be applied, our approach can be extended to multiple applications running concurrently on the mobile device. For example, approximation can be applied to simultaneous background sensing and foreground applications. Furthermore, while so far we have focused on approximation via accuracy-energy tradeoff, we can also consider other resources available on the mobile device such as network bandwidth, memory, and CPU cycles.

Chapter 4

Light-weight Object Detection and Decision Making via Approximate Computing in Resource-constrained Mobile Robots

In this chapter we use object detection application implemented via popular computer vision algorithms as a case study to solve the problem of limited resource availability in mobile robots. There has been a growing interest in supplementing geometric maps used for indoor navigation with semantic information (e.g., object detections) using computer vision algorithms. Unfortunately, there is a disconnect between the relatively heavy computational requirements of these computer vision solutions, and the limited computation capacity available on mobile autonomous platforms. In this chapter, we propose to bridge this gap with a novel MDP framework that adapts the parameters of the vision algorithms to the incoming video data rather than fixing them a priori. This approach leverages the offline phase explained in the previous chapter to identify the parameter space. As a concrete example, we test our framework on a object detection and tracking task, showing significant benefits in terms of energy consumption without considerable loss in accuracy, using a combination of publicly available and novel datasets.

4.1 Introduction

Autonomous navigation in indoor environments (e.g., with robotic drones) can be achieved through the use of mapping, planning, and control solutions that rely purely on *geometric* information (such as occupancy grids or point clouds). There exist an opportunity, however, to improve these techniques for robots that need to work in close proximity to humans, e.g., in controlled environments such as warehouses or in search and rescue missions. There has been a growing interest in supplementing geometric maps with *semantic* information, by using vision information to reach some understanding of the environment [57]. As a motivating application, we posit that the autonomous robots could use the same existing infrastructure developed for humans, such as traffic signs, to

navigate. In other words, signs that have intuitive meaning for humans can be used to influence the behavior of robots; for instance, if a robot detects a “Stop" sign, it might pause its motion until some environment-specific condition is met.

To realize these functions, it is necessary to run object detection algorithms on the incoming data (video stream) in *near real-time*. However, running computationally-intensive vision algorithms using on-board processors can cause delay in detection and significant battery consumption leading to reduction in the mission time. In robotics, offloading operations to clouds has been proposed as a way to handle the constraints on the robots, targeting collaborative applications such as localization [58, 59] and object recognition [60, 61]. However, such approach faces challenges such as a variable network connectivity or a high latency, leading to performance bottlenecks, especially for real-time/interactive applications that require low response times.

In recent years there has been tremendous improvement in the performance of detection algorithms; however, these algorithms have been mostly designed for accuracy, and are too slow for resource-constrained autonomous systems (since they typically rely on deep neural architectures or model ensembles). We argue that the selection of the computer vision algorithm, along with its parameters, should depend on the input data, resources available with the device (battery capacity, and CPU frequency), and the accuracy of the results that is acceptable to the user or application. To this end, we propose a decision framework that selects the parameters of computationally intensive algorithm that are “good-enough", i.e., parameters that give acceptable accuracy in each frame of the video with significant savings in time and energy. A good-enough parameter for a particular frame is selected from a parameter space that is created using the concept of transformation of workflows explained in Chapter 3.

Our framework exploits the temporal correlation between the continuous stream of frames obtained from the camera sensors, learns the good-enough parameters from the first few frames, and applies these parameters to the subsequent frames. We cast the problem of selecting the algorithm and input parameters for a video as a MDP. MDPs provides a mathematical framework for modeling decision making in a stochastic environment. We design a reward function for the MDP that achieves acceptable accuracy for the application with minimum cost (in terms of time). While in this chapter we focus on a specific case study, the selection of good-enough parameters via MDP is a generic technique that may be applied to a wide range of computationally-intensive algorithms.

Our proposed framework works well in variety of challenging conditions, such as those that exists in object detection application (e.g., background clutter or poor illumination).

The problem of object detection has been considered exhaustively in the computer-vision literature. However, our focus is on reducing the computational requirement of state-of-the art detector to better fit real-time applications on resource-constrained platforms. Likewise, MDPs have also been used in the computer vision community for the problem of object detection [62, 63], multi-object tracking [64], human path predictions [65], and videogame play [66]. However, they have not been considered with the goal of achieving good-enough performance for video data on resource-constrained devices.

We make these contributions in this chapter:

- We use object detection and tracking as a case study to motivate that traditional "one-size-fits-all" approaches are energy and time inefficient for real-time applications, while we show via simulations and experiments that our solution shows improved performance.

- We identify, via offline experiments, the algorithm and parameters of an object detection application that can be approximated, showing 42% increases in speed while maintaining 80% accuracy.

- We validate the proposed solution on publicly available datasets, and on a new dataset of videos collected with a Bebop drone; we achieve up to $30\,\mathrm{fps}$ for 480×270 video frames with an accuracy drop of less than 2% with respect to a fixed choice of object detection parameters.

In Sect. 4.2 we give different works in the area of object detection. In Sect. 4.3 we discuss the application of object detection and tracking and explain two popular computer vision algorithms to implement them. In Sect. 4.4 we introduce the entities of our decision framework, cast the problem of selecting the parameter-algorithm combination as an MDP, and present its application to a popular object detection algorithm. In Sect. 4.5, we present the application of our proposed decision framework to object detection and tracking application. In Sect. 4.6, we provide details of our experimental setup and study the performance of our solution under various conditions. Finally, in Sect. 4.7, we conclude our work.

4.2 Related Work

We review here how the issue of limited resource availability has been handled while implementing object detection in mobile devices and also discuss the use of MDPs in computer vision applications.

Mobile computation offloading: Many applications using object detection in videos for example mobile gaming, automatic landmark recognition [67], and face detection [68] rely on offloading costly (compute and energy-intensive) tasks to cloud resources in a transparent manner. However, these approaches are not suitable for enabling data-intensive applications in real time due to prohibitive communication cost and response times, significant energy footprint, and the curse of extreme centralization. To circumvent these challenges, the concept of mobile device cloud has been introduced [32]. This paradigm was based on splitting the applications into tasks, and then executing the tasks in parallel on nearby mobile devices. However, the local resource pool may suffer from scarcity of capable devices, or from uncertain network connectivity and device availability. Also, it may not be possible to split tasks in an application to independent sub-tasks (so to take advantage of their parallel execution) or the computation gain obtained by parallel execution of sub-tasks may not be able to surpass the communication costs. Our work, on the other hand, introduces a layer of approximation over tradition mobile computing to overcome the aforementioned challenges.

Object detection and tracking: The problem of tracking has been considered exhaustively in the computer-vision literature [69] but the focus has been on developing on improving the accuracy of trackers to handle various challenges arising due to variations in illumination and appearance changes of the object due to variations in object viewpoints and rotation of object between frames. However, our focus is on reducing the computational requirement of state-of-the art trackers to better fit real-time applications in resource-constrained platforms. The use of MDPs in computer vision is not new, and they have been applied to the problem of object detection [62, 63], multi-object tracking [64], human path predictions [65], and videogame play [66]. However, they have not been considered in the context of approximate computing with the goal of achieving good-enough performance on mobile devices. In [63] the authors present an anytime object recognition algorithm. Here, the focus is running object recognition application on static images whereas we have focused on videos.

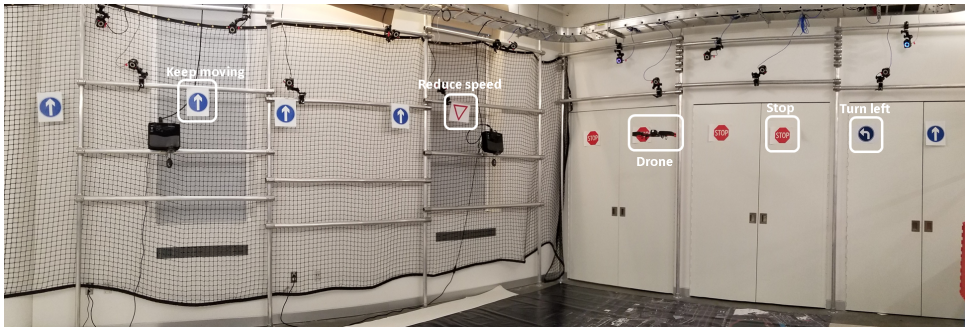


Figure 4.1: One of the areas used in the Robotics Lab, Boston University for collecting our dataset. The arena simulates a warehouse where human-interpretable signs (road signs) were used as cues to help the drone navigate the warehouse. Each road sign is associated with a specific control command.

4.3 Case Study: Drone Navigation via Object Detection and Tracking

To validate our novel approach, in this chapter we focus on the problem of road-sign detection and tracking in videos. This particular problem is encountered not only in mobile computing, but also in robotics. For instance, we envision that many warehouses in the future will use teams of autonomous mobile robots that work in close proximity to humans to improve productivity. These autonomous mobile robots could use a camera and the existing infrastructure developed for humans to navigate, locate objects and prevent collisions, without the need of ad-hoc, machine-readable-only signatures (e.g., QR codes). Recognition of objects (such as signs) that have common meaning between robots and humans can also be used by the human operators to indirectly but intuitively influence the behavior of the robots. For instance, an operator could put up a “Do not enter" sign to avoid that human and robot alike cross an area that would be normally open, but that is currently unavailable due to repairs or other similar operations.

Navigation via detecting such signs by using computer vision techniques is a computationally intensive task that may delay the time taken to detect road signs and consume significant energy. Our goal is to reduce to execution time of running these computer vision techniques and give results in a timely manner. Lowering the execution time will also reduce the energy consumed by the drone’s battery and hence effectively extend the possible mission time. We give a simple illustration of the aforementioned setup using experiment spaces located in the Robotics Lab at Boston University, as shown in Fig. 4.1. We now give details of computer vision algorithms that we will use to realize the above mentioned scenario.

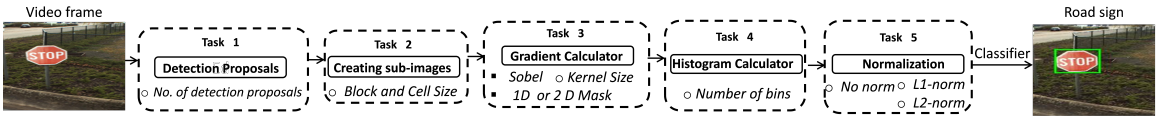


Figure 4.2: Object detection workflow using *Histogram of Gradients (HoG)* and *Edgeboxes* [2] algorithm. Each dashed block contains multiple implementations of approximable task types (with varying degree of complexities) and parameters.

4.3.1 Technical Algorithms Considered

In this study we employ in conjunction two well known algorithm in the computer vision community, namely, HoG and Edgeboxes [2] for object detection and tracking on video frames. Figure 4.2 shows the different tasks and their functions and input parameters that are approximated for the object detection and tracking workflow. We now briefly explain the different components of this computer vision workflow.

Edgeboxes: In an object detection workflow the first task (shown in first task in Fig. 4.2) is to identify the region in the image where it is likely to find an object such as person, car, animal as opposed to a background such as grass, cloud etc. These regions, called *detection proposals* (or bounding boxes) in the computer vision literature, are represented as rectangular boxes on the images. The object proposals are generated in a manner that is agnostic to the type of object being detected, and they typically reduce the number of false positives detected in the video. They also reduce the computational load, because instead of searching the whole image for the object we only look for road-signs in the detection proposals. In Fig. 4.3 we motivate the need for adaptive control of proposals generated by the algorithm. We can see that based on the background clutter, illumination, and camera angle, different number of bounding boxes are required for successful recognition. It is represented in Task 1 in Fig. 4.2.

Histogram of Gradients: HoG is a popular image feature used for object detection. The different tasks in HoG feature extraction from an image are given in Tasks 2-5 in Fig. 4.2. It decomposes the image into square cells of a given size, computes a histogram of oriented gradient in each cell, and then normalizes the histogram in each cell.

Classifier: The features extracted using the HoG algorithm are given to the classifier and a score (c) is given as output by the classifier. The score quantifies the quality of prediction at a location in the image (in our case whether there is an object or not). Higher score values are typically correlated

(a) (b) (c) (d) (e) (f)

Figure 4.3: Motivation figures to illustrate that parameters of the object detection algorithms can be adapted to input data to achieve savings in time and energy. Figures (a,b,c) show variation in the amount of background clutter; (d and e) show variation in illumination, and (f) shows variation in camera viewpoint. These variation lead to differences in the minimum number of bounding boxes parameter (detection proposals) required for object detection. Lower detection proposals translate to lower execution time.

with better predictions. We calculate the threshold T_c over which the detection is successful from the training data. Since during run-time we do not have the ground truth indicating the presence and location of object in the image, we rely on the classifier score to assess the accuracy of our prediction. In our case study we are focusing on object detection of certain road signs. HoG features for each road sign are constructed from a set of training examples. Also HoG features from each road sign are extracted from images that do not contain any road sign also called negative images. Features from both positive and negative images are given to the classifier to create a model for the road sign.

4.3.2 Applying Approximation to the Object Detection Workflow

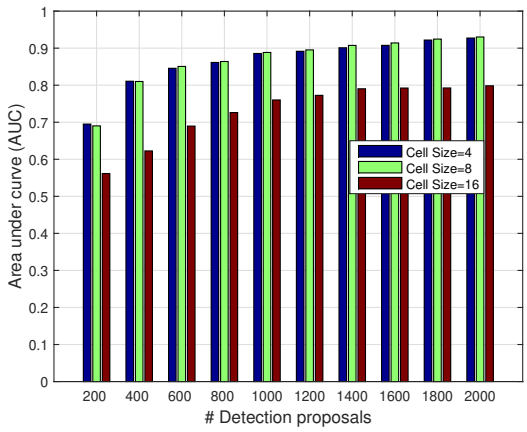
We envision that instead of using fixed algorithm and input parameters for object detection we let the system choose from a wide range of algorithm and parameters such that these actions achieve the classifier score expected by the application at the lowest computational cost. As already explained, we use the classifier score as a proxy for accuracy of detection, since this information is not available at test time. We now explain how the paradigm of approximate computing is used to create the parameter and algorithm space for the object detection workflow.

Each task represented in Fig. 4.2 of the object detection workflow is represented by an (i) executable code/function and (ii) a set of input parameters. The executable code/function represents a functionality that cannot be split further e.g., in Task 3 gradient calculator, the function could be using a Sobel filter or Gaussian filter. The object detection worklfow has the parameter tuple: {`BB`, HoG_i}, where `BB` is the number of bounding boxes or detection proposals used for an image and HoG_i indicates a set of parameters of different tasks in the HoG algorithm. These parameters are:

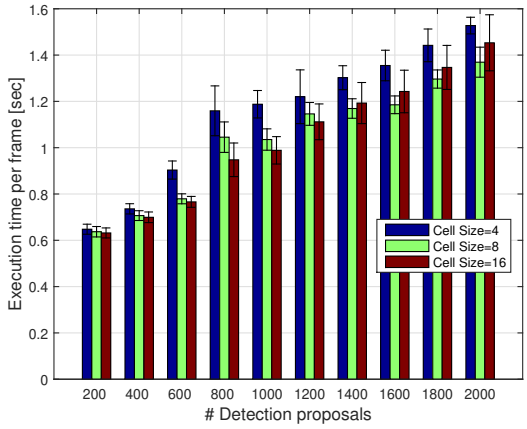
$HoG_i = \{$`bsize`, `csize`, `fsize`, `nhist`$\}$, block size (Task 2), cell size (Task 2), filter size (Task 3), and number of orientation (Task 4), respectively. Interested readers can read the details on how to construct the parameter space in Chapter 3. The approximation can be applied to the tasks in the object detection workflow shown by applying the following transformations in Fig. 4.2.

Substitution transformation: as explained earlier in Chapter 3, substitution transformation substitutes the task(s) of an application in exact computation (highest achievable accuracy) with those with lower degree of complexity, i.e., it requires substitution of a computation task by a simpler task, with potential loss in accuracy. This could be achieved at the function level, for example by substituting the calculation of an image gradient (Task 3 in the object detection workflow) with a simpler filter; or it could be achieved at the parameter level, for example by varying the number of detection proposals or the cell size in the object detection workflow (Task 1 in the object detection workflow). These transformations requires domain knowledge. In [6] the number of detection proposals was fixed at 1000 detection proposals independently from the image. However, as mentioned earlier we argue that this should not be fixed, and Fig. 4.3 visually shows an example to make the case that the parameters of the object detection algorithm should not be fixed, but should be adapted to the scene. For instance, in Fig. 4.3(a,b,c) the minimum number of bounding box required per image varies as the background clutter (quantified using [70]) varies from 1 for image 4.3(a) to 100 for image 4.3(b). Similarly, due to variation in illumination, image 4.3(d) requires less detection proposals than 4.3(e) and for image 4.3(f) the detection of the road sign (on the bottom-right corner) is not possible because of the camera's viewpoint.

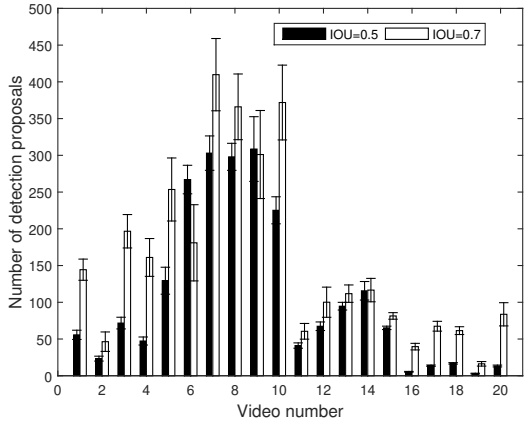
Out of the HoG parameters ($\{$`bsize`, `csize`, `fsize`, `nhist`$\}$), `csize` variation in cell size in the object detection workflow, shows the highest gains in terms of execution time for minor loss in accuracy. We pick cell size 4 (HoG_1), 8 (HoG_2), 16 (HoG_3) in our analysis. In Fig. 4.4 (a,b) we can see the variation in accuracy with the number of detection proposals (Task 1 of the object detection workflow) for a fixed cell size (Task 2 of the object detection workflow). We define accuracy here in terms of Area Under Curve (AUC), which is the measure of overlap between ground-truth and object location generated by the HoG detector. We see a 10% loss in accuracy for a 42% decrease in execution time when the number of detection proposal is decreased from 1200 to 200 and AUC decreases from 0.9 to 0.81. In Fig. 4.4 (c) we observe that in different videos of the dataset [3] the "number of detection proposal (BBs)" parameter in the object detection workflow also varies.



(a)



(b)



(c)

Figure 4.4: Representing the tradeoff between (a) accuracy in terms of area under the curve (AUC) and (b) execution time per frame for different values of parameters, namely, number of detection proposals and cell size, of the object detection workflow in Fig. 4.2; (c) Figure motivates that for computationally-intensive object detection and tracking algorithms “one size fits all" approach is not optimal and the parameters required for the object detection and tracking algorithm varies with input data. We observe that different videos of the dataset [3] the “number of detection proposal (BBs)" parameter in the object detection workflow (Fig. 4.2) also varies.

Discarding transformation: as explained earlier in Chapter 3, discarding transformation involves not executing certain tasks in the application to reduce the execution time, incurring a reduced accuracy. For example, we can discard the normalization step in the object detection workflow. From the experiments we did not observe any significant gain in time by removing the normalization tasks (Task 5). However, removal of any other task caused significant loss in accuracy, and hence these have not been considered. We will now leverage these approximation techniques to explain our proposed algorithm.

4.4 Proposed Algorithm

In this section we give the details of our proposed MDP-based decision framework for selecting good-enough parameters for an object detection and tracking. In MDPs, an agent takes *actions* based on its *state*, and receives a corresponding *reward*. The goal of the MDP is to maximize the sum of rewards. In our case, the rewards are designed such that the system achieves acceptable accuracy with minimum cost (in terms of execution time). To this end, a *policy* is learned for an MDP, which gives the optimal action that should be taken from each state of the MDP to maximize its sum of rewards achieved from that state. In our scenario the agent is represented by the vision-based navigation system in the drone, actions corresponds to choices of the parameter of the object detection workflow that will be used on the received frame, the current state is represented by whether the object is likely to have been successfully detected in the current frame, and the reward is given by a combination of execution time and confidence levels returned by the object detector and tracker. We first detail our decision framework, and then show its application to a popular object detection algorithm.

Decision framework: As the video is captured by the drone's camera, the frames are given to the MDP, which estimates the good-enough parameters of an object detection algorithm from the first few frames, and then reuses them in the subsequent frames. The system continues to use the good-enough parameters as long as the classifier score is above a threshold, otherwise the MDP is triggered again to re-estimate the good-enough parameters. If, for a given frame, the MDP cannot find parameters that lead to a sufficient score after a set number of attempts, we consider this as a sign of an unlikely detection, and drop the frame. Our framework is composed of these entities:

States: Each state $s \in \mathcal{S}$ of the MDP is represented by a tuple given as $\mathtt{s} \doteq \{\mathtt{c}, \mathtt{params}\}$, where `c` is the classifier score and `params` includes all tunable parameters of an object detection algorithm. We first identify all combinations of tunable parameters in `params`, thus generating different instances. Next, only those instances of `params` that are on the Pareto-optimal front with respect to the accuracy/execution time tradeoff are considered in $\mathcal{S}$. See Fig. 4.5 for an example. This is similar to how the Pareto optimal parameters were selected in Chapter 3 to create the optimized workflow.

Actions: The action space of the MDP consists of choosing good-enough parameters (`params`) for object detection, and applying them to the incoming frame.

Reward: The real-valued reward function $\mathcal{R}$: $\mathcal{R}(s, s', a)$, is the immediate reward received after transition to state s' from state s by taking action a. The exact definition of this function is explained later in this section.

State transition function: $\mathcal{P}$: $\mathcal{P}(s, s', a) = \Pr(s_{t+1} = s' \mid s_t = s,\, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$.

Reward function: The reward is the way of communicating to the MDP the goal we want to achieve, that is, the agent should obtain a high positive reward when it achieves the desired goal, and a lower reward otherwise. Our goal is to achieve acceptable accuracy with minimum execution time. We use the classifier score as a proxy for detection accuracy; while the latter, naturally, cannot be computed at runtime due to the absence of ground truth annotations, the classifier score correlates well with the quality of the prediction (e.g., see Fig. 4.6), with higher scores indicating better predictions.

In order to define our reward function, we first delineate three categories for the classifier score c that we obtain after running the object detection algorithm: optimal-accuracy score (c_{opt}), good-enough score (c_{ge}), and unacceptable score (c_{ua}) with $c_{opt} > c_{ge} > c_{ua}$, and with associated weights w_{opt}, w_{ge}, w_{ua}, respectively. The category of weights are obtained from offline training of an algorithm, as explained later for a specific object detection algorithm.

When an action is executed (i.e., the object detection algorithm is run with parameters corresponding to a state s'), the MDP receives a classifier score, and thus a weight $w \in \{w_{opt}, w_{ge}, w_{ua}\}$. Our proposed reward function is:

$$R(s', w) = \frac{w}{r(s')^{\alpha}}, \tag{4.1}$$

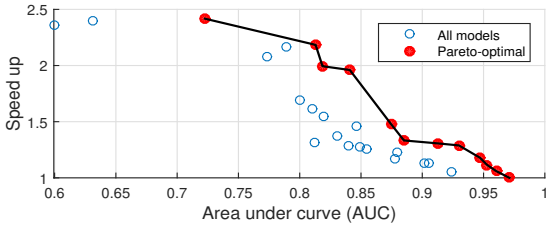


Figure 4.5: Representing the tradeoff between accuracy in terms of area under the curve (AUC) and speed up in execution time per frame for different values of parameter tuple of the object detection pipeline, namely, number of detection proposals and cell size in object detection pipeline

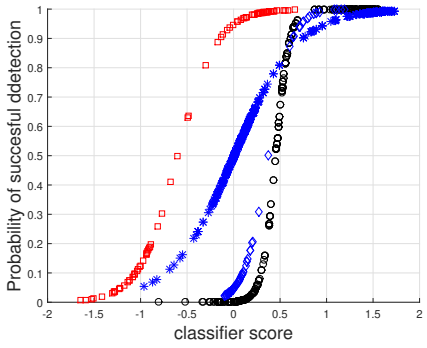


Figure 4.6: Illustration of variation of probability of successful detection with classifier score for various road sign in KUL traffic sign dataset [4]. The SVM scores have been converted to probabilities using Platt Scaling [5] and we determine the values of c_{opt}, c_{ge}, and c_{ua} for the reward function.

where $r(s') = \frac{T_{s'}}{T_{min}}$, gives the factor of increase in execution time $T_{s'}$ when choosing the tuple for state s' with respect to the parameter tuple which takes minimum time T_{min}, and α is a tuning parameter. In order to encourage the MDP to select parameters with acceptable (good-enough or better) accuracy and minimum execution cost we let $w_{ge} \gg w_{opt}$.

Policy: In the context of MDP a “policy" is defined as a function π that specifies the action $\pi(s)$ that the agent will choose when in state s. The goal of an MDP is to choose an optimal policy π^*, i.e., the optimal action from state s that will maximize some cumulative function of the random rewards, $\Sigma_{t=0}^{\infty}\gamma^t R(s_t, s_{t+1}, \pi(s_t))$, typically the expected discounted sum over a potentially infinite horizon, where γ is the discount factor and satisfies $0 \leq \gamma < 1$. To estimate the optimal policy we estimate a value function that expresses how good it is for the agent to be in a given state. The notion of “how good" here is defined in terms of future rewards that can be expected starting from

state s and following π thereafter. For MDP we define $v_\pi(s)$ formally as

$$v_\pi(s) := \sum_{s'} P_{\pi(s)}(s, s')\big[R_{\pi(s)}(s, s') + \gamma v(s')\big]. \tag{4.2}$$

Value functions define a partial ordering over policies. There is always at least one policy that is better than or equal to all other policies. This is an optimal policy, defined by the Bellman optimality equation. The optimal policy or action from a state is the one that gives the maximizes the value or function or expected sum of rewards for that state, i.e., optimal policy $v_*(s) := \max_\pi v_\pi(s)$. Also, higher the value of γ higher is the weight given to the future reward. When $\gamma = 0$, the agent is myopic and is only interested in maximizing immediate rewards. When $\gamma = 1$, the rewards are undiscounted and the agent is interested in maximizing rewards but is not concerned when it achieves the rewards.

4.5 Application to Object Detection

We now apply our framework to the object detection workflow comprised of HoG and Edgeboxes algorithms which were explained earlier. The parameter tuple for the object detection worklfow is given by `params` $\doteq$ {`BB`, `bsize`, `csize`, `nhist`}, that is: number of bounding boxes or detection proposals, block size, cell size, and number of bins in the histogram, where detection proposals is a parameter of Edgeboxes, and the others belong to HoG. Fig 4.6(a) shows the tradeoff between accuracy in terms of AUC and speed up in execution time. Each marker in the figure is one instance of `params`, with red markers highlighting the Pareto-optimal instances. We see that we can achieve a reduction in execution time by 50% for a 10% loss in accuracy.

Thresholds: We use a Support Vector Machine (SVM) classifier [71] for which the score c is calculated as $c = w^T x + b$, where w is a weight vector, b is the bias term, and x is the HoG feature extracted from a detection proposal. Features from images of both positive (containing road sign) and negative (with no road sign) training sets are used to train the SVM classifier (i.e., find the optimal values for w and b). Cross-validation is used to determine the threshold score T_c for each road sign over which the probability of correct detection is high. In Fig. 4.6(b) we give an example of how categories for a classifier score are selected. Here, the SVM scores have been converted to probabilities using Platt Scaling [5]. Using this graph, we set the threshold classifier scores for the

three categories as $c_{opt} = 0.9$, $c_{ge} = 0$, and $c_{ua} = -0.5$. We set the value of classifier threshold T_c to be equal to c_{ge}. The parameter α is set in the interval $[0, 1]$.

Run-time decision framework: When a new frame enters the navigation system, the MDP framework identifies the good-enough parameters of the object detection algorithm that give classifier score above the threshold T_c with minimum execution time. If after running the object detection application on the frame with good-enough parameters we get score greater than T_c, we initialize a Kanade Lucas Tomasi (KLT) object tracker [72] with the object location in that frame. The good-enough parameters given by the MDP are used in the subsequent frames with the combination of the results from the KLT tracker on those frames. In the previous chapter our goal was to identify at run-time the parameters that should be selected to achieve the speed up that is expected by the user. In this section we focus on selecting the parameters based on the input data and the MDP helps in identifying which parameters will give us acceptable accuracy.

Object tracker: The KLT tracker produces a bounding box B_t for the current frame propagated from the previous frame, with the results given by the detection algorithm, which gives another bounding box B_d from the current frame. We calculate the overlap between the bounding boxes from the good-enough object detector B_d and the KLT tracker B_t using the *Intersection over Union (IoU)* metric:

$$\theta(B_d, B_t) = \frac{B_d \cap B_t}{B_d \cup B_t}. \tag{4.3}$$

The MDP is re-triggered when either the classifier score is below the threshold T_c, or when the overlap of the bounding boxes $\theta(B_d, B_t)$ is lower than a pre-defined threshold (θ_{th}).

4.6 Performance Evaluation

This section focuses on how our proposed MDP-based decision framework can reduce the time required for running object detection algorithms on resource-constrained robots. We evaluate our solution on publicly available datasets and data collected by our team via a drone platform. Figure 4.7 shows our run-time experimental setup.

Experimental setup: As mentioned in the introduction, we envision aiding the navigation of robots in a warehouse by using the same visual cues as humans to execute an associated control task

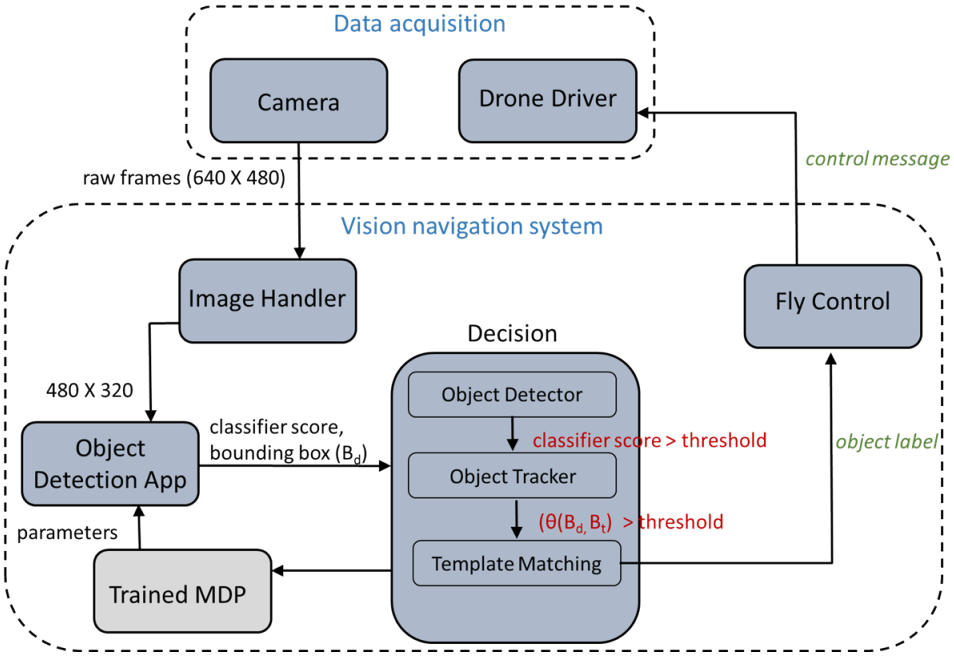


Figure 4.7: The block diagram showing different elements of our run-time decision framework. The Drone's Robot Operating System (ROS) (acts as a master) communicates with the Laptop's ROS (acts as a slave). The decision block based on the label ('Stop', 'Turn Left') of the detected object, sends control commands to the drone via ROS.

(e.g., reducing speed, changing direction). Currently, there is no public dataset for this specific application; instead, as a proof of concept, we use the Katholieke Universiteit Leuven (KUL) dataset, a traffic road sign dataset [4]. Videos taken from a moving camera containing traffic signs while traveling along a road is very similar to those collected by a drone in a warehouse where road signs can be used to guide its motion.

In addition, to test the performance of our system in more realistic conditions, we created the *BU-RU dataset* [73], based on a setup similar to a warehouse in the Robotics Lab at Boston University; see Fig. 4.1 for an example. The dataset contains videos of a drone (Parrot Bebop) that simulate a warehouse where human-interpretable signs (road signs) were used as cues to help the drone navigate. Each road sign is associated with a specific control command. We collect videos in three different locations in the lab, namely, an hallway, a dedicated flying arena, and an office area. In each case, the drone is tele-operated with a laptop through a WiFi connection. The resulting dataset contains scenarios with different conditions in terms of clutter and illumination (see Figs. 4.3)

Data preprocessing: We give now a few more details about the two datasets used in this work.

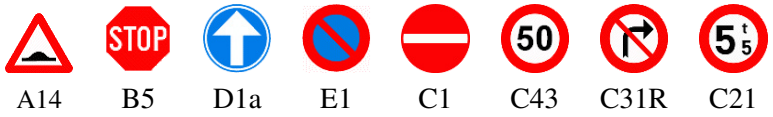


Figure 4.8: Traffic signs from the KUL dataset [4] (with the codes assigned to them in the dataset) that we have considered for performance evaluation of our proposed solution.

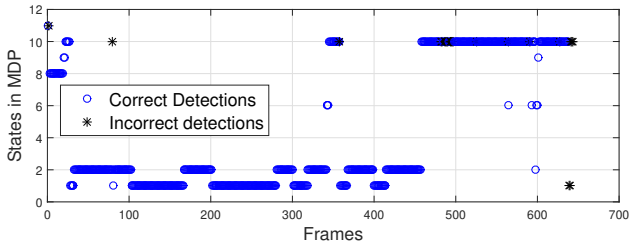
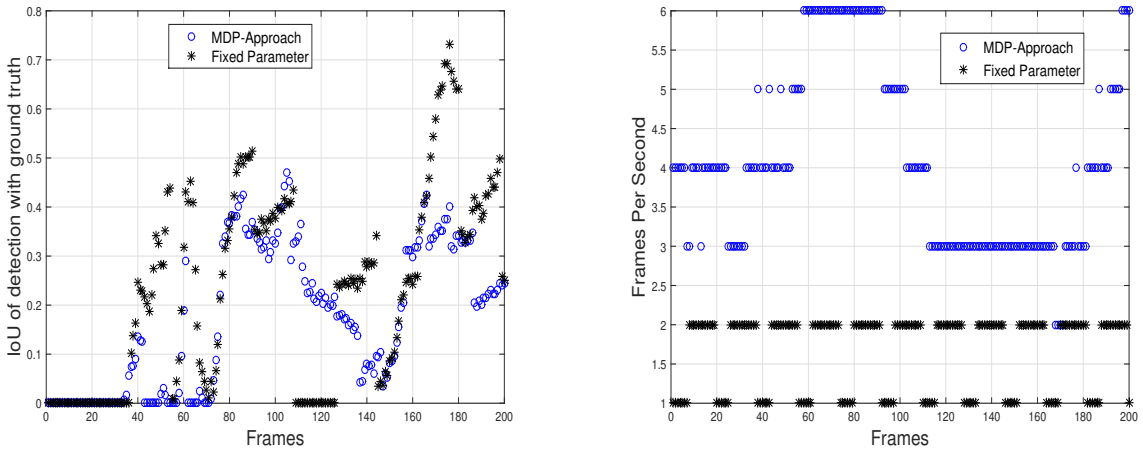


Figure 4.9: Illustration of the MDP state space for the object detection algorithm that the navigation system accesses for an example video sequence from the BU-RU dataset.

KUL dataset: In Fig. 4.8 we show the different road signs used in our evaluation. This set of signs was selected to span sufficient shape variability while appearing often enough in the dataset (i.e., in a sufficiently high number of video frames) so to have statistically relevant results. We extract the portion of the video frames from the dataset which continuously include the selected signs. Typically only one road sign per video frame is present, with few exceptions. These extracted videos are, in average, 40 frames long. Since the original dataset does not provide the ground truth location of the objects in all the video frames, we manually annotated this information in our smaller videos. Unfortunately, given the limits of the instances in which a sign appears continuously in the KUL videos, this dataset is not sufficient to fully test our MDP-based approach. Thus, we have collected a new dataset with longer sequences.

BU-RU dataset: The video streams for this dataset were shot by a 14 mega-pixel 180° fisheye camera on a Parrot Bebop drone. The raw video stream is cropped and rectified to a plain video with about 80° (horizontal) by 50° (vertical) field of view. The quality of the video stream is limited to $480 \times 270\,\text{px}$ at $30\,\text{fps}$. We collected 10 videos each for different values of background clutter (low, medium, high). Each video is about $10\,\text{s}$ long (i.e., around 300 frames). We also collected 6 videos in poorly illuminated locations, where each video is also $10\,\text{s}$ long. We also collected 200 negative training images with no road signs in them. Our framework is run on a laptop with an Intel i7 CPU $3.4\,\text{GHz}$ processor.

MDP states for an example video sequence: We now present an example video sequence

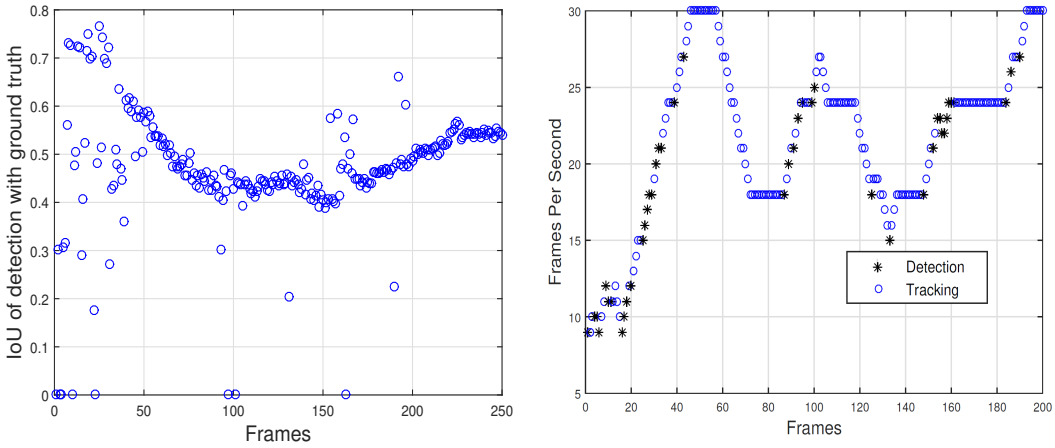


(a) (b)

Figure 4.10: Comparison of performance for an example video sequence in terms of (a) IoU of ground truth with detected locations; (b) Frames per second achieved for an object detection algorithm by parameter selection versus fixed parameter approach.

to show the advantage of selecting different parameters for road sign detection for HoG object detection algorithm. We choose a sequence from the BU-RU dataset. When the first frame arrives, the MDP decision framework is triggered and good-enough parameters {`BB`, `bsize`, `csize`, `nhist`} are selected by the MDP. If after executing the object detection algorithm with these parameters the classifier score is greater than the threshold T_c, then the navigation system uses the same set of parameters for the next frame otherwise the MDP is triggered again. In Fig. 4.9 we can see that due to the temporal correlation between frames, the same parameter values (i.e., state) are reused for multiple video frames.

Note that, while the drone is processing a particular frame, all the other incoming frames are dropped until the drone finishes its processing. Our decision framework focuses on reducing the time taken to process each frame so that the number of frames processed is closer to the incoming frame rate. We now compare the performance of our MDP-based framework against a fixed choice of parameters; for the latter, we use the tuple {1000,2,8,9} as in [6]. The deviation of the detected road sign from the ground truth is shown in Fig. 4.10(a). We see that our framework achieves up to 6 fps, while the fixed parameter approach tops out at 2 fps, as shown in Fig. 4.10(b). This is because our framework adapts the parameters using the first few frames, lowering the execution time for the other frames.



(a) (b)

Figure 4.11: Real-time performance of up to 30 fps for some frames and the lower bound of 10 fps for frames in a sequence when MDP based parameter selection is implemented along with tracker. The IoU is greater than 0.4 for majority of frames.

(a) (b)

Figure 4.12: Video frames from one of the video sequences in the KUL dataset. The tracker was initialized using the location from given by the detector in (a) i frame. The tracker has drifted away from the object location in the (b) $i + 5$ frame and should be re-initialized via location generated by object detection algorithm.

Achieving real-time performance: Even with the incorporation of our approach, the processing rate achievable through pure object detection is far below the frame rate of the drone platform (30 fps). To improve our real-time performance and also improve the IoU between detected object and ground truth, we can choose to not trigger the MDP for every new frame. After the MDP finishes adapting from the first few frames, we can check the classifier score achieved on the successive frame: if this score is greater than the predefined threshold T_c, the KLT tracker is initialized with the detected object. Here, instead of discarding the remaining frames during processing, as we did

Thresholds	Value
Re-trigger MDP	100 frames
Re-initilize KLT	10 frames
Score thresh T_c (KUL)	0
Score thresh T_c (BU-RU)	-1
Access MDP per image	5
KLT thresh θ_{th}	0.5
Template matching T_m	0.5
Max. object proposals	2000
IoU for evaluation	0.5

Figure 4.13: Various thresholds considered in the performance evaluation of our solution.

	Average		Good Enough			
Codes	**P**	**E [s]**	**P**	**E [s]**	**Speed-up [%]**	**Accuracy Loss [%]**
A14	0.85	1.85 ± 0.3	0.71	1.59 ± 0.3	14.05	16.40
B5	0.57	2.21 ± 0.11	0.46	2.6 ± 0.51	-	
D1a	1	2.1 ± 0.13	0.78	1.10 ± 0.02	40.1	22.0
E1	0.99	2.1 ± 0.28	0.95	2.14 ± 0.06	45.7	4.0
C1	0.75	2.06 ± 0.16	0.61	1.59 ± 0.1	22.8	6.7
C43	0.82	2.09 ± 0.05	0.60	1.95 ± 0.01	6.7	26.8
C31R	0.85	2.04 ± 0.16	0.80	1.89 ± 0.7	7.4	5.0
C21	0.50	1.88 ± 0.1	0.50	1.56 ± 0.1	17.0	0

Figure 4.14: KUL dataset: Comparison of precision and execution time of road signs (codes) in KUL dataset for fixed parameters [6] and good-enough parameters given by MDP approach for object detection algorithm.

earlier, we run the KLT tracker on these frames in parallel. Interested readers can see the video showing the real-time performance of our proposed framework at [74].

We use as template of the road sign an average of several images obtained from the training data. We continue to track until the normalized distance between the histograms of the template and detected region by the tracker is greater than a predefined threshold T_m (equal to 0.5 in our experiments). In Fig. 4.11(a) we see that this approach leads to an improvement in performance and we get up to 30 fps for some frames and the lower bounds on FPS achieved for this sequence is 10 fps with IoU greater than 0.4 for majority of frames. Also, Fig. 4.11(b) shows that the MDP (shown by black dots) is triggered when the pairwise distance between the histograms of the template and

detected region by the tracker is below T_m. KLT tracker is computationally cheaper and gives us higher FPS performance than using only an object detector. However, object trackers accumulate error during runtime (drift) and typically fail if the object disappears from the camera view. Hence, a combination of detector and tracking helps in achieving high performance with low execution time. Our solution brings further benefits in reducing time by choosing parameters of the object detection algorithm adaptively. Figure 4.12 gives an example of how we cannot solely rely on the tracker and so we use a combination of detector and tracker.

Performance on the KUL dataset: We measure quantitatively the performance benefits in terms of execution time and accuracy of our proposed MDP-based approach for selecting good-enough parameters. The different thresholds considered in our performance evaluation are shown in Fig. 4.13. We quantify the benefits of approximation in Fig. 4.14. Here the execution time corresponds to the time taken to run the object detection workflow per frame. The results show that approximation at the parameter level can indeed bring benefits in terms of execution time for the KUL dataset. The first column of the table indicates the codes of these roads signs used in the dataset as shown in Fig. 4.8; for most of the traffic signs we get a gain in execution time for a small penalty of accuracy. For instance, for road signs E1, C1, and C31R, and C21 we are able to get a speed up (in percentage) of 45.7, 22.8, and 7.4, and 17.0 and get accuracy loss of 4.0, 6.7, 5.0, and 0.0, respectively.

Performance with varying clutter and illumination: Here we test the performance of our solution under different values of background clutter and illumination on our BU-RU dataset. In Fig. 4.15(a,b,c) we show the performance of the MDP when the background clutter is low (e.g., as in Fig. 4.3(a)) and when the background clutter is high (e.g., as in Fig. 4.3(c)). We quantify the background clutter using [70] as 4.05 ± 0.53 for high clutter, and 2.0 ± 0.63 for low clutter. The time taken by different components of our framework is given in Table 4.1.

We compare the following scenarios: (i) Detection via fixed parameters (we run the object detection algorithm on each frame with the same parameters); (ii) Random parameter selection with tracker (detection is done using a random parameter tuple); (iii) MDP-based detection (we adapt the object detection parameters using our MDP, using the same parameters as long as the IoU of location detected by tracker and object detector is higher than a pre-defined threshold θ_{th}); (iv) Fixed parameter detection with tracking (fixed parameters are used for object detection, but the

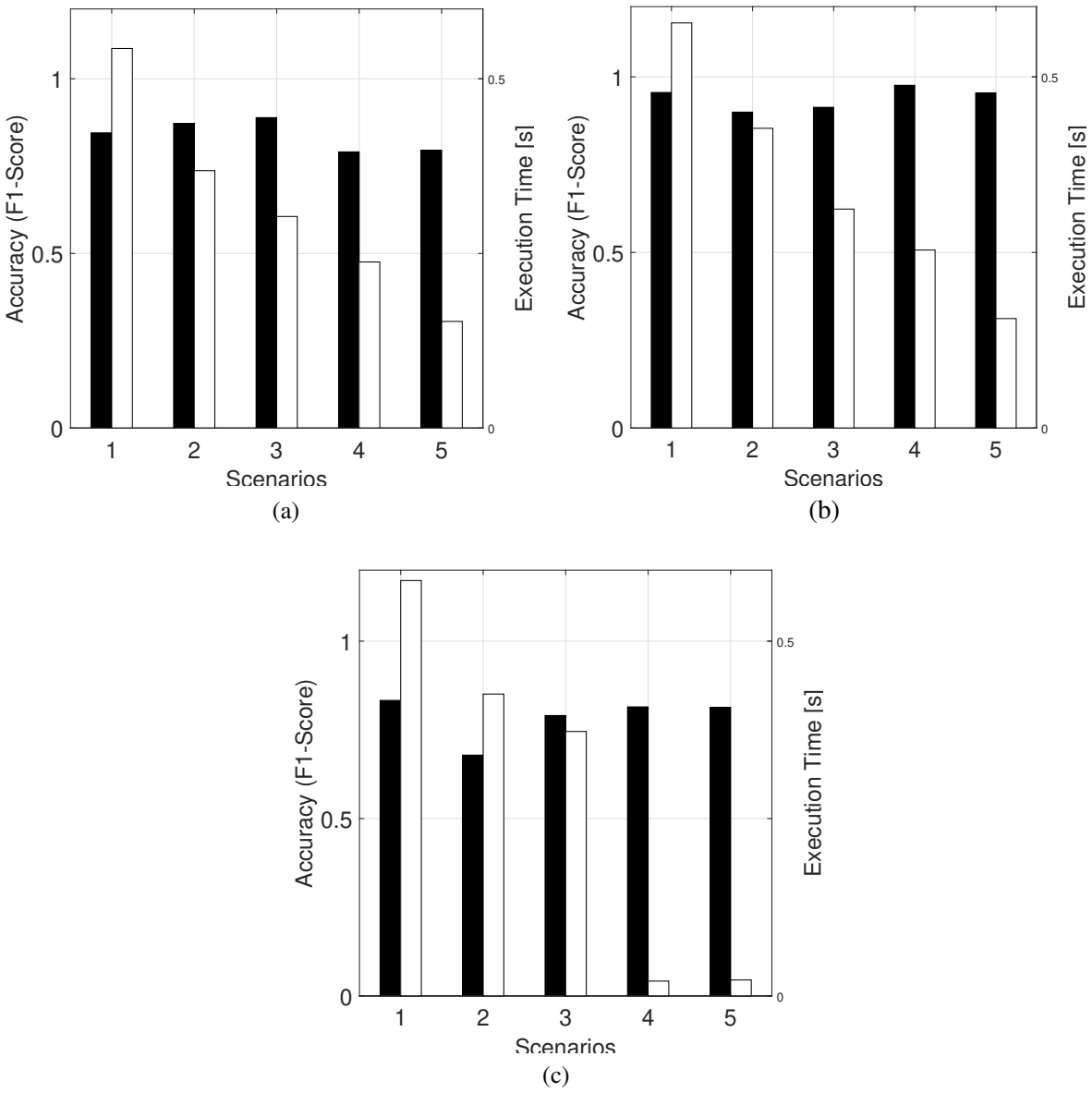


Figure 4.15: BU-RU dataset: Performance of object detection in a video under different scenarios, (i) Fixed parameter, (ii) Random parameter, (iii) MDP-based parameter, (iv) Fixed parameter and tracking, and (v) MDP-based approach and tracking for different conditions (a) low and (b) high background clutter; (c) poor illumination.

object detector is only run when the overlap of location detected by object tracker and template is less than a pre-defined threshold T_m); and finally (v) MDP-based detection with tracking (as the previous one, but the parameters of the object detector are identified by the MDP).

Background clutter: Figure 4.15(a) shows results for low background clutter condition. The fixed parameter approach (Scenario i) of [6] has the best performance but it also takes the highest

Component	Runtime [ms]
Communication	< 3
ROS preprocessing	5.4
MDP Parameter search	36
Feature extraction (variable)	100-800
Object tracker	3.7

Table 4.1: Time taken by different components of the framework for object detection in a video frame on an Intel i7 CPU 3.4 GHz processor.

execution time. The MDP approach (Scenario iii) brings 39% gain in time 4% loss in accuracy. In the case where tracking is used (scenarios (i) and (iv)), we see that MDP based brings further gains in execution time of 35%. Triggering the MDP only at specific intervals (Scenario (v)) when the performance of tracker fails gives us a 50% better performance than triggering the MD at every frame (Scenario (iii)). The benefits of the MDP approach are visible also in the case of high background clutter, as shown in Fig. 4.15(b). In particular, we see that MDP-based approach with tracking (Scenario (v)) brings a 38% gain in execution time at a cost of 2% loss in accuracy.

Poor illumination: In Fig. 4.15(c) we show the performance of the MDP when the illumination is low (e.g., as in Fig. 4.3(e)). We see that MDP approach (Scenario (iii)) again achieves a gain of 35% in comparison to fixed parameter approach (Scenario (i)). However, along with the tracker both fixed parameter and MDP-based approach give similar results because the detector is not triggered frequently and is able to achieve acceptable performance (overlap of tracker and template above T_m) with the tracker. As, a result the execution time of this two scenarios is significantly lower than other results (since the tracker is computationally much cheaper than the object detector).

4.7 Summary

In this chapter, we presented a method to handle the problem of resource constraints in mobile robots. We targeted the object detection problem and presented a solution based on Markov Decision Processes to select the parameters of computationally-intensive computer vision algorithms and bring benefits in terms of time for long-term object-detection in videos. We showed the benefit of our formulation with experiments on a public dataset and a new dedicated dataset via experiments. We showed a decrease in the execution time of a object detection application by 20-70% with an accuracy of 98-100% in comparison to fixed parameter approach in existing works.

Chapter 5

Robust Distributed Dictionary Learning for In-network Image Compression

In this chapter we move from a single camera system to a distributed camera system. Camera networks are resource-constrained distributed systems that communicate over wireless networks to make decisions collaboratively. For surveillance applications, these camera nodes take decisions about an object of interest within incoming videos by coordinating with neighboring nodes, which is a costly process in terms of both time and energy. Data-compression methods can bring significant energy savings in camera nodes while transmitting or storing data in the network. Signal representation using sparse approximations and overcomplete dictionaries have received considerable attention in recent years and have been shown to outperform traditional compression methods. However, distributed dictionary learning itself relies on consensus-building algorithms, which involve communicating with neighboring nodes until convergence is achieved. We design a novel protocol to enable energy-efficient and robust dictionary learning in distributed camera networks by leveraging the spatial correlation of collected multimedia data. To this end, we employ low-computational-complexity metrics to quantify the correlation across cameras nodes. The performance of the proposed approach is validated through extensive simulations using a network simulator and public datasets as well as via real-world experiments on a testbed of Raspberry Pi nodes.

5.1 Introduction

Cameras networks are real-time distributed systems that cover large spaces and communicate over (wireless) networks to make decisions collaboratively. These cameras refer to a system of physically distributed camera nodes that may or may not have overlapping fields of view [75] and that allow us to see a subject of interest from several angles, helping resolve the problem of occlusion faced by individual cameras. Camera networks have been used in surveillance applications where the

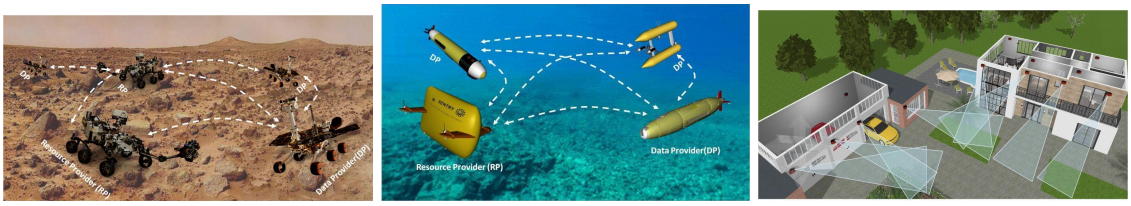
videos by the cameras are sent to a centralized location and are analyzed offline. Sending video data (generated throughout the time of operation at 30-60 fps) from a large number of cameras in a network to a central server is expensive (in terms of time and energy of camera nodes) and inherently unscalable. The combination of large numbers of nodes, security concerns, expectation of fast response times, and delay in communicating data to the central node pushes us away from server-based architectures.

There has been some recent works focusing on *smart distributed camera networks* that combine video sensing, processing, and communication on a single embedded platform. These camera nodes analyze the object of interest online on incoming videos and take decisions by running computationally-intensive computer-vision algorithms for detection and tracking of object of interest [76]. This requires communication between the camera nodes that involves multiple transmissions of raw data, which is a costly process (in terms of time and energy) for the resource-constrained nodes and reduces the mission lifetime of the nodes. These limitations call for efficient image-compression methods for transmission and storing of data in the network. Signal representation using sparse approximations and overcomplete dictionaries [77] have received considerable attention in recent years in the area of image feature extraction, data compression, and bit-rate reduction [78–80] for both storage and transmission. The goal of dictionary learning is to learn an overcomplete dictionary D such that data samples, represented as a matrix Y, are well approximated by no more than T_0 columns of D. Mathematically, the problem of dictionary learning can be expressed as

$$\big(D, X\big) = \arg\min_{D,X} \|Y - DX\|_F^2 \ \text{ s.t. } \forall s, \|x\|_s \leq T_0, \tag{5.1}$$

where $D \in \mathbb{R}^{n \times K}$ with $K > n$ is an overcomplete dictionary having unit l_2-norm columns, Y is the data available at a centralized location, $X \in \mathbb{R}^{K \times S}$ are the sparse coefficients of the data having no more than $T_0 \ll n$ nonzero coefficient per sample, and x_s represents the s^{th} column in X. The value of T_0 is selected based on the sparsity expected by the user.

Once a dictionary is learned, each node can learn a sparse approximation of the signal that can be used along with an image-compression technique to help save significant space when storing the data and energy when offloading data to a centralized location or to neighboring nodes. Consensus forms the communication primitive to be used between neighboring camera nodes for dictionary



(a) (b) (c)

Figure 5.1: Illustration of three different types of distributed camera networks where image compression using dictionary learning can bring potential benefits in saving battery capacity of camera nodes in the network. Each sensor node can serve as a Data Provider (DP) or a Resource Provider (RP). The RP nodes have sophisticated computational capabilities and higher battery capacity than DP nodes. (a) Mars exploration with *Curiosity* rover [7]; (b) Underwater environment covered with a network of Autonomous Underwater Vehicles (AUVs). AUV serving as DP continuously capture raw data, while RPs—such as *Mesobot* and *Sentry* [8]—can create bathymetric maps and perform sidescan of the seafloor by taking and processing images in deep sea and oceans; (c) Deployment of multiview distributed cameras in a smart home application designed using [9].

learning in a setting where the data is distributed such as in distributed camera networks [81,82]. In particular, consensus is an iterative process where the camera nodes communicate with their neighbors for a fixed number of iterations or until convergence. This iterative communication introduces an additional overhead of energy consumption on nodes in the network.

We focus on developing energy-efficient dictionary-learning techniques for distributed camera networks. We consider a distributed camera network where the camera nodes have *heterogeneous* capabilities in terms of battery capacity and processor capabilities. We divide these camera nodes into two categories, namely, *Data Provider (DP)* and *Resource Provider (RP)*. The DP nodes are *dumb nodes* in the sense that they are severely resource constrained and have limited computational capabilities. The RPs, on the other hand, have sophisticated computational capabilities and higher battery capacity than DP nodes. Figure 5.1 shows examples of various futuristic applications of the proposed distributed solution in harsh environments—such as on Mars and in underwater environments—as well as for consumer applications—such as in smart home cameras.

We design a protocol to identify the camera nodes in the network with spatial correlated data by using only local knowledge of the network available to the camera nodes. We use computationally cheap metrics to quantify the correlation of data among camera nodes. We also present a study of the performance of different parameters of the distributed dictionary learning algorithm in terms of accuracy of the algorithm and energy consumed by the nodes to implement the algorithm. Distributed dictionary learning can be used to enable a variety of applications in a distributed camera

network such as background modeling and object classification [82, 83]; in this chapter, we focus on in-network image compression.

The following are the main contributions of this chapter.

- We present via simulations the scalability of consensus in a wireless distributed computing infrastructure by varying the wireless network environment variables (i.e., density, connectivity, and number of consensus iterations).
- We develop a protocol to enable energy-efficient, robust consensus process on resource-constrained camera nodes by identifying the network nodes with correlated data.
- We quantify the trade-off in energy savings and loss in accuracy achieved for in-network image compression by implementing our protocol on a testbed of Rasberry Pi nodes.

The remainder of this chapter is organized as follows. In Sect. 5.2, we cover the work done in the area of distributed systems employing consensus and analyze the difference of these approaches from ours. In Sect. 5.3, we study how different consensus parameters impact the feasibility of convergence. In Sect. 5.4, we present details of the design of our protocol that is proposed to bring energy savings to the distributed dictionary-learning algorithm executed in a camera network. In Sect. 5.5, we provide quantitative results that demonstrate the merits of our contributions. Finally, in Sect. 5.6, we conclude our chapter.

5.2 Related Work

We compare here the work done in the area of distributed camera networks, dictionary learning, and image compression. We cover different communities in which distributed consensus was studied and consider works that tackled the problem of energy-efficient consensus.

Mobile cloud computing: In Chapter 2 we have discussed the various works done in the domain mobile computing to solve the issue of limited computational capability of devices. We now briefly reiterate some of these works here. These works have been divided into two categories: (i) where the resource-constrained device offloads its tasks to a remote cloud. This research involves code partitioning to determine which tasks in the application will be executed locally and which in the cloud [84]. COSMOS (proposed in [85]) suggests a system to fill the gap between the demands for

computing resources by individual mobile devices and the ones cloud providers offer; and (ii) offloads tasks to mobile devices in the vicinity, which execute the tasks in parallel and return results back to the device. Authors in [86] have implemented a preliminary ad-hoc distributed prototype to offload portions of a service from a resource-constrained device to a nearby server. A participatory computing system is proposed in [38] to derive the minimal workload for individual participating devices to achieve the overall system performance requirement. Serendipity [87] also exploits the remote computational resources available in other mobile systems in a collaborative manner.

Similar solutions can be seen in the area of distributed camera networks. These works have focused on conserving energy for applications via (i) offloading or handover [88] of computationally intensive to other nodes and (ii) using dynamic power management schemes [89]. Our solution looks at the problem of energy-efficient computation in distributed networks from a different angle. Dictionary learning is a energy-intensive algorithm because of the high communication between the nodes. The computation cost of executing distributed dictionary learning on nodes is much lower than the communication cost. In the later sections we give more detail on the energy cost of both computation and communication tasks in dictionary learning algorithm. To reduce the energy cost in distributed dictionary learning we focus on identifying a subset of nodes on which dictionary learning can be executed. To identify these nodes we rely on the data captured by the nodes as a result of which the lifetime of nodes is extended.

Distributed consensus: Many existing works have considered consensus for ad-hoc networks via gossip algorithms where each node uniformly at random contacts a neighbor within its transmission range and exchanges data [90]. There have been a few recent works that have focused on gossiping via broadcasting [91] and geographic gossiping [92]. However, these works make some unrealistic assumptions that are not consistent with real-word deployment scenarios such as knowledge of topology, unit disk graph models, uniform distribution of nodes, and homogeneous battery capacity of nodes. In our work we do not make such assumptions and rely on the local knowledge available to the nodes to make decisions.

Dictionary learning: Dictionary learning can be performed in either centralized [77] or distributed structures [81]. Centralized dictionary learning is not efficient as (i) the nodes have to communicate with a central node (usually at a far distance), which imposes high communications costs, especially in extreme environments; (ii) move the nodes in order to get closer to the central

node imposes extra delays, which challenges the real-time processing; (iii) the structure is vulnerable to break if the centralized node fails to provide the required support. This failure can happen for energy, communication, or computational restrictions in a network. On the other hand, when there are multiple sensors in the network, e.g., camera network, where images are collected and stored in multiple geographic locations, a distributed solution is energy efficient, avoids a single point of failure, and provides more flexibility and robustness against the topology variations of the network. Our focus in this chapter is to present a solution to make distributed dictionary learning energy-efficient so that it can be beneficial to run on resource-constrained nodes.

Image compression: Image compression is a popular tool for image data size reduction, so that data transmission in bandwidth-limited and error-prone channels become feasible and efficient. Compared to the traditional lossy image compression methods—such as in Joint Photographic Experts Group (JPEG) and Discrete Wavelet Transform—dictionary learning can be used to enhance image compression algorithms. A sparse model seems to fit the natural images as well as the human visual perception of the images [93]. An improved sparse representation algorithm was proposed in [94], in which dictionaries with overlapping atoms are generated in the wavelet coefficient domain and in the pixel domain so as to remove blocking artifacts. Distributed dictionary learning has shown to achieve dramatic improvement over JPEG/JPEG2000 compression techniques [79, 95]. In our work, we utilize image-compression via distributed dictionary learning and provide an energy-efficient way to implement it. In the next section we will explain the different tasks of distributed dictionary learning algorithm.

5.3 Dictionary Learning and Consensus in a Distributed Camera Network

In this section we first present different steps of the distributed dictionary learning algorithm. We then show the performance of consensus and distributed dictionary learning in terms of accuracy of the algorithms and energy consumed by the nodes to run these algorithms in a network. This section will help us to motivate the need to save energy during the execution of a distributed dictionary learning algorithm.

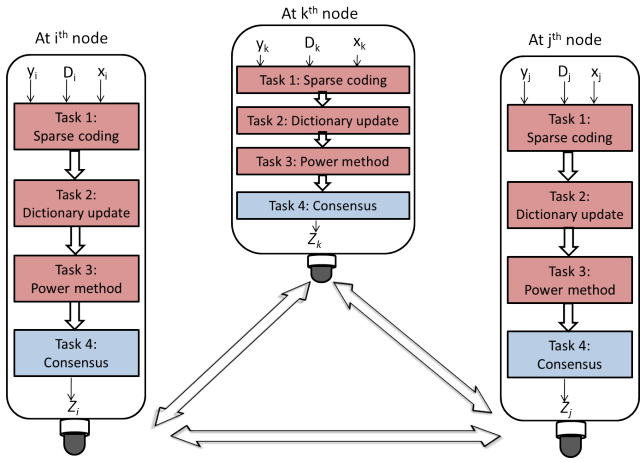


Figure 5.2: Workflow for distributed dictionary learning executed at each camera node in the network. Task 1, 2, and 3 are executed locally at each node; whereas Task 4, i.e., consensus, involves communication between nodes that are in communication range of each other.

5.3.1 Distributed Dictionary Learning

Assume N camera nodes used for surveillance form an ad-hoc network and communicate over a wireless network. Any wireless communications technology such as Bluetooth, WiFi Direct, 802.11 ad-hoc mode, or Near-field Communication can be used in this scenario without relying on any existing fixed infrastructure. Two camera nodes are considered neighbors when they are in the communication range of each other. Each node is capable of storing some local data, performing computations on it, and exchanging messages with its neighbors. Next, we assume each node i has a collection of local data, expressed as a matrix $Y_i \in \mathbb{R}^{n \times S_i}$, with S_i representing the number of data samples at the i^{th} node. In case of distributed camera networks, each camera node is taking images of a scene from a different angle, which form the data samples at the i^{th} node. We can express all this distributed data into a single matrix $Y = [Y_1 \cdots Y_N] \in \mathbb{R}^{n \times S}$, where $S = \sum_{i=1}^{N} S_i$ denotes the total number of data samples distributed across the N nodes.

In a distributed setting, the goal of dictionary learning is to have individual nodes collaboratively learn dictionaries $\{\widehat{D}_i\}_{i \in N}$ from global data Y such that these collaborative dictionaries are close to a dictionary D that could have been learned from the global data Y in a centralized fashion. We now explain the different tasks of this algorithm, as described in [82] (Fig. 5.2).

Task 1–Sparse coding: Each i^{th} node uses the local estimate $\widehat{D}_i$ of the centralized dictionary D and calculates the sparse coefficients of its local data by solving the following equation without

collaborating with other nodes,

$$\tilde{x}_{i,s}^{(t)} = \arg\min_{x \in \mathbb{R}^K} \|y_{i,s} - \widehat{D}_i^{(t-1)} x\|_2^2 \text{ s.t.} \|x\|_0 \leq T_0, \ \forall s \in \{1, \ldots, S_i\}, \tag{5.2}$$

where $y_{i,s}$ and $\tilde{x}_{i,s}^{(t)}$ represent the s^{th} sample of the data and its coefficients at node i at the t^{th} dictionary learning iteration.

Task 2–Dictionary update: Dictionary update stage involves computing the dominant left (u_1)- and right-singular (v_1) vectors of the "reduced" error matrix $\widehat{E}_{i,k,R}^{(t)}$. The *Cloud K-SVD* algorithm of [82] defines $\widehat{d}_{i,k}^{(t)} = u_1$ and $\widehat{x}_{i,k,R}^{(t)} = \widehat{d}_{i,k}^{(t)^T} \widehat{E}_{i,k,R}^{(t)}$, where u_1 is the dominant vector of a matrix $\widehat{M}^{(t)}$ that is formally described in [82]. We need to only worry about calculating u_1 collaboratively. To this end, at each node, $\widehat{M}_i^{(t)} = \widehat{E}_{i,k,R}^{(t)} \widehat{E}_{i,k,R}^{(t)}$ is calculated. Our goal now is computing the dominant eigenvector of $\widehat{M}^{(t)}$ in a collaborative manner. In order to estimate the eigenvectors in the distributed network, Cloud K-SVD algorithm uses the distributed power method.

Task 3–Power method: Power method is an iterative procedure used to estimate the eigenvectors of a matrix. Power method is run for a fixed number of iterations (t_p) or until convergence. Cloud K-SVD algorithm is interested in distributed power method as M_i's are distributed in the network. To this end, each site calculates $z_i = \widehat{M}_i^{(t)} \widehat{q}_i^{(t_p-1)}$ locally, where $\widehat{q}_i^{(t_p-1)}$ denotes an estimate of the dominant eigenvector of $\widehat{M}^{(t)}$ at i^{th} site after $(t_p - 1)$ power method iterations. We initialize each site with the same value (q^{init}). In Cloud K-SVD, one of the ways that this can be achieved is if each node uses the same seed for a random number generator. Next, the sites collaborate to calculate $\widehat{v}_i^{(t_p)} = \sum_i \widehat{M}_i^{(t)} \widehat{q}_i^{(t_p-1)}$ at each site. The normalized $\widehat{v}_i^{(t_p)}$ is an estimate of the dominant eigenvector of $\widehat{M}^{(t)}$.

Task 4–Consensus averaging: To calculate the approximation of $\sum_i \widehat{M}_i^{(t)} \widehat{q}_i^{(t_p-1)}$, nodes in the network make use of the distributed consensus technique. Each node is initialized as $z_i^{(0)} = \widehat{M}_i^{(t)} \widehat{q}_i^{(t_p-1)}$ at each node. We now give an example of how the consensus-averaging problem in a network can be defined. Let $Z^{(0)} = [z_1^{(0)}, \cdots, z_N^{(0)}]^\top$ be the initial value at each node. Each node achieves perfect consensus averaging as $t_c \to \infty$, where t_c is the number of consensus iterations, and obtains ${Z_{i,T}^{(\infty)}}^\top = \frac{1}{N} \sum_{j=1}^N z_j^{(0)} = \frac{1}{N} \sum_{j=1}^N \widehat{M}_j^{(t)} \widehat{q}_j^{(t_p-1)}$.

We take advantage of the wireless medium being inherently broadcast, and hence use a broadcasting based consensus method proposed in [91]. The asynchronous broadcast consensus assumes

each node i broadcasts its own data to its $\mathcal{N}_i$ neighboring nodes within its communication range. The neighbors, which received the data, update their data according to the weighted average of their current data as follows [91]:

$$z_k^{(t_c+1)} = \gamma z_k^{(t_c)} + (1-\gamma) z_i^{(t_c)}, \forall k \in \mathcal{N}_i, \tag{5.3}$$

where $\gamma \in (0,1)$ stands for the mixing parameter, which gives the weight assigned to data values from each node, and $\mathcal{N}_i$ denotes the neighboring nodes of the i^{th} node. The remaining nodes in the network update their values as,

$$z_k^{(t_c+1)} = z_k^{(t_c)}, \forall k \notin \mathcal{N}_i. \tag{5.4}$$

The optimal mixing parameter, minimizing a worst-case convergence rate, is given by $\gamma^* = \frac{N-\lambda_{N-1}(L)}{2N-\lambda_{N-1}(L)}$, where N is the number of nodes in the network, $\lambda_{N-1}(L)$ is referred to as the algebraic connectivity of the graph and L is the Laplacian matrix of the graph [96]. Both distributed power method (Task 3) and broadcast consensus (Task 4) are part of dictionary update stage, i.e., both these tasks are executed to update each column (atom) of the dictionary. As a result, for each execution of the dictionary update stage Task 3 and Task 4 are run for K times, where K is the number of columns in a dictionary. Dictionary update stage is itself run t_d times, which is the number of dictionary learning iteration.

Image compression scheme: Once a dictionary has been learned by the algorithm explained above, it can be used in an image-compression scheme in the following way. At the i^{th} node, for a new incoming image or an image that is to be sent to the centralized server, we first form the set of vectors from non-overlapping patches of the image. This is denoted by $\{y_j\}_{j=1}^{L}$ or Y, where L is the number of patches. Next, we estimate the sparse approximation X of Y using the estimated dictionary $\widehat{D}_i$. The non-zero coefficients are quantized, for which a uniform quantization scheme along with thresholding is used [97]. The quantized X matrix is then entropy coded to form a bit sequence before ending to the server.

5.3.2 Network Performance of Distributed Dictionary Learning

We now provide the specifics of our simulator in which consensus and dictionary learning algorithms are implemented and present the public datasets used to study the performance of these

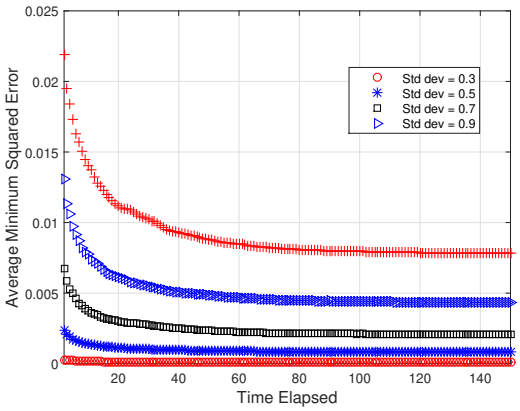
algorithms.

Simulation setup: We consider a static network of 20 randomly deployed digital nodes in a 1000×1000 region in a discrete-event Network Simulator (NS), NS-2 [98]. For our experiments, NS-2 was configured to use (i) a physical layer including a two-ray ground radio propagation model supporting propagation delay, wireless effects, and carrier sense as well as (ii) the IEEE 802.11 Medium Access Control protocol based on Distributed Coordination Function [99]. The NS-2 packet class only allows to define the size of the packet and does not allow to include the payload with real data values. Since, in Task 4 of distributed dictionary learning we are interested in identifying when the nodes have reached consensus for which we have to transmit actual data values between nodes. To this end, we create a new packet class and use byte arrays to handle the data exchanged between nodes. In the new packet class created we initialize variables for packet size, offset which points to the position in the NS-2 byte array where the header is stored, and array to store data values.

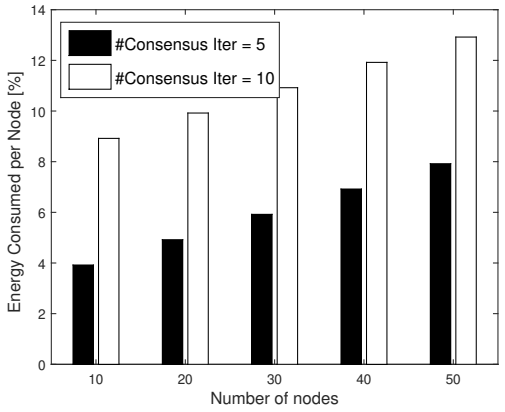
Datasets: We consider both simulated dataset and publicly available datasets to study the performance of our algorithms. For the simulated dataset, we assume that the data at the nodes have dimension $n = 64$ and they follow a Gaussian distribution with fixed mean and standard deviation. We also use the publicly available Multi-camera Pedestrian Videos dataset by EPFL [10].

Distributed consensus: We first study the performance of the consensus algorithm for a connected network topology of 20 nodes. We modify the existing node functionality in NS-2 to make the nodes work asynchronously, which is done via a timer functionality. The value of the delay variable supplied to the timer function determines the time delay from the present instant after which a node should broadcast its data. In our simulator, each node broadcasts 20 times after which the timer is not rescheduled. We calculate the mean square error at each node as the mean square of the difference between average of the data values under perfect consensus ($t_c \to \infty$) and the data values at the nodes after a fixed number of consensus iterations. In Fig. 5.3(a), we plot mean square error of data value averaged over number of nodes; we see that, as the standard deviation of the data is increased from 0.3 to 0.9, the consensus algorithm converges with a higher error due to the finite and fixed number of iterations.

Battery capacity: We now show the battery consumed by nodes in a distributed camera network when the consensus is implemented. These measurements have been made using the energy model



(a)

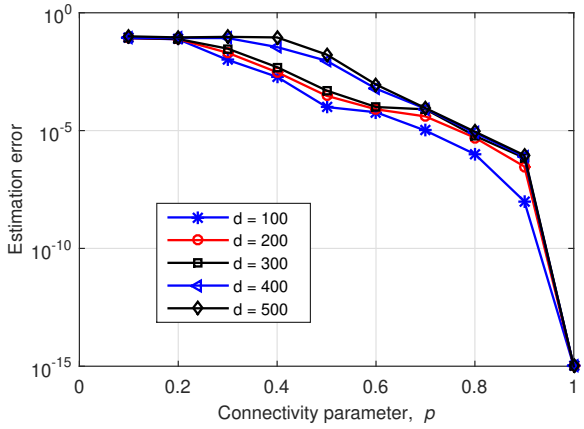


(b)

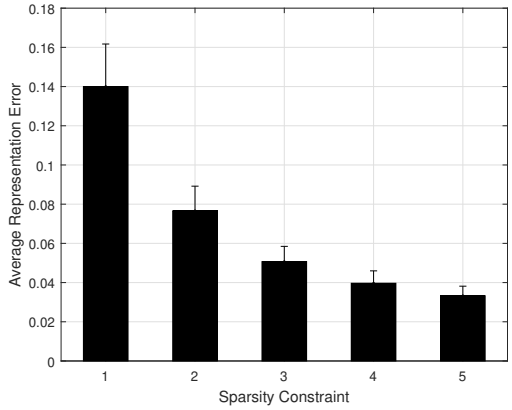
Figure 5.3: Average mean square error of consensus as the standard deviation of data is varied; (b) Average energy consumed per node in the network for different number of consensus iterations.

defined in the NS-2 simulator. This model sets the transmit power of the node at 0.2 W and the received power at 0.1 W. In Fig. 5.3(b), we show the percentage of battery capacity consumed by the nodes for different values of consensus iterations, i.e., number of consensus iterations as 5 and 10. We see that for battery consumed for consensus iterations $t_c = 10$ along with $t_p = 10$ power iterations, the battery consumed by the nodes is up to 12%. To extend the mission lifetime of the nodes, we will present in the next section how the communication cost in a network can be reduced with a small loss in accuracy.

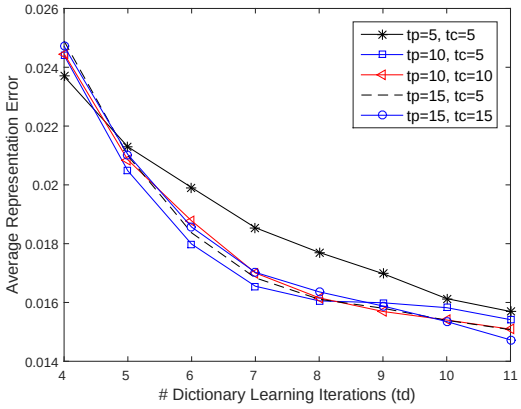
Graph connectivity: To study the impact of graph connectivity on the distributed power method algorithm, we implement a network graph in MATLAB. We generate a random graph based on the Erdős-Rényi model, which takes as input the number of nodes/vertices and ρ gives the probability of an edge between any two nodes in the network, with $\rho = 1$ indicating that the network graph is



(a)



(b)



(c)

Figure 5.4: (a) Average error in eigenvector estimates of distributed power method obtained by varying the connectivity parameter ρ in Erdős-Rényi graph; (b) Average representation error of Cloud K-SVD as a function of the number of non-zero coefficients per sparse representation (sparsity constraint) with 95% confidence interval; (c) Average representation error of Cloud K-SVD as the dictionary learning iterations are varied for different values of consensus and power iterations.

fully connected. Erdős-Rényi model is used because of its simplicity of analysis due to the independence of the links between the nodes [100]. These edges can be considered wireless links which are

represented by independent channels that are either on (with probability ρ) or off (with probability 1 - ρ) [101]. Other models such as unit disk graph model [102] popularly used in wireless communication can also be used. In Fig. 5.4(a), we study the impact of graph connectivity on average error in eigenvector estimates of distributed power method. We observe that as the connectivity parameter increases ρ, i.e, the probability of an edge between two nodes in the network increases, the estimation error in distributed power method also decreases. This is because as ρ increases each node is able to reach a higher number of neighboring nodes in every broadcast transmission, which causes faster information dissemination in the network. As a result of which, for a fixed number of consensus iterations when ρ increases the consensus error decreases which leads to lower error in distributed power method.

Sparsity constraints: We now study the impact of the sparsity of sparse representations, i.e., maximum number of non-zero coefficients per sample x_s on the average representation error of Cloud K-SVD, defined as $\frac{1}{nS}\sum_{i=1}^{N}\sum_{j=1}^{S_i}\|y_{i,j} - Dx_{i,j}\|_2$. We first create a set of training and test images using the dataset [10]. Next, we learn the dictionary using the Cloud K-SVD algorithm. The learned dictionary is used to find sparse representations of the test dataset, and we then calculate the representation error. In Fig. 5.4(b), we plot the average representation error and observe that, as the number of non-zero coefficients per sparse representation is decreased, the error increases.

Number of iterations: In the distributed dictionary learning algorithm [82], three iterations are defined, namely, dictionary-learning iterations, power-method iterations, and consensus iterations. In Fig. 5.4(c), we see that the dictionary-learning iterations have the highest impact on the representation error, i.e., for different values of consensus and power iterations, when the dictionary-learning iterations are increased we observe a decrease in the representation error. Among the power and consensus iterations we see that increase in power iterations for fixed consensus iterations lead to higher reduction in representation error. As we increase the number of power iterations from $t_p = 5$ to $t_p = 15$, with $t_c = 5$, we observe the representation error falls by 38.2%, however, when we increase consensus iterations from $t_c = 5$ to $t_c = 15$, with $t_p = 15$, the representation error of the dictionary learning algorithm falls by 26.5%.

5.4 Energy Efficient Dictionary Learning

Dictionary learning is an energy-intensive technique which reduces the mission lifetime of nodes in the network and if the nodes shutdown due to exhaustion of battery it can cause the network to be disconnected and as a result of which the network fails to achieve consensus. Our goal in this chapter is to reduce the energy consumed by the nodes to perform dictionary learning in a camera network. To this end, we design a protocol to select a subset of nodes in the network to perform distributed dictionary learning and extend the lifetime of the network.

5.4.1 System Model

We consider a scenario with a number of heterogeneous camera nodes, e.g., in an Internet of Things scenario, as shown in Fig. 5.1(c). The nodes have different roles, i.e., DPs and RPs, where RPs are camera nodes with sophisticated computational capabilities and higher batter capacity than DPs. In a homogeneous environment where there is no difference between the hardware capabilities of devices, we can assume that the role of RPs can be randomly selected or given to nodes that have residual battery capacity over a certain value. The configuration of camera at each node is given by its Field of View (FoV), which refers to the directional view of a camera sensor and is assumed to be an isosceles triangle (two-dimensional approximation) with vertex angle θ and length of the congruent sides R_s (sensing range of the sensor), and orientation α [103].

5.4.2 Quantifying Spatially-correlated Nodes

We now present various techniques to identify nodes in a camera network that have correlated camera data. Silencing nodes, i.e., these nodes do not participate in the consensus process, with redundant data will help save the battery capacity of the network and will extend the mission lifetime of the camera network. The cameras with spatially correlated data can be identified in two ways, namely, (i) using camera configuration, i.e., area of overlap between FoV of two cameras and (ii) using data collected by the nodes. The former technique is more helpful when the nodes have been deployed with known FoV whereas the latter is helpful when the camera configuration is not known in advance. We allocate a fixed energy budget for the RPs for this phase, i.e., we assume that each RP can only spend a pre-defined E_{setup} [kWh] to identify spatially correlated nodes in its

neighborhood. This is done in such a way that the overhead of our solution is kept low. We now explain in detail how these two techniques are implemented.

Identifying correlated nodes: Since the FoV of cameras is limited to the area they observe, the information they get is directly related to the directional sensing and configuration of the cameras. Assume a camera's FoV is described by $(P, R, \vec{V}, \alpha)$ as in [103], in which P stands for the location of the camera, R represents the sensing radius, $\vec{V}$ indicates the sensing direction (i.e., the center line of sight of the camera's FoV), and α is the offset angle. Focal length of each camera, its location, and its sensing direction (shown by f, P, and $\vec{V}$, respectively) can be estimated as shown in [104]. A model for the spatial correlation can be derived based on the above parameters as can be seen in [103]. However, this information is not always available or may change over time.

If the camera configuration is not available, then the online data collected by the nodes is used to estimate the nodes in the network that have correlated data. Each DP identifies the RP node with which it will communicate based on metrics such as max received signal strength or residual battery capacity. Next, the RPs have to identify which of the DPs are spatially correlated and for that the DPs send a histogram of their collected online image data; such payload would include {`nodeID`, H_{nodeID}}. We choose a normalized similarity metric $S(i, j)$ that gives the intersection between the histograms of two images [105]. This metric is used to quantify data correlation between different camera nodes as,

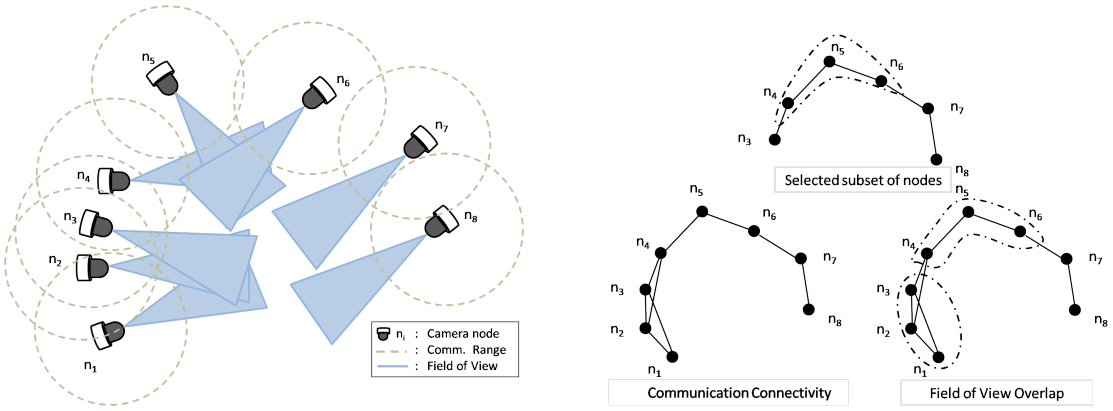
$$S(i,j) = \frac{\sum_{k=0}^{b-1} \min(H_i^k, H_j^k)}{\sum_{k=0}^{b-1} H_i^k} \tag{5.5}$$

where histogram H_i of node i is a b-dimensional vector. The data collected by the cameras are in the RGB space. So to calculate $S(i, j)$ we first calculate the histogram (10 bins) of each color channel separately for both image i and j. Next, we calculate the intersection of histogram of each channel separately for the two images and then take an average across the channels.

5.4.3 Challenges to Selection of Correlated Nodes

We now present the challenges associated with selection of correlated nodes, namely, *critical nodes* and *field of view*.

Critical nodes: A node is defined as a *critical node* if, when removed from the network, it will



(a) (b)

Figure 5.5: (a) Field of View (FoV) and the communications range of the cameras for scenario that was mentioned in Fig. 5.1(c)—as an example, the FoVs' of nodes n_1–n_3 are overlapping, but the FoVs' of n_7–n_8 have zero overlap although they are in the communication range of each other; (b) Connectivity of the cameras, based on their communication range and FoVs' overlap—the subsets of nodes are selected in such a way as to leverage the overlap of the neighboring nodes.

cause any of its neighboring nodes to be disconnected from the rest of the network. For example, in Fig. 5.5(a) n_7 is critical to node n_8 because when node n_7 is removed, n_8 cannot communicate with the rest of the network. We identify the nodes that have only one neighbor as the end nodes (n_e). Any of the end nodes cannot be critical because removing the end nodes will not make the wireless network disconnected.

FoV: We make a note here that the nodes that have overlapping FoVs or are spatially correlated may not necessarily be within the communication range of each other or can communicate with the same RP. For example, in Fig. 5.5(b), we see that nodes n_1–n_3 have overlapping FoVs and are also in the communication range of each other, while nodes n_4–n_6 have overlapping FoVs, but are not in the communication range of each other and only have n_5 as their common communication neighbor; finally, nodes n_7 and n_8 are only in communication neighborhood of each other but do not have overlapping FoVs. In such a scenario, each RP broadcasts the data packets that it has received from its neighboring RP. This will be only done if the RP has not exhausted its energy budget (E_{setup}). In Fig. 5.6, we illustrate the timing diagram for nodes in the network and the corresponding tasks performed by them in order to enable robust in-network image compression.

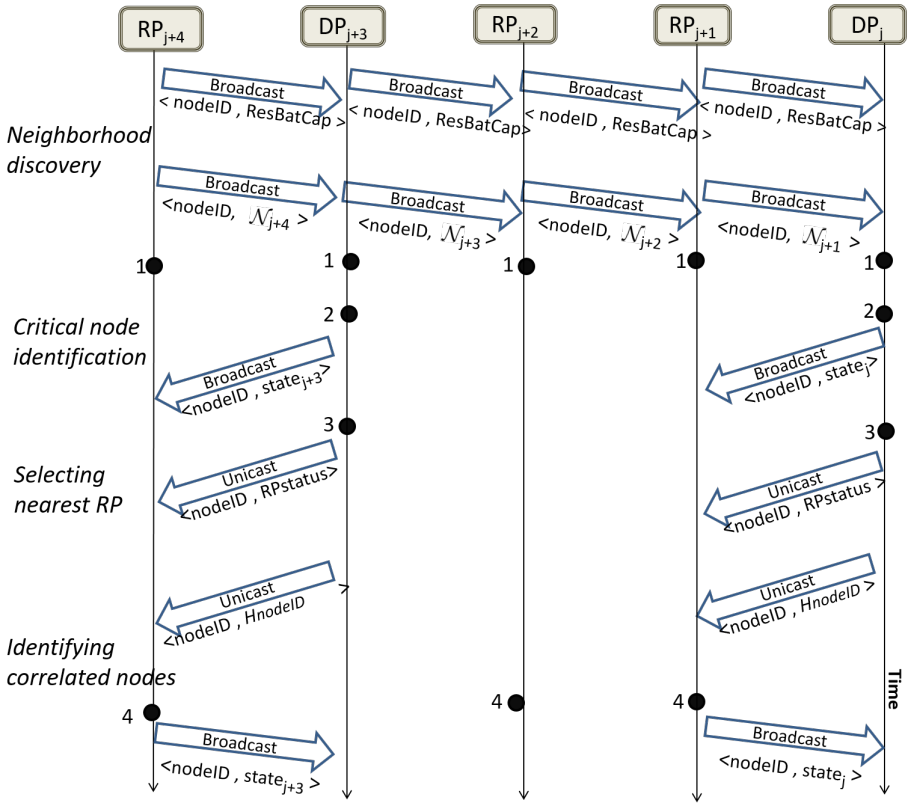


Figure 5.6: Illustration of the proposed protocol to reduce energy consumption of the nodes in the network to execute distributed dictionary learning. Tasks performed locally at each node are shown using black dots. Task 1 identifies neighbor of each node in the network, Task 2 involves critical node selection procedure at the DPs, Task 3 involves selecting the nearest RP of each DP, and Task 4 involves identifying the spatially-correlated nodes.

5.4.4 Selecting a Subset of Nodes

We now present our protocol to intelligently select a subset of nodes in the network on which dictionary learning algorithm will be implemented while also keeping in mind the challenges mentioned above. Only the subset of nodes that are selected run the dictionary learning algorithm to collaboratively learn the dictionary for the network. Since, only a subset of nodes are run it leads to saving the energy of the network and increasing the mission lifetime of the network.

Neighborhood discovery: Neighborhood discovery is a two-step process where each camera node identifies its own neighbors as well as the neighbors of its neighbors. To this end, each node broadcasts a packet with the following information: node ID and residual battery capacity {`nodeID`, `ResBatCap`}. Once each node receives the information from its neighbors, it updates its data structure storing nodeID of its neighbors. Each node also broadcasts its neighbor's nodeID by using the

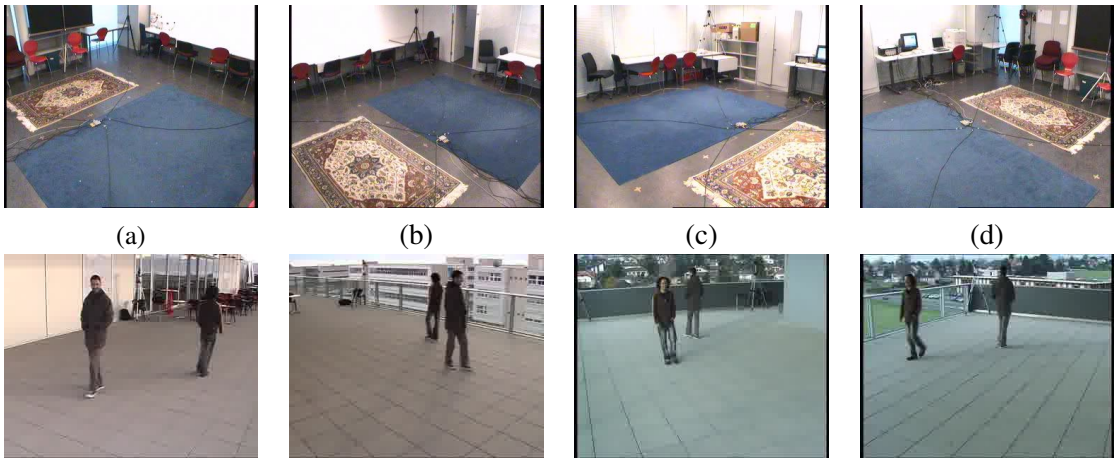
packet payload: {nodeID, $\mathcal{N}_{nodeID}$}. Upon receiving this packet, the neighboring node updates its data structure, which stores information about the neighbors of its neighbors.

Identifying critical nodes: We present three steps that each node can use based on its local information available in order to identify if it is a critical node. For illustration of these steps, we consider two neighboring nodes, n_i and n_j, and assume the following:

1. All RPs are considered critical nodes;
2. If a node has no neighbor other than critical nodes and it is not a n_e, then that node is identified as a critical node;
3. If the neighbors of node n_j are a subset of node n_i's neighbors, then node n_i becomes critical to node n_j.

Various works have been done in this area that use distributed Depth First Search (DFS) and other spanning tree based algorithms to find the critical nodes in the network [106, 107]. These methods require passing multiple messages and increase the network traffic and total consumed energy by the nodes. We on the other hand, use only local knowledge available to the nodes to find the set of critical nodes. The conditions mentioned above may not identify the critical nodes of the network as the global knowledge of the topology of the graph is not available to the nodes. They however, help each node in making a local decision if it is a critical node to its neighboring nodes and on removing the node its neighbors will become disconnected from the network.

Communication protocol: Figure 5.6 illustrates the timing diagram for the set of nodes and the corresponding tasks. Here we consider two DPs that are exchanging their data with the chosen RPs. Broadcast signal is propagated from RPs and the DPs acknowledge the signal with the requested information. Our first goal is to identify nodes whose data collected is highly correlated by utilizing the techniques mentioned earlier. Once each node has identified the node to which it is spatially correlated, they inform each other. One of the node among the pair of spatially correlated node is considered as the member of the subset of nodes selected to execute distributed dictionary learning. The nodes among the group that is spatially correlated are members of the subset. If there are multiple nodes with data correlation higher than a threshold then only one of those is active and other nodes to not participate in the dictionary learning process. If one or more nodes among the



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.7: Example scenarios—(a-d) room and (e-f) terrace—at a particular time slot taken from cameras with varying viewing angles and field of view. These images are part of a multiview dataset [10] and are used in our experiments to study the energy-efficiency performance of distributed dictionary learning.

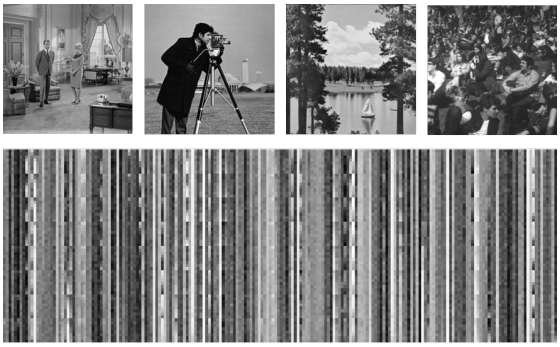


Figure 5.8: Illustration of how the training data Y is collected from the Berkeley Segmentation Dataset for dictionary learning. We collect patches of size 8×8 from an image, which are flattened and placed at random column indices in Y. We repeat this with multiple images to create training data matrix Y.

group of nodes is critical then the critical nodes are part of the subset of nodes that participates in dictionary learning. This is because we have to make sure that the topology in the subset remains connected otherwise some of the nodes will not be able to take part in the consensus process. If none of them is critical, then the node that has the highest number of neighbors within its communication range is in subset. The packets forwarded by RP to its neighboring RP consists of {`RPnodeID`, `DPnodeIDs`, H_i, H_j}.

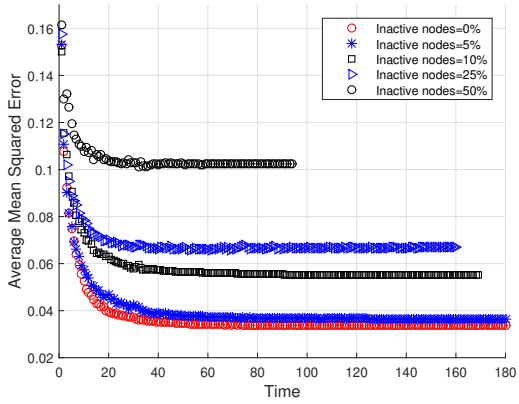


Figure 5.9: Average mean square error of consensus as the number of nodes participating in the consensus process is varied.

5.5 Performance Evaluation

The goal of this section is to present the performance of distributed dictionary learning algorithm in terms of energy consumed and accuracy achieved using our proposed protocol. We present our results on publicly available multiview camera datasets and dataset collected via experiments. We also discuss experimental results obtained on a testbed composed of Raspberry Pi nodes to study the performance of the proposed solution in a real-world setting.

5.5.1 Energy-efficient Dictionary Learning

We now present performance of our proposed solution on a public dataset. We use the Multi-camera Pedestrian Videos dataset [10] by EPFL. We will discuss the different steps required to prepare the data for distributed dictionary learning, cost of dictionary learning, and performance of distributed learning on this dataset and using our proposed solution.

Datasets: The dataset [10] consists of three different scenarios captured by four digital video cameras placed in different locations. Unfortunately the dataset does not provide any information about the camera configuration; hence, we identify nodes that are spatially correlated using online data. The video format is DV PAL, downsampled to 360×288 pixels at 25 fps. The dataset consists of 4 different locations and we show two example sequences from the dataset in Fig. 5.7.

Reducing the data size: To start the dictionary-learning process, each camera node captures images. The matrix Y_i is an $n \times S_i$ matrix, where S_i is the number of training samples. We can

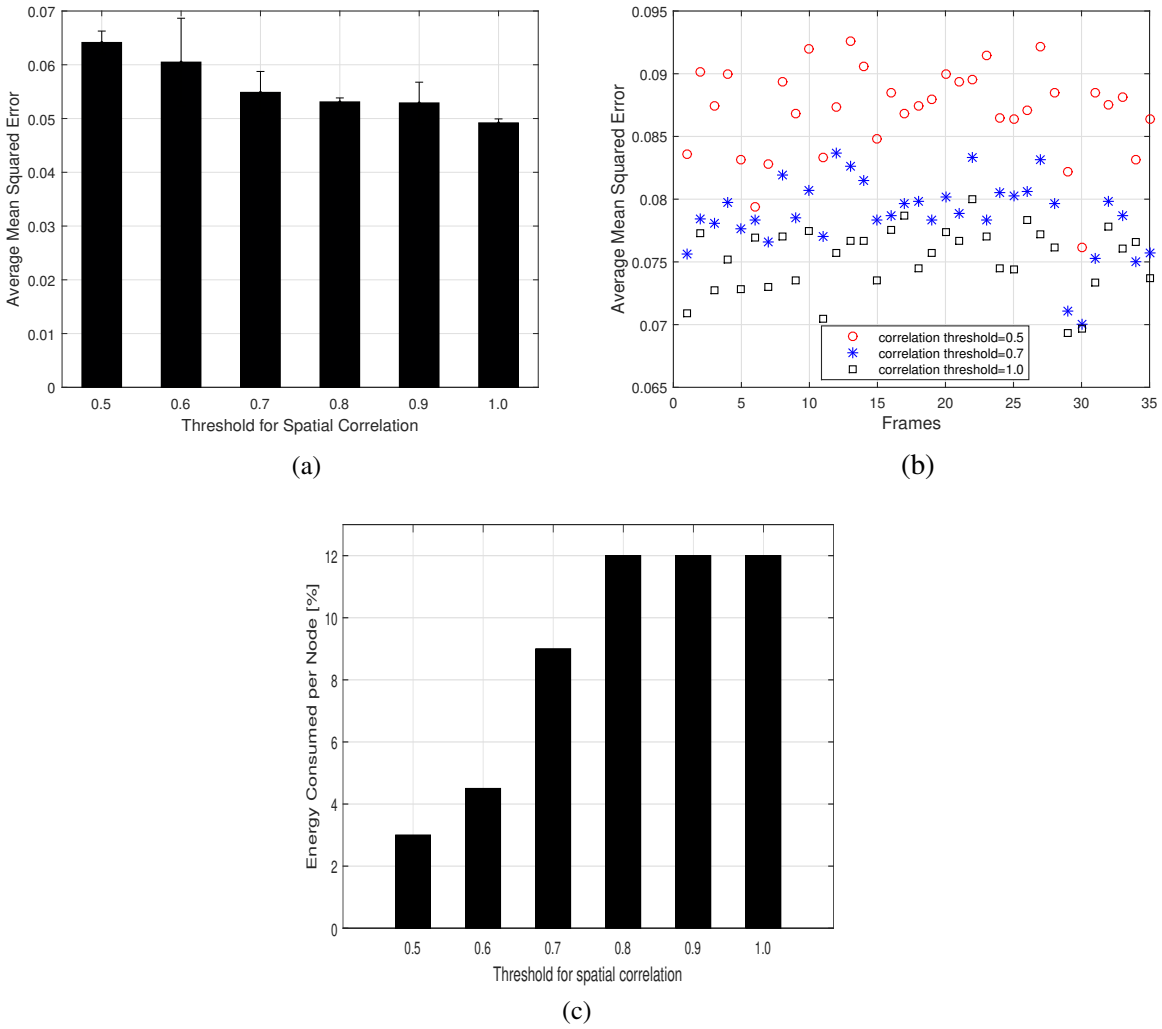


Figure 5.10: (a) Average representation error achieved in distributed dictionary learning when nodes with spatially correlated data (above a threshold) are identified and one of them is selected along with nodes which do not have correlated data with any other node; (b) Average representation error achieved over incoming test frames by using dictionaries learned by using training dataset that has removed data that is spatially correlated with other nodes; (c) Energy savings obtained in the dictionary learning process by removing spatially correlated nodes.

consider each image captured by the camera to be a training sample. However, this would require the cameras to capture a large number of images and also would increase the computational complexity of the problem. For example, each image in the dataset [10] is of size 360×288 pixels, which leads to n being greater than $10,000$, which would increase tremendously the computational complexity of the problem. To overcome this problem, we give an example in Fig. 5.8, where we take 8×8 patches from the image, convert each patch to one-dimensional array, and put it in a column of data matrix Y. We combine multiple such samples from different images taken by the camera node and

place them at random column indices. This method significantly reduces the computational cost at each node. We also pre-process each image by first converting the color images to gray scale, subtracting the mean from the images, and normalizing the images.

Inactive nodes: We consider a scenario where a certain percentage of randomly selected nodes from the network are not allowed to participate in the consensus process. As mentioned earlier, we assume there are 20 nodes in the network and we fix the number of times a node can broadcast its data to be 20. We again plot average mean square error, averaged over the number of nodes. We observe in Fig. 5.9 that, as the number of nodes participating in the consensus process reduces, the mean squared error of the network increases. We also see that the consensus process ends earlier as fewer nodes are broadcasting in comparison to when all the nodes are participating in consensus.

Spatial correlation: In this scenario, since no information is given about the camera configuration and the nodes are located in a small room, we assume all the nodes to be fully connected. We vary the threshold of correlation coefficient and identify the nodes that have correlated data. We divide this dataset into training and testing frames, and use the training images to estimate the dictionary. To this end, we first identify the nodes whose data will be part of the training process. We use the intersection of histogram metric to quantify the spatial correlation between data at different nodes. Each node creates a set that includes itself and the other nodes with which it has correlated data above a certain threshold. We randomly select one of the nodes from this set. Nodes that do not have data correlated with any other node are also included in the dictionary-learning process.

In Fig. 5.10(a), we plot the average representation error of distributed dictionary learning; as the threshold for spatial correlation changes from 1.0 to 0.7, we get an accuracy loss of 3.7% and the energy consumption per node reduces from 12% to 9% in Fig. 5.10(c). In Fig. 5.10(b) we note that the dictionary created for lower threshold coefficient performs worse and for a particular threshold coefficient the performance remains constant over the next 35 incoming frames for which the tests are performed.

5.5.2 Energy vs Accuracy Tradeoff

The existing work in the development of dictionary learning algorithms have focused on maximizing the performance of the algorithm in terms of accuracy achieved by the algorithm. Our work on the other hand, as explained in Chapter 4, focuses on identifying “good-enough" parameters, i.e.,

parameters that give acceptable accuracy in each frame of the video with significant savings in time and energy as these algorithms have to be run on resource-constrained devices. To this end, we study the accuracy energy tradeoff of various parameters in the dictionary learning algorithm and use it to select the parameters that can be used in the implementation of dictionary learning algorithm on a resource-constrained device for a real-world application.

To this end, we study the impact of the following three parameters, namely, dictionary size or number of atoms in the dictionary (K), number training samples (L), and dictionary learning iterations (t_d) on the performance of these algorithms. We show the performance of dictionary learning algorithm using the Mean Squared Error (MSE) metric which is the pixel wise mean squared difference between image reconstructed using estimated dictionary and sparse coefficients and the original image. To study the performance of object detection algorithm, we use Single-shot detector [108] convolution neural network and use AUC metric to quantify its performance. AUC is the area under the Receiver Operating Characteristic (ROC) curve. Since our goal is to compress image and process them at the server for detecting authorized or unauthorized objects of interests, we study the object detection accuracy achieved at the server. We also calculate the energy consumed by the Raspberry Pi device in kWh to quantify the cost of using a particular parameter of the algorithm. To measure the power we use a power monitor, PortaPower 3-20V Dual USB Power Monitor that displays current, voltage, and amount of current received by devices. We now present the accuracy energy tradeoff of various parameters of the distributed dictionary learning algorithm.

Number of atoms: We now see the performance as the number of dictionary atoms (K) are varied. We observe that as we increase the number of dictionary atoms, the performance of the dictionary learning algorithm increases. We see that in Fig. 5.11(a) the left y-axis shows the MSE of dictionary learning algorithm and right y-axis shows the object detection accuracy. MSE reduces by 1.33% and the object detection accuracy increases by 2% as the number of dictionary atoms are increased from 100 to 300. At the same time there is significant increase in energy costs as shown in Fig. 5.11(b). The energy cost increases by 45% as the number of dictionary atoms are increased from 100 to 300. This increase in energy costs can be attributed to the increase in communication cost with the increase in number of dictionary atoms. Nodes in the network communicate with each other to update each atom of the dictionary sequentially. Now, as the number of dictionary learning atoms increase the communication cost in the network also increases.

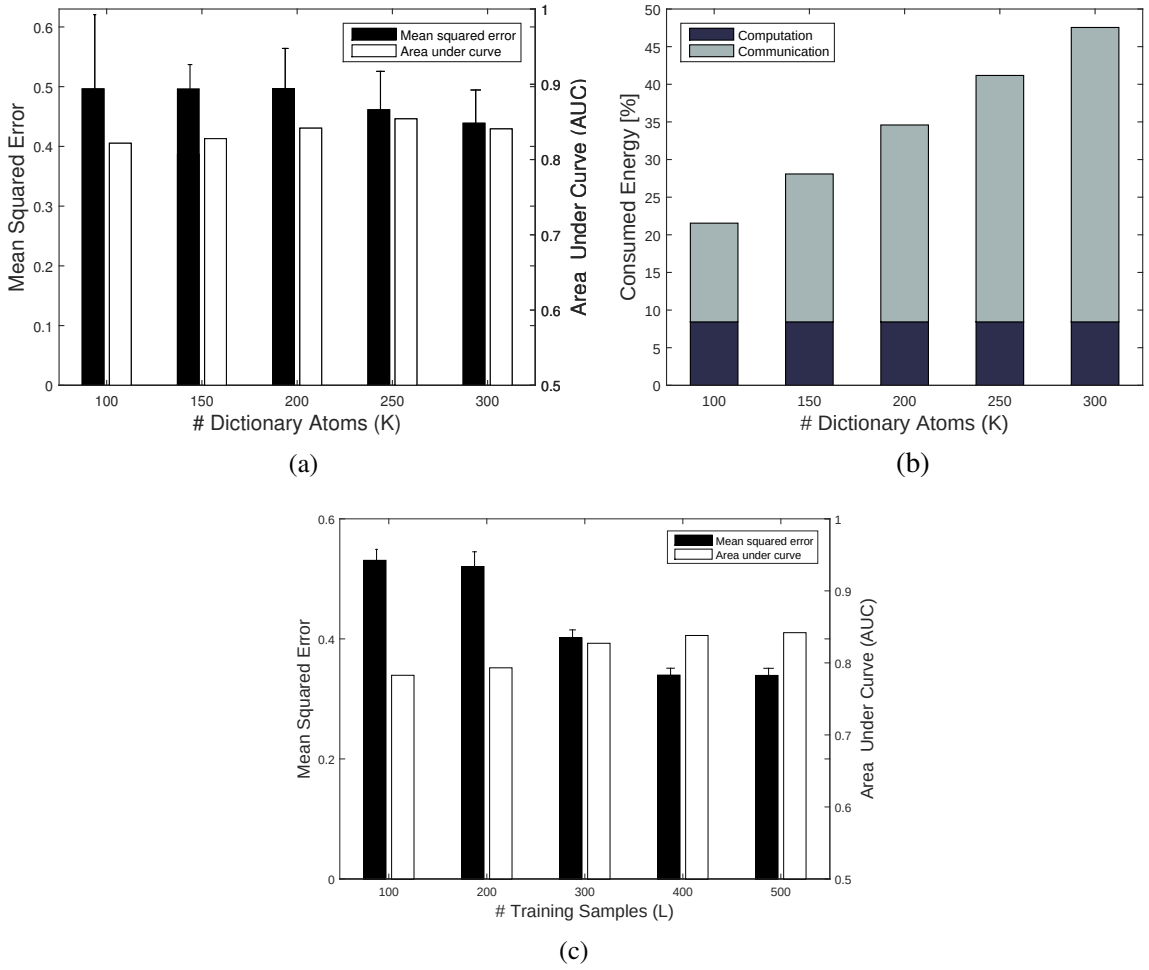


Figure 5.11: Accuracy-energy tradeoff of different parameters of distributed learning algorithm. (a) Accuracy obtained by varying number of atoms (K) in a dictionary. Left y-axis is mean squared error and Right y-axis is the object detection accuracy in terms of area under curve of ROC curve; (b) Energy consumed by the dictionary learning algorithm by varying number of atoms in a dictionary; (c) Accuracy obtained by varying number of training samples in the data (L). Left y-axis is mean squared error and Right y-axis is the object detection accuracy in terms of Area Under Curve (AUC) of the Receiver Operative Characteristic (ROC) curve.

Number of training samples: We now see the performance as the number of training samples (L) are varied. We observe that as we increase the number of training samples , the performance of the dictionary algorithm increases. In Fig. 5.11(c) We see that the MSE reduces by 50% and the object detection accuracy increases by 10% as the number of training samples are increased form 100 to 400. The object detection rises steeply and then saturates. However, since increasing the training samples only impacts the sparse coding task which is executed locally at the nodes (Task 1 in Fig. 5.2) it does not significantly increase the energy consumed as can be seen in Fig. 5.12(a).

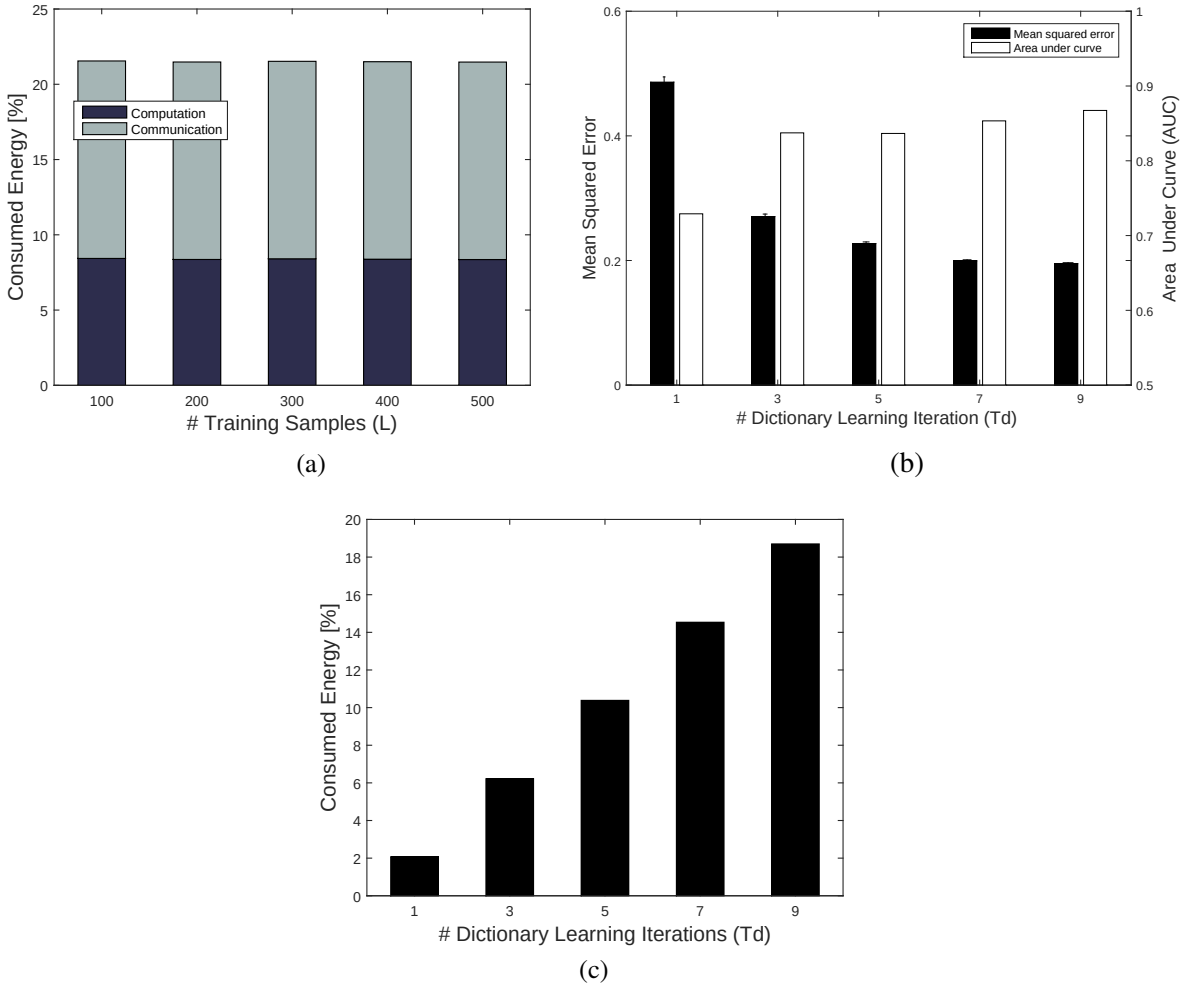


Figure 5.12: (a) Energy consumed by the dictionary learning algorithm by varying number of training samples (L) in the data; (b) Accuracy obtained by varying the number of dictionary learning iterations (t_d). Left y-axis is mean squared error and Right y-axis is the object detection accuracy in terms of area under curve of ROC curve; (c) Energy consumed by the dictionary learning algorithm by varying the number of dictionary learning iterations.

Number of dictionary learning iterations: We now see the performance as the number of dictionary iterations (t_d) are varied. In Fig. 5.12(b) we see that as we increase the number of dictionary learning iteration increases from 1 to 9 the MSE decreases by 62% and the object detection accuracy increases by 16%. However, this comes at the cost of significant increase in energy as shown in Fig. 5.12(c). The increase in cost can be quantified in Fig. 5.12(c) as the cost increases by 50% as the number of dictionary learning iterations are increased from 1 to 9. This increase in energy costs is due to increase in communication costs with increase in number of iterations.

Parameter selection at run-time: Among the parameters we have discussed above, the size

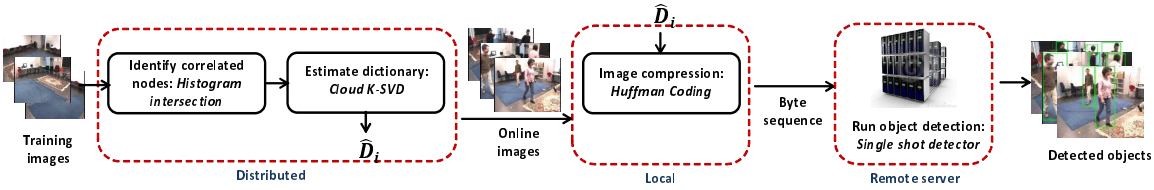


Figure 5.13: Our envisioned pipeline for image compression using dictionary learned collaboratively and the compressed images are send to the cloud for storage or further processing. The figure shows different tasks pf this pipleine and the location where they are implemented.

of dictionary, i.e, number of atoms in the dictionary, impacts the cost of image compression at run-time. This is because the sparse coefficients are estimated using estimated dictionary and input data and these sparse coefficients are later used as input to the image compression algorithm. The size of dictionary to use can be calculated at run-time depending on the resources available with the device. Higher the energy available with the nodes, higher size of dictionary can be used which will result in lower accuracy loss in object detection done on compressed images (as shown in Fig. 5.11(a)). As the energy of the nodes decreases over time the nodes can also switch to lower dictionary sizes. This selection of dictionary size can be dynamic and the decision can be based on the current energy of each node. To this end, our proposed decision framework in Chapter 4 can be trained to select the dictionary size at run-time.

5.5.3 Experimental Testbed

We now give details for our experimental setup used to test the performance of distributed dictionary learning algorithm in terms of accuracy and energy consumption of the nodes. We also study the benefits obtained by our proposed solution.

We now present our envisioned scenario in a distributed surveillance systems as shown in Fig. 5.1(c). We assume that a set of resource-constrained nodes with camera as the sensor and limited battery capacity are used. Our scenario as shown in Fig 5.13 is that a network of local nodes perform distributed dictionary learning. The estimated dictionary is the used for compression of incoming images. The compressed images are then send to the cloud for storage or person detection or detection of unauthorized objects (such as detection guns and alarming the building security).

Setup: Our testbed consists of 10 Raspberry Pi. We use Raspberry Pi nodes, which are small single-board computers with the following characteristics—Model 3B, 1.2GHz 64-bit quad-core

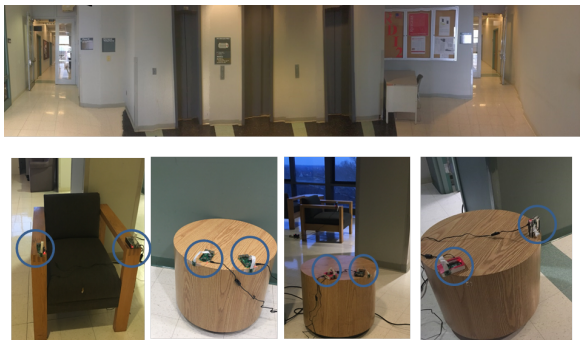


Figure 5.14: Testbed set up in the ECE Department at Rutgers University. The goal of the testbed is to simulate a surveillance system in which images that are sent to the server for further processing are compressed using distributed dictionary learning. The testbed of 10 raspberry Pi nodes and camera covers a region in a hallway with thee elevators and two doors.

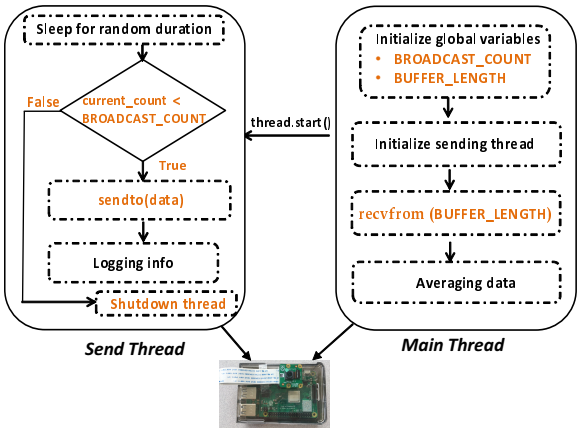


Figure 5.15: Communication workflow implemented at the Raspberry Pi nodes our testbed. The communication workflow is used to enable consensus in the network. Two separate threads for receiving and transmitting data at each node.

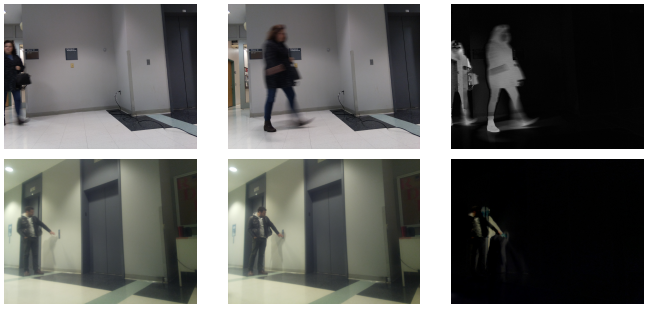
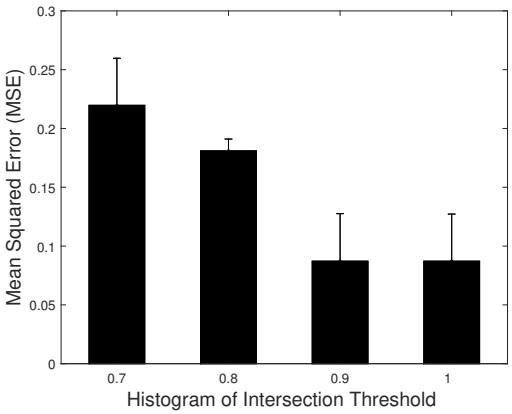
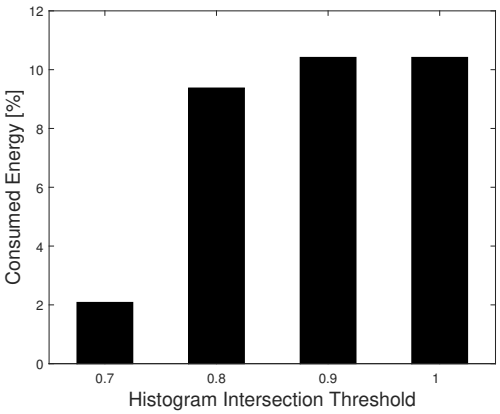


Figure 5.16: Pre-processing using frame differencing and thresholding of incoming images at the local nodes is done to identify if there is a significant change in the consecutive images. If the number of non-zero pixels in the difference image is above a pre-determined threshold value then the image is compressed and sent to the cloud. Frame differencing is a low-complexity technique to identify that we are not sending frames with redundant information. (Top) High movement; (Below) Low movement



(a)

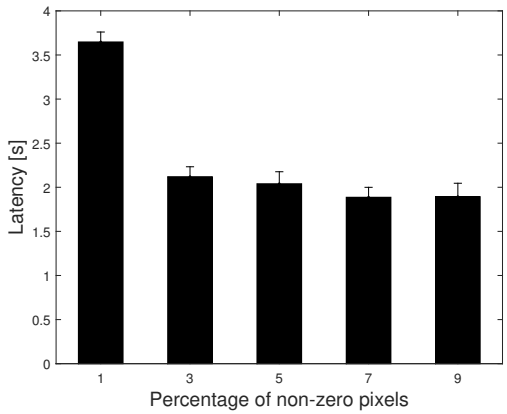


(b)

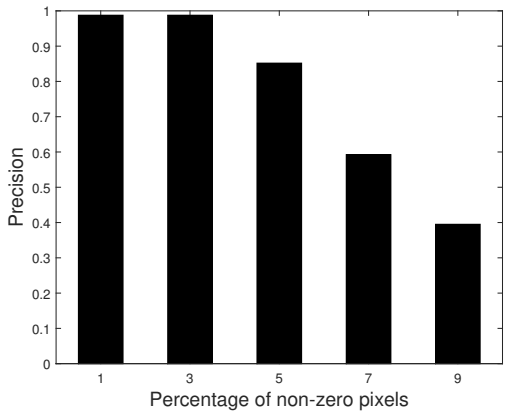
Figure 5.17: (a) Mean squared error (MSE) performance of our testbed; (b) Energy consumed by the nodes when we vary the threshold for histogram intersection coefficient. Only nodes that have value above value above this participate in the dictionary learning algorithm.

processor, 1 GB RAM and with Wireless LAN and Bluetooth capabilities. The camera used in our testbed is Sony IMX219 8-megapixel sensor 1080P, along with Picamera Python library. To measure the power we use a power monitor, PortaPower 3-20V Dual USB Power Monitor that displays current, voltage, and amount of current received by devices. Figure 5.14 indicates our setup for a surveillance application.

Communication workflow: Figure 5.15 shows the workflow that is used by the nodes to communicate with other nodes in the network to achieve consensus. Each node maintains two threads, a main thread which is responsible for receiving data and another thread which is responsible for broadcasting data. Each node has a pre-defined value of number of times it will broadcast its data values. Each node waits a predefined amount of time before it transmits the data. Each time a node



(a)

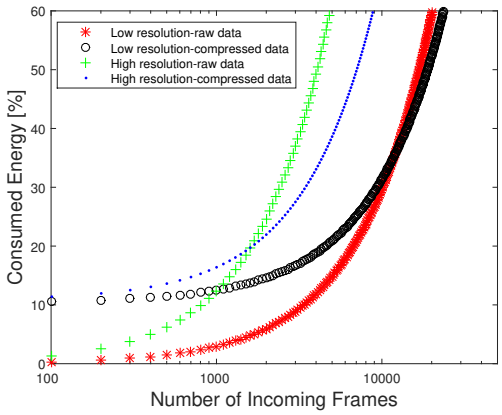


(b)

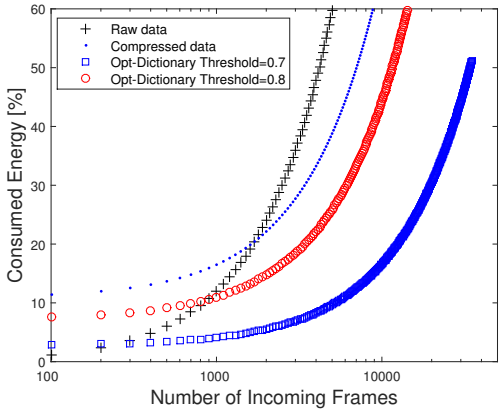
Figure 5.18: (a) Latency (time taken from the instant image is captured by the camera till the images reaches the server) in our experimental testbed when estimated dictionary is used to compress images; (b) Precision of the object detection algorithm for various percentage of non-zero pixels in the difference image of consecutive images.

receives data it takes an average of its current data and received data and update it as its current data value. Once a node broadcasts its value for a fixed number of times the transmit thread is shutdown.

Image differencing: We want a camera in the network to send a frame to the server for object detection only when there is a significant change from the last frame that was sent by the camera node. This will help in reducing the energy required to compress and transmit images which have redundant information. To identify which image to send, we first take the difference of the consecutive images at the camera node and use Otsu thresholding [109] to calculate a global threshold. See Fig. 5.16 for visualized results. All the pixels in the difference image which are below the threshold are given a zero pixel value. In the binary image if the percentage of pixels that are above the threshold (selected as 0.02 in experiments) are greater than 10% then we compress those frames



(a)



(b)

Figure 5.19: Comparison of energy costs consumed by the network of raspberry nodes in the network (a) Raw images are send to the cloud vs Compressed images are send to the cloud after compressing using the estimated dictionary. We consider here two image resolutions high resolution (1920, 1080) and low resolution (366, 288); (b) Images are compressed using optimized dictionary learning at various thresholds of histogram intersection.

and send them to the remote server.

Communication with remote server: We initialize an Amazon EC2 compute instance (t2.micro instance) where the object detection algorithm is run on the compressed image. We use the public DNS of the created instance to securely transmit data from the Raspberry Pi to the server. Once the images have been received at the server the object detection algorithm is executed. To this end, we use single-shot detector [108] convolution neural network for object detection.

Performance in the testbed: We study the performance of our testbed in terms of mean squared performance, energy costs, and latency incurred by processing images in our testbed. We vary the threshold for histogram intersection from 1.0 to 0.7 and see that in Fig. 5.17 the cost of dictionary

learning falls by 75% and the MSE increases by 35%. We also study the latency as the percentage of non-zero pixels vary in Fig. 5.18. Latency is the time taken to compress the image and send to the server. We see that as we increase the percentage of non-zero pixels, the latency decreases and the precision also decreases. This is because fewer images are sent to the server. Since the object detection algorithm misses detecting the object in the frames that were not sent, the precision decreases.

Energy costs: We now discuss the long term energy costs of running image compression using estimated dictionary. In Fig. 5.19 (a) we see the impact of resolution on energy cost of compressing image using dictionary and sending to the cloud. We study two resolutions, high resolution Raspberry Pi with 1920×1080 and low resolution EPFL with 366×288. For the high resolution images we see that after sending 2000 compressed images we see that sending raw data leads to more energy consumption of the nodes in comparison to compression using dictionary learning whereas for low resolution it is after $100,000$ images. In Fig. 5.19 (b) we see that we use dictionary created using our proposed algorithm at different threshold of histogram intersection. We see that sending raw data leads to more energy consumption of the nodes in comparison to compression using dictionary learning much faster than when we used only all nodes in the network. This is because the energy cost of initially estimating the dictionary is much lower when we use our proposed method and hence we are able to gain benefits much faster.

5.6 Summary

In this chapter we presented an energy-efficient technique to reduce the cost of dictionary learning algorithm in a distributed camera network to support real-time in-network image compression. We designed a protocol that identifies spatially-correlated nodes using local knowledge of the network available to the camera nodes. To this end, we utilized low-computational-complexity metrics to quantify the correlation of data among camera nodes. We also presented via simulations the performance in terms of accuracy and energy consumed by the nodes to run consensus and dictionary-learning algorithms. The performance of the proposed approach was validated through extensive simulations using a network simulator and public datasets as well as via real-world experiments on a small testbed of Raspberry Pi nodes.

Chapter 6

Conclusion and Future Directions

This chapter summarizes the main contributions of this dissertation and discusses future research directions that are worth investigation and can leverage the frameworks proposed in this dissertation.

6.1 Summary of Dissertation Contributions

This dissertation describes novel solutions to enable real-time computation-intensive mobile applications in resource-limited and uncertain environments. To this end, this dissertation leverages the paradigm of approximate computing to exploit the untapped potential of mobile distributed computing and to enable real-time pervasive applications in a resource-constrained. Firstly, an accuracy and resource aware framework is introduced that determines offline the approximable tasks in an application via powerful workflow representation and data approximation schemes. Also, a lightweight, online algorithm to select in real time the approximable tasks to be executed is proposed. The effectiveness of the proposed approach was validated through extensive simulations and testbed experiments by taking as motivating example three different computer vision algorithms. Secondly, a new method to handle the problem of resource constraints in mobile robots was presented. A solution based on MDPs to select the parameters of computationally-intensive computer vision algorithms, and bring benefits in terms of time and energy for long-term object-detection in videos. The benefit of our formulation in a robot navigation setting with experiments on a public dataset and a new dedicated dataset via experiments. Finally, an energy-efficient technique to reduce the cost of dictionary learning in a distributed camera network to support real-time in-network image compression was proposed. A protocol was designed that identifies spatially-correlated nodes using local knowledge of the network available to the camera nodes. To this end, we utilized low-computational-complexity metrics to quantify the correlation of data among camera nodes. Also the performance of the solution was validated using experimental and public datasets.

6.2 Future Directions

The avenues for further research in the following areas have been identified:

Applying approximation to concurrent applications: While so far we have studied isolated applications running on a mobile device to which approximation can be applied, our approach can be extended to multiple applications running concurrently on the mobile device. For example, approximation can be applied to simultaneous background sensing and foreground applications. Furthermore, while so far we have focused on approximation via accuracy-energy tradeoff, we can also consider other resources available on the mobile device such as network bandwidth, memory, and CPU cycles.

Light-weight object detection: To estimate good-enough parameters so far we have looked at estimating parameters of the input data via MDP. As a next step we can extract certain features from the image which quantify the clutter, illumination, uniformity etc. This feature will then be used to classify the parameters of the object detection algorithm.

Dynamic dictionary learning: So far we have looked at estimating dictionary for a fixed environment and as future work, it can be identified when the dictionary-learning algorithm should be re-triggered as the environment in which camera nodes are situated changes. Our goal will be to obtain high-accuracy results from the algorithm while at the same time not incur high energy consumption due to frequent re-triggering of the algorithm. Another, interesting question is how the current estimation of dictionary be used to speed up or improve the accuracy of the dictionary in a new environment. The network can collaborate to learn dictionary of various sizes (number of atoms). At run-time a decision can be made to select the size of dictionary based on the energy available at nodes. Higher the energy available with the nodes, higher size of the dictionary can be used which will result in lower accuracy loss in object detection done on compressed images. As the energy of the nodes decreases over time the nodes can also switch to lower dictionary sizes. This selection of dictionary size can be dynamic and the decision can be based on the current energy of each node. To this end, our proposed decision framework in Chapter 4 can be trained to select the dictionary size at run-time.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[2] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Proc. of European Conference on Computer Vision (ECCV)*, (Zurich, Switzerland), 2014.

[3] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, (Dallas, TX, USA), 2013.

[4] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognition-How far are we from the solution?," in *Proc. of IEEE International Conference on Neural Networks (IJCNN)*, (Dallas, TX, USA), 2013.

[5] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in Large Margin Classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[6] B. Alexe, T. Deselaers, and V. Ferrari, "Measuring the objectness of image windows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2189–2202, 2012.

[7] "NASA's Mars Science Laboratory." `https://mars.nasa.gov/msl`.

[8] "Woods Hole Oceanographic Institution (WHOI) Autonomous Underwater Vehicles (AUVs)." `http://www.whoi.edu/main/auvs`.

[9] "Home Design 3D." `https://en.homedesign3d.net/`.

[10] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple Object Tracking using K-Shortest Paths Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[11] "Augmented Reality." http://cacm.acm.org/magazines/2014/9/177938-augmented-reality/fulltext.

[12] P. Lee, D. Stewart, and C. Calugar-Pop, "Technology, Media & Telecommunications Predictions," tech. rep., Technical report, Deloitte Global Services Limited, 2011.

[13] C. Julien and G.-A. Nazri, *Solid state batteries: materials design and optimization*, vol. 271. Springer Science & Business Media, 2013.

[14] H. Kim, K.-Y. Park, J. Hong, and K. Kang, "All-graphene-battery: bridging the gap between supercapacitors and lithium ion batteries," *Scientific reports*, vol. 4, p. 5278, 2014.

[15] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proc. of the Intl. Conf. on Mobile Systems, Applications, and Services (MobiSys)*, (San Francisco, CA), June 2010.

[16] P. Pandey and D. Pompili, "$MobiDiC$: Exploiting the Untapped Potential of Mobile Distributed Computing via Approximation," in *Proc. of IEEE Intl. Conference on Pervasive Computing and Communications (PerCom)*, (Sydney, Australia), March 2016.

[17] P. Pandey and D. Pompili, "Exploiting the Untapped Potential of Mobile Distributed Computing via Approximation," *Pervasive and Mobile Computing*, vol. 38, no. 2, pp. 381–395, 2017.

[18] P. Pandey, Q. He, D. Pompili, and R. Tron, "Light-Weight Object Detection and Decision Making via Approximate Computing in Resource-Constrained Mobile Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain), Oct 2018.

[19] P. Pandey and D. Pompili, "Handling limited resources in mobile computing via closed-loop approximate computations," *accepted for publication in IEEE Pervasive Computing Magazine*, 2019.

[20] P. Pandey, M. Rahmati, D. Pompili, and W. U. Bajwa, "Robust distributed dictionary learning for in-network image compression," in *IEEE International Conference on Autonomic Computing (ICAC)*, (Trento, Italy), Sep. 2018.

[21] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, 2001.

[22] "Pricing at Amazon EC2." `https://aws.amazon.com/ec2/pricing/`.

[23] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications," in *Proc. of the ACM/IFIP/USENIX Intl. Conf. on Middleware (Middleware)*, (Urbanna Champaign, IL), Nov. 2009.

[24] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution between Mobile Device and Cloud," in *Proc. of The European Professional Society on Computer Systems (EuroSys)*, (Salzburg, Austria), Apr. 2011.

[25] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling Interactive Perception Applications on Mobile Devices," in *Proc. of Intl. Conference on Mobile Systems, Applications, and Services (MobiSys)*, (Bethesda, MD), June 2011.

[26] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: Code Offload by Migrating Execution Transparently," in *Proc. of the USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, (Hollywood, CA), Oct. 2012.

[27] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, Oct.-Dec. 2009.

[28] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*, pp. 59–79, Springer, 2010.

[29] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud," in *Proc. of Conference on Computer Systems*, (Salzburg, Austria), April 2011.

[30] M. D. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *Proc. of International Conference on Pervasive Computing and Communications (PerCom)*, pp. 217–226, IEEE, 2010.

[31] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili, "Uncertainty-aware autonomic resource provisioning for mobile cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 8, pp. 2363–2372, 2015.

[32] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices," in *Proc. of International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, (Hilton Head Island, SC), June 2012.

[33] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing Performance vs. Accuracy Trade-offs with Loop Perforation," in *Proc. of ACM SIGSOFT Symposium*, (Szeged, Hungary), Sept 2011.

[34] J. Flinn, D. Narayanan, and M. Satyanarayanan, "Self-tuned remote execution for pervasive computing," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pp. 61–66, IEEE, 2001.

[35] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 273–286, ACM, 2003.

[36] "International Telecommunication Union." https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf.

[37] "Verizon-Travel Pass." http://money.cnn.com/2015/11/12/technology/verizon-travelpass/.

[38] Z. Dong, L. Kong, P. Cheng, L. He, Y. Gu, L. Fang, T. Zhu, and C. Liu, "REPC: Reliable and efficient participatory computing for mobile devices," in *Proc. of International Conference on Sensing, Communication, and Networking (SECON)*, pp. 257–265, IEEE, 2014.

[39] A. Mtibaa, K. Harras, A. Fahim, *et al.*, "Towards Computational Offloading in Mobile Device Clouds," in *Proc. of International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 1, pp. 331–338, IEEE, 2013.

[40] "AllJoyn." https://allseenalliance.org/.

[41] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[42] W. Baek and T. M. Chilimbi, "Green: A Framework for Supporting Energy-conscious Programming using Controlled Approximation," *ACM Sigplan Notices*, vol. 45, no. 6, pp. 198–209, 2010.

[43] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger, "Eon: A Language and Runtime System for Perpetual Systems," in *Proc. of Iternational Conference on Embedded Networked Sensor Systems (SenSys)*, (Sydney, Australia), Nov 2007.

[44] J. W. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao, *Algorithms for Scheduling Imprecise Computations*. Springer, 1991.

[45] D. Narayanan and M. Satyanarayanan, "Predictive Resource Management for Wearable Computing," in *Proc. of International Conference on Mobile systems, applications and services*, pp. 113–128, ACM, 2003.

[46] L. J. Van Vliet, I. T. Young, and P. W. Verbeek, "Recursive Gaussian Derivative Filters," in *Proc. of Intl. Conference on Pattern Recognition.*, vol. 1, pp. 509–514, 1998.

[47] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate Signal Processing," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 15, no. 1-2, pp. 177–200, 1997.

[48] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *et al.*, *Discrete-time signal processing*, vol. 2. Prentice hall Englewood Cliffs, NJ, 1989.

[49] "IBM ILOG CPLEX Optimizer." `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`.

[50] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[51] D. G. Lowe, "Distinctive Image Features from Scale-invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[52] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[53] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *Proc. of the IEEE International Conference on Hardware/software Codesign and System Synthesis*, pp. 105–114, ACM, 2010.

[54] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," in *Proc. of International Conference on Computer Vision*, vol. 2, pp. 416–423, July 2001.

[55] "INRIA Person Dataset." http://pascal.inrialpes.fr/data/human/.

[56] J. Gong and A. Maher, "Use of Mobile Lidar Data to Assess Hurricane Damage and Visualize Community Vulnerability," *Journal of the Transportation Research Board*, vol. 2459, no. 1, pp. 119–126, 2014.

[57] A. Paolillo, A. Faragasso, G. Oriolo, and M. Vendittelli, "Vision-based maze navigation for humanoid robots," *Autonomous Robots*, vol. 41, no. 2, pp. 293–309, 2017.

[58] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation sharing in distributed robotic systems: A case study on SLAM," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 410–422, 2015.

[59] G. Mohanarajah, V. Usenko, M. Singh, R. D'Andrea, and M. Waibel, "Cloud-based collaborative 3D mapping in real-time with low-cost robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 423–431, 2015.

[60] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The Roboearth Cloud Engine," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, (Karlsruhe, Germany), May 2013.

[61] O. Zweigle, R. van de Molengraft, R. d'Andrea, and K. Häussermann, "RoboEarth: Connecting Robots Worldwide," in *Proc. of ACM International Conference on Interaction Sciences: Information Technology, Culture and Human*, (Seoul, Korea), 2009.

[62] L. Paletta, G. Fritz, and C. Seifert, "Q-learning of sequential attention for visual object recognition from informative local descriptors," in *Proceedings of the 22nd international conference on Machine learning*, pp. 649–656, ACM, 2005.

[63] S. Karayev, M. Fritz, and T. Darrell, "Anytime recognition of objects and scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 572–579, 2014.

[64] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *IEEE International Conference on Computer Vision*, pp. 4705–4713, 2015.

[65] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *European Conference on Computer Vision*, pp. 201–214, Springer, 2012.

[66] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[67] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile Recommender Systems in Tourism," *Journal of Network and Computer Applications*, vol. 39, pp. 319–333, 2014.

[68] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, Real-time Object Recognition on Mobile Devices," in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, (Seoul, South Korea), ACM, 2015.

[69] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[70] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin, "Feature Congestion: A Measure of Display Clutter," in *Proc. of the ACM SIGCHI conference on Human Factors in Computing Systems*, (Portland, Oregon, USA), 2005.

[71] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[72] C. Tomasi and T. Kanade, "Detection and tracking of point features," *International Journal of Computer Vision*, vol. 9, no. 3, pp. 137–154, 1991.

[73] P. Pandey and Q. He, "BU-RU Dataset." https://github.com/parul1985/Experimental-Data-at-BU, 2017.

[74] P. Pandey, Q. He, D. Pompili, and R. Tron, "Video accompanying paper Light-Weight Object Detection and Decision Making via Approximate Computing in Resource-Constrained Mobile Robots accepted at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)." https://youtu.be/7sLntOoQ6oo, 2018.

[75] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1565–1575, 2008.

[76] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof, and B. Strobl, "Autonomous multicamera tracking on embedded smart cameras," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 092827, 2007.

[77] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[78] Q. Zhang and B. Li, "Discriminative K-SVD for dictionary learning in face recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2691–2698, IEEE, 2010.

[79] I. Horev, O. Bryt, and R. Rubinstein, "Adaptive image compression using sparse dictionaries," in *Proceedings of the 19th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 592–595, IEEE, 2012.

[80] J.-W. Kang, C.-C. J. Kuo, R. Cohen, and A. Vetro, "Efficient dictionary based video coding with reduced side information," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 109–112, IEEE, 2011.

[81] J. Liang, M. Zhang, X. Zeng, and G. Yu, "Distributed dictionary learning for sparse representation in sensor networks," *IEEE Transactions on Image Processing*, vol. 23, no. 6, pp. 2528–2541, 2014.

[82] H. Raja and W. U. Bajwa, "Cloud K-SVD: A collaborative dictionary learning algorithm for big, distributed data," *IEEE Transactions on Signal Processing*, vol. 64, no. 1, pp. 173–188, 2016.

[83] Z. Shakeri, H. Raja, and W. U. Bajwa, "Dictionary learning based nonlinear classifier training from distributed data," in *Proc. 2nd IEEE Global Conf. Signal and Information Processing (GlobalSIP'14), Symposium on Network Theory*, (Atlanta, GA), pp. 759–763, Dec. 2014.

[84] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[85] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: Computation offloading as a service for mobile devices," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pp. 287–296, ACM, 2014.

[86] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. J. Giuli, and X. Gu, "Towards a distributed platform for resource-constrained devices," in *Proceedings of the International Conference on Distributed Computing Systems*, pp. 43–51, IEEE, 2002.

[87] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the 13th ACM international symposium on Mobile Ad Hoc Networking and Computing*, pp. 145–154, ACM, 2012.

[88] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner, "Socio-economic vision graph generation and handover in distributed smart camera networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 2, p. 20, 2014.

[89] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 96, 2014.

[90] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[91] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, "Broadcast gossip algorithms for consensus," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2748–2761, 2009.

[92] A. D. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic gossip: Efficient averaging for sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 3, pp. 1205–1216, 2008.

[93] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1," *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[94] K. Skretting and K. Engan, "Image compression using learned dictionaries by RLS-DLA and compared with K-SVD," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1517–1520, IEEE, 2011.

[95] O. Bryt and M. Elad, "Compression of facial images using the k-svd algorithm," *Journal of Visual Communication and Image Representation*, vol. 19, no. 4, pp. 270–282, 2008.

[96] N. Alon and V. D. Milman, "$\lambda 1$, isoperimetric inequalities for graphs, and superconcentrators," *Journal of Combinatorial Theory, Series B*, vol. 38, no. 1, pp. 73–88, 1985.

[97] K. Skretting, "Image Compression Tools for Matlab."

[98] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer Science & Business Media, 2011.

[99] G. Bianchi, "Performance analysis of the ieee 802.11 distributed coordination function," *IEEE Journal on selected areas in communications*, vol. 18, no. 3, pp. 535–547, 2000.

[100] M. E. Yildiz, R. Pagliari, A. Ozdaglar, and A. Scaglione, "Voting models in random networks," in *2010 Information Theory and Applications Workshop (ITA)*, pp. 1–7, IEEE, 2010.

[101] F. Yavuz, J. Zhao, O. Yağan, and V. Gligor, "Toward k-connectivity of the random graph induced by a pairwise key predistribution scheme with unreliable links," *IEEE Transactions on Information Theory*, vol. 61, no. 11, pp. 6251–6271, 2015.

[102] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete mathematics*, vol. 86, no. 1-3, pp. 165–177, 1990.

[103] R. Dai and I. F. Akyildiz, "A spatial correlation model for visual information in wireless multimedia sensor networks," *IEEE Transactions on Multimedia*, vol. 11, no. 6, pp. 1148–1159, 2009.

[104] D. Devarajan, Z. Cheng, and R. J. Radke, "Calibrating distributed camera networks," *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1625–1639, 2008.

[105] S.-H. Cha and S. N. Srihari, "On measuring the distance between histograms," *Pattern Recognition*, vol. 35, no. 6, pp. 1355–1370, 2002.

[106] S. Xiong and J. Li, "An efficient algorithm for cut vertex detection in wireless sensor networks," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, pp. 368–377, IEEE, 2010.

[107] O. Dagdeviren and V. K. Akram, "An energy-efficient distributed cut vertex detection algorithm for wireless sensor networks," *The Computer Journal*, vol. 57, no. 12, pp. 1852–1869, 2014.

[108] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[109] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.