May 2014

ENERGY-EFFICIENT LIGHTWEIGHT ALGORITHMS FOR EMBEDDED SMART CAMERAS: DESIGN, IMPLEMENTATION AND PERFORMANCE ANALYSIS

Mauricio Casares
Syracuse University

Abstract

An embedded smart camera is a stand-alone unit that not only captures images, but also includes a processor, memory and communication interface. Battery-powered, embedded smart cameras introduce many additional challenges since they have very limited resources, such as energy, processing power and memory. When camera sensors are added to an embedded system, the problem of limited resources becomes even more pronounced. Hence, computer vision algorithms running on these camera boards should be light-weight and efficient. This thesis is about designing and developing computer vision algorithms, which are aware and successfully overcome the limitations of embedded platforms (in terms of power consumption and memory usage). Particularly, we are interested in object detection and tracking methodologies and the impact of them on the performance and battery life of the CITRIC camera (embedded smart camera employed in this research). This thesis aims to prolong the life time of the Embedded Smart platform, without affecting the reliability of the system during surveillance tasks. Therefore, the reader is walked through the whole designing process, from the development and simulation, followed by the implementation and optimization, to the testing and performance analysis. The work presented in this thesis carries out not only software optimization, but also hardware-level operations during the stages of object detection and tracking. The performance of the algorithms introduced in this thesis are comparable to state-of-the-art object detection and tracking methods, such as Mixture of Gaussians, Eigen segmentation, color and coordinate tracking. Unlike the traditional methods, the newly-designed algorithms present notable reduction of the memory requirements, as well as

the reduction of memory accesses per pixel. To accomplish the proposed goals, this work attempts to interconnect different levels of the embedded system architecture to make the platform more efficient in terms of energy and resource savings. Thus, the algorithms proposed are optimized at the API, middleware, and hardware levels to access the pixel information of the CMOS sensor directly. Only the required pixels are acquired in order to reduce the unnecessary communications overhead. Experimental results show that when exploiting the architecture capabilities of an embedded platform, 41.24% decrease in energy consumption, and 107.2% increase in battery-life can be accomplished. Compared to traditional object detection and tracking methods, the proposed work provides an additional 8 hours of continuous processing on 4 AA batteries, increasing the lifetime of the camera to 15.5 hours.

ENERGY-EFFICIENT LIGHTWEIGHT ALGORITHMS FOR EMBEDDED SMART CAMERAS: DESIGN, IMPLEMENTATION AND PERFORMANCE ANALYSIS

---

by

Mauricio Casares

Ph.D., Syracuse University, 2014

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in *Electrical and Computer Engineering.*

Syracuse University

May 2014

Dedicatoria

Dedicada a mis padres por todo el carino, amor e incazable esfuerzo de construir un hogar lleno de alegria y harmonia, confiaza y apoyo, llevandome a sobresalir y sobrepasar cualquir obstaculo por dificil que este fuese.

Acknowledgments

“Determine that the thing can and shall be done and then... find the way”

– Abraham Lincoln

After many years of hard, challenging, and yet fascinating work, Ive come to conquer one of the most important goal towards ones self realization. Graduate school has given me different perspectives of life, and has led me to understand and be more curious about everyday experiences. Despite all of my effort and determination during the pursue of my Doctorate Degree, it would not have been possible without the support and guidance of the kindest and dearest people surrounding me.

I want to give my most sincere thanks to my adviser Dr. Senem Velipasalar for bringing me into a fascinating area of studies. I am really thankful for all her guidance and support during the process of obtaining my degree. Her knowledge and teaching skills were very valuable throughout the whole process, and more so in my first steps of becoming the researcher I am now. She is a vivid example of a great and strong mentor, who would always support and encourage me to explore new possibilities in our area of research, and most importantly, whom with her example would show that everything you are willing to achieve, can be accomplished. Her high research standards and quality of work led me to all the great accomplishments I had through my path as a Doctorate student.

Many thanks to my co-workers for all their help with the development and understanding of my research, especially Youlu Wang. Thanks to the University of Nebraska-Lincoln, in particular to the department of Electrical Engineering for giving me the chance to demonstrate my knowledge. I would like to thank the faculty members who willingly answered every question I had, and cleared any doubt regarding my research and studies. Special thanks to Dr. Gursoy, Dr. Vuran, Dr. Hoffman and Dr. Boye.

Last but not least, I would like to thank Syracuse University for taking me as part of their academic community during the last year of my studies, as well as my co-workers in the Smart Vision Systems Laboratory. Meeting new researchers, and getting new perspectives and ideas from a new environment, was a great addition to my set of experiences that led me to obtaining my degree. It was a very enriching experience to help Dr. Velipasalar set up a new laboratory, and learn a whole new set of skills in the process. Many thanks to Akhan and Koray for their contributions in the development of my last experiments and for making the lab a great working environment.

Thanks to God, family and friends. “A journey will always be more exciting when there is good company involved.” I want to give my most profound thanks to my parents; without all their effort and support, none of this would have been possible. To both of my brothers, Daniel and Diego, who always made me laugh, event in situations of really high stress. Special thanks to my uncles Juliana Cordero and Eduardo Gonzales, who helped and supported me every time I needed them. With great appreciation I would like to thank all my friends, who share the challenge of living apart from their families, and that are now part of mine, making this whole experience one of the best I will remember in my life. Tomas Murtagh and Jorge Venegas, who always gave me encouraging words to continue, Alvaro Pinto, Fabio Parigi, Andres Doblado, Tania Toruno, Rebbeca Duar and Angelo Bee for their unconditional friendship. My dearest thanks to Maria Isabel Quintero who gave me all her love and patience, and for being next to me helping me finish my thesis.

Contents

Page

1 Introduction **1**

1.1 Overview 1

1.1.1 Embedded Smart Cameras: A short history 2

1.2 Thesis Contribution 9

1.3 Publications 10

1.3.1 Peer-reviewed Published Journal Papers 11

1.3.2 Peer-reviewed Published Conference Papers 11

2 CITRIC camera: Architecture **15**

2.1 The CITRIC Camera 15

2.2 The Microprocessor 16

2.3 The Image Sensor 17

2.4 The TelosB Mote 19

3 Lightweight salient foreground detection for embedded smart cameras **22**

3.1 Introduction 22

3.2 Proposed Method 26

3.2.1 **Building the Background Model** 27

3.2.2 **Updating the Counters** 29

3.2.3 **Salient Foreground Detection** 31

3.2.4 Adaptive background model update 33

3.2.5 Adaptive number of memory accesses and instructions 34
3.3 Experimental Results 38
3.4 Conclusions 46

4 Resource-Efficient Salient Foreground Detection in battery-Powered Embedded smart cameras by feedback tracking 58

4.1 Introduction 58
4.2 Wireless Embedded Smart Camera Platform 61
4.3 Motivation: Energy Consumption Analysis 62
4.4 FeedbackMethod: Resource-Efficient Salient Foreground Detection by Feedback Tracking 64
4.4.1 Determining the Search Regions 65
4.5 Experimental Results 69
4.5.1 Comparison of the Energy Consumptions of the Feedback and Sequential Methods 69
4.6 Conclusions 71

5 Resource-Efficient Salient Foreground Detection in battery-Powered Embedded smart cameras by adaptive tracking methodologies 72

5.1 Motivation: Adaptive methodologies 72
5.1.1 Empty-Scene Mode 75
5.2 Fixed-Rate Tracking Mode 77
5.3 Adaptive Tracking Mode 80
5.4 Combined Method for Further Energy Efficiency 84
5.5 Experimental Results 85
5.5.1 Comparison of the Energy Consumptions of the Adaptive Methodology and the Sequential Method 86
5.5.2 Energy Savings Provided by the Combined Method 87

5.5.3 Energy Consumption Analysis over a Longer Time Window 89
5.5.4 Comparison of Battery Lives 91
5.5.5 Analysis of the Tracking Performance 93
5.6 Conclusions 96

6 Energy-efficient Feedback Tracking on Embedded Smart Cameras by Hardware-level Optimization 97

6.1 Frame Capture Operation 97
6.1.1 CITRIC Middleware Interface 99
6.2 Hardware-Level Image Processing Tasks: Scaling and Cropping 102
6.3 Savings in Energy Consumption 105
6.3.1 Analysis of grabbing a QVGA frame 105
6.3.2 Analysis of hardware-Level image/video processing tasks: Object detection and tracking 107
6.4 Longer Tracking Experiment 114
6.5 Outdoor experiments 115
6.5.1 Tracking multiple objects 118
6.6 Increase in Battery-Life 120
6.7 Conclusion 121

7 Conclusions 122

List of Figures

1.1 Linux based embedded smart camera prototypes. 4
1.2 BlueLYNX camera mote. 5
1.3 Embedded smart cameras running TinyOS. 6
1.4 Wireless camera architecture introduced by Kleihorst et al. [55] 6
1.5 Other embedded smart cameras examples. 7
1.6 CMUcam embedded smart cameras. 7
1.7 CMUcam3 featuring a SunSPOT wireless mote. 8
1.8 CITRIC embedded smart camera Mote 8

2.1 The CITRIC camera mote. 15
2.2 The block diagram of the CITRIC camera. 16
2.3 Intel PXA270 block diagram. 18
2.4 OV9655-QCI interconnection 19
2.5 The TelosB mote 19
2.6 The TelosB architecture 20

3.1 Memory required for a pixel with the proposed method 26
3.2 Output of the temporal difference after applying a threshold. 29
3.3 The background model is gradually built as moving objects change their location. . 30
3.4 Illustration of how $h(i, j)$ is computed. 31
3.5 Original frame and the plot of the counter values $h(i, j)$ for different pixel locations (i, j). Higher values correspond to outer boundaries of multiple fountains, indicating regions with low reliability and non-salient motion. 32

3.6 Illustration of the behavior of a pixel's location (i, j) in both reliable and unreliable cases 34

3.7 Illustration of unreliable areas due to swaying tress and sun reflections (circled). Large peaks revel them reporting high counts kept in $h^t_{t-50}(i, j)$. 35

3.8 Video of a fountain: number of pixels with $R(i, j) = 0$ vs. the frame number plot. . 36

3.9 Traffic light sequence: number of pixels with $R(i, j) = 0$ vs. the frame number plot. 36

3.10 Rain sequence: number of pixels with $R(i, j) = 0$ vs. the frame number plot. . . . 37

3.11 Rain sequence: number of pixels with $R(i, j) = 0$ vs. the frame number plot. . . . 38

3.12 Per frame memory requirements of different background subtraction methods when one color channel is used. 40

3.13 ROC curves of different background subtraction methods. 42

3.14 Processing time (ms) versus the frame number for two different versions of the algorithm when there is a foreground object in the scene. 44

3.15 Processing time (ms) versus the frame number for two different versions of the algorithm when there is a foreground object in the scene. 44

3.16 Camera setup ready to perform the required energy measurements. 45

3.17 Variations in the operating current during the processing of three consecutive frames containing a foreground object. The method presented in this chapter (blue plot) is faster than the method presented in [4] (red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of Casares et al. [4]) 45

3.18 Variations in the operating current during the processing of three consecutive frames of an empty scene. The method presented in this chapter (blue plot) provides speed gaining at frame numbers that are multiple of 25. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of Casares et al. [4].) 46

3.19 Foreground detection results of different algorithms on a challenging indoor's video sequence with flickering lights. Outputs are obtained without morphological operations. 48

3.20 Foreground detection results of different algorithms on a challenging video of a windy scene. Outputs are obtained without morphological operations. 49

3.21 Foreground detection results of different algorithms on a challenging video in a windy day at the Airport. Outputs are obtained without morphological operations. . 50

3.22 Foreground detection results of different algorithms on a challenging video of another windy scene in a parking lot. Outputs are obtained without morphological operations. 51

3.23 Foreground detection results of different algorithms on a video of a rainy scene. Outputs are obtained without morphological operations. 52

3.24 Foreground detection results of different algorithms on a video of another rainy scene. Outputs are obtained without morphological operations. 53

3.25 Foreground detection results of different algorithms on a challenging video of a lake. Compared to (eg), the proposed method can eliminate the non-salient motion better. Although (d) has less noisy pixels, it misses the person and the dog. Outputs are obtained without morphological operations. 54

3.26 Foreground detection results of different algorithms on a video of a street. Outputs are obtained without morphological operations. 55

3.27 Foreground detection results of different algorithms on a video of a street. Outputs are obtained without morphological operations. 56

3.28 Comparison of foreground detection results of different algorithms on a video of a fountain with a significant lighting difference 57

4.1 Processing time in milliseconds when an object is at different distances from the camera. 62

4.2 Illustration of the flow diagrams for sequential and feedback tracking methodologies. 64

4.3 Displacement in the horizontal and vertical directions. 66

4.4 Operating current of the camera board with the feedback and sequential methods when tracking (a) one, (b) two, and (c) three remote-controlled cars. 68

5.1 Camera setup ready to perform the required energy measurements. 73

5.2 Operating current of the camera board during different tasks 74

5.3 Camera dropping frames to save energy as illustration of the main goal of the algorithm. 74

5.4 Empty-scene mode: red and blue plots are the operating current values when camera captures frame continuously, and when the microprocessor is put into idle state, respectively. 76

5.5 (a) Detecting a speed change. (b) Overlapping bounding boxes for a faster car. . . . 78

5.6 Updating the idle state duration based on the fastest object in the scene. 79

5.7 Car increasing its speed. 82

5.8 Operating current waveform when the idle time is changed based on the object speed. 83

5.9 Idle duration is increased in the combined method by employing the feedback method and the adaptive methodology together. 84

5.10 Operating current of the camera when tracking one car with the adaptive methodology (blue) and the sequential method (red). 87

5.11 Operating current of the camera board when employing the feedback method by itself and the combined method. 88

5.12 Operating current of the camera board when employing the adaptive methodology and the combined method. 89

5.13 Output frames obtained while tracking one car. 90

5.14 Output frames obtained when tracking two targets. 90

5.15 Operating current of the camera board when tracking one car with different algorithms for 20 min. 91

5.16 Characteristic curves of the batteries. 92

5.17 Scenario wherein the bounding boxes before and after the idle state do not overlap. 94

5.18 Tracking with a preventive mechanism that can handle the gradual increases in speed and resolve the issue seen in Figure 5.17. 95

6.1 Timing diagram for grabbing a frame using the Quick Capture Interface. 98

6.2 Interconnection of OV9655 and the Intel Quick Capture Interface on ARM PXA270. 99

6.3 Camera Driver Internal architecture. 100

6.4 Software architecture handling the CITRIC camera board. 101

6.5 Hardware-Level Image processing tasks: Scaling and Cropping 102

6.6 Interaction among components used in the Software based-Feedback method (Chapter 4.) 103

6.7 Interaction among components used in the Hardware based-Feedback method. . . . 104

6.8 Area cropped (a) by software using the API libraries (b) by hardware using the micro-controller of the OV9655, (c) background subtraction output on the cropped region 105

6.9 QVGA images captured by (a) using the API software library down-sampling subroutines and (b) performing hardware–level down-sampling on the micro-controller of the OV9655. 106

6.10 Operating currents of the camera board while grabbing a QVGA frame using the API sub-sampling subroutines and using the image micro-controller of the OV9655. 107

6.11 Illustration of saving gain by using hardware level operations. 108

6.12 Background subtraction output on a frame grabbed by using the API software libraries to down-sample to QVGA resolution (left column), and by using hardware level down-sampling (right column). 109

6.13 Operating currents when grabbing/buffering a frame and performing background segmentation using the API sub-sampling subroutines versus collaborating with the OmniVision OV9655. 110

6.14 Operating currents when (i) obtaining QVGA images with software-level down-sampling, and performing all processing on the main microprocessor ; (ii) performing down-sampling at hardwarelevel on the micro-controller of the OV9655 sensor, and performing foreground object detection and tracking at the main microprocessor. 111

6.15 (a) Last QVGA frame captured while computing pixel displacement of the tracked object; (b) Search regions cropped at hardware level. 112

6.16 Operating currents when performing foreground object detection an tracking on cropped search regions obtained by software versus hardware-level cropping. . . . 113

6.17 (a) Detecting a speed change. (b) Overlapping bounding boxes for a faster car. . . . 113

6.18 Operating currents when performing foreground object detection and tracking during 1-second time interval. 115

6.19 Outdoor experiment: Detection and Tracking of a person by employing hardware level operations. 117

6.20 Alternating BGS outputs from two objects being detected and tracked. 118

6.21 Alternating tracking of a person and a vehicle on hardware scaled and cropped frame areas. 119

List of Tables

3.1 Salient foreground detection algorithm 28
3.2 Memory requirement for the data saved for each pixel for different background subtraction methods (for one color channel) 39

4.1 Comparison of the Processing Times of the Proposed Feedback Method and the Sequential Approach. 69
4.2 Energy Consumptions for the Feedback and the Sequential Methods When Tracking One Car Continuously 70
4.3 Energy Consumptions for the Feedback and the Sequential Methods When Tracking One and Then Two Cars 70
4.4 Energy Consumptions for the Feedback and the Sequential Methods When a Car Enters and Leaves Twice 70

5.1 Energy Analysis of the Empty-Scene Mode 76
5.2 Energy Analysis of the Fixed-Rate Tracking Mode 80
5.3 Energy Analysis of the Adaptive-Rate Tracking Mode 83
5.4 Energy Analysis 84
5.5 Energy Consumption Comparison Between Adaptive and Sequential Methods . . . 86
5.6 Energy Consumption Comparison Between the Feedback, Adaptive and Combined Methods 87
5.7 Energy Consumption Comparison Between the Combined and Sequential Methods 89
5.8 Energy Consumption and Savings Comparison When Tracking One Car 91
5.9 Battery lifetime projection 92

6.1 Energy consumption when grabbing a QVGA frame using the API software libraries versus performing down-sampling at hardware-level. 107

6.2 Energy consumption when grabbing a QVGA frame at Software versus Hardware-level, and performing detection at the main microprocessor. 110

6.3 Energy consumption when grabbing and cropping a search region (100x100) at software versus hardware-level and performing detection at the main microprocessor. 114

6.4 Energy Consumption when performing detection and tracking during a 3-second time interval at software versus hardware level. 115

6.5 Battery life projection. 121

Chapter 1

Introduction

1.1 Overview

Computer vision has been a fast growing field of studies. In recent decades it is now viable to accomplish demanding computer vision tasks in real time. Researchers in the field are developing and testing complicated computer vision algorithms, and running them in real-time; tasks that could not be accomplished in the near past. Yet, the faster and powerful the processor is, the more energy it consumes. Thus, as the attention is being directed towards mobile applications and mobile platforms with limited processing and energy resources, special attention has to be paid to computational efficiency and energy consumption.

As opposed to general-purpose wall-powered computer systems, which have constant sources of energy, embedded platforms such as cell phones, wireless sensors, smart cameras, medical monitoring devices, and tablets have limited energy provided by on-board battery packets. Relying on a limited source of energy limits the design of the embedded devices. Special attention has to be paid the size and number of components utilized to build the actual platform. It is even more challenging when the embedded platform captures and processes image and video data, which is the case with wireless smart cameras. Since battery-life is limited, and video processing tasks consume considerable amount of energy, it is essential to have lightweight algorithms, and methodologies to increase the energy-efficiency of each camera. For instance, the design of algorithms to be imported on the embedded platforms should take into account important issues such as energy

consumption, memory usage and processing time. Designing algorithms that consider the energy consumption as well as the memory usage on embedded smart cameras has not received much attention until now.

An embedded smart camera can be summarized as a vision system, which not only captures images, but also incorporates on-board processing and communication. As opposed to regular cameras, an embedded smart camera, not only captures images, but also provides on-board computation capabilities to extract useful information from the captured images, detect certain events of interest and create alerts that are used in an intelligent and automated system. Thus, rather than transferring all the data to a back-end server, they can process images, and extract data locally.

This thesis focuses on lightweight algorithm design for embedded smart cameras, and methodologies to increase the battery-life of the embedded smart camera. We focus on the performance of the embedded smart cameras, and present the impact of designing and running well-suited lightweight computer vision algorithms on the battery-life of the embedded platform. The thesis emphasizes the advantages of designing lightweight algorithms that are well integrated with the cameras architecture, opposed to using algorithms designed for wall-powered platforms. The goal of the algorithms and the methodologies is to prolong the lifetime of the embedded smart platforms, without affecting the reliability of the system during surveillance tasks. The reader is walked through the whole process, starting with the design and simulation, followed by implementation and optimization, ending with the testing and performance analysis.

1.1.1 Embedded Smart Cameras: A short history

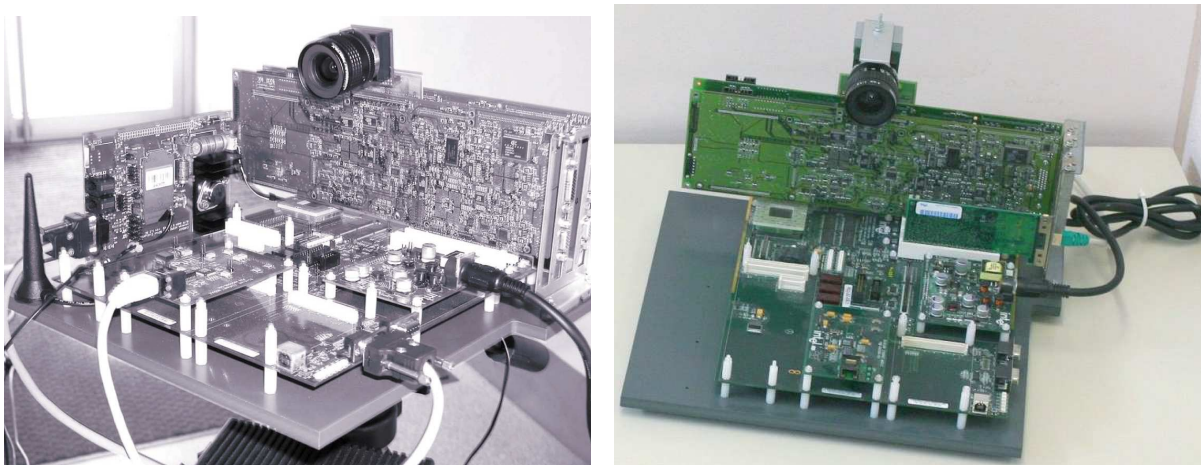
Even though embedded smart cameras do not have a very long history, since the first time the concept was introduced, they have been exposed to a series of adaptations and changes in recent decades. This section brings to the reader a brief summary on how embedded cameras have been developed during the past decade.

Wolf et al. [26] introduced one of the early examples of embedded smart cameras. Since then, embedded smart cameras have received a lot of attention from both academia and industry due

to the wide range of applications for which they can be used. In recent years, embedded smart cameras have grown popular not only for their small size and ease of deployment, but also for the diverse applications that could not be accomplished on centralized, general purpose vision systems. Consequently, they have become effective means of rapidly implementing simple machine-vision systems due to their reliability, cost effectiveness and ease of integration. Additionally, since they are self-contained units, embedded smart cameras can be used as a single unit as well as for network applications. Furthermore, due to the growing variety and complexity of the vision algorithms, research on embedded camera design and development needs to keep up with the demanding pace of computer vision applications. An embedded smart camera performs real-time analysis to extract useful information from captured images. They are employed in a variety of applications including surveillance, medicine, sports, industry and military applications. Embedded cameras are also used for on-site data acquisition, and customer behavior analysis in marketing and advertisement. Most of the effort to improve the performance of embedded smart cameras has been expended to accomplish real-time processing tasks with acceptable levels of accuracy and reliability. In order to achieve this goal embedded smart cameras are becoming powerful devices which require a better management of their energy source. Common computing platforms for smart cameras are FPGAs , digital signal processors (DSPs), and/or general purpose microprocessors [60]. Many embedded vision platforms, designed for wireless sensor networks, have been developed recently.

Due to compatibility issues, using dedicated micro-controllers without an underlying Operating System (OS) makes it difficult to create distributed network of embedded smart cameras that operate in a plug and play fashion. During the Workshop on Embedded Middleware for Smart Camera and Visual Sensor Networks (eMCAM), which was held at Stanford University in 2008, it was concluded that there is a need for having an OS running on the smart cameras as central management unit. The idea of having embedded Linux running on the smart cameras has been explored in recent years, and it is becoming more and more common. Bramberger et al. [48] along with the Australian Research Centers (ARC) designed an innovative smart camera which consists of a network processor and a variable number of DSPs (Figure 1.1 (a)). Their design is targeted for

distributed embedded surveillance, focusing on power consumption, QoS management, and limited resources. Even though the platform provides sufficient capabilities for image processing with a processing power of 9600 MIPS and on-board memory of 784 MB, it still requires an average power consumption of 35 Watts (Rinner et al. [61]).



(a) The board by Bramberger et al. [48] (b) The borad by Quaritsch et al [54]

Figure 1.1: Linux based embedded smart camera prototypes.

Quaritsch et al. [54] employed smart cameras with multiple DSP processors, as shown in Figure 1.1 (b), for data processing. Thus, having multiple DSP processors would require the use of an Operating System on top of the design. Even though the authors did not report any information regarding the power consumption of their prototype, the power consumption of using multiple DSPs would be comparable to the analysis presented in Rinner et al. [61].

Fleck et al. [51] presented a network of smart cameras for tracking multiple people. They used commercial IP-based cameras, which consist of a CCD image sensor, a Xilinx FPGA and a Motorola PowerPC shown in Figure 1.2. Chalimbaud and Berry [52] presented a smart camera based on FPGAs. Similar to Bramberger et al. [48], the hardware architecture introduced by Fleck et al. [51] requires an operating system that reliably manages the software tasks among the multiple processing units and their peripherals.

Even though running embedded Linux as a central management unit brought scalability as well as flexibility to the design, and a wider range of algorithms could now be implemented on the

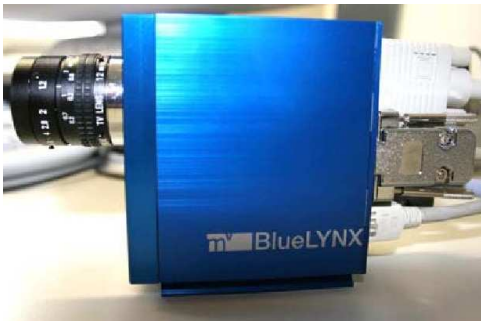


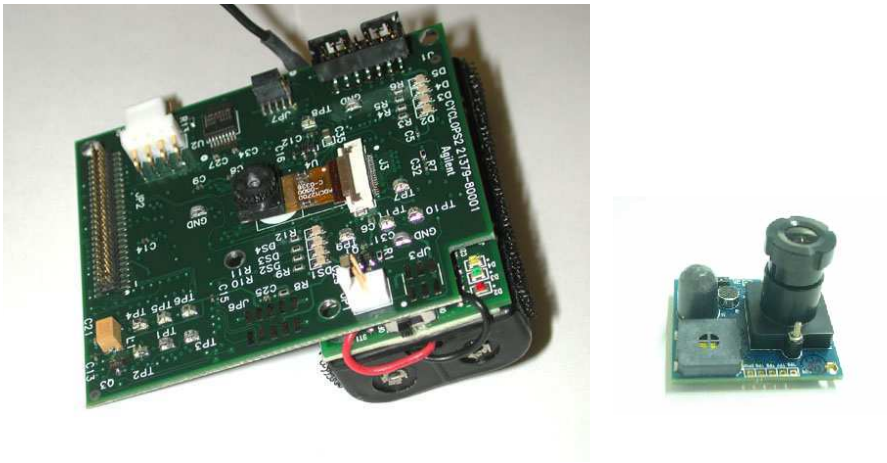
Figure 1.2: BlueLYNX camera mote.

smart cameras, the architectures of these platforms were still big in size and consumed significant amount of energy. The advances in integrated micro-chips allowed embedded smart cameras to be sizable and energy efficient.

Embracing the success in the field of sensor networks, along with the availability of low-cost micro-sensors, applications involving multimedia vision sensor networks have drawn attention from the research community. In the proposed architectures, vision capability was added to a host mote which featured a dedicated micro-controller managed by a simpler Operating System called TinyOS [47].

The Cyclops [43] (Figure 1.3 (a)) and Imote2 [62] (Figure 1.3 (b)) are two examples of this architectural trend. Even though these smart cameras were surprisingly small and yet powerful with processors running at frequencies of 7.3 MHz and 12 MHz, respectively, their capabilities were still very limited to support a more complex variety of algorithms.

Another type of embedded smart cameras involved platforms featuring dedicated microcontrollers, which instead of using an OS, employed their own custom API libraries. These platforms were mostly application-dependent with some limitations in terms of scalability and ease of deployment. Despite being limited to a specific range of applications, and designed to be wall-powered, their APIs were optimal and reliable reaching high processing rates in the order of 30 fps. For instance, Kleihorst et al. [55], presented a smart camera mote with a Xetal-II high-performance, yet low-power single-instruction multiple data processor shown in Figure 1.4. The camera's processor is equipped with dedicated peripherals for frame-based real-time video analysis. The pro-



(a) Cyclops (b) Imote2

Figure 1.3: Embedded smart cameras running TinyOS.

cessor handles interrupts from the Data Input/Output processor (DIP/DOP), communicates with the outside world and configures other blocks. The average power consumption of the processor is 600 mW when working at 84Mhz.

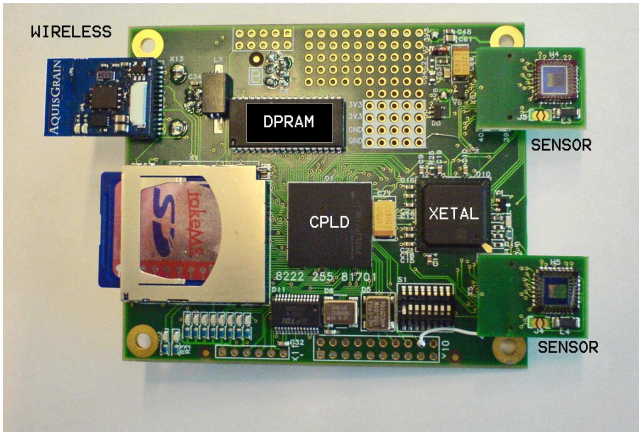


Figure 1.4: Wireless camera architecture introduced by Kleihorst et al. [55]

Others platforms following this type of architecture were the MeshEye [53], XYZ [44], and Panoptes [39], shown in Figure 1.5 (a-c). These platforms have flexibility and scalability issues in general. Moreover, they employ processors running at 206 Mhz, 55Mhz, and 56.7 Mhz (to control demanding peripherals), with higher energy consumption.

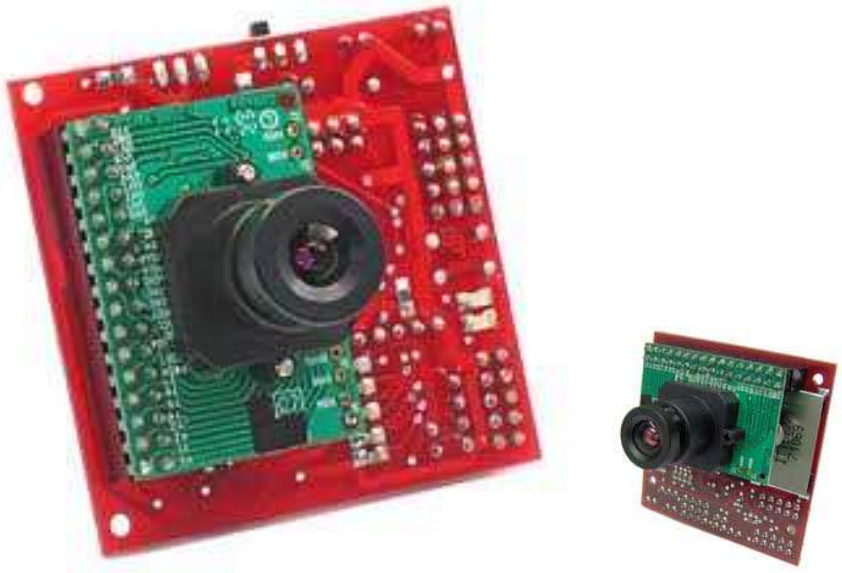
With the advancement in semiconductors and RISC micro-controllers, a new era in the embedded smart camera design started; smart cameras became smaller in size and more efficient in terms

(a) MeshEye (b) XYZ (c) Panoptes

Figure 1.5: Other embedded smart cameras examples.

of power consumption.

The CMUcam2 [46] shown in Figure 1.6 (a) is a low-cost embedded camera that could be categorized between the previous two classes. Having a 75MHz-RISC processor and 384KB SRAM, and being equipped with a wireless mote running tinyOS, they were powerful enough to run a larger set of applications. Additionally, they contain efficient computer vision API libraries.



(a) CMUcam 2 (b) CMUcam 3

Figure 1.6: CMUcam embedded smart cameras.

The camera was small, flexible and easy to deploy. Their design was intensively studied and highly accepted in the sensor network community. However, due to the limited memory and processing power, only low-level image processing could be performed. Later on, the CMUcam3 [56], shown in Figure 1.6 (b), was introduced, but the design was still lacking processing power as re-

ported by Casares et al. [64]. Consequently, they proposed an improvement to the existing design by attaching a SunSPOT mote (from SUN [57]) as shown in Figure 1.7. Hence, some of the processing demand could be handled at the ARM micro-controller in the mote. They proposed their own middleware interface so the camera and the mote could efficiently communicate with each other.

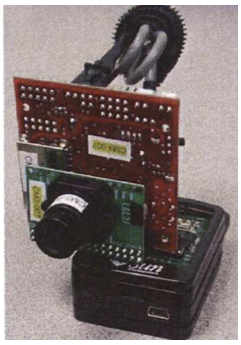


Figure 1.7: CMUcam3 featuring a SunSPOT wireless mote.

With the advancement in embedded micro-computing, embedded smart cameras have become sophisticated Systems on Chip (SoCs), with dedicated hardware that support complex vision algorithms and video/image analysis. In particular due to the remarkable improvements in ARM (Advance Risc Micro-controllers) technology, the idea of having Linux running on the cameras became feasible and scalable. The CITRIC camera [63], shown in Figure 1.8, is a great example of an efficient low-power architecture, which contains enough resources to run demanding vision algorithms on a real-time basis. We have used the CITRIC platform for our experiments.

Figure 1.8: CITRIC embedded smart camera Mote

1.2 Thesis Contribution

The primary contributions of this thesis are (i) the design, implementation, and testing of lightweight computer vision algorithms, which are aware and successfully overcome the limitations of embedded platforms (in terms of power consumption and memory usage), (ii) the development of adaptive methodologies to increase battery lifetime of the embedded smart cameras, and (iii) development and implementation of hardware-level operations to increase the energy efficiency further. The contribution in this dissertation is divided into three parts.

The first part presents a background subtraction algorithm for an object detection system to be imported into an ARM micro-controller. A lightweight and efficient algorithm for salient foreground detection is presented; it is highly robust against lighting variations and non-static backgrounds (i.e. scenes with swaying trees, water fountains, as well as strong lighting changes). The performance of the algorithm is better than or comparable to state-of-the-art background subtraction methods, such as mixture of Gaussians, Eigen- or Codebook-based background subtraction methods while providing a notable reduction in the memory requirements, as well as the reduction in the number of memory accesses per pixel.

The second part presents a feedback-based object detection and tracking algorithm to decrease the processing time of a frame. The algorithm estimates positions of the objects being tracked and feeds this information to the background subtraction stage. Hence, the detection process in the subsequent frames become localized, which leads to decrease in the processing time and the energy consumption.

The third part is related to the optimization of the algorithm at different levels of the embedded architecture. This algorithm is optimized at the API, middleware, and hardware levels to directly access the pixel information of the CMOS sensor. Only the required pixels in the predicted area (based on location prediction) are acquired in order to reduce the unnecessary communications overhead.

The algorithms were initially designed and tested by using MATLAB. They were then coded in C/C++ to be imported on to the ROM memory of the embedded smart camera. The execution of

the algorithms rely on embedded Linux as central management unit. The camera used for testing is a CITRIC camera [63] shown in Figure 1.8.

Chapter 2 describes the hardware architecture of the camera used in this project. It describes different components of the CITRIC embedded smart camera. In addition, a brief description of the wireless communication capabilities of the camera mote is provided in Chapter 2.

Chapter 3 presents our light-weight and efficient background modeling and foreground detection algorithm. This algorithm runs on the camera boards in order to detect and segment moving objects (person, cars, etc.). It is highly robust against lighting variations and non-static backgrounds. The memory requirement per pixel and the allocation of it is described. The number of memory accesses and instructions are adaptive, and are decreased according to the amount of activity in the scene and on a pixel's history.

Chapter 4 describes the feedback-based background subtraction and tracking algorithm, which provides significant savings in processing time. Then in chapter 5, an adaptive methodology is presented that can send the camera to idle state not only when the scene is empty but also when there are target objects. Subsequently, a combined method is introduced, that employs the feedback method and the adaptive methodology together providing further savings in energy consumption. Finally, a detailed comparison of these methods is presented along with the gains in processing time as well as the significant savings in energy consumption and battery life increase.

Hardware/software interactions are discussed in Chapter 6. Operations are performed at hardware-level to (i) change the image resolution, and (ii) perform image cropping based on search regions obtained from the tracking stage. Moreover, experimental results are presented to show the advantages of implementing hardware operations.

1.3 Publications

I have received a third place award with my work titled "Energy-efficient Feedback Tracking on Embedded Smart Cameras by Hardware-level Optimization" at the fifth ACM/IEEE International Conference on Distributed Smart Cameras in Gent Belgium, 2011.

My research work during my Ph.D. studies has resulted in the following articles published in prestigious and peer-reviewed journals and conference proceedings.

1.3.1 Peer-reviewed Published Journal Papers

[J1] K. Ozcan, A. K. Mahabalagiri, M. Casares, and S. Velipasalar, "Automatic Fall Detection and Activity Classification by a Wearable Embedded Smart Camera", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 125–136, June 2013

[J2] M. Casares and S. Velipasalar, "Adaptive Methodologies for Energy-efficient Object Detection and Tracking with Battery powered Embedded Smart Cameras," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, issue 10, pp. 1438–1452, October 2011.

[J3] A. Sharma, D. Bullock, S. Velipasalar, M. Casares, J. Schmitz, N. Burnett, "Improving Safety and Mobility at High Speed Intersections with Innovations in Sensor Technology," *Transportation Research Record, TRR 2259, Journal of the Transportation Research Board*, pp. 253–263, 2011.

[J4] Y. Wang, M. Casares, and S. Velipasalar, "Cooperative object tracking and composite event detection with wireless embedded smart cameras," *IEEE Trans. Image Process.*, vol. 19, no. 10, pp. 2614–2633, Oct. 2010.

[J5] M. Casares, S. Velipasalar, A. Pinto, "Light-weight Salient Foreground Detection for Embedded Smart Cameras," *Computer Vision and Image Understanding*, vol. 114, issue 11, pp. 1223–1237, 2010.

1.3.2 Peer-reviewed Published Conference Papers

[C1] A. Almagambetov, M. Casares, S. Velipasalar, "Autonomous Tracking of Vehicle Rear Lights and Detection of Breaks and Turn Signals," *Proc. of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, July 2012.

[C2] M. Casares, A. Almagambetov and S. Velipasalar, "A Robust Algorithm for the Detection of Vehicle Turn Signals and Brake Lights," *Proc. of the IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance (AVSS)*, Sept. 2012.

[C3] M. Casares, K. Ozcan, A. Almagambetov and S. Velipasalar, "Automatic Fall Detection by a Wearable Embedded Smart Camera," *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Oct.30-Nov.2, 2012.

[C4] A. Almagambetov, M. Casares and S. Velipasalar, "Autonomous Tracking of Vehicle Taillights from a Mobile Platform using an Embedded Smart Camera," *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Oct.30-Nov.2, 2012.

[C5] A. Sharma, M. Casares, S. Velipasalar, D. Bullock, " Wide Area Detection for Reducing Dilemma Zone Incursions at Isolated High Speed Intersections: Opportunities and Challenges," *Proc. of the 18th World Congress on Intelligent Transportation Systems*, 2011.

[C6] M. Casares, P. Santinelli, S. Velipasalar, R. Cucchiara, A. Prati, "Energy-efficient Feedback Tracking on Embedded Smart Cameras by Hardware-level Optimization", *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras*, August 2011 (**received the 3rd place paper award**).

[C7] M. Casares, P. Santinelli, S. Velipasalar, A. Prati and R. Cucchiara, "Energy-efficient Foreground Object Detection on Embedded Smart Cameras by Hardware-level Operations," *Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2011.

[C8] A. Sharma, D. Bullock, S. Velipasalar, M. Casares, J. Schmitz, N. Burnett, "Improving Safety and Mobility at High-Speed Intersection with Innovations in Sensor Technology," *Proc. of the Transportation Research Board Annual Meeting*, Jan. 2011

[C9] M. Casares and S. Velipasalar, "Resource-Efficient Salient Foreground Detection for Embedded Smart Cameras by Tracking Feedback, *Proc. of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, 2010.

[C10] M. Casares and S. Velipasalar, "An Adaptive Method for Energy-Efficiency in Battery-Powered Embedded Smart Cameras," *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, September 2010.

[C11] Y. Wang, M. Casares and S. Velipasalar, "Cooperative Object Tracking and Event Detection with Wireless Smart Cameras," *Proc. of the IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance*, pp. 394–399, 2009.

[C12] M. Casares, A. Pinto, Y. Wang and S. Velipasalar, "Power Consumption and Performance Analysis of Object Tracking and Event Detection with Wireless Embedded Smart Cameras," *Proc. of the Int'l Conf. on Signal Processing and Communication Systems (ICSPCS)*, 2009.

[C13] M. Casares and S. Velipasalar, "Light-weight Salient Foreground Detection for Embedded Smart Cameras,"*Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, pp. 1–7, Sept. 2008.

[C14] M. Casares, M. C. Vuran and S. Velipasalar, "Design of a Wireless Vision Sensor for Object Tracking in Wireless Vision Sensor Networks," *Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), Workshop on Embedded Middleware for Smart Camera and Visual Sensor Networks (eMCAM)*, pp. 1–9, Sept. 2008.

[C15] Y. Zhao, M. Casares and S. Velipasalar, "Continuous Background Update and Object Detection with Moving Cameras,"*Proc. of the IEEE International Conference on Advanced Video and Signal Based Surveillance*, pp. 309–316, Sept. 2008.

Our work related to background subtraction presented in chapter 3, is published in part in [J2], [J3], [C13]. Our foreground object detection algorithm is designed for embedded smart cameras. The algorithm is implemented and imported to an embedded camera and the results are reported at the end of chapter 3. This lightweight background subtraction is also utilized in chapters 4 5 6 as foreground detection stage for object tracking purposes. Our work on feedback tracking and adaptive methodologies for increasing battery-life is published in [J4],[C9],[C10],[C12]. Finally

the work related to hardware level optimization presented in chapter 6, is published in [C6] and [C7].

Chapter 2

CITRIC camera: Architecture

This chapter presents the details of the architecture of the CITRIC embedded smart camera [63]. It provides an understanding the limitations and challenges involved when designing algorithms to be imported on to embedded platforms. It will also introduce the terminology to be used in the following chapters. Even though the majority of the components of the CITRIC camera are described in this chapter, 6 is where the hardware/software interactions are explained. Understanding the camera's architecture and its challenges provides further motivation to design lightweight algorithms suitable for embedded image/video processing tasks.

2.1 The CITRIC Camera

Figure 2.1: The CITRIC camera mote.

The CITRIC camera depicted in Figure 2.1 is a fully programmable embedded platform with communication capabilities. The CITRIC camera offers wireless communication using a Telos B (wireless sensor mote) attached to it. The block diagram shown in Figure 2.2 represents the

hardware architecture of the camera. The camera is equipped with a general-purpose processor running embedded Linux (see Section 2.2), an image sensor (see Section 2.3), external memories and other supporting circuitry. The ARM PXA270 microprocessor is a fixed-point processor from Marvell with a maximum speed of 624 MHz. The typical frequencies supported by the CITRIC camera range from 208 to 520MHz. The board also incorporates a wireless MMX co-processor to accelerate multimedia operations. In terms of memory resources, the CITRIC camera comes with 256 KB of internal Synchronous RAM (SRAM) while the available external memory is composed of 64 MB of SDRAM, and 16 MB of NOR FLASH. The latter has the capability to execute code directly out of the non-volatile memory on bootstrap (eXecution-In-Place, XIP) and is natively supported by the PXA270 processor.

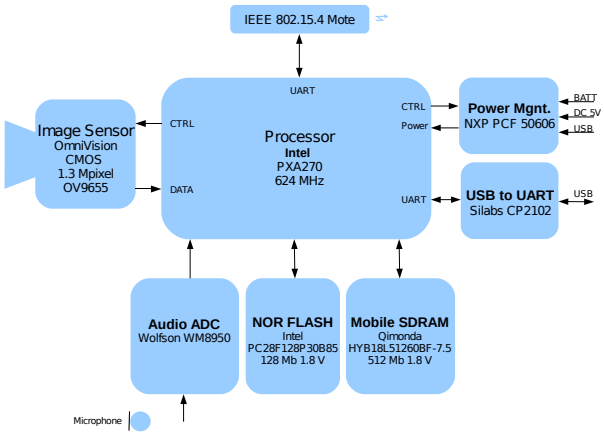


Figure 2.2: The block diagram of the CITRIC camera.

2.2 The Microprocessor

The CITRIC camera platform is equipped with a general-purpose processor (Intel PXA270 processor [75]) running embedded Linux. This facilitates the development of software applications using higher level programming languages such as C/C++. The PXA27x processor is a fixed point integrated system-on-a-chip microprocessor for high-performance, low-power, portable, handheld and handset devices. It incorporates the Intel XScale technology with on-the-fly voltage and frequency scaling and sophisticated power management to provide industry-leading MIPs/mW performance.

The PXA27x processor complies with the ARM Architecture V5TE instruction set (excluding floating point instructions). It also supports Intel Wireless MMX integer instructions to accelerate applications involving audio and video processing. The PXA27x processor memory interface supports a variety of external memory types to allow design flexibility. The processor also provides four 64-Kbyte banks of on-chip SRAM, which can be used for program code or multimedia data. Each bank can be configured to retain its contents when the processor enters a low-power mode. An integrated LCD panel controller provides support for displays up to 800 x 600 pixels. It permits 1, 2, and 4-bit gray scale and 8- or 16-bit color pixels. A 256-entry palette RAM provides flexibility in color mapping. A set of serial devices and general system resources provides computational and connectivity capability for a variety of applications. The PXA27x processor incorporates a comprehensive set of system and peripheral functions that makes it useful in a variety of low-power applications. Figure 2.3 the block diagram of the processor. The diagram shows a primary system bus with the Intel XScale core attached, along with an LCD controller, USB host controller, and 256 KB of internal memory. The system bus is connected to a memory controller to allow communication with a variety of external memory or companion-chip devices, and it is also connected to a DMA controller/bridge to allow communication with the on-chip peripherals. Some of these peripheral functions provide the ability to handle directly the image sensor. In particular, the Quick Capture Interface (QCI) provides a connection between the processor and the image sensor (as shown in Figure 2.4). The QCI is able to acquire data and control signals and performs the appropriate data formatting before routing the data to the memory using direct memory access (DMA). The I2C interface is directly connected to the Serial Camera Control Bus (SCCB) interface of the image sensor, and it is used to access the configuration register set.

2.3 The Image Sensor

The image sensor on the CITRIC camera is a OmniVision OV9655 [76], which is a low voltage SXGA CMOS image sensor with an image micro-controller on board. It supports image sizes SXGA (1280 x 1024), VGA (640 x 480), CIF (352 x 288), and any size scaling down from CIF to

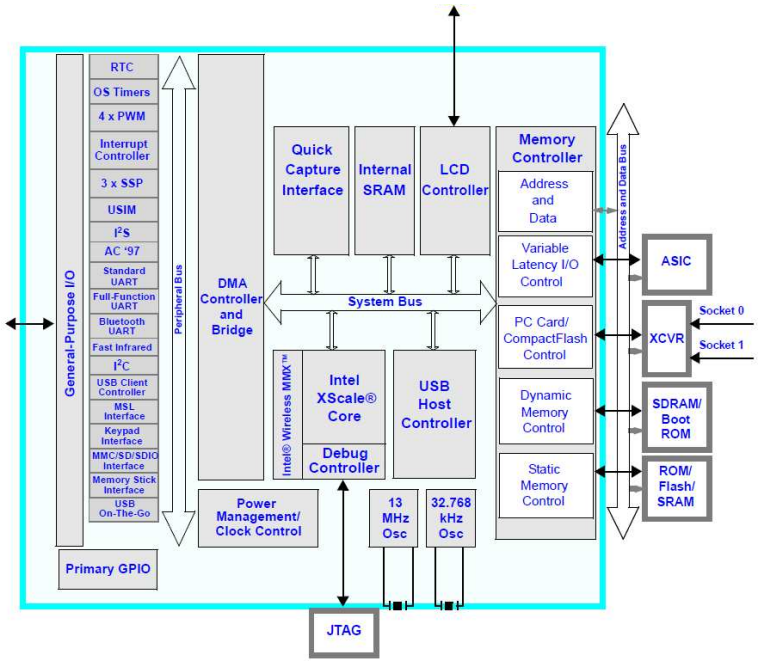


Figure 2.3: Intel PXA270 block diagram.

40 x 30, and provides 8-bit/10-bit data formats [63]. It can operate up to 15 frames per second (f/s) in SXGA mode and up to 30 fps working in VarioPixel(R)[1] mode when performing sub-sampling. Figure 2.4 shows the interconnection of OV9655 and the Intel Quick Capture Interface on ARM PXA270. The image sensor offers the full functionality of a camera and image micro-controller on a single chip. There is a complete control over image quality, formatting and output data transfer and all required image processing functions are also programmable. The Serial Camera Control Bus (SCCB) interface is used to program the sensor behavior by setting all the control registers in the device. It is an Inter-Integrated Circuit (I2C) compatible hardware interface. The Digital Video Port, used to capture images, provides a connection between the sensor and the CITRIC camera main processor PXA270. It is used to capture the image data. It is a unidirectional communication bus transferring 10-bit data signals and the line and frame synchronization signals [75].

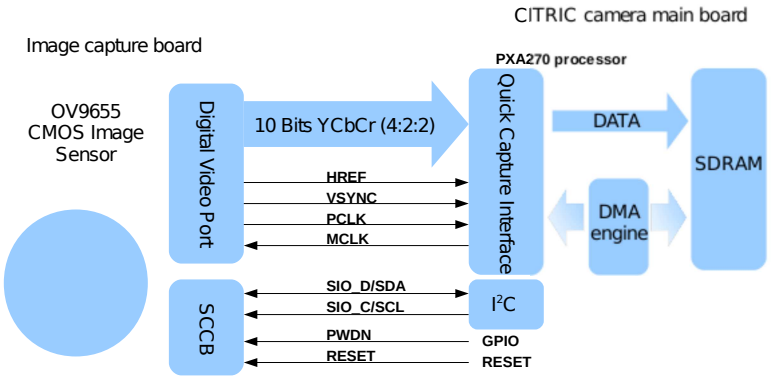


Figure 2.4: Interconnection of OV9655 and the Intel Quick Capture Interface on ARM PXA 270.

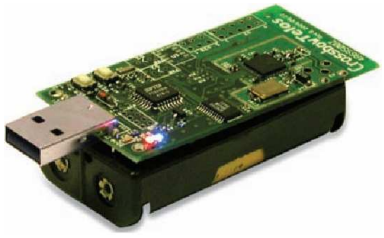


Figure 2.5: The TelosB mote

2.4 The TelosB Mote

The CITRIC camera provides a Joint Test Action Group (JTAG) port offering serial and I2C communications for data transferring to external devices. The port in the embedded platform is used to connect to a TelosB (wireless sensor mote) for wireless communication purposes. The TelosB is a wireless mote from Crossbow Technology. It is an ultra low power wireless sensor module (mote) developed by UC Berkeley.

The camera communicates with the mote using a dedicated asynchronous serial interface. The main features of the mote are: minimal power consumption, easy to use, and software and hardware robustness. TelosB [77] is based on the Texas Instruments MSP430 microcontroller, Chipcon CC2420, IEEE 802.15.4-compliant radio, and USB. The maximum data rate of 802.15.4 is 250 kbps per frequency channel (16 channels available in the 2.4 GHz band). Even though, the TelosB is capable of frame streaming over the wireless channel, its maximum rate is too low to achieve

[1]VarioPixel: Newly Developing technology that uses multiple pixels acting as a single pixel in order to improve the performance of the chips. Thus, significantly improving low light performance and enhance the video capture.

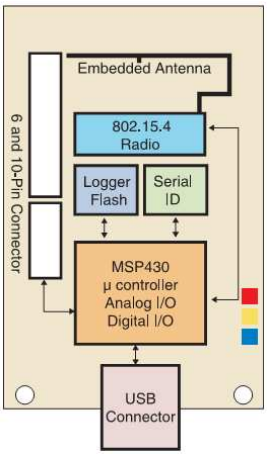


Figure 2.6: The TelosB architecture

real-time image streaming from the camera back to the server at high quality. On the other hand, the existing rate is optimal for sending extracted features over the network if an event of interest occurs. Since TinyOS [47] is the operating system running on the mote, it offers the capability of substituting different standard routing protocols to suite the particular needs of an application.

TinyOS is a component based operating system suitable for research in wireless embedded systems for sensor networks. TinyOS was developed for tiny, low-power nodes, whose imported applications operate with severe memory and power constraints. TinyOS is the current platform of choice in the sensor network community. It helps developers face the challenges of limited resources, low-power operation, and event-centric concurrent applications. TinyOS has a component-based programming model, codified in NesC language [30], a dialect of C. It is not an OS in the traditional sense; it is a programming framework for embedded systems, and a set of components that enable the compilation of an application-specific OS into the user's application. The architecture of the system and composition of the components allow researchers to work at any level, from details of link layer communication protocols up to the application semantics [78].

In TinyOS, the hardware primitives, such as register access and module flags, are exposed through a hardware presentation layer (HPL). A platform-dependent hardware abstraction layer (HAL) exposes hardware module functionality so that the full power of the hardware may be used. On top of the HAL abstraction, there is a platform independent radio stack (link protocol and

physical layer access) for the CC2420 transceiver that provides register access to the radio device; the radio stack then acts as a library that uses these primitives to control the radio.

Chapter 3

Lightweight salient foreground detection for embedded smart cameras

3.1 Introduction

An embedded smart camera is a stand-alone unit that not only captures images, but also includes a processor, memory and communication interface. With battery-powered and embedded smart cameras, it has become viable to install many spatially-distributed cameras interconnected by wireless links. However, wireless and battery-powered smart-camera networks introduce many additional challenges since they have very limited resources, such as energy, processing power, memory and bandwidth. The algorithms running on the camera boards should be lightweight and efficient. They should require less memory for storage, and consume less power. In addition to the accuracy of an algorithm, it is very important to consider its efficiency, memory requirements and portability to an embedded processor during algorithm design.

This chapter presents a lightweight and efficient background modeling and salient foreground detection algorithm that is highly robust against lighting variations and non-static backgrounds such as scenes with swaying trees, water fountains, rippling water effects and rain. The memory requirement for the data saved for each pixel is very small in the proposed algorithm, and this is achieved without sacrificing accuracy. Moreover, the number of memory accesses and instructions are adaptive, and are decreased even more depending on the amount of activity in the scene and on a pixel's history.

Foreground detection is the first step in most of the object tracking applications. Existing methods for foreground detection can be broadly classified into two categories: temporal difference methods [2,3], and background subtraction methods [5–7,9,12,14,16,23,25,29,36,49]. Temporal difference methods subtract two consecutive frames and then apply a threshold to the output. These methods perform well when the background changes over time, however they cannot detect all the pixels of a moving object. Background subtraction methods build a model of the background and subtract this from the current image to detect objects in the scene. In order to adapt to changes in the environment, the background model is usually updated over time [6, 12, 14, 16, 23, 25, 36, 49]. The method proposed in this chapter is a hybrid method, and it employs temporal difference to build the background model.

Horprasert et al. [8] obtain expected chromaticity by the arithmetic mean of the RGB values calculated over a number of background images. By using several thresholds, pixels are classified as foreground, background, shadow and highlighted background. Hidden Markov Models (HMMs) have been employed to represent the variations in the pixel intensity as discrete states [15, 22]. Nonparametric background models have been used in [12, 28, 38].

Oliver et al. [21] present an eigenbackground method, where images of a static background are collected, and PCA is employed to reduce the dimensionality of space. Input images are projected onto the PCA subspace, and a threshold is applied to the difference between the projected and current image to find the foreground regions.

Adaptive Mixture of Gaussians (MoG), introduced by Stauffer and Grimson [16], is one of the most commonly used background subtraction methods to model complex and non-static backgrounds. However, a few Gaussian distributions are usually not sufficient to accurately model backgrounds having fast variations. Methods have been introduced later that are based on Gaussian mixtures [24, 33, 35, 37]. Zivkovic [33] proposed an improved adaptive MoG model to constantly update the parameters of a Gaussian mixture and to simultaneously select the appropriate number of components for each pixel.

Kim et al. [32] proposed an algorithm for background modeling, where sample background

values at each pixel are quantized into codebooks during training, which represent a compressed form of the background model. This algorithm performs well when background is non-static or there are lighting variations. However, its performance on different video sequences is dependent on the choice of multiple threshold values.

Although many methods have been introduced for foreground object detection, much less attention has been paid to the memory requirement and the portability of these algorithms to an embedded processor. Lighting variations and non-static backgrounds make the foreground detection problem even more challenging, since we are interested only in *salient* motion in tracking applications. We need to separate cases of uninteresting motion, such as swaying trees and water fountains, from the salient motion regions. The necessity of handling these challenging cases increases the algorithm complexity, and thus memory requirements.

In this chapter, we present a lightweight method that is highly robust against lighting variations and non-static backgrounds. The memory requirement of the proposed method for the data saved for each pixel is very small compared to many traditional background subtraction methods. For instance, Stauffer and Grimson [16] use multiple (three to five) Gaussian distributions per pixel, to model non-static backgrounds. Kim et al. [32] form codewords for each pixel to capture the different values at that pixel location. Each codeword for each pixel has nine entries, and on the average 6.5 codewords are needed for a pixel. The MoG method requires 23 to 32 bytes per pixel if three Gaussian distributions and one color channel are used. The codebook method requires 91 bytes on the average for one color channel. Whereas, in our method, at most 6.25 bytes are needed per pixel. We provide a detailed comparison of the memory requirements in Section 3.3.

The proposed algorithm differentiates between salient and non-salient motion based on the history and reliability of a pixel's location, and by considering neighborhood information. The concept of reliability will be explained in detail below. The background model is selectively updated with an automatically adaptive rate, thus can adapt to rapid changes. For instance, if a location is deduced to be very reliable based on its history, a reliability flag is set to 1 for this location, and a higher background update rate is used, i. e. this location is incorporated to the background faster.

As opposed to existing methods, each pixel is treated differently based on their histories. Instead of requiring the same number of memory accesses and instructions for every pixel, we require less memory access and less instructions for stable background pixels, i.e. for pixels whose reliability flag is set to 1. If a car enters the scene, for example, then the reliability flags of the pixels occluded by the car will be set back to 0. Thus, if we plot the number of pixels with reliability bit 0 versus the frame number, the changes and peaks in this plot will indicate the portions of the video with activity. Thus, this plot can serve as a tool for activity summary.

Unlike many traditional methods treating each pixel individually, in the proposed method, information is obtained from neighboring pixels and incorporated into decision making, which increases accuracy and robustness. The algorithm can use only intensity, or one color channel, and still provides very reliable results. The experimental results presented in Section 3.3 were obtained by using the red color channel only. The experiments were performed on different video sequences, with non-static backgrounds and varying levels of difficulty, and the same threshold values were used for all of them. Thus, the dependency on the threshold values is low. The experimental results also demonstrate the success of the proposed lightweight method in challenging situations such as scenes with water fountains, swaying trees, and strong wind and rain.

We presented an initial version of the proposed algorithm in [65]. In this previous version, static foreground objects are not pushed into the background. In [4], we proposed a new version with which static foreground objects can be pushed into the background *if* desired. This version also has less memory requirement as well as memory access. If the functionality of incorporating static objects into the background is added, the memory requirement of the previous version [65] is 7.25 bytes per pixel, which is more than the 6.25 required by the improved version [4].

We then modified and improved our previous work [4, 65] in terms of the number of memory accesses, number of instructions, and thus speed. The decision about whether a pixel is a foreground pixel is made differently and more efficiently. In addition, we implemented our previous algorithm [4], and the version presented in this chapter on the microprocessor of an actual embedded smart camera, and compared them in terms of processing speed, and the operating current of

the camera board. To measure the current, we used a precise oscilloscope and a 1-ohm resistor configuration placed at the input of the supply source. Also, we compare the proposed method in detail with other state-of-the-art background subtraction algorithms in terms of their memory requirement, accuracy and processing time. We ran the presented algorithm and the other methods on challenging outdoor and indoor video sequences, and here show the results obtained with nine different videos. These video sequences include videos of two different windy scenes, two different rainy scenes, a video of a fountain, a video of a lake and videos of two different streets. We also present the Receiver Operation Characteristics (ROC) curves for different background subtraction algorithms.

The rest of this chapter is organized as follows: The details of the proposed method is explained in Section 3.2. Specifically, building of the background model, counters and how they are updated, salient foreground detection, adaptive background model update, and adaptive number of memory accesses and instructions are described in Sections 3.2.1 through 3.2.5, respectively. Experimental results are presented in Section 3.3, and the chapter is concluded with a summary in Section 3.4.

3.2 Proposed Method

The proposed algorithm employs a temporal difference method until a complete background model is built. It differentiates between salient and non-salient motion based on the history of a pixel's location, and by incorporating neighborhood information. At each frame, each pixel is classified either as a background or a foreground pixel, and its state is set to be 0 or 1, respectively. For a

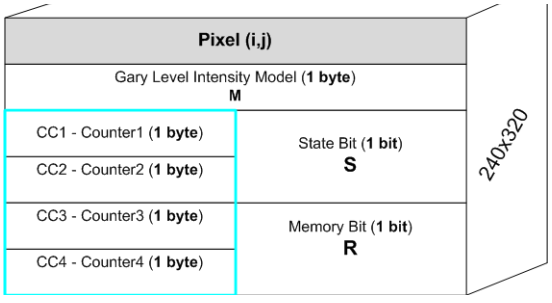


Figure 3.1: Memory required for a pixel with the proposed method

pixel at location (i, j), a counter $h(i, j)$ holds the number of changes in the state of this pixel during the last 100 frames, i. e. the counter $h(i, j)$ keeps the number of times a pixel's state changes from 0 to 1 or vice versa. The stability of a pixel at location (i, j) is determined by this counter $h(i, j)$. The motivation is that the lower the value of $h(i, j)$, the more stable and reliable that location is, or vice versa. Until a complete background model is built, the state of a pixel is determined by using temporal difference.

The algorithm has an adaptive background model update rate. If a pixel location is determined to be consistently stable and very reliable, then the value of this pixel is incorporated to the background model with a higher weight. Instead of treating each pixel independently, information from neighboring pixels is used to differentiate between salient and non-salient motion, and in turn to classify a pixel as a foreground or background pixel. The details of the proposed algorithm will be explained by referring to the pseudo-code provided in Table 3.1. Additionally, Figure 3.1 shows the amount of memory required for a pixel with the proposed method.

3.2.1 Building the Background Model

A temporal difference-based method is used to build a complete background model, M. In order to detect slow motions or stopping objects, a weighted accumulation, I_t^{ac}, is used for temporal difference. At pixel location (i, j), I_t^{ac} is defined as:

$$I_t^{ac}(i,j) = (1 - w_{ac})I_{t-1}^{ac}(i,j) + w_{ac}|I_t(i,j) - I_{t-1}(i,j)| \tag{3.1}$$

where t is the current frame number, I_t is the current image frame, and w_{ac} is the weight. I_0^{ac} is set to be an empty image, and w_{ac} is set to be 0.5.

At the beginning, the background model is an empty array. In Table 3.1, M denotes the background model, and $s(i, j)$ denotes the state of a pixel at location (i, j), which is defined as:

$$s(i,j) = \begin{cases} 1 & I_{diff}(i,j) > \tau \\ 0 & Otherwise. \end{cases} \tag{3.2}$$

Set $M(i,j) = -1$ for all i,j; Set $s(i,j) = 0$, $R(i,j) = 0$ for all i,j;
Set $I_1 =$ first frame; Set $model\_complete = false$;
for every frame $t > 1$
  Set $I_t = t^{th}$ frame, and set $I_{outp}(i,j) = 0$ for all i,j;
  if $\exists\ i,j$ for which $M(i,j) = -1$
    compute I_t^{ac}; set $I_{diff} = I_t^{ac}$; $\tau = \tau_d$;
  else
    set $model\_complete = true$;
    compute $I_t^{md} = |I_t - M|$; set $I_{diff} = I_t^{md}$; $\tau = \tau_m$;
  for all i,j
    if $I_{diff} > \tau$
      if $(s(i,j) == 0)$, set $s(i,j) = 1$; update CC_k, for $k \in \{1 \ldots 4\}$;
    else
      if $(s(i,j) == 1)$, set $s(i,j) = 0$; update CC_k, for $k \in \{1 \ldots 4\}$;
      if $model\_complete == false$
        if $M(i,j)$ is not equal to -1
          $M(i,j) = \alpha I_t(i,j) + (1-\alpha)M(i,j)$;
        else
          $M(i,j) = I_t(i,j)$;
    if $model\_complete == true$
      if $I_t^{md}(i,j) > \tau$
        if $R(i,j)$ == 0
          Compute $h(i,j) = \sum_{i=1}^{4} CC_i$;
          if $h(i,j) < \tau_p$
            Set $I_{outp}(i,j) = 1$; Set $R(i,j) = 0$;
          else
            Set $neighb(i,j)$ to be 3×3 neighb. of $h(i,j)$
            if N $> 0.7(2w+1)^2$
              $I_{outp}(i,j) = 1; R(i,j) = 0$;
            else
              $M(i,j) = \alpha I_t(i,j) + (1-\alpha)M(i,j)$;
        else
          $I_{outp}(i,j) = 1; R(i,j) = 0$;
      else
        Reset $FG\_duration(i,j) = 0$;
        if t is a multiple of 25
          if $R(i,j)$ == 0
            Compute $h_{t-50}^{t}(i,j)$;
            if $(h_{t-50}^{t}(i,j) \leq 2)$, set $R(i,j) = 1$;
        if $R(i,j)$ == 1
          $M(i,j) = 0.5 I_t(i,j) + 0.5 M(i,j)$;
        else $M(i,j) = \alpha I_t(i,j) + (1-\alpha)M(i,j)$;
      if $I_{outp}(i,j) == 1$ and t is a multiple of 100
        Create and/or increase $FG\_duration(i,j)$;
        if $100 \times FG\_duration(i,j) > T$
          $M(i,j) = 0.5 \times I_t(i,j) + 0.5 \times M(i,j)$;
  if $model\_complete == false$
    Set $I_{t-1} = I_t$;
return I_{outp}

Table 3.1: Salient foreground detection algorithm

During the model building period, $I_{diff}(i,j)$ is set to be $I_t^{ac}(i,j)$, and τ is set to be $\tau_d = 15$. After the background model M is complete, τ is set to be $\tau_m = 25$, and $I_{diff}(i,j)$ is obtained by using the model M, as explained below. Since temporal difference is based on consecutive frames, and tends to give smaller differences, τ_d has a smaller value than τ_m.

When $s(i,j) = 1$, i. e. when the pixel is classified as foreground, this pixel location in the model ($M(i,j)$), is not updated/changed. On the other hand, if $s(i,j)$ is 0, the current value of $M(i,j)$ is checked. If $M(i,j)$ is not filled yet, $M(i,j)$ is set to be $I_t(i,j)$, which is the current pixel value. If $M(i,j)$ is already filled, its value is set to be $M(i,j) = 0.95M(i,j) + 0.05I_t(i,j)$. Thresholded temporal difference cannot detect all the pixels of a moving object as depicted in Figure 3.2. Thus, existing model is given a 95% weight not to corrupt it by direct use of the values coming from the internal region of a moving object. As moving objects in the scene change their location, the M will gradually be filled as seen in Figure 3.3. The process of building the background model ends when no empty location is left in M. When M is complete, temporal difference is not used anymore.

Figure 3.2: Output of the temporal difference after applying a threshold.

3.2.2 Updating the Counters

As stated previously, the stability of a pixel at location (i,j) is determined by a counter $h(i,j)$, which keeps the number of times a pixel's state changes from 0 to 1, or vice versa, in the last 100

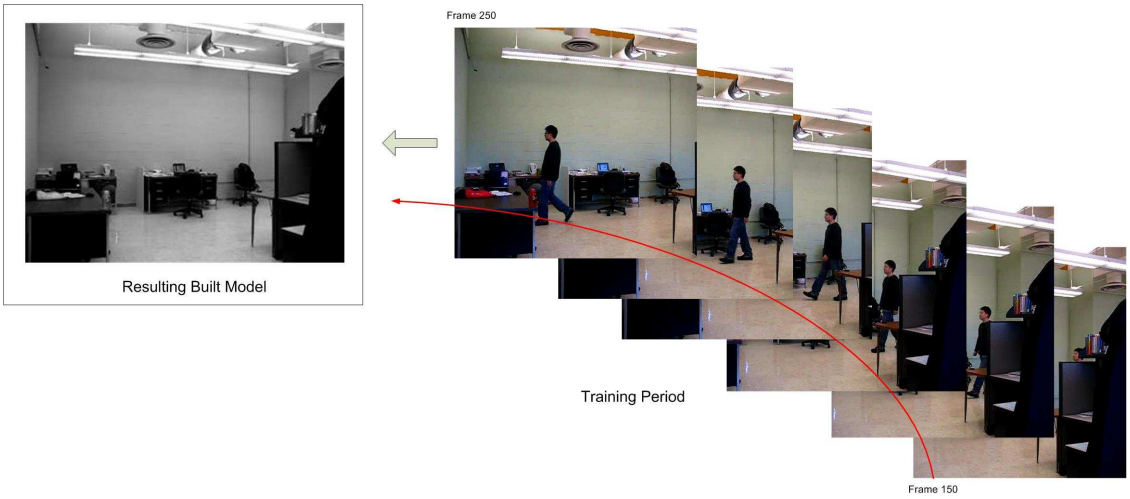


Figure 3.3: The background model is gradually built as moving objects change their location.

frames. The motivation is that the lower the value of $h(i,j)$, the more stable and reliable that pixel location is.

Although it may look like an implementation detail, the computation of $h(i,j)$ for each pixel at each frame is worth emphasizing since we want fast processing, and we need to take the memory requirements into account. At any frame t, we want the number of changes in a pixel's state between frames $t-100$ and t. This requires saving the frame number each time a change occurs in a pixel's state. For locations with non-salient motion, this, in turn, requires an array with potentially high dimension for each pixel. Instead, we quantize the 100-frame window into 4 intervals, and keep a counter $CC_k(i,j), k \in \{1,\ldots,4\}$, for each interval for pixel (i,j). The approach is illustrated in Figure 3.4. Between frames 1 and 25, the counter CC_1 is increased each time the pixel's state changes, between frames 26 to 50 the counter CC_2 is increased etc. At the end of the 100-frame period, the counter CC_1 is reset and its value is increased until frame 125 is reached, and the other counters are updated similarly. This avoids saving the frame instances of each change. Then,

$$h(i,j) = \sum_{k=1}^{4} CC_k(i,j).$$

Counters $h(i,j)$ are updated during the building of the model as well. Figure 3.5 shows a

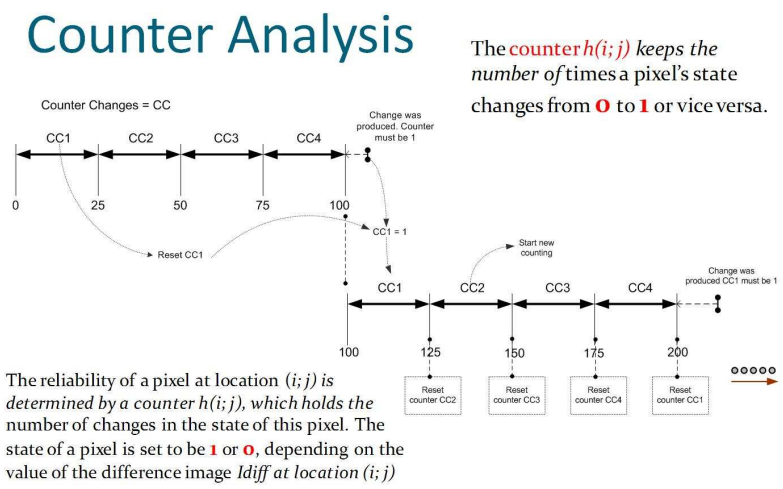


Figure 3.4: Illustration of how $h(i, j)$ is computed.

frame from a video containing a fountain, and a plot of the counter values $h(i, j)$ for different pixel locations (i, j). As can be seen, the counters are higher around the outer boundaries of the multiple fountains, where the water is constantly moving and splashing. The high counters indicate regions with low reliability and non-salient motion.

It should be noted that this approach provides only an approximation of the number of changes in a pixel's state without having to save the frame numbers of every state change. For instance, at frame 101, it gives the number of changes that happened between frames 25 and 101. Other approaches can be used, and have been tried, that can give better approximations. However, they either require introducing additional variables, and/or additional instructions, and thus increase the memory requirement and decrease the algorithm speed. The presented approach is adapted for small memory requirement and better computational speed.

3.2.3 Salient Foreground Detection

As can be seen in Table 3.1, after the background model is built, then the difference image is set to be $I_{diff} = I_t^{md} = |I_t - M|$.

If $I_t^{md}(i, j) \leq \tau$, then the pixel location (i, j) is classified as background. On the other hand, as opposed to many traditional model-based background subtraction approaches, in the proposed

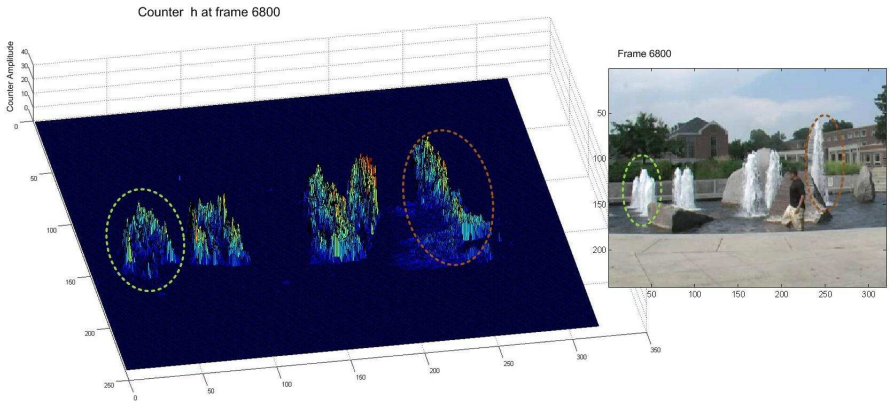


Figure 3.5: Original frame and the plot of the counter values $h(i,j)$ for different pixel locations (i,j). Higher values correspond to outer boundaries of multiple fountains, indicating regions with low reliability and non-salient motion.

scheme, satisfying $I_t^{md}(i,j) > \tau$ is not enough for the pixel location (i,j) to be classified as foreground. Instead, reliability constraints are employed to differentiate between salient and non-salient motion. A pixel location satisfying $I_t^{md}(i,j) > \tau$ is classified as foreground only if its counter $h(i,j)$ satisfies $h(i,j) < \tau_p$, where $\tau_p = 15$ is the percentage threshold. The reasoning is that if $h(i,j) < 15$, then it means that the state of the pixel at this location changed less than 15% of the time during the last 100 frames making this location a reliable one. In other words, this location is not likely to be in a non-salient motion region. Thus, the intensity difference greater than τ is caused by a salient motion with high probability.

If $I_t^{md}(i,j) > \tau$ *and* $h(i,j) \geq \tau_p$, then we do not classify this location as background right away. We take a $(2w+1) \times (2w+1)$-window neighborhood, where $w = 1$, around location (i,j) and check the h counter for all the neighbors. In Table 3.1, N is the number of neighbors whose counter h is less than τ_p. If the majority of the neighbors (more than 70%) have a low counter, i. e. $h < \tau_p$, then location (i,j) is set to be a foreground pixel or vice versa. This way, we take into account the fact that neighboring pixels are not independent from each other. We obtain information from neighbors, which increases accuracy and robustness.

3.2.4 Adaptive background model update

In order to adapt to changes in the environment, such as lighting changes, the background model needs to be updated over time. We perform the update of the background model M in a selective way, and with an automatically adaptive rate. The motivation is that when a pixel's location is deduced to be consistently reliable and stable, then the value at that location is incorporated into the background model with a higher weight.

If $I_t^{md}(i,j) \leq \tau$, then the algorithm concludes that it is safe to update the background model at this location. However, by looking at the summary of the recent past of a pixel, a higher weight can be given to the current pixel value, and better adapt to faster changes in the background. In other words, the background update rate an automatically adaptive.

The very compact *summary* of a pixel's history is formed as follows: Rather than saving many values for each pixel location, such as averages for three color values, multiple Gaussian distribution means and variances, multiple codewords with multiple entries, we use two of the four counters (CC_k, $k \in \{1,\ldots,4\}$) corresponding to the last 50 frames. Let $h_{t-50}^t(i,j)$ denote the sum of these two counters. Thus, $h_{t-50}^t(i,j)$ holds the number of state changes at pixel location (i,j) during the last 50 frames. If $h_{t-50}^t(i,j) \leq 2$, it means that the state of this pixel has changed only two times or less during the last 50 frames, i.e. this location is very reliable. We perform this check every 25 frames, and if the condition is satisfied, we set the boolean variable $R(i,j)$, which is a reliability flag, to be 1. This location is then incorporated to the background model with a 50% weight.

On the other hand, if $I_t^{md}(i,j) \leq \tau$ (Figure 3.6) and $R(i,j)$ is equal to 0, then 95% and 5% weights are given to the existing model value and the current pixel value, respectively. Figure 3.7 shows unreliable pixel locations in a parking lot area. They are produced due to swaying tress and sun reflections on the buildings' roofs. If a pixel at location (i,j) is classified as a foreground pixel, then $M(i,j)$ is not updated, which prevents corrupting the existing model. However, if a pixel location (i,j) is classified consecutively as foreground for a specified period of time (T) due to a static foreground object, then we start to push this location to the background by giving it 50%

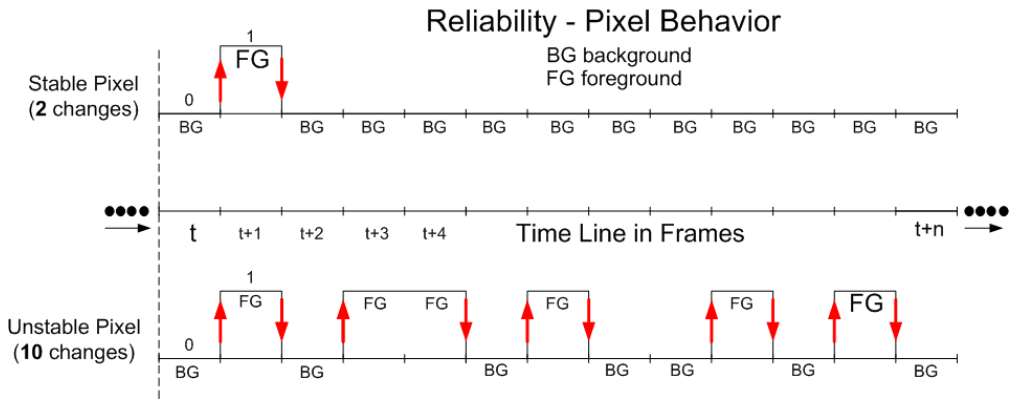


Figure 3.6: Illustration of the behavior of a pixel's location (i, j) in both reliable and unreliable cases

weight. T is set by the user, and determines how much time a stopped object should be static to be considered as part of the background.

3.2.5 Adaptive number of memory accesses and instructions

In Section 3.2.4, we described how we set the value of $R(i, j)$. If $I_t^{md} \leq \tau$, and current $R(i, j)$ is 0, and the frame number is a multiple of 25, we compute the value of $h_{t-50}^t(i, j)$. A small $h_{t-50}^t(i, j)$ indicates that the pixel's state has not changed much in the last 50 frames, and thus $R(i, j)$ is set to be 1.

At the beginning, for each pixel, 1 byte is allocated for each CC_k, where $k \in \{1, \ldots, 4\}$, 1 byte for the value saved in $M(i, j)$, 1 byte for the previous frame value, 1 bit for the state variable $s(i, j)$, and 1 bit for the reliability flag $R(i, j)$ making the total memory allocation 50 bits per pixel. After the background model is built, the pixel values of the previous frame are no longer needed. Instead, the memory allocated for the previous frame values is used for the $FG\_duration$ variable.

If the value of $R(i, j)$ is 1, this indicates that this pixel is a very reliable and stable background pixel. With the presented method, first type of saving occurs when there is a foreground object in the scene covering reliable background pixels. When $I_t^{md}(i, j) > \tau$, $h(i, j)$ is not calculated for very reliable background pixels, i.e. pixels for which $R(i, j)$ is 1. The reasoning is the following: $h(i, j)$ is employed to determine the stability of a pixel by looking at its state changes in the last 100

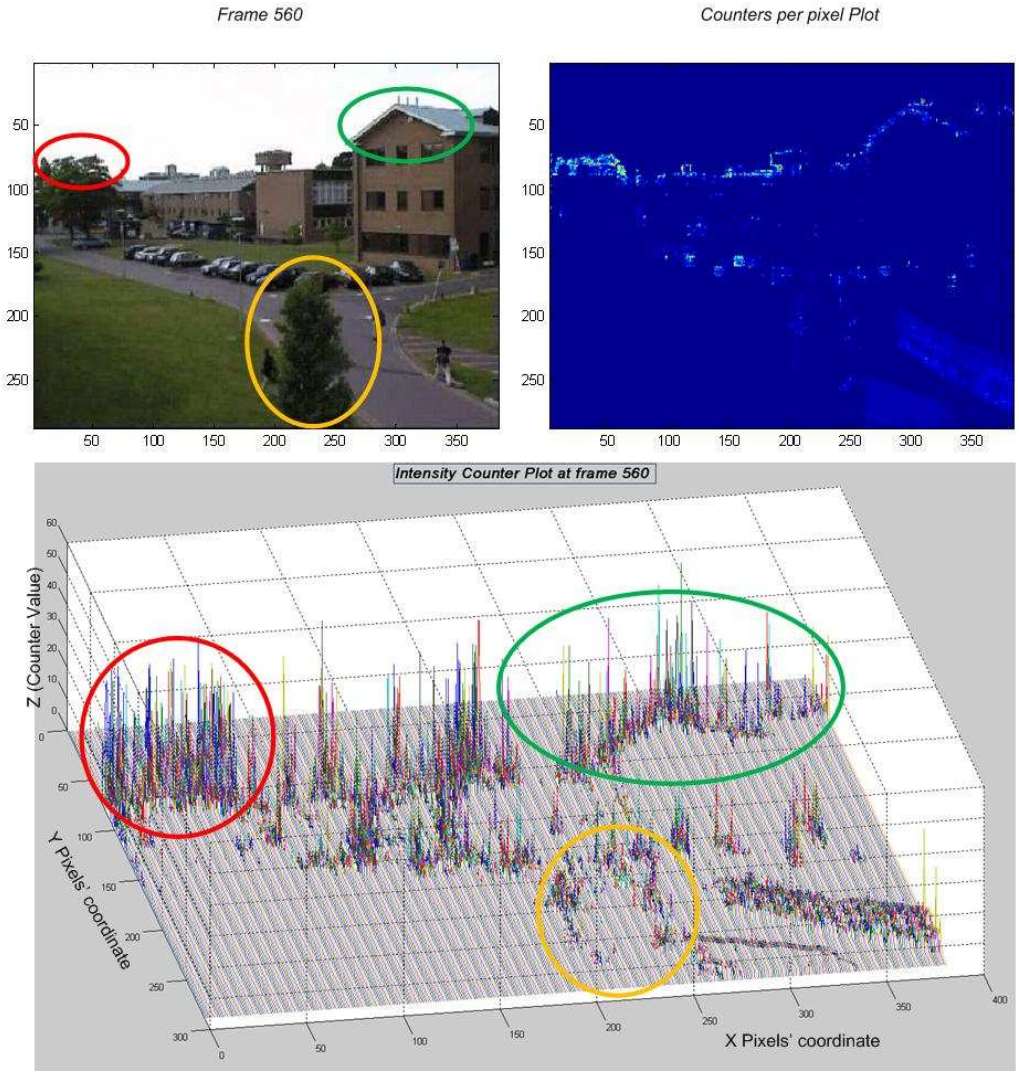


Figure 3.7: Illustration of unreliable areas due to swaying tress and sun reflections (circled). Large peaks revel them reporting high counts kept in $h^t_{t-50}(i,j)$.

frames. If $R(i,j)$ is 1, it is already known that this location is very reliable, thus we do not need to calculate and check the value of $h(i,j)$. In addition, we do not need to check the counters of the neighboring pixels either. This provides significant savings in terms of the number of memory accesses and instructions

The second type of savings occurs every 25 frames. If $R(i,j)$ is currently 1, then there is no need to compute $h^t_{t-50}(i,j)$, which provides additional savings. The detailed comparison of the method presented here and its previous version presented in [4], in terms of the processing speed,

will be presented in Section 3.3.

As described above, for very reliable and stable background pixels $R(i, j)$ is set to 1. Thus, the plot of the number of pixels, whose reliability flag $R(i, j)$ is 0, versus the frame number serves as a tool for activity summary. The changes and peaks in this plot will indicate the portions of the video with activity. Figures. 3.8 - 3.11 show these plots obtained for different video sequences.

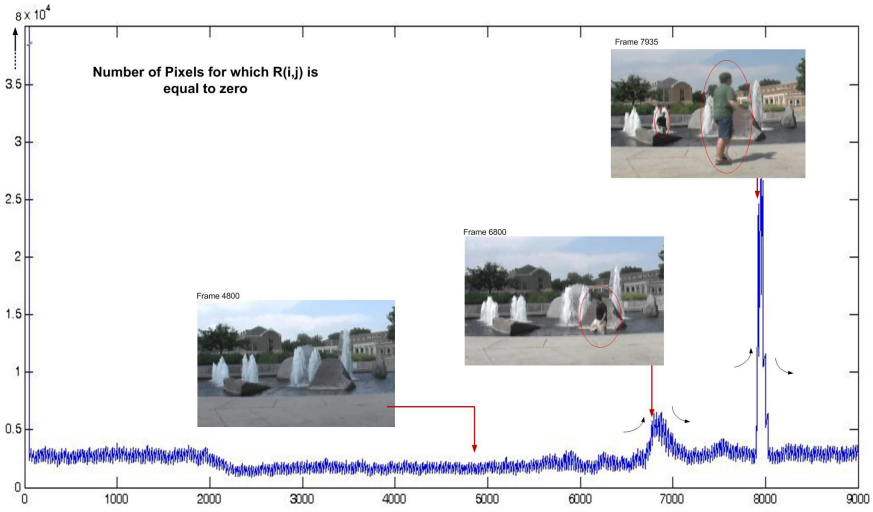


Figure 3.8: Video of a fountain: number of pixels with $R(i, j) = 0$ vs. the frame number plot.

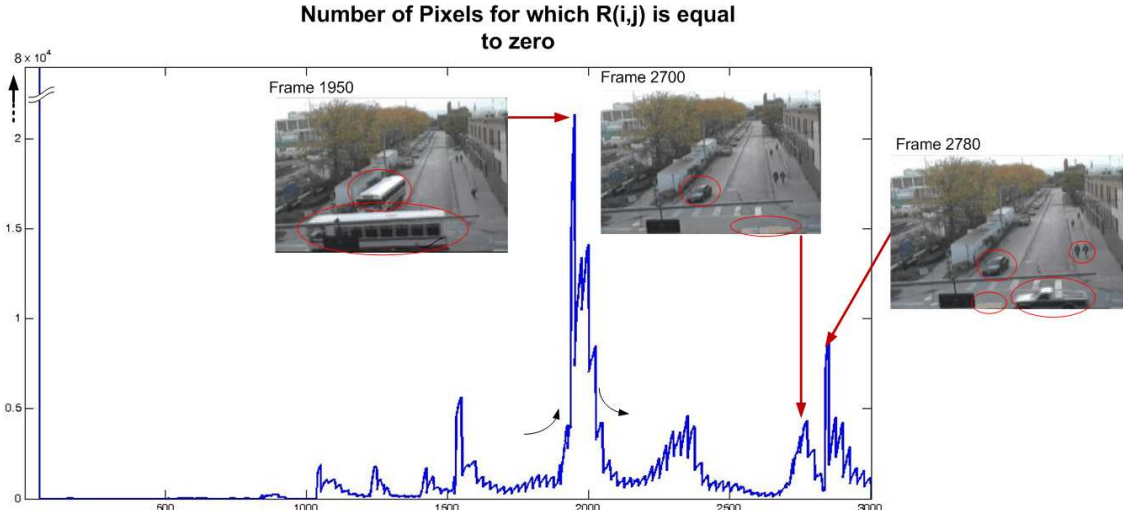


Figure 3.9: Traffic light sequence: number of pixels with $R(i, j) = 0$ vs. the frame number plot.

Figure 3.8 shows the number of pixels with $R(i, j) = 0$ for a video sequence of a fountain.

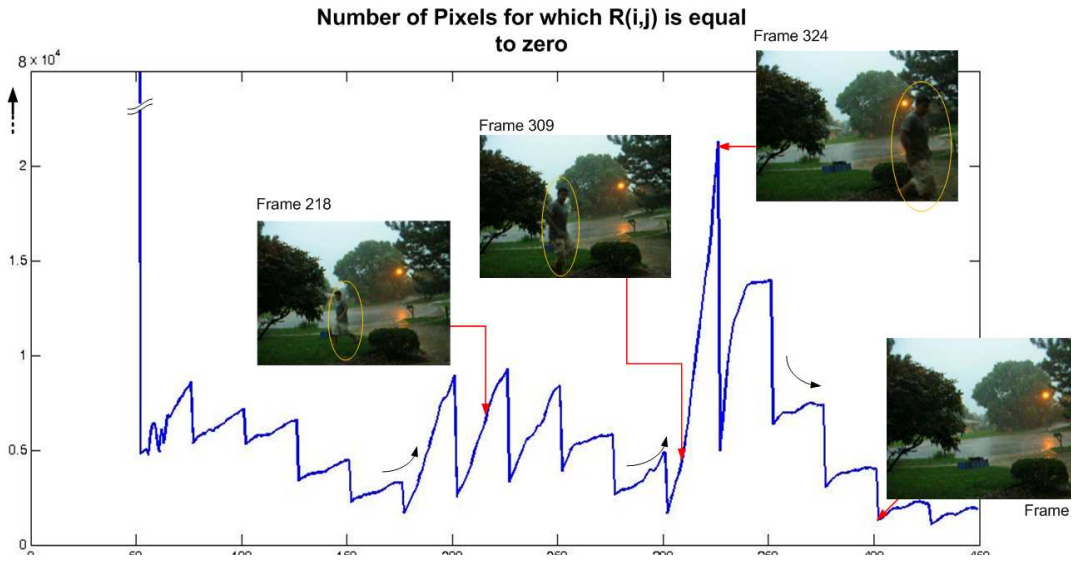


Figure 3.10: Rain sequence: number of pixels with $R(i,j) = 0$ vs. the frame number plot.

Frames 1-100 correspond to the model building period, during which $R(i,j) = 0$ for all the pixels. After the model is built, and stable background pixels are determined, the number of pixels whose $R(i,j)$ is 0 drops significantly to about 2500 pixels per frame, and it remains around this value until some activity starts in the scene. For example, at frame 6800, there is a person walking in front of the camera. This creates a peak in the plot. A similar situation occurs at frame 7935, where the detected person is closer to the camera and thus its size is larger than the previous scenario. A bigger object covers more pixels, and causes them to be classified as foreground pixels. Thus, the $R(i,j)$ is set back to zero for these affected pixels. This is why the peak at frame 7935 is higher than the one at frame 6800.

The savings provided by the proposed method increases with increasing number of reliable background pixels, i.e. pixels whose $R(i,j)$ is 1. In Figure 3.8, low values correspond to frames with small number of unreliable pixels, and thus more number of reliable background pixels. Thus, in these portions of the video, the number of memory accesses, and the number of instructions will be less with the proposed method. More speed analysis will be provided in Section 3.3.

Another interesting video sequence captured at a traffic light shows a continuous flow of cars going in the north–south direction. The number of pixels with $R(i,j) = 0$ vs. the frame number

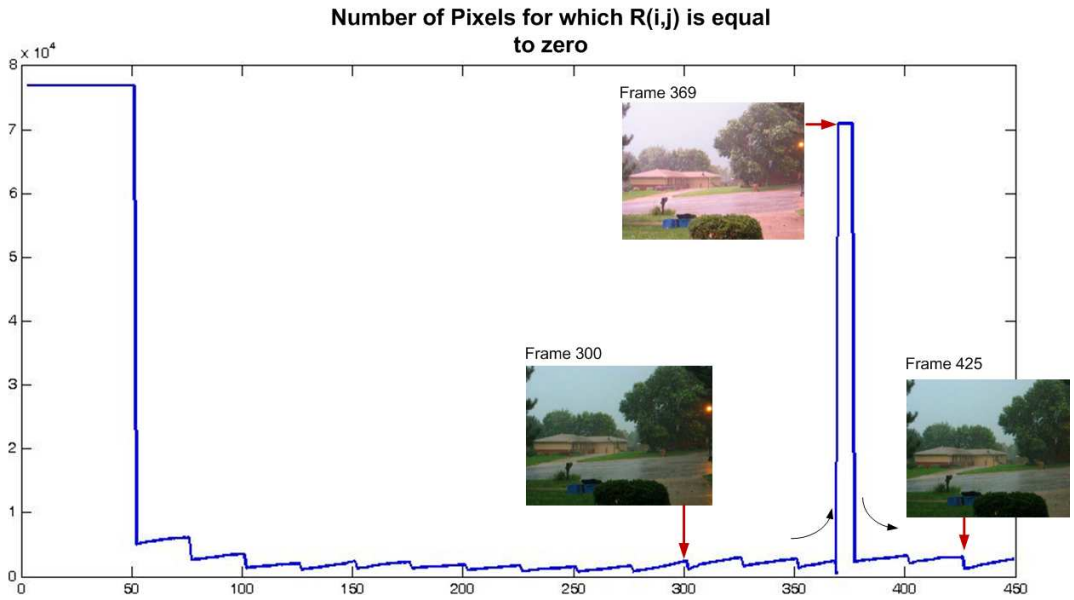


Figure 3.11: Rain sequence: number of pixels with $R(i,j) = 0$ vs. the frame number plot.

plot for this sequence is displayed in Figure 3.9. At frame 1950, there are two buses in the scene occupying a larger area than the smaller sedans and trucks that are seen at frames 2700 and 2780. The highest peak shown in the plot corresponds to this instance.

Figure 3.10 shows the number of pixels with $R(i,j) = 0$ for a rainy scenario in which a person goes through the view of the camera twice. The first time, the individual was farther away from the camera while in the second pass, he is closer to it resulting in a higher number of pixels with reliability bit set to zero memory. A more extreme example is presented in Figure 3.11, in which a sudden lightning causes a complete intensity change in the whole image at frame 369. As a result, a large peak is observed in the plot. After this, the total number of pixels with $R(i,j) = 0$ drops again.

3.3 Experimental Results

In this section, the proposed method is compared with five other background subtraction methods, including its previous version presented in [65], on 11 different video sequences with varying levels of difficulty. Henceforth, these algorithms will be referred to as follows: ALW: Adaptive

	CB	Org-MoG	EB	LW [65]	ALW
Bytes per pixel	91	32	28	7.25	6.25

Table 3.2: Memory requirement for the data saved for each pixel for different background subtraction methods (for one color channel)

lightweight algorithm (the method presented in this chapter), LW: lightweight algorithm [65], Org-MoG: Original MoG [16], Impr-MoG: Improved MoG [33], CB: Codebook [32], EB: Eigenbackground [21]. In addition, we provide a detailed comparison of the proposed (ALW) method with other state-of-the-art background subtraction algorithms in terms of their memory requirement, accuracy and processing time. We also present the Receiver Operation Characteristics (ROC) curves for different background subtraction algorithms.

Since embedded smart cameras have limited processing power and memory, it is very important to design lightweight algorithms that require less memory for storage. First, the proposed algorithm is compared with others in terms of the memory requirement for the data saved for each pixel. The algorithm was run on the red channel, and its memory requirement is detailed in Section 3.2.5. For different background subtraction techniques, Table 3.2 lists the number of bytes necessary for the data saved for each pixel, for one color channel.

The memory requirements for the other background subtraction methods are computed as follows. Let n denote the number of Gaussian distributions used in Org-MoG. Org-MoG requires two floating point numbers per Gaussian distribution, per color channel, per pixel (one for the mean and one for the variance of a Gaussian distribution). It also requires $n-1$ many floating point numbers for the weights of distributions. Thus, if n is picked to be 3, eight floating point numbers are needed per color channel. If three color channels are used the memory required per pixel is 96 bytes. If one color channel is used, it is 32 bytes. Even if the mean for each distribution is rounded so that it can be represented by a byte, the memory requirement is still 23 bytes per color channel.

The codebook-based method (CB) uses 3 floating point numbers for the means of the RGB channels, 2 bytes for the minimum and maximum brightness values that the codeword accepted, 1 integer for the frequency of the codeword, 1 integer for the maximum negative run-length, and 2

integers for the first and last access times. Thus, the total memory needed is 22 bytes per codeword. If only one color channel it is 14 bytes per codeword. In [32], it is stated an average of 6.5 codewords is needed per pixel codebook. Thus, the average memory requirement per pixel is 91 bytes.

For the EB method, the memory requirement per pixel is the number of the best eigen-backgrounds. During the training time the method requires allocation for all the training images. In general, 7 floating point numbers are required per pixel. Thus, the memory needed is 28 bytes.

The LW algorithm presented in [65] requires 7.25 bytes per pixel when the functionality of pushing the static foreground objects to the background is incorporated. For different methods, Figure 3.12 shows a bar graph of the memory requirement (in bytes) per frame for a 240×320 frame.

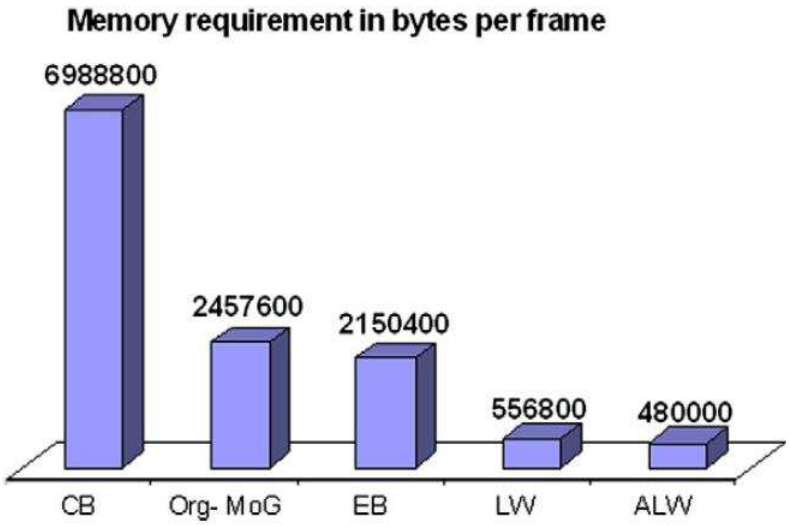


Figure 3.12: Per frame memory requirements of different background subtraction methods when one color channel is used.

The proposed method was tested on 11 challenging video sequences, and compared it with five other background subtraction methods including our previous work. It should be noted that all the displayed outputs below are the images obtained *without applying any* morphological or post-processing operations. All the results of our algorithm were obtained by using the same threshold values for all videos, specifically, $\tau_d = 15$, $\tau_m = 25$, $\tau_p = 15$, and $\alpha = 0.05$. Overall,

the proposed method requires the least amount of memory per pixel while providing better or comparable outputs at the same time.

Figures 3.20 and 3.22 display the outputs obtained on videos of two different windy scenes. All the algorithms were run on one channel except the CB and Impr-MoG. As can be seen, the proposed method provides the least amount of noisy pixels, and good detection at the same time.

Figures 3.23 and 3.24 show the outputs for challenging videos of rainy scenes. Again, the proposed method provides comparable if not better outputs compared to the other algorithms, while requiring the least amount of memory at the same time.

Figure 3.25 shows the outputs for another challenging video of a lake, where there are rippling water effects on the lake, and swaying trees in the background. Compared to Impr-MoG, EB and CB, the proposed method can differentiate the non-salient motion better. It gives the least amount of noisy pixels. The Org-MoG, on the other hand, has less noisy pixels than the proposed method. However, it misses the person and the dog, which should be detected as foreground objects. The outputs obtained on two other outdoor videos showing two different streets are shown in Figures 3.26 and 3.27.

The results displayed in Figure 3.28 were obtained from a video of a scene with a fountain, where the water level goes up and down. Moreover, during the video, lighting changes due to moving clouds, as seen in Figure 3.28. As the figure illustrates, since the eigenbackground method does not update the background model, it cannot handle the lighting change. The improved MoG method cannot detect most of the foreground pixels. The proposed method provides good detection, and can eliminate most of the non-salient motion caused by the fountains.

Figure 3.19 displays the outputs obtained from an indoor sequence. Although the video was captured indoors, the flickering of the overhead lights affects the performance of the algorithms.

Figures 3.21 and 3.22 present common surveillance scenarios. Figure 3.21 shows the output of the algorithm on an airport video during regular daily activities while Figure 3.22 was captured at a parking lot. The latter shows the robustness of the algorithm against non-salient motion introduced by swaying trees.

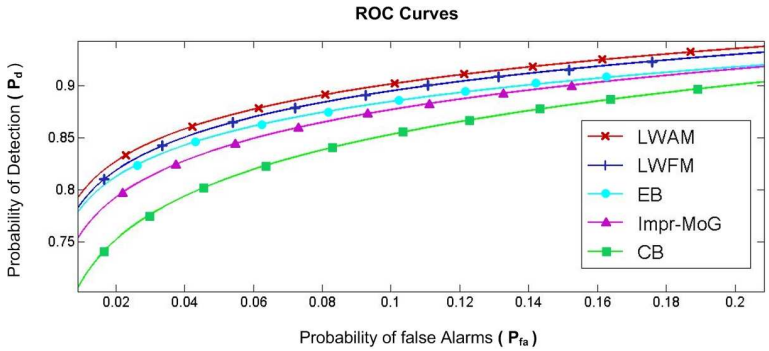


Figure 3.13: ROC curves of different background subtraction methods.

We also compared the processing times of these algorithms on a PC. However, the codes for the ALW, EB, CB and Org-MoG are written in MATLAB, whereas the code for Impr-MoG is written in C. Also, these codes are not equally optimized. Hence, it is difficult to make a comparison of the processing times. We will list the frames/sec rates to give the reader a general idea. The algorithms were run on a video with 240×320 frame size. ALW and EB run at 35 frames/sec and 49.5 frames/sec, respectively, in MATLAB. It should be noted that EB does not update the background model. The Org-MoG and the CB run at 0.2 frames/sec and 0.24 frames/sec, respectively, in MATLAB. The C++ version of the CB method runs at around 55 frames/sec, and the Impr-MoG runs at 59 frames/sec in C.

In addition, we performed a comparison of the different algorithms in terms of their probability of detection (P_d) and probability of false alarm (P_{fa}) rates, and plotted their Receiver Operation Characteristics (ROC) curves [1, 10, 13, 31]. ROC curves are employed often when comparing background subtraction algorithms. Alongside the outputs obtained by different algorithms, ROC analysis provides us with a quantitative comparison. We obtained the ground truth for the foreground objects manually, and plotted the ROC curve for each algorithm. These curves are displayed in Figure 3.13. As can be seen, for the same P_d rate, the proposed method has the least P_{fa}, and for the same P_{fa} rate it has the highest P_d.

As described above, compared to the initial version presented in [4], the method presented here provides more savings, in terms of number of memory accesses and number of instructions, and

thus speed and efficiency, in two different ways. To demonstrate these savings, we performed three different experiment

First two experiments compare the processing speed of two versions when a foreground object is in the scene. As discussed in detail above, with the method presented here, first type of savings occurs when there is a foreground object in the scene covering reliable background pixels. When $I_t^{md}(i,j) > \tau$ and $R(i,j) = 1$, $h(i,j)$ is not calculated for these reliable background pixels, i.e. pixels for which $R(i,j)$ is 1. In addition, we do not need to check the counters of the neighboring pixels either. This provides significant savings in terms of the number of memory accesses and instructions. For these experiments, we imported and implemented the two versions of our algorithm on an embedded smart camera node.

Figure 3.14 shows a plot of the processing time (in milliseconds on the microprocessor of the camera) for two different versions during an interval when there is an object in the scene. The blue and red plots correspond to the methods presented in this chapter and in [4], respectively. As can be seen, on the average, the method presented here performs 2.82 milliseconds faster per frame. Also, the speed gain provided by this method increases with increasing object size and also increasing number of objects in the scene as seen in Figure 3.15. Since the foreground object is larger the proposed method runs 4.5 milliseconds faster per frame on the average. This gain is obtained in part by not accessing CC_k, $k \in \{1, ..., 4\}$, and not performing $\sum_{k=1}^{4} CC_k$ for reliable background pixels.

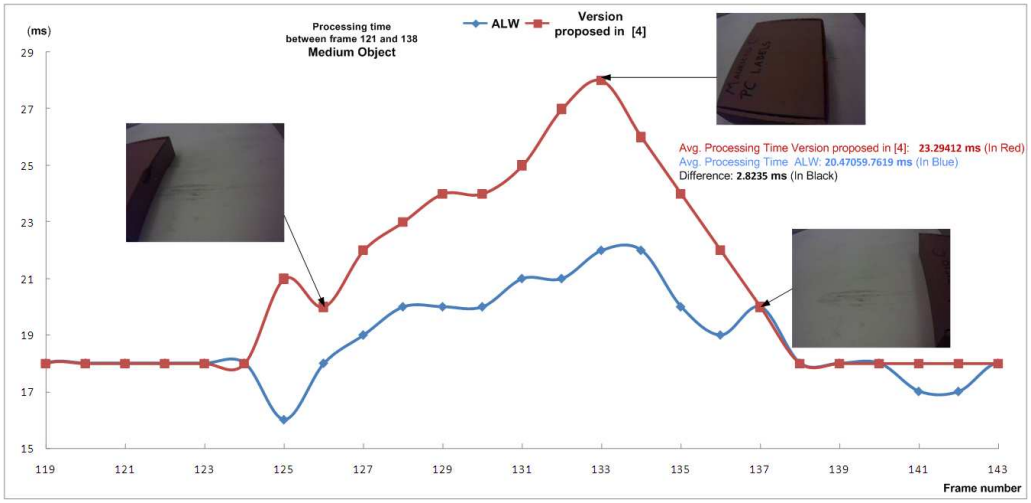


Figure 3.14: Processing time (ms) versus the frame number for two different versions of the algorithm when there is a foreground object in the scene.

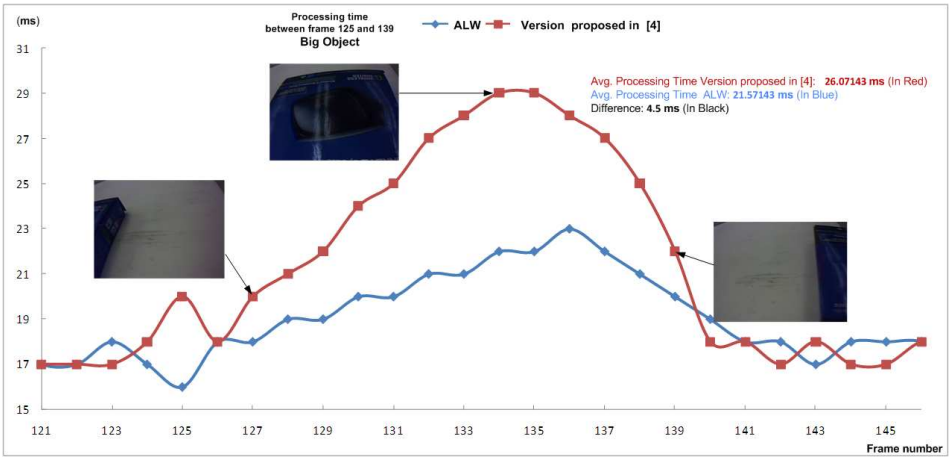


Figure 3.15: Processing time (ms) versus the frame number for two different versions of the algorithm when there is a foreground object in the scene.

In the second experiment, we ran the different versions on the embedded smart camera board, and measured the operating current of the board. The operating current increases or decreases based on the workload of the processor (number of instructions per task), the supply voltage source and the frequency at which the processor is working. To measure the current, we used a precise oscilloscope and a 1-ohm resistor configuration placed at the input of the supply source (battery

pack) as shown in Figure 3.16.

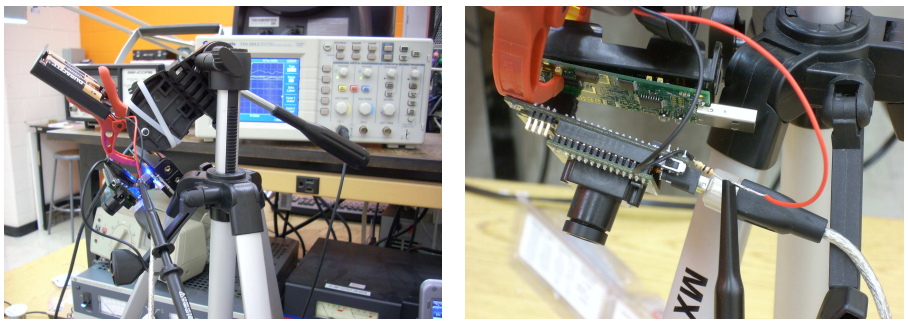


Figure 3.16: Camera setup ready to perform the required energy measurements.

Figure 3.17 shows the variations in the current during the processing of three consecutive frames containing a foreground object. As can be seen the proposed method (blue plot) finishes processing the first frame 8 milliseconds earlier than the method presented in [4]. It also finishes processing the following two frames 7 and 8 milliseconds faster.

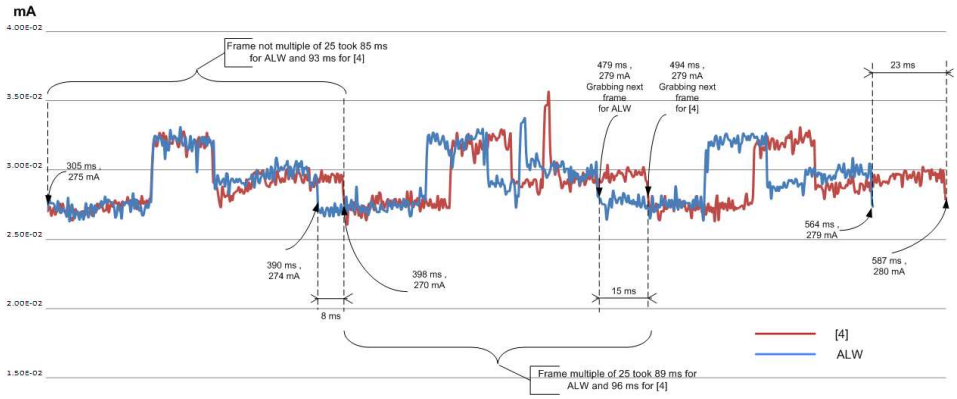


Figure 3.17: Variations in the operating current during the processing of three consecutive frames containing a foreground object. The method presented in this chapter (blue plot) is faster than the method presented in [4] (red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of Casares et al. [4])

The proposed method provides second type of savings, over our previous work, every 25 frames. As described before, if $R(i,j)$ is currently 1, then there is no need to compute $h^t_{t-50}(i,j)$, which provides additional savings. In order to demonstrate these savings, we performed another experiment and measured the operating current of the camera board over time with an oscilloscope.

To measure the gain obtained only from not calculating $h^t_{t-50}(i,j)$ at every 25 frames, we used an empty scene. Figure 3.18 shows the waveforms obtained. The blue and red plots correspond to the methods presented in this chapter and in [4], respectively. As can be seen, when the frame number is multiple of 25, the proposed method performs 5 milliseconds faster than the method in [4].

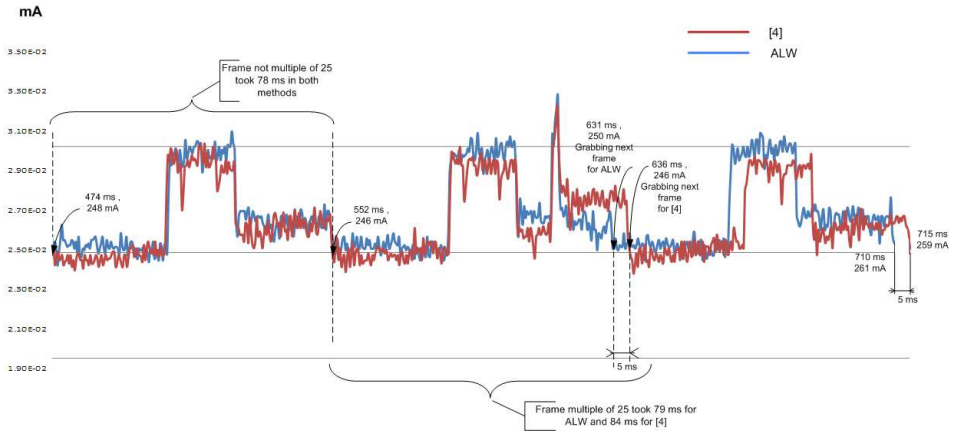


Figure 3.18: Variations in the operating current during the processing of three consecutive frames of an empty scene. The method presented in this chapter (blue plot) provides speed gaining at frame numbers that are multiple of 25. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of Casares et al. [4].)

3.4 Conclusions

We presented a lightweight salient foreground detection algorithm that is highly robust against challenging non-static backgrounds. Contrary to many traditional methods, the memory requirement for the data saved for each pixel is very small in the proposed algorithm, which is very important for portability to an embedded smart camera. Moreover, the number of memory accesses and instructions are adaptive, and are decreased even more depending on the amount of activity in the scene and on a pixel's history. Each pixel is treated differently based on its history, and instead of requiring the same number of memory accesses, and thus, instructions for every pixel, we require less instructions for stable background pixels. This, in turn, increases the processing speed. The algorithm achieves this without sacrificing accuracy. The plot of the number of unstable pixels at

each frame also serves as a tool to find the video portions with high activity.

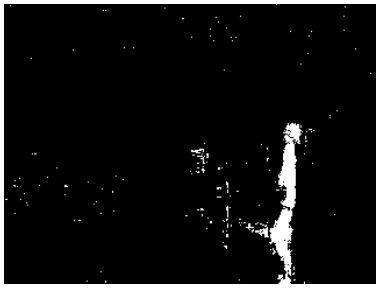
The proposed method selectively updates the background model with an automatically adaptive rate, thus can adapt to rapid changes. As opposed to traditional methods, pixels are not always treated individually, and information about neighbors is incorporated into decision making, which increases accuracy and robustness. The algorithm can use only intensity, or one color channel, and still provides very reliable results. The results obtained with nine different challenging outdoor and indoor sequences were presented, and compared with the results of different state-of-the-art background subtraction methods. All the results of our algorithm were obtained by using the same threshold values for all videos. The ROC curves of different background subtraction methods are also provided. The memory requirements of the different algorithms have been compared as well, and it has been shown that the proposed method requires the least amount of memory per pixel. The experimental results demonstrate the success of the proposed lightweight method in challenging situations such as scenes with water fountains, swaying trees, and strong rain.

The method presented in this chapter modifies and optimizes our previous work [4] in terms of the memory access, number of instructions, and thus, speed. The decision about whether a pixel is a foreground pixel is made differently and more efficiently. These methods were compared in terms of processing speed with three different experiments performed with an embedded smart camera running these algorithms. It was shown that the presented method runs faster on the smart camera nodes.

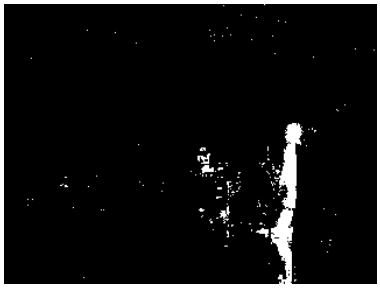
(a) Sample Frame

(b) Proposed method

(c) Previous method



(d) Mixture of Gaussians



(e) Improved MoG

(f) Eigenbackground

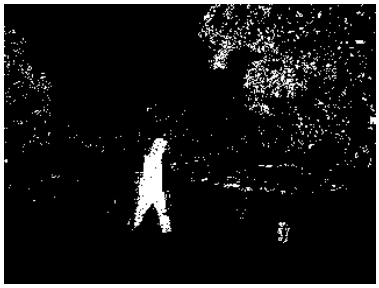
(g) CodeBook

Figure 3.19: Foreground detection results of different algorithms on a challenging indoor's video sequence with flickering lights. Outputs are obtained without morphological operations.

(a) Sample Frame

(b) Proposed method

(c) Previous method



(d) Mixture of Gaussians

(e) Improved MoG

(f) Eigenbackground

(g) CodeBook

Figure 3.20: Foreground detection results of different algorithms on a challenging video of a windy scene. Outputs are obtained without morphological operations.

(a) Sample Frame

(b) Proposed method

(c) Previous method

(d) Mixture of Gaussians

(e) Improved MoG

(f) Eigenbackground

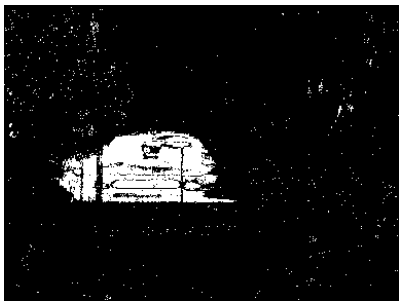
(g) CodeBook

Figure 3.21: Foreground detection results of different algorithms on a challenging video in a windy day at the Airport. Outputs are obtained without morphological operations.

(a) Sample Frame

(b) Proposed method

(c) Previous method



(d) Mixture of Gaussians

(e) Improved MoG

(f) Eigenbackground

(g) CodeBook

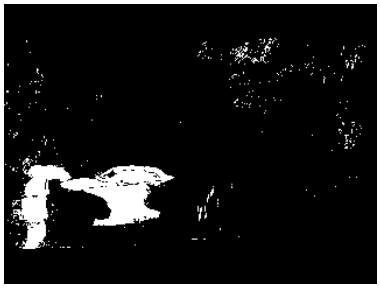
Figure 3.22: Foreground detection results of different algorithms on a challenging video of another windy scene in a parking lot. Outputs are obtained without morphological operations.

(a) Sample Frame

(b) Proposed method

(c) Previous method

(d) Mixture of Gaussians



(e) Improved MoG

(f) Eigenbackground

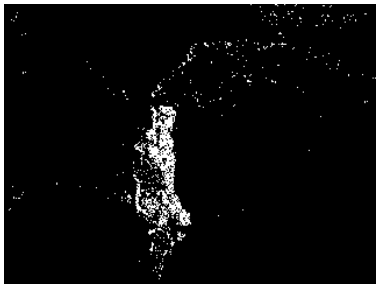
(g) CodeBook

Figure 3.23: Foreground detection results of different algorithms on a video of a rainy scene. Outputs are obtained without morphological operations.

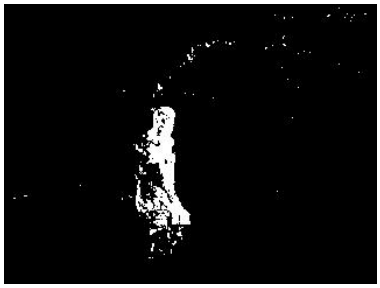
(a) Sample Frame

(b) Proposed method

(c) Previous method



(d) Mixture of Gaussians



(e) Improved MoG

(f) Eigenbackground

(g) CodeBook

Figure 3.24: Foreground detection results of different algorithms on a video of another rainy scene. Outputs are obtained without morphological operations.

(a) Sample Frame

(b) Proposed method

(c) Previous method

(d) Mixture of Gaussians

(e) Improved MoG

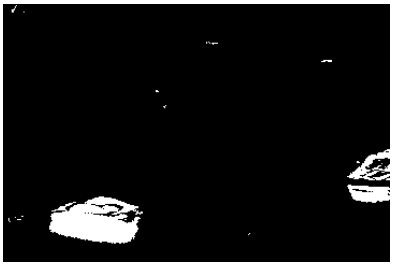
(f) Eigenbackground

(g) CodeBook

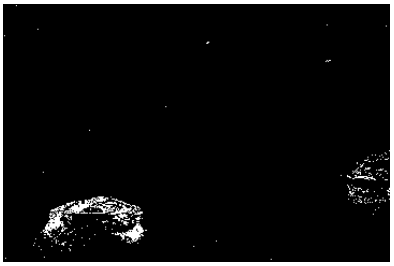
Figure 3.25: Foreground detection results of different algorithms on a challenging video of a lake. Compared to (eg), the proposed method can eliminate the non-salient motion better. Although (d) has less noisy pixels, it misses the person and the dog. Outputs are obtained without morphological operations.

(a) Sample Frame

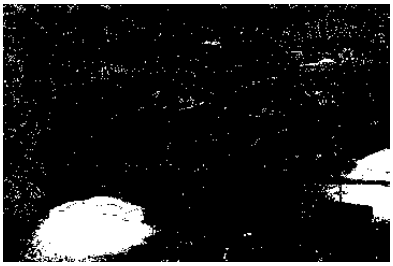
(b) Proposed method



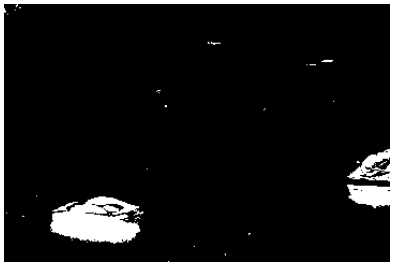
(c) Previous method



(d) Mixture of Gaussians



(e) Improved MoG



(f) Eigenbackground

(g) CodeBook

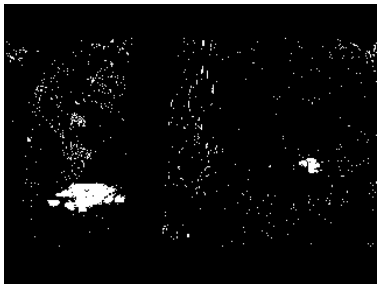
Figure 3.26: Foreground detection results of different algorithms on a video of a street. Outputs are obtained without morphological operations.

(a) Sample Frame

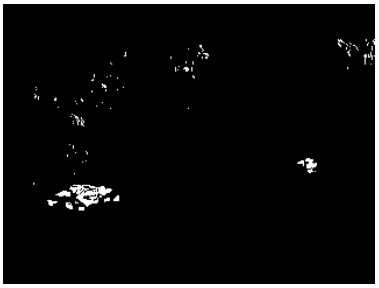
(b) Proposed method

(c) Previous method

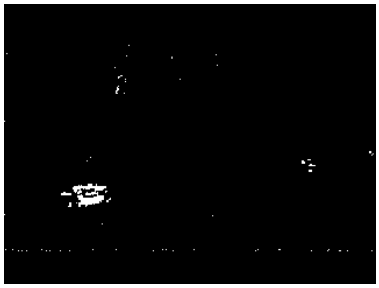
(d) Mixture of Gaussians



(e) Improved MoG



(f) Eigenbackground



(g) CodeBook

Figure 3.27: Foreground detection results of different algorithms on a video of a street. Outputs are obtained without morphological operations.

(a) Frame 6323

(b) Frame 7171

(c) Proposed method

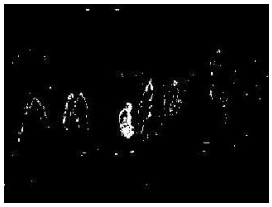
(d) Proposed method

(e) Previous method

(f) Previous method

(g) Mixture of Gaussians

(h) Mixture of Gaussians



(i) Improved MoG

(j) Improved MoG

(k) Eigenbackground

(l) Eigenbackground

(m) Codebook

(n) Codebook

Figure 3.28: Comparison of foreground detection results of different algorithms on a video of a fountain with a significant lighting difference

Chapter 4

Resource-Efficient Salient Foreground Detection in battery-Powered Embedded smart cameras by feedback tracking

4.1 Introduction

Battery-powered wireless embedded smart cameras have limited processing power, memory, and energy. Since video processing tasks consume a considerable amount of energy, it is essential to have lightweight algorithms to increase the energy efficiency of camera nodes. Moreover, just grabbing and buffering a frame requires a significant amount of energy. Thus, it is not sufficient to only focus on the vision algorithms. Methodologies are needed to determine when and how long a camera can be idle. This chapter introduces a feedback method for detection and tracking, which provides significant savings in processing time. Experimental results are performed to show the gains in processing time as well as the significant savings in energy consumption and battery life increase.

Wireless embedded smart cameras are stand-alone units that can capture images and perform on-board computation and communication. Rather than transferring all the data to a back-end server, they can process images, extract relevant data locally, and decrease communication bandwidth requirements. They also provide flexibility in terms of quantities and placement of cameras. On the other hand, battery-powered embedded smart cameras have limited computational power,

memory, and energy. Since battery life is limited and video processing tasks, such as foreground detection and tracking, consume a considerable amount of energy, it is essential to have efficient algorithms to optimize the energy expenditure of each camera node and thus, the overall lifetime of the network.

As shown below, even with no computer vision processing, only grabbing and buffering a frame requires a significant amount of energy. Thus, it is not sufficient to only focus on vision algorithms. Hence, there is the need for methodologies to adaptively reduce the processing time per frame according to the number and size of the objects being tracked. Tracking multiple objects is an important and challenging problem, which constitutes wide-ranging application areas. Even though many methods have been introduced for multi-object tracking , [18], [19], [20], [74], most of the existing tracking systems do not focus on embedded platforms and energy efficiency.

As mentioned in Chapter 3, common computing platforms for smart cameras are field programmable gate arrays (FPGAs), digital signal processors (DSPs), and/or general purpose microprocessors [60]. Additionally, in the sensor network community, detection and tracking methods have been proposed, that focus on different types of sensors other than cameras. Examples include magnetic, acoustic, and radar sensors. Arora et al. [34] presented a wireless sensor network for distributed intrusion detection, that employs magnetic and radar sensors. They studied the degradation in application performance in sensor networks as a function of network unreliability. Dutta et al. [42] presented a sensor network platform for detecting and classifying rare, random and ephemeral events. They used infrared, magnetic, and acoustic sensors. The infrared and acoustic sensors are designed for low-power continuous operation and include asynchronous processor wake up circuitry. Benbasat and Paradiso [58] presented a framework for power-efficient detection in wearable sensors. They used accelerometers and gyroscopes in their test scenario. State detection is structured as a decision tree classifier that dynamically orders the activation and adjusts the sampling rate of the sensors, such that only the data necessary to determine the system state is collected at any given time. Jiang et al. [59] presented a sleep scheduling algorithm for multiple target tracking to improve energy efficiency. A target tracking algorithm for wireless acoustic

sensor networks was introduced by Yu et al. [67]. Yet, systems based on scalar sensors can have problems when tracking multiple targets. Moreover, the aforementioned studies do not focus on camera sensors, on vision algorithms running on camera boards, nor in the energy consumption of the embedded camera nodes.

Many traditional tracking systems perform foreground object detection and tracking at each frame independently and in a sequential manner. On the other hand, Quast and Kaup [74] presented an object tracking system, wherein the object masks generated in the detection stage are used for constructing asymmetric kernels for the mean-shift based tracking stage.

This chapter is mainly focused on the design of a tracking algorithm capable of reducing the processing time per frame without affecting the performance and reliability of the overall foreground detection and the tracking system. The goal of the lightweight algorithm is to increase the energy efficiency and battery life of an embedded smart camera node.

A feedback method to increase the energy efficiency of the salient foreground detection and tracking is presented. Instead of performing foreground detection and tracking independently and sequentially at each frame, the feedback method incorporates the information from the tracking stage into the foreground detection stage. This way, foreground detection is performed in smaller regions as opposed to whole frame. The feedback method significantly reduces the processing time of a frame. To take advantage of these savings the microprocessor is sent to idle state at the end of processing a frame without causing tracking failure. This type of approaches were previously introduced by Casares et al. in conference proceedings [70] and [71], respectively.

The additional and different contributions presented in this chapter are as follows: 1) the feedback method is analyzed in detail in terms of energy consumption and gain in battery life; 2) the proposed method is compared with a sequential tracking approach; the way in which the proposed methodologies can send the microprocessor to idle state while tracking objects, and preserve the tracking performance will be shown.

The methodology presented in this Chapter is not intended for applications involving crowded scenes. There are two main reasons. 1) In a crowded scene, there will be search regions around

every object, and the area that needs to be processed will be close to the whole image. Thus, there may not be considerable savings in processing time. 2) Interactions, such as merges and splits, will be more likely in crowded scenes. It is not preferable to send the camera to idle state just before or during these interactions, since when the camera wakes up, there might be errors associating trackers with correct targets. In addition, during these interactions, it may be beneficial to capture more frames in case of an interesting event.

Intended applications include military surveillance, wildlife monitoring, elder care, and surveillance of surroundings of facilities. The remainder of this chapter is organized as follows: Section 4.2 shortly describes the embedded smart camera platform used in our experiments, which was introduced in more detail in chapter 2. Section 4.3 provides motivation for designing methodologies that decrease the processing time as well as the energy consumption of the camera node. One of the goals is to reduce the precessing time to send the camera to idle state, thus decreasing the energy consumption. Idling of the camera is merely mentioned in this chapter. Later, Chapter 5 introduces a more in depth analysis of the advantages offered by ideling the camera node. The feedback method is described in Section 4.4. Experimental results are presented in Section 4.5. This chapter is concluded in Section 4.6.

4.2 Wireless Embedded Smart Camera Platform

The wireless embedded smart camera employed in our experiments is a CITRIC mote [63] which runs embedded Linux Operating System. It consists of a camera board and a wireless mote. The camera board is composed of an image sensor, a microprocessor, external memories, and other supporting circuits. The image sensor is a Omni Vision OV9655, which is a low voltage SXGA CMOS image sensor. It supports image sizes SXGA (1280 1024), VGA (640480), and any size scaling down from VGA. The camera is capable of operating at 30 frames per second (f/s) in VGA resolution. Attached to the camera board is a TelosB mote from Crossbow Technology with a maximum data rate of 250 kb/s. The TelosB uses a Texas Instruments MSP430 microcontroller and Chipcon CC2420 IEEE 802.15.4-compliant radio, both for low-power operation [63]. Details

on the camera architecture are introduced in chapter 2.

4.3 Motivation: Energy Consumption Analysis

In Casares et al. [73], there are analyzed cases related to the size of objects being tracked. Tracking targets that are close to or far from the camera report different results in terms of processing time. The bar graph in Figure 4.1 shows the frame processing times when tracking an object in a close, middle and far range from the camera, together with the size of the object. The size of the bounding box of the object is displayed inside the bars. As expected, the processing time increases when the object is closer to the camera, since the object size, and thus, the area to be processed increases.

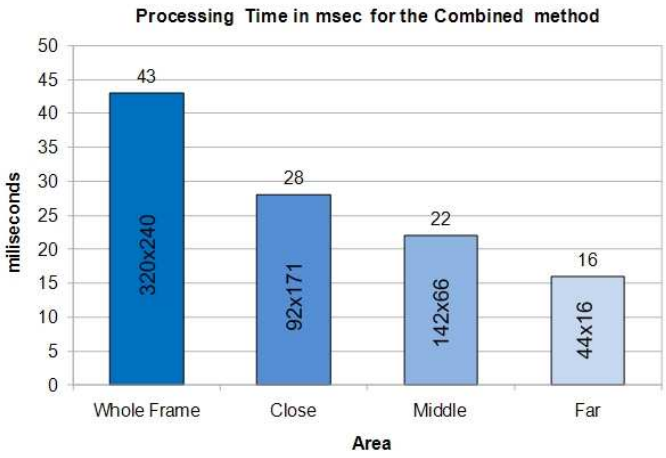


Figure 4.1: Processing time in milliseconds when an object is at different distances from the camera.

Hence, focusing only on vision algorithms is not sufficient. There is a need for self-adapting methodologies capable of increasing the overall life time of the camera mote.

The findings illustrated in Figure 4.1 encourage us to design methodologies and efficient algorithms to adaptively decrease the processing time of a frame reducing the accesses to memory per pixel. Moreover, preliminary results from section 4.4 will show that sending the microprocessor of the camera to idle state significantly reduces the overall energy consumption of the camera. Hence, new important challenges are sending the microprocessor to idle state even when the scene is not

empty, and determining adaptively how long the microprocessor can remain idle without affecting the performance and reliability of the overall foreground detection and the tracking system. This topic is fully covered in chapter 5.

As mention above, the main focus of this chapter is the design of a feedback method to increase the energy efficiency of the foreground detection. Additionally, it aims to show the reduction in terms of energy compared to the traditional ways to do tracking. This method significantly reduces the processing time of a frame. To take advantage of these savings, after done processing a frame, the microprocessor is reliably sent to idle state without causing tracking failure.

After grabbing and buffering a frame, the embedded smart camera performs foreground object detection and tracking. Casares et al. [68] presented a lightweight and efficient algorithm for salient foreground detection. This algorithm takes into account the memory requirements as well as the computational complexity. It is highly robust against lighting variations and non-static backgrounds including scenes with swaying trees, water fountains, and rain. The logic of the algorithm is explained in detail in chapter 3. As opposed to state of the art background subtraction, whose memory usages are ranged from 32 to 91 bytes per pixel, the object segmentation employed and described in Chapter 3 required 6.25 bytes per pixel. Additionally, the number of memory accesses and instructions per pixel are adaptive, and are decreased even more depending on the amount of activity in the scene and on a pixel's history.

A **sequential** term will be used throughout this chapter to refer to tracking methodology in which at every frame, the above foreground detection algorithm runs on the whole image to detect foreground pixels. The algorithm groups them together to form foreground blobs, and then match the foreground blobs to existing trackers. Most traditional tracking algorithms operate in this sequential manner.

The feedback method is described in Sections 4.4. As mentioned, Chapter 5 presents methodologies related to the idling of the embedded smart camera, which combined with the methodology introduced in this Chapter, will bring a third energy efficient algorithm named the Combined method, also explained in Chapter 5.

4.4 FeedbackMethod: Resource-Efficient Salient Foreground Detection by Feedback Tracking

The method presented in this section will be referred to as the feedback method. Instead of performing foreground detection and tracking independently at each frame, the feedback method incorporates the information from the tracking stage into the foreground detection stage that employs our algorithm summarized above. The diagram presented in Figure 4.2 illustrates the flow diagram followed by the feedback algorithm in comparison to the sequential one. Hence, foreground detection is performed in smaller regions as opposed to whole frame. Thus, significant savings in terms of energy consumption are expected since the energy expenditure is proportional to the size of, not only the object being tracked, but also the frame being captured and processed.

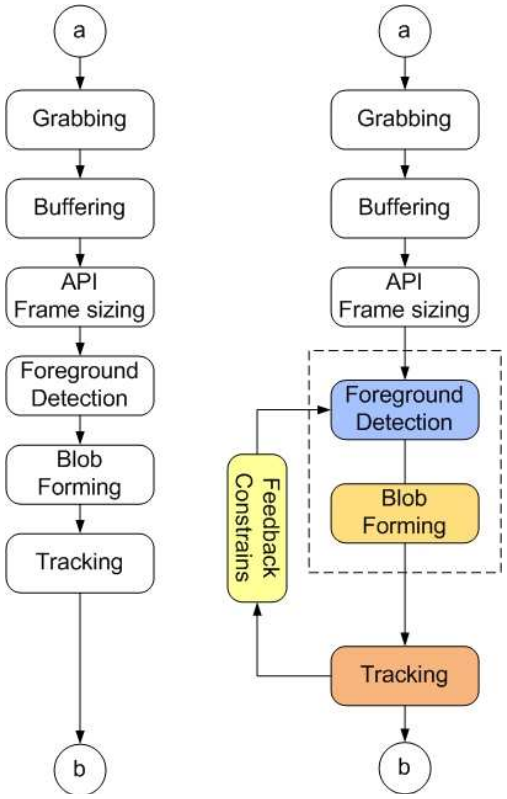


Figure 4.2: Illustration of the flow diagrams for sequential and feedback tracking methodologies.

4.4.1 Determining the Search Regions

When a foreground blob is detected in the scene, a bounding box is formed around it, and a new tracker is created. The intensity histogram of the foreground object is built and saved as the model histogram of the tracker (intensity histogram is used to keep the computational complexity low). The tracker also holds the coordinates of the bounding box of this object. Let $T = T^1(t1), T^2(t1)...T^n(t1)$ denote the set of existing trackers at frame $t1$. At frame t, a detected blob $B^i(t)$ will be matched to one of the trackers in the set T by using a matching criteria based on bounding box intersection and the Bhattacharyya coefficient [18], [72]. The Bhattacharyya coefficient is derived from the sample data by using

$$\hat{\rho}(y) = \rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\mathbf{y})\hat{q}_u} \tag{4.1}$$

Where $\hat{\mathbf{q}} = \hat{q}_{u=1...m}$, and $\hat{\mathbf{p}}(\mathbf{y}) = \hat{p}_u(\mathbf{y})_{u=1...m}$ are the probabilities estimated from the m-bin histogram of the model in the tracker and the candidate blobs, respectively. If the bounding box of a blob intersects with that of the tracker, the Bhattacharyya coefficient between the model histogram of the tracker and the histogram of the foreground blob is calculated by using 4.1. The tracker is assigned to the foreground blob which results in the highest Bhattacharyya coefficient. After blob $B^i(t)$ is matched to tracker $T^j(t1)$ (which holds the bounding box location from frame $t1$), the displacement of the centroid of the tracker's bounding box is calculated in x and y directions to obtain Δx and Δy, respectively (Figure 4.3). At frame $t+1$, for each foreground object i, a search region $R^i(t+1)$ is determined by using Δx, Δy, W and H, where W and H are the width and height of the bounding box of $B^i(t)$.

Then, the background subtraction and blob forming in the search regions $R^i(t+1)$ is performed as opposed to doing it on the whole frame. As shown in Table 4.1 searching for and forming foreground blobs in smaller regions significantly reduce the processing time. After the search regions are determined, the bounding box of the tracker T^j is updated to be the bounding box of $B^i(t)$.

The center of the search region $R^i(t+1)$ is found by using 4.2, where $B^i_x(t)$ and $B^i_y(t)$ are the x and y coordinates of the center of the blob B^i at frame t. $\Delta x(t)$ and $\Delta y(t)$ are the displacements in the x and y directions calculated between frames $t1$ and t as shown in Figure 4.3.

$$R^i_x(t+1) = B^i_x(t) + \Delta x(t) \tag{4.2}$$
$$R^i_y(t+1) = B^i_y(t) + \Delta y(t)$$

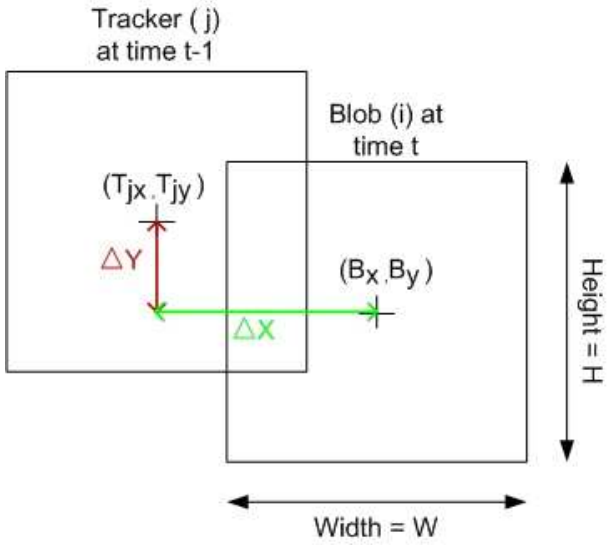


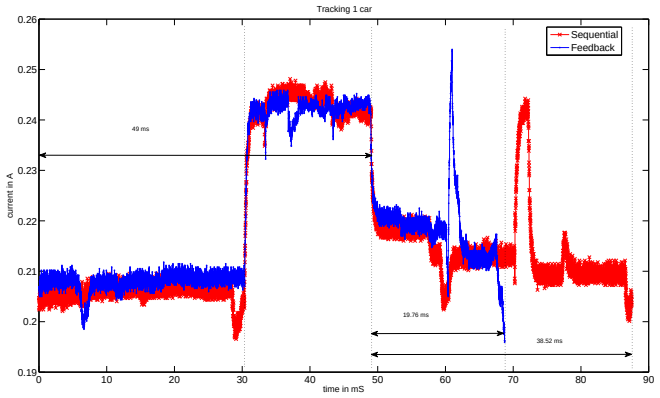
Figure 4.3: Displacement in the horizontal and vertical directions.

The boundaries of the search region are determined by using the equation 4.3. Foreground detection at frame $t+1$ will be performed in the search regions formed around the estimated locations of objects that were detected at frame t.

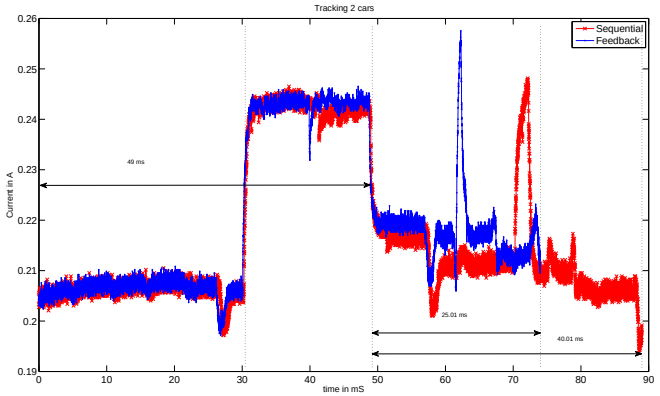
$$
\begin{aligned}
R^i_{x\_min}(t+1) &= R^i_x(t+1) - \Delta x(t) \\
R^i_{x\_max}(t+1) &= R^i_x(t+1) + \Delta x(t) \\
R^i_{y\_min}(t+1) &= R^i_y(t+1) - \Delta y(t) \\
R^i_{y\_max}(t+1) &= R^i_y(t+1) + \Delta y(t)
\end{aligned}
\tag{4.3}
$$

The camera's capture rate is 15 frames per second (f/s). Since, the algorithm becomes localized around the regions R^i, the foreground detection runs on the whole frame every 500 ms. In this way,the system is able to detect new objects in the scene, and update the background model. Compared to the sequential method, this mechanism reduces the processing time significantly. To exploit the advantage of these savings the microprocessor is sent to idle state at the end of processing a frame.

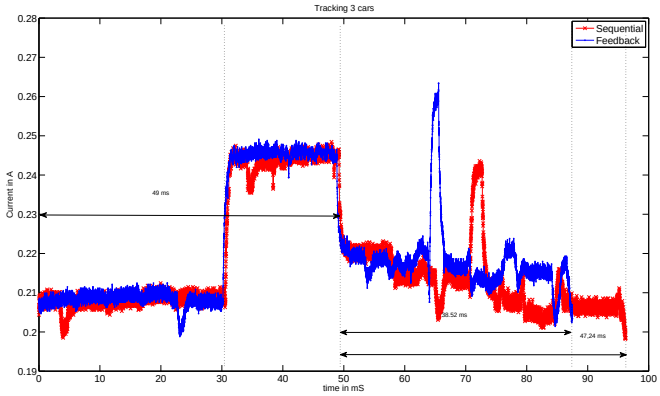
Both, the sequential and the feedback methods were run on embedded cameras to compare their processing times. Experiments tracking one, two and three remote-controlled cars were conducted. The blue and red plots in Figure 4.4(a) show the operating currents of the camera board when running the feedback method and the sequential method, respectively. The grabbing and buffering of a frame take 49 ms. The feedback method and the sequential method finish the processing of the frame in 19.7 ms and 38.5 ms, respectively, and the feedback method provides 48.7% decrease in the processing time. Figures 4.4(b) and 4.4(c) show operating currents when tracking two and three cars, respectively. As expected, the gain in processing time decreases with increasing the number of tracked objects. Though, the feedback method still outperforms the sequential method. The processing times and the results of the comparison are summarized in Table 4.1. Additionally, an experiment to measure the energy consumption when running the feedback and the sequential methods was performed. The comparison is presented in Section 4.5.



(a)



(b)



(c)

Figure 4.4: Operating current of the camera board with the feedback and sequential methods when tracking (a) one, (b) two, and (c) three remote-controlled cars.

4.5 Experimental Results

As mentioned previously, in most traditional tracking algorithms, background subtraction and tracking run independently, and operate in a sequential manner. In other words, background subtraction is performed first on the whole frame, and then trackers are matched to detected objects. In Section 4.4, the feedback method was presented. In this section the sequential and feedback methods are compared, and the gain in processing time provided by the feedback method showed. The feedback method takes advantage of the savings in processing time by sending the microprocessor to idle state at the end of processing a frame. Section 4.5.1 will compare the energy consumption of the feedback and the sequential methods.

All the algorithms run on the microprocessor of the camera board. The image size used in all the experiments is 320×240. The clock frequency of the microprocessor is 520 MHz.

4.5.1 Comparison of the Energy Consumptions of the Feedback and Sequential Methods

In this section, a set of experiments were conducted. Three different tracking scenarios to measure the energy consumption of the camera were used to run the feedback and the sequential method. In all three cases, remote-controlled cars are tracked for the same amount of time (5 min) so that energy consumptions for different scenarios can be compared.

In the first scenario, a remote-controlled car is tracked continuously for 5 min. In other words, the car is always in the field of view, and the scene is never empty. When tracking one car, the feedback method finishes the processing of a frame, on the average, 18 ms earlier than the

Method	1 Car (ms)	2 Cars (ms)	3 Cars (ms)
feedback (ms)	19.76	25.01	38.52
sequential (ms)	38.52	40.01	47.24
Savings	48.702%	37.49 %	18.45 %

Table 4.1: Comparison of the Processing Times of the Proposed Feedback Method and the Sequential Approach.

sequential method, and sends the microprocessor to idle state for 18 ms at the end of processing each frame. This way, the two methods process about the same number of frames during the 5-min period. It should also be noted that with the feedback method, the camera still processes the whole frame every 500 ms, to detect new objects and update the background model. Even in this case, using the feedback method provides 9.63% savings in energy consumption as seen in Table 4.2.

In the second scenario, the scene is empty for the first 100 sec. Subsequently, a car enters the scene, and is tracked for 100 sec. Then, a second car enters the field of view of the camera, and two cars are tracked for another 100 sec. Table 4.3 shows the total energy consumptions while running each method during the 5-min experiment. The feedback method provides 17.34% savings in energy consumption. Compared to the previous scenario, the savings in energy consumption increase, since the scene is empty for the first 100 sec.

Method	Energy (J)
Feedback	304.25
Sequential	336.69
Savings	9.63%

Table 4.2: Energy Consumptions for the Feedback and the Sequential Methods When Tracking One Car Continuously

Method	Energy (J)
Feedback	274.7057
Sequential	332.3419
Savings	17.34%

Table 4.3: Energy Consumptions for the Feedback and the Sequential Methods When Tracking One and Then Two Cars

Method	Energy (J)
Feedback	242.6787
Sequential	330.8194
Savings	26.6%

Table 4.4: Energy Consumptions for the Feedback and the Sequential Methods When a Car Enters and Leaves Twice

The third scenario is as follows. During the first 100 sec the scene is empty. Then, a remote-controlled car enters the scene, stays in the view of the camera for 50 sec, and leaves the field of view. After 100 sec, the car enters the view again, and stays there 50 more seconds. Table 4.4 shows the total energy consumption while running each method during the 5-min experiment. The feedback method provides a 26.6% decrease in energy consumption.

This chapter was dedicated to the introduction of a new methodology which can reduce the processing time per frame required by the embedded camera. As shown in the results, it will have an impact on the battery life of the camera due to the reduction in the energy consumption of the embedded node. Chapter 5 will introduce two more new methodologies to increase even further the battery life of the camera. Thus, a more comprehensive section of experiments, including the Feedback method and two new algorithms with outdoors scenarios will be presented.

4.6 Conclusions

A lightweight algorithm to increase the energy efficiency of an embedded smart camera node was presented. The feedback method for detection and tracking provides significant savings in processing time. We presented experimental results showing the gains in processing time as well as the savings in energy consumption and the gain in battery life. In summary, the feedback method provides 48.7% decrease in the processing time of a frame, and 10.44% savings in energy consumption, compared to traditional sequential tracking when tracking one object. We show that the presented methodology does not affect the tracking performance. On the other hand, strong shadows can be a problem for the tracking algorithm, since they are also detected as foreground regions. We are planning to design a shadow removal algorithm without significant increase in the memory requirements.

Chapter 5

Resource-Efficient Salient Foreground Detection in battery-Powered Embedded smart cameras by adaptive tracking methodologies

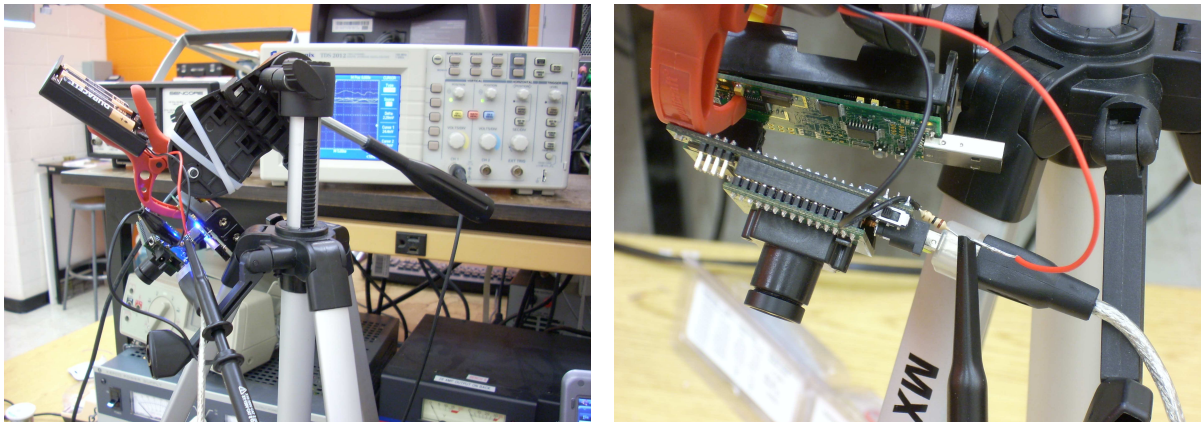
As discussed in chapter 4 section 4.3, grabbing and buffering a frame require significant amount of energy, even when no processing is performed. Hence, it is not sufficient to only focus on vision algorithms. There is a need for effective and self-adapting methodologies to be able to drop frames even when the scene is not empty.

The findings presented in chapter 4 motivate us to design and implement methodologies and efficient algorithms to adaptively drop frames, decrease processing time of a frame, and increase idle durations. This will bring new important challenges such as sending the microprocessor to idle state even when the scene is not empty. Moreover, determining adaptively how long the microprocessor can remain idle without affecting the performance and reliability of the overall tracking is even more complex.

5.1 Motivation: Adaptive methodologies

This section presents the energy consumption analysis of an actual embedded smart camera at the stages of grabbing, buffering, and processing a frame. This analysis provides the motivation to develop methodologies that will increase the battery life of the camera. The operating current of

the embedded smart camera during the tasks of grabbing, buffering and processing a frame were measured as follows. To measure the current, a 500 MHz LeCroy oscilloscope was used, and a $1Ohm$ resistor was placed at the input of the supply source as shown in Figure 5.1



(a) 500 MHz LeCroy oscilloscope (b) $1Ohm$ shunt resistor

Figure 5.1: Camera setup ready to perform the required energy measurements.

The processing of a frame will refer to performing foreground detection and tracking, while the grabbing and buffering of a frame are considered two separate actions requiring different energy levels. For instance, Figure 5.2 shows the operating current of the camera board when running a sequential tracking method (i.e., performing background subtraction on the whole frame, and then tracking) to track one remote-controlled car. As can be seen, the grabbing and buffering take 49 ms, and the processing of the frame takes 38.5 ms. In addition, grabbing and buffering consume 54.1 mJ of energy while detection and tracking consume 42.2 mJ. Thus, grabbing and buffering are even costlier than processing, and demand a significant amount of energy even when no computer vision processing task is performed. Thus, it motives us to develop methodologies that are capable of grabbing, buffering and processing the optimal number of frames while still having a reliable tracking system.

This chapter focuses on developing an adaptive tracking-based method that significantly decreases the energy consumption of the camera. The microprocessor of the camera can be sent to idle state to save energy even when there are moving objects in the scene. The idle-state duration is adaptively changed based on the amount of activity in the scene and speed of tracked objects.

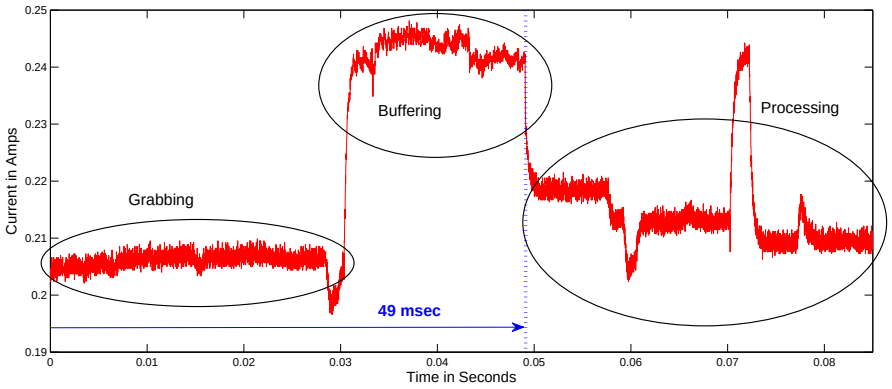


Figure 5.2: Operating current of the camera board during different tasks

Instead of continuously capturing and processing every frame, the camera drops frames during idle state, while preserving the tracking performance and thus system reliability at the same time. The idea behind the algorithm is to save energy by processing the optimal required number of captured frames to reliably track objects. Figure 5.3 illustrates the process in which a car enters to the view of the camera; the speed of the car is estimated, and then the camera only grabs the necessary number of frames to reliably track the vehicle.

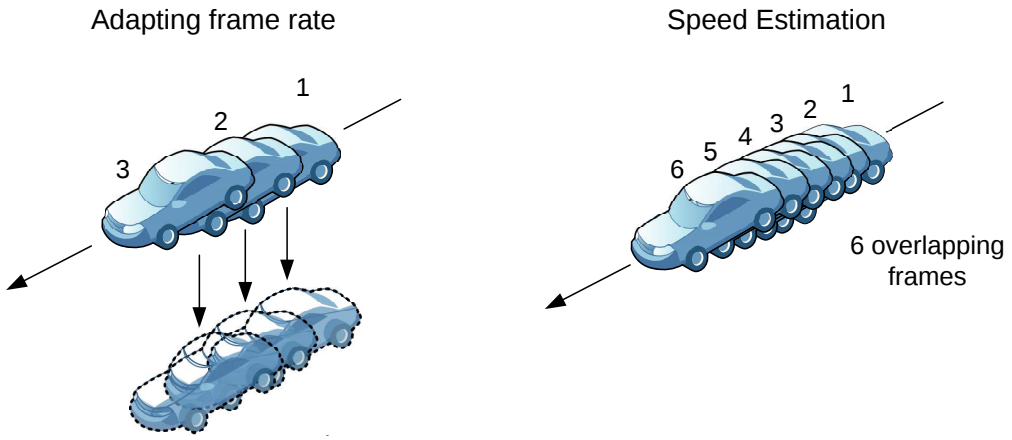


Figure 5.3: Camera dropping frames to save energy as illustration of the main goal of the algorithm.

This significantly prolongs the battery life. The experimental results including graphs of camera's operating current over time, and power and energy tables showing the energy-efficiency of the proposed method as well as the gain in battery life are also presented in Section 5.5. In order to increase energy-efficiency, the system puts the microprocessor in an idle state during which cer-

tain number of frames are dropped. Since the processor is running embedded Linux, commands such as “usleep()” and “nanosleep()” are available to suspend execution, and send the processor in idle state. The function “usleep()” takes the number of desired microseconds as argument. The important challenge is to determine how long the microprocessor can remain in the idle state without affecting the performance and reliability of the overall foreground detection and the tracking system. If the camera drops too many frames, then the tracking algorithm will most likely have problems associating the currently detected object with the most recent model and location. To increase energy-efficiency, three operation modes are developed: empty-scene mode, fixed-rate tracking mode, and adaptive tracking mode. Henceforth, empty scene will refer to the case when there are no foreground objects in the scene. To detect whether the scene is empty or not, the lightweight salient foreground detection algorithm presented in chapter 3 introduced by in Casares et. al [68] is used. The algorithm was implemented in C/C++ and imported on the microprocessor of the cameras. After detecting foreground pixels, connected-component analysis is performed to remove small pixel regions, and form object blobs. Then, we have used an efficient and robust tracking algorithm for object tracking purposes.The tracking algorithm has also been imported to the camera board, and the details of it are explained in chapter 4.4. Together, foreground detection and tracking run at 10.5 f/s on the microprocessor when there is one object in the scene.

5.1.1 Empty-Scene Mode

When no object is detected in the scene, since no tracking has to be performed, the idle durations can be longer, and thus more frames can be dropped. In the empty-scene mode, the algorithm determines the idle duration so that the camera grabs 2 f/s. The operating current values of the camera board were measured when the camera was continuously capturing frames (no idle state) and when the camera was sent to idle state for a fixed amount of time. Figure 5.4 shows a 715 ms segment from the current waveforms obtained during a 5-min experiment. The data was acquired with a NI 6221 data acquisition card at a sampling rate of 10 kHz. Red and blue plots are the operating current values of the camera when it continuously captures and processes frames, and

	Duration (min)	Energy (J)
Continuous frame capture	5	341.818
Empty-scene mode	5	223.855
Savings		34.5%

Table 5.1: Energy Analysis of the Empty-Scene Mode

when the microprocessor is sent into idle state, respectively.

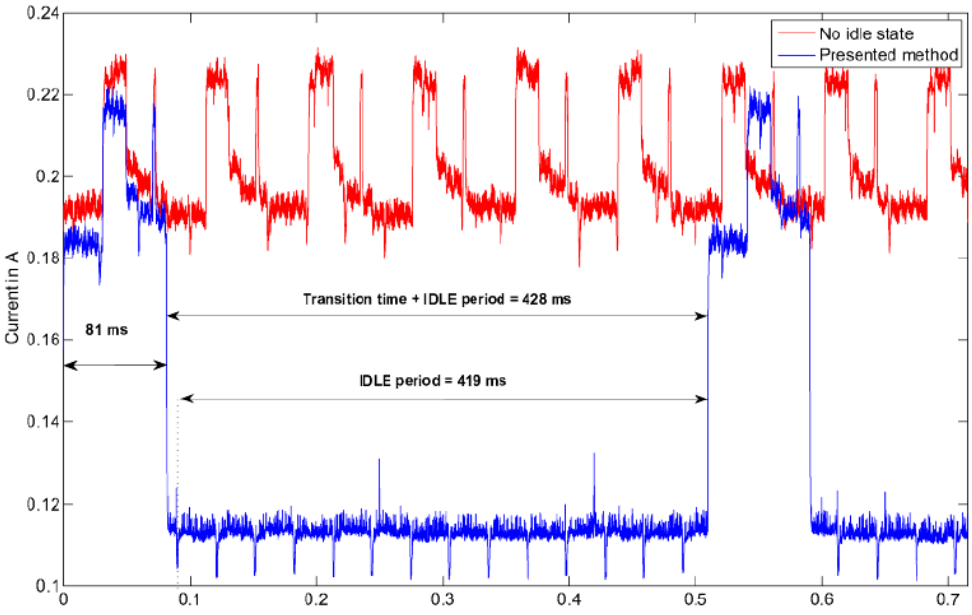


Figure 5.4: Empty-scene mode: red and blue plots are the operating current values when camera captures frame continuously, and when the microprocessor is put into idle state, respectively.

After a frame is grabbed, the background model is updated, and the elapsed time from grabbing the frame to finishing model update is determined. As seen in the red plot in Figure 5.4, it takes 81 ms from grabbing the frame to finishing the update of the background model. Then, the idle state duration is determined to be $t = 50081 = 419$ ms. At the end of the idle duration, the camera performs foreground detection to determine whether the scene is still empty. If no foreground objects are detected, the background model is updated, and the microprocessor is sent to idle state again. Table 5.1 lists the computed energy consumptions during a 5-min interval. As can be seen, the empty-scene mode provides 34.5% savings in the energy consumption.

5.2 Fixed-Rate Tracking Mode

When foreground objects are detected, the tracking mode is employed. In the fixed-rate tracking mode, idle state duration is determined based on the fastest moving object in the scene, and the same duration is used until a faster object enters the scene. The assumption is that the speed of the objects does not change significantly. Section 5.3 will present an adaptive methodology, which changes the idle state duration if a change in the object's speed is detected.

When a new object O^i is detected in the scene, a new tracker T^i is created, and the bounding box (B_1^i) formed around this object is saved in memory. Also, a counter ($N_{overlap}$) is set to 1. In the following frames, the object O^i is tracked. At each frame, it is checked if the bounding box of the object at that frame overlaps with B_1^i. If they overlap, $N_{overlap}$ is incremented by 1. This process is illustrated in Figure 5.5(a), where the first bounding box is B_1^i. Blue regions are the overlapping areas between B_1^i and bounding boxes B_2^i through B_5^i. As seen in Figure 5.5(a), the last overlap occurs between B_1^i and $Bi5$, i.e., B_1^i and B_6^i do not overlap. At this point, the value of $N_{overlap}$ is 5, and this value is used to calculate the duration of the idle state without affecting the tracking performance. To calculate the idle time, T_{idle}, equation 5.1 is used.

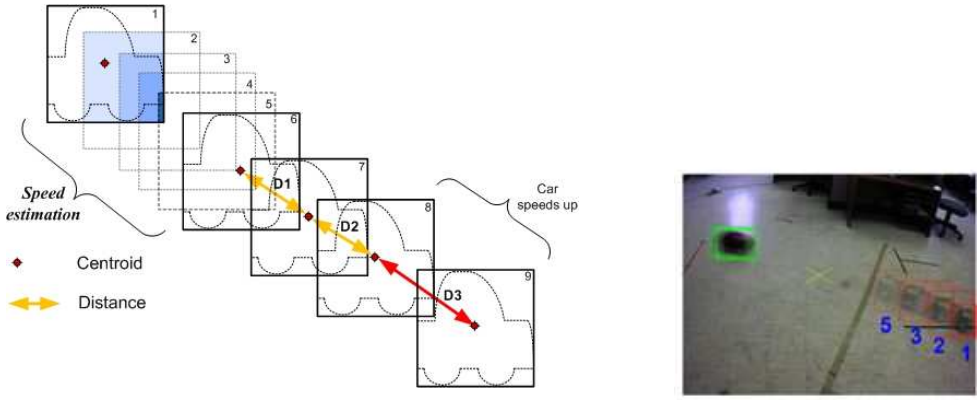
$$T_{idle} = 1000 \times \frac{N_{overlap}}{2} \times R_{capture} \tag{5.1}$$

Where $R_{capture}$ is the camera's capture time per frame. The camera captures 15 f/s, thus $R_{capture}$ is 67 ms. In this case, after processing the first frame, $\sum_{k=2^5} P_k$ ms pass until the fifth frame is processed, where P_k is the time it takes to capture and process the k_{th} frame. $P_k > R_{capture}$, and at the fifth frame, overlapping still continues.

However, the time that has passed since the first frame is an upper bound for the idle duration, since object pattern or speed may change. Thus, equation 5.1 takes a conservative approach to account for these changes while calculating the idle duration. First, it uses $R_{capture}$, instead of the time it takes to process a frame, and $R_{capture}$ is always less than P_k. Second, $N_{overlap}/2$ is used instead of $N_{overlap}$ in the formula to address sudden speed increases, and make sure that overlap

continues at the end of the idle duration by assuming that the object cannot more than double its speed in order of milliseconds. In 5.1, 1000 is used to convert milliseconds to microseconds so that T_{idle} can be used as the argument of the function usleep(). Figure 5.6 shows the current waveform while estimating the idle duration and during the idle states.

In Section 5.3, adaptive methodology is introduced, where the idle state duration is changed if the object's speed changes significantly. In Section 5.5.5, a detailed analysis of a scenario where the object's speed increases gradually and very slowly is presented. Consequently, in the scenario, the bounding boxes do not overlap after coming back from idle duration. Thus, a solution of how to overcome this problem is also presented.



(a) (b)

Figure 5.5: (a) Detecting a speed change. (b) Overlapping bounding boxes for a faster car.

To be able to successfully track every object, idle state duration has to be based on the fastest object in the scene. If another object enters the scene, T_{idle} is calculated again, and if it is less than the current value used, the idle state duration is changed. Figure 5.6 shows the operating current waveform, obtained from a NI 6221 data acquisition card, for a scenario that involves two cars. When the first car enters the scene, the computed value for T_{idle} is 435.5 ms and the camera is sent to idle state for this amount of time. About 8 s later, a faster car enters the scene, and a new T_{idle} is calculated based on this new car. Since the new value, 201 ms, is less than 435.5 ms, the idle state duration is changed, and the camera is sent to idle state for a shorter time period as seen in Figure

5.6.

Experiments were performed when there were one and/or two cars in the scene. The energy consumption was measured over a 100 sec window. In the first 50 sec, there is one car in the scene. When the car enters the scene, $N_{overlap}$ is computed, and idle state duration is calculated. The value obtained for $N_{overlap}$ is 21. Applying equation 5.1, the idle state duration is obtained to be 703 ms. After 50 sec, a second car enters the scene whose speed is higher. The new idle state duration is 134 ms. The idle state duration of the camera is shortened based on the fastest moving object in the scene. Figure 5.5(b) illustrates the bounding boxes at different frames. For this case, the fifth bounding box does not overlap with the first one.

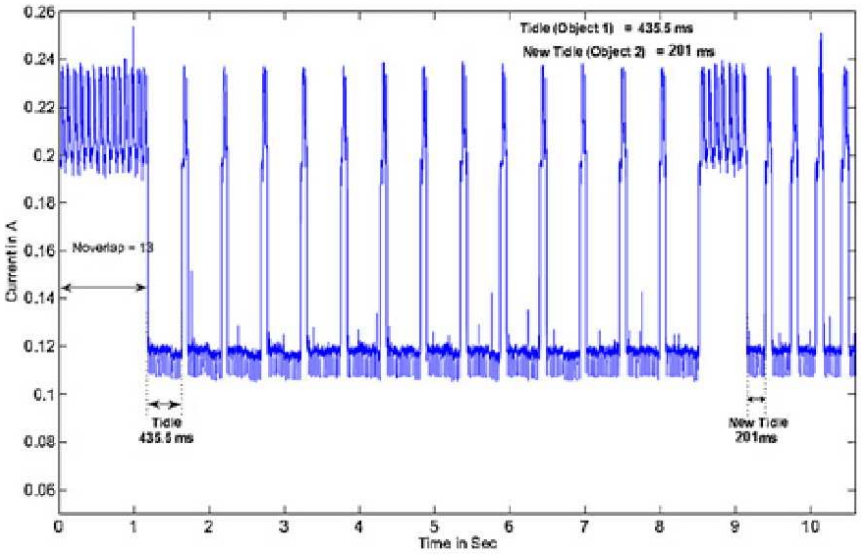


Figure 5.6: Updating the idle state duration based on the fastest object in the scene.

Table 5.2 lists the computed average energy consumption when there is one and two cars in the scene with and without using the fixed-rate tracking mode. As can be seen, the fixed rate tracking mode provides 36.5% and 25.7% savings in the energy consumption for one car and two car cases, respectively. Since the second car is faster, the idle state duration becomes shorter, which explains the decrease in savings.

Continuous frame capture	Time(s)	Avg. Power (W)	Energy (J)
1 car	50	1.1463	57.3141
2 cars	50	1.1448	57.2401
Fixed-rate method	Time(s)	Avg. Power (W)	Energy (J)
1 car	50	0.7272	36.3616
2 cars	50	0.8502	42.5102

Table 5.2: Energy Analysis of the Fixed-Rate Tracking Mode

5.3 Adaptive Tracking Mode

Objects in the scene can continuously increase or decrease their speeds. When the object is first detected, an idle state duration is calculated by using the method described in Section 5.2. Later on, if the object slows down, using the same idle duration will not negatively affect the tracking performance, i.e., it will not cause the tracker to lose the object. On the other hand, if the speed of the object continuously increases, using the same idle state duration might cause tracking failure. To handle these cases, a method that adapts the idle state duration is introduced, when a significant change in the object's speed detected. In the former case, where the object slows down, the idle state duration can be increased accordingly to increase the energy efficiency even further.

Detecting Speed Change

When an object enters the scene, an initial T_{idle} is computed as described in Section 5.2. Consistent with the notation in Section 5.2, let B_1^i denote the bounding box of object i, when it is first detected, and let B_n^i denote the bounding box at the n_{th} frame. In the scenario shown in Figure 5.5 (a), B_6^i is the first bounding box that does not overlap with B_1^i, thus $N_{overlap}$ is 5. At this point, B_6^i is saved as B_{last}^i, and the camera is sent to idle state for T_{idle} microseconds. At the end of the idle state, when camera captures and processes the seventh frame, the distance D_{curr} between the centers of B_{last}^i and B_7^i is calculated. D_{curr} is saved as D_{prev}, and B_{last}^i is set to be B_7^i. Then, the camera is sent to idle state again for T_{idle} microsec. At the end of the idle duration, the camera captures and processes the eighth frame, and calculates the distance D_{curr} between the centers of B_{last} and B_8^i.

The main idea is comparing D_{prev} and D_{curr} to detect a speed change. However, the following

scenarios need to be handled. When an object is moving toward the camera, it is going to appear larger, and D_{curr} can be greater than D_{prev} even if the object is moving with constant speed, or when an object is moving away from the camera, it is going to appear smaller, and D_{curr} can be smaller than D_{prev} even if the speed does not change. As a result, these situations could be mistaken for a speed increase/decrease. To address these cases, equation 5.2 applies a normalization to the center coordinates before calculating the distances.

$$\left(\bar{x}_n^i, \bar{y}_n^i\right) = \frac{(x_n^i, y_n^i)}{max\left\{W_n^i, H_n^i\right\}} \tag{5.2}$$

where W_n^i and H_n^i are the width and height of the bounding box B_n^i, respectively, and $\bar{x}_n^i$ and $\bar{y}_n^i$ are the normalized coordinates of its center. After normalization, the distance $\bar{D}_{curr}$ is calculated by using

$$\bar{D}_{curr} = \sqrt{\left|\bar{x}_n^i - \bar{x}_{last}^i\right|^2 + \left|\bar{y}_n^i - \bar{y}_{last}^i\right|^2} \tag{5.3}$$

where $\bar{x}_{last}^i$ and $\bar{y}_{last}^i$ are the normalized center coordinates of the bounding box B_{last}^i. Then, the ratio $R = \bar{D}_{curr}/\bar{D}_{prev}$ which initially was equal to 1 is calculated. If $R \geq 1.25$, then the idle state duration is recalculated by using equation 5.4. Initial idle state duration is determined by using equation 5.1. changing the idle state duration when $R \geq 1.25$ is going to handle cases of increasing speed while avoiding tracking failure at the same time. Performing idle state duration update only when $R \geq 1.25$ avoids recomputing a new duration when there is not a significant speed change. In Section 5.5 the tracking performance for different scenarios when using the adaptive methodology will be analyzed.

$$T_{idle}^{new} = \frac{1}{R} \times T_{idle} \tag{5.4}$$

Different kind of experiments were performed to measure the gain in energy consumption when using the adaptive methodology. In the first experiment, a car enters the scene, and then speeds up. Figure 5.7 shows an example of consecutive frames processed by the camera. Between frames 118

and 119 the car increases its speed (the distance between the centers of bounding boxes is larger than the previously computed distance between frames 117 and 118).

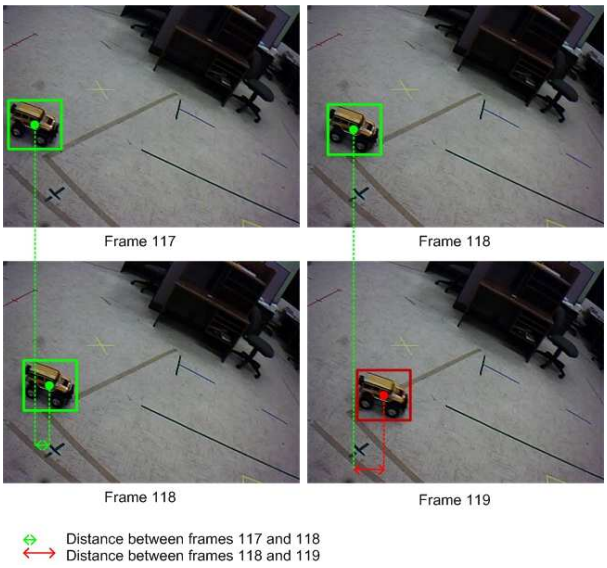


Figure 5.7: Car increasing its speed.

Figure 5.8 shows the operating current waveform obtained with the oscilloscope during this experiment. When it is first detected, $N_{overlap}$ is computed to be 7. Then, the idle state duration is calculated to be 234.5 ms, by using equation 5.1, and the microprocessor of the camera is sent to idle state. Between $t = 0.9s$ and $t = 1.6s$, the car follows a path that is not parallel to the camera's image plane, i.e., it either moves toward the camera or away from it, with approximately constant speed. Thus, the movement pattern should not affect the idle state duration. This scenario was successfully handled by the aforementioned normalization method. As seen in Figure 5.8, between $t = 0.9s$ and $t = 1.6s$, the idle state duration does not change. At some point after $t = 1.6s$, when the camera returns from idle state, the calculated distance ratio (R) is 1.267, and thus a new idle state duration is calculated by using equation 5.1. The value obtained for T_{idle}^{new} is 185 ms. As a result, the camera was sent to idle state for a shorter period of time, to handle increasing object speed. As seen in Table 5.3, the presented adaptive methodology provides 37% saving in terms of energy consumption.

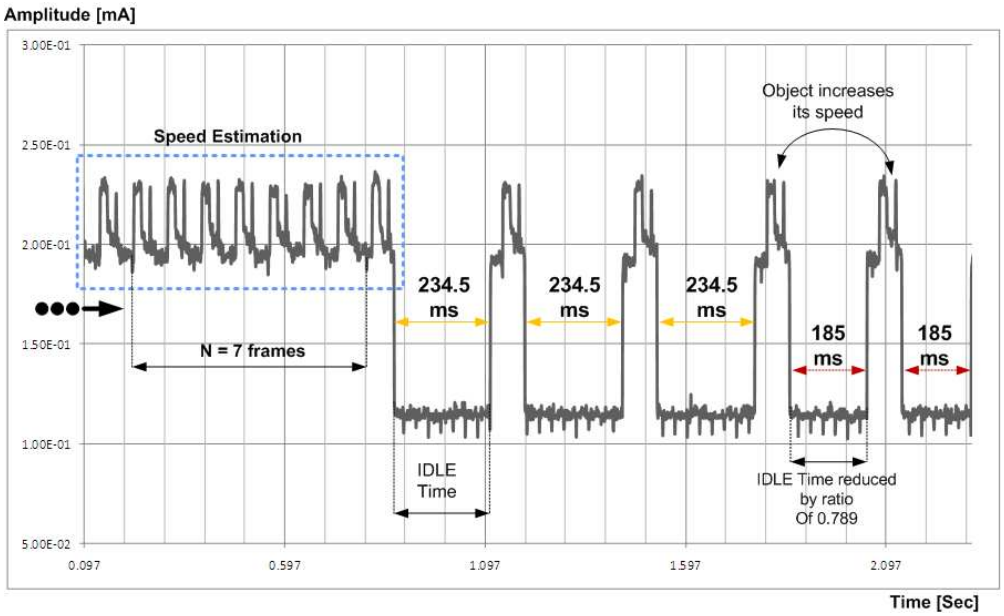


Figure 5.8: Operating current waveform when the idle time is changed based on the object speed.

	Time(ms)	Power (W)	Energy (J)
Cont. capture	3664	1.235	4.525
Adaptive-rate	3664	0.776	2.843
Savings			37.17%

Table 5.3: Energy Analysis of the Adaptive-Rate Tracking Mode

As mentioned above, when an object slows down, continuing to use the initially determined idle state duration will not cause any tracking failure. However, with the same method described above, the system can detect the speed decrease, and then increase the idle state duration to further increase the energy efficiency and the battery life. Thus, if $R < 0.75$, the idle duration can be recalculated by using equation 5.4, and send the camera to idle state for a longer time period. In another experiment, to analyze the energy savings, the fixed-rate tracking mode and the adaptive tracking mode were compared in a scenario where the tracked object slows down. Since the adaptive tracking method detects the decreasing speed, it increases T_{idle}, accordingly. The camera stays in idle state longer (143 ms), and compared to the fixed-rate mode, this provides additional 7.8% savings in the energy consumption as seen in Table 5.4.

In the rest of this chapter, the adaptive methodology will be used when analyzing the energy

	Time(s)	Avg. Power (W)	Energy (J)
Cont. capture	1.6	1.1457	1.8333
Fixed-rate	1.6	0.8924	1.4279
Adaptive-rate	1.6	0.8226	1.3163

Table 5.4: Energy Analysis

savings, since the adaptive methodology is more robust compared to the fixed-rate method, and provides more savings for objects slowing down.

5.4 Combined Method for Further Energy Efficiency

As discussed in chapter 4, the feedback method for salient foreground object detection provides significant savings in processing time of a frame. On the other hand, the adaptive methodology described in Section 5.3 allows us to send the microprocessor to idle state, even when the scene is not empty, and also can increase/decrease the idle duration based on object speed.

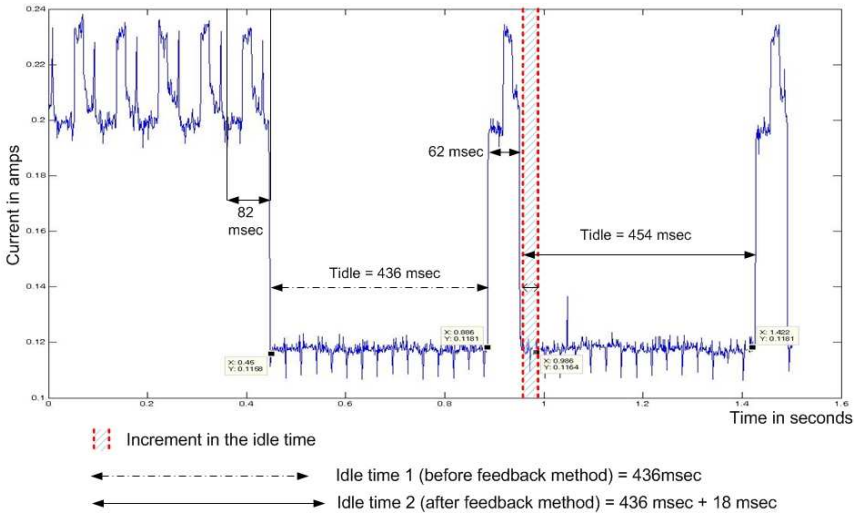


Figure 5.9: Idle duration is increased in the combined method by employing the feedback method and the adaptive methodology together.

To leverage the advantages of both, the feedback method and the adaptive methodology, these two methods were combined. First, when an object enters into the view of the camera, the method

described in Section 5.2 is employed to compute $N_{overlap}$ and the initial idle duration, T_{idle}. During this period, the foreground detection is applied on the whole frame, and the average processing time, T_{avg}, of a frame is computed. After the camera comes back from the first idle state, the feedback method is employed, and the processing time of the frame, T_{feedb}, is found. Then, $Tadd = T_{avg} - T_{feedb}$ is computed, where T_{add} is the extra idle duration gained. From this point on, the microprocessor is sent to idle state for $T_{idle} + T_{add}$ ms. These steps are illustrated in Figure 5.9, which shows the contribution of each method in an experiment wherein a car enters into the view of the camera. After it is detected, $N_{overlap}$ is computed to be 13 frames, and by using 5.1, the idle duration is calculated to be 436 ms. The shaded region in Figure 5.9 shows the savings in processing time of a frame provided by the feedback method. The idle duration is increased by this amount, i.e., it is increased from 436 ms to 454 ms. In section 5.5, a detailed analysis and comparison of the feedback method, the adaptive methodology, and the combined method in terms of their energy consumptions and battery life of the camera board will be provided.

5.5 Experimental Results

As mentioned previously, in most traditional tracking algorithms, background subtraction and tracking run independently, and operate in a sequential manner. In other words, background subtraction is performed first on the whole frame, and then trackers are matched to detected objects. In the previous chapter, the feedback method was presented, the sequential and feedback methods compared, and the gain in processing time provided by the feedback method was shown.

The adaptive methodology presented in Section 5.3 uses the sequential method for frame processing, but can send the microprocessor to idle state even when the scene is not empty. The combined method described in Section 5.4 employs the feedback method and the adaptive methodology together. It uses the feedback method for frame processing, and allows to send the microprocessor to idle state, even when there are moving objects in the scene, for longer periods of time. The energy consumption comparison of the adaptive methodology and the sequential method is presented in Section 5.5.1. Section 5.5.2 presents the energy savings provided by the combined method.

Method	Time(ms)	Power (W)	Energy (J)
Adaptive	458	0.7213	0.3304
Sequential	458	1.1041	0.5057
Savings			34.7%

Table 5.5: Energy Consumption Comparison Between Adaptive and Sequential Methods

All the algorithms run on the microprocessor of the camera board. The image size used in all the experiments is 320×240. The clock frequency of the microprocessor is 520 MHz.

As mentioned before in chapter 4, our work is not intended for applications involving crowded scenes. There are two main reasons. 1) In a crowded scene, there will be search regions around every object, and the area that needs to be processed will be close to the whole image. Thus, there may not be considerable savings in processing time. 2) Interactions, such as merges and splits, will be more likely in crowded scenes. It is not preferable to send the camera to idle state just before or during these interactions, since when the camera wakes up, there might be errors associating trackers with correct targets. In addition, during these interactions, it may be beneficial to capture more frames in case of an interesting event. For this reason, the system disables the function of going idle when objects in the scene get close to each other.

5.5.1 Comparison of the Energy Consumptions of the Adaptive Methodology and the Sequential Method

The energy consumption of the adaptive methodology and the sequential method during 458 ms when tracking one remote-controlled car were calculated. In this experiment, after the car enters into the view of the camera, the idle duration T_{idle} is obtained as described in Section 5.2. The number of overlapping frames ($N_{overlap}$) was 11. By using equation 5.1, T_{idle} was computed to be 368.5 ms. Figure 5.10 shows the operating current of the camera board when running the two methods. As can be seen, the adaptive method processes one frame, and then goes into idle state for 368.5 ms. During the same time interval (458 ms) the adaptive method processes only one frame, whereas the sequential method processes five frames. The adaptive methodology provides 34.7% savings in energy consumption as shown in Table 5.5.

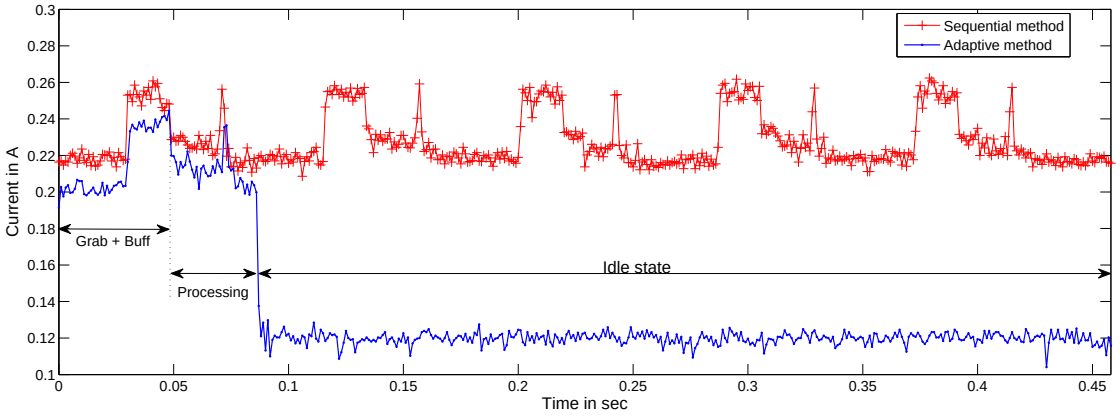


Figure 5.10: Operating current of the camera when tracking one car with the adaptive methodology (blue) and the sequential method (red).

Method	Time(ms)	Power (W)	Energy (J)
Feedback	458	1.0216	0.4679
Adaptive	458	0.7213	0.3304
Combined	458	0.6986	0.3199

Table 5.6: Energy Consumption Comparison Between the Feedback, Adaptive and Combined Methods

5.5.2 Energy Savings Provided by the Combined Method

The combined method described in Section 5.4 employs the feedback method and the adaptive methodology together. It uses the feedback method for frame processing, and allows us to send the microprocessor to idle state, even when there are moving objects in the scene, for longer periods of time.

Figure 5.11 shows the operating current graphs of the camera board when running the feedback method only (green plot), and when running the combined method (black plot). During this experiment, the camera is tracking one remote-controlled car, and the feedback method finishes processing a frame 18 ms faster, on average, compared to the sequential method. Thus, the microprocessor is sent to idle state for 18 ms at the end of processing a frame. On the other hand, the idle duration for the combined method is 394 ms. During a 458 ms time window, the feedback method captures and processes five frames, whereas the combined method only captures and processes

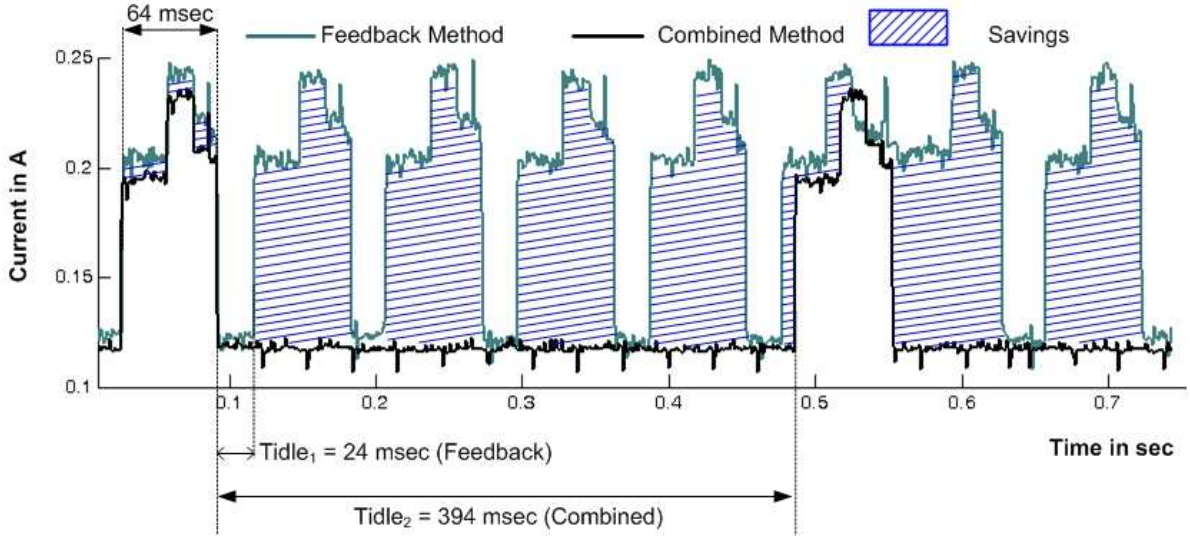


Figure 5.11: Operating current of the camera board when employing the feedback method by itself and the combined method.

one. The adaptive methodology was also employed for the same scenario. Figure 5.12 shows the operating current of the camera board when using the adaptive methodology (red plot) and the combined method (black plot). As can be seen, thanks to the additional idle duration provided by the feedback method, the camera stays in the idle state 18 ms longer in the combined method, compared to the adaptive methodology that uses the sequential tracking for frame processing. Table 5.6 lists the power requirements and energy consumptions of the feedback, adaptive, and combined methods.

Outdoor experiments were also performed with vehicles and people. Figures 5.13 and 5.14 show different output images obtained when tracking one car, and one person and one car, respectively. Thus, the energy consumption of the combined method and the sequential method were computed on a 2-min segment to obtain the savings provided by the combined method. In this segment, a car enters the scene after one min., and stays in the scene for the following one min. As seen in Table 5.7, the combined method provides 39% savings in the energy consumption during the period when the car is in the scene.

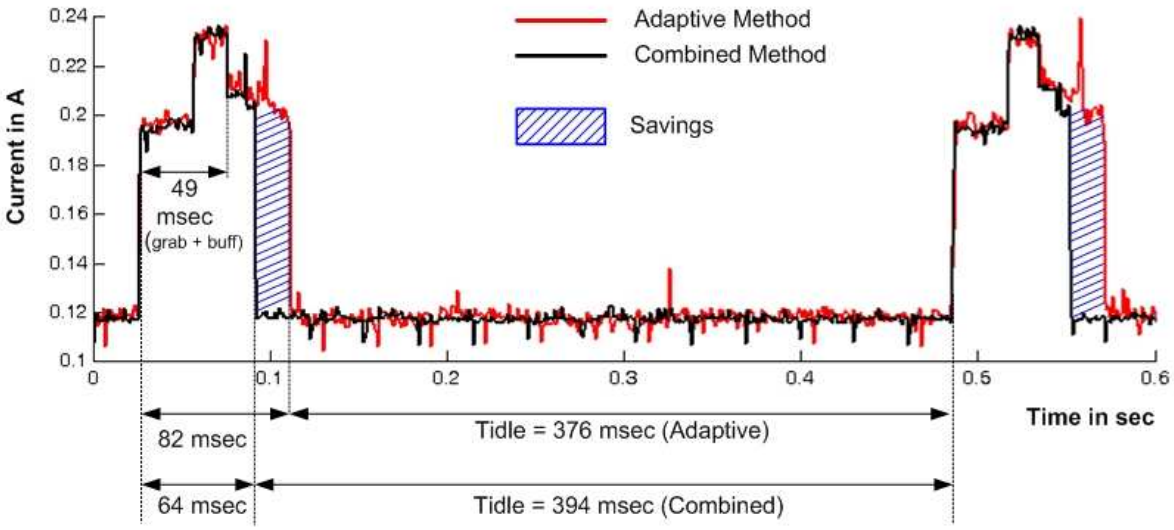


Figure 5.12: Operating current of the camera board when employing the adaptive methodology and the combined method.

Method	Empty (60 s) Energy (J)	1 Car (60 s) Energy (J)	Total (120 s) Energy (J)
Sequential method	67.8688	65.867	133.7358
Combined method	40.3074	40.164	80.4714
Savings	40.61%	39.02%	39.83%

Table 5.7: Energy Consumption Comparison Between the Combined and Sequential Methods

5.5.3 Energy Consumption Analysis over a Longer Time Window

To further analyze the energy consumption, and better project the battery life, an experiment for a longer period of time was conducted. The camera tracked an object for 20 min. Figure 5.15 shows a segment of the operating currents for all the algorithms/methods described in this chapter The red and dark blue plots are the operating currents when running the sequential detection/tracking and the feedback method, respectively. The light blue and green plots correspond to the adaptive and combined methods, respectively. A zoomed in version of these operating current plots is also shown in the same figure.

The pairwise energy savings for different combinations of these algorithms/methods over a

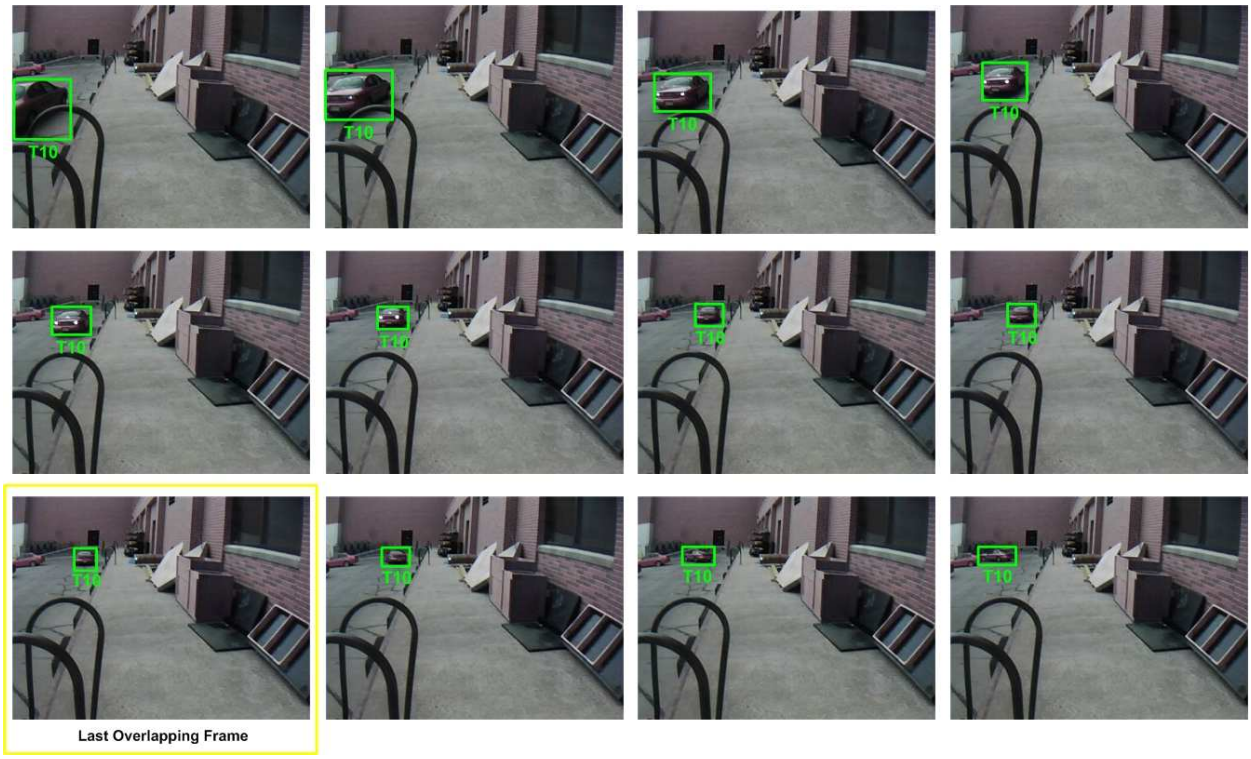


Figure 5.13: Output frames obtained while tracking one car.

Figure 5.14: Output frames obtained when tracking two targets.

20 min time window are listed in Table 5.8. The diagonal entries are the energy consumption of each method. 36.89%, for instance, is the savings in energy provided by the combined method compared to using sequential detection/tracking continuously (i.e., without dropping frames). Sim-

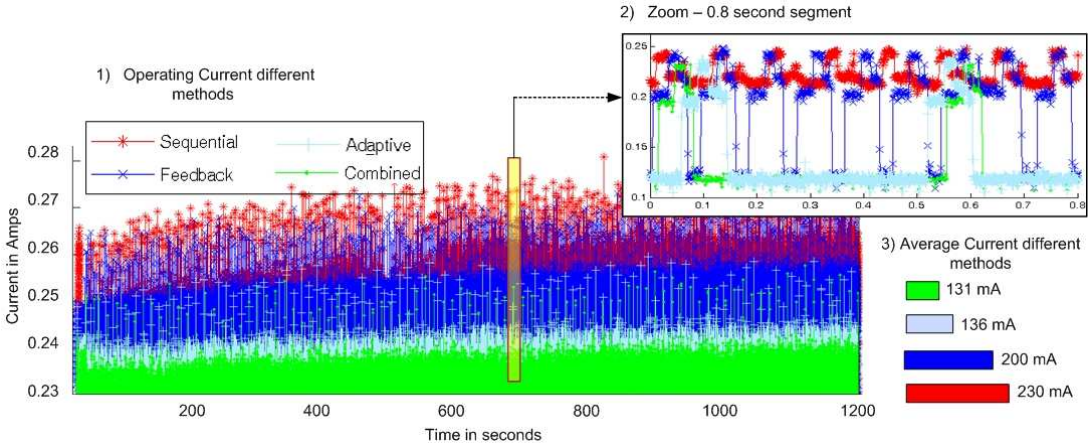


Figure 5.15: Operating current of the camera board when tracking one car with different algorithms for 20 min.

	Sequential	Feedback	Adaptive	Combined
Sequential	1329.5 J	10.44%	34.66%	36.89%
Feedback		1190.7 J	27.04%	29.54%
Adaptive			868.713 J	3.42%
Combined				839.0146 J

Table 5.8: Energy Consumption and Savings Comparison When Tracking One Car

ilarly, 27.04% is the energy savings provided by the adaptive methodology compared to the feedback method.

5.5.4 Comparison of Battery Lives

The battery life of the camera node when employing the algorithms described in this chapter has also been estimated. The algorithms include the sequential method, the feedback method, the adaptive methodology, and the combined method. For this analysis, characteristic curves provided by the manufacturer of the batteries were used. These curves are shown in Figure 5.16. The battery lifetimes were predicted for a scenario, where there are always two cars in the scene, i.e., the scene was never empty. When the sequential detection/tracking method is used, and the camera continuously captures frames, the average current drawn is 0.230 A, and the estimated lifetime is

7 h. For the feedback method, the average current drawn is 0.2 A, and the estimated lifetime is 8.4 h. Since the scene is never empty, this gain is solely thanks to the savings in processing time. The adaptive method confers the ability to send the camera to idle state, even when the scene is not empty. Thus, the gain in battery life increases. The average current in this case is 0.136 A, and the battery life is 15.58 h. Finally, if the combined method is employed, the average current drawn is 0.131 A, and the lifetime increases to 16.17 h. It should be emphasized that the estimated lifetimes are based on the scenario that there will always be two objects to track in the scene. Table 5.9 summarizes the battery lifetimes when using the different algorithms.

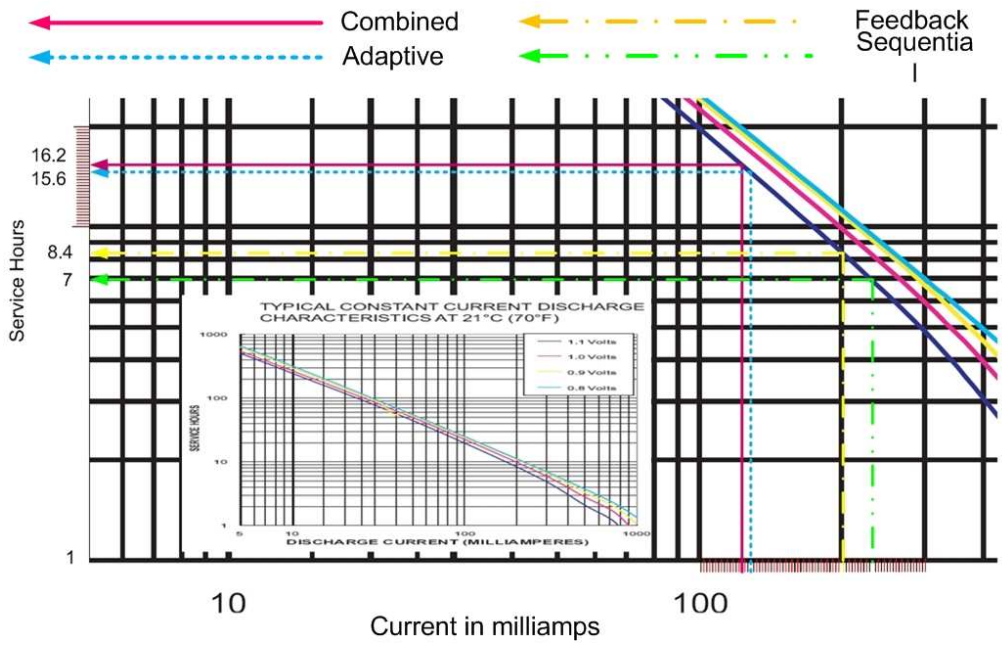


Figure 5.16: Characteristic curves of the batteries.

Method	Battery Lifetime (hours)	Percentage gain(%)
Sequential	7	-
Feedback	8.4	20%
Adaptive	15.58	122%
Combined	16.17	131%

Table 5.9: Battery lifetime projection

5.5.5 Analysis of the Tracking Performance

The adaptive methodology, and thus the combined method, depend on the tracking results to calculate the idle durations. Assuming that the tracking algorithm performs well when the camera continuously captures frames, one of the goals is to have no negative effect on the performance by going idle. If the bounding boxes before and after the idle duration do not overlap, it is considered as a failure caused by sending the camera to idle state. Below, a detailed analysis of the effect of these methodologies on tracking performance is provided. Additionally, this section shows how to handle objects that increase their speeds gradually, which is a very challenging scenario. In all the performed experiments, both indoor and outdoor, the combined methodology did not affect the tracking performance. Without dropping any frames, at a processing rate of 12.2 f/s, the tracking algorithm is highly robust and reliable with the car running at full speed, i.e., the bounding boxes of the car at consecutive frames always overlap. When dropping frames, the pixel displacement between the last saved bounding box, and the one obtained when the camera comes back from the idle state will be larger. If the camera drops too many frames, these boxes will not overlap.

In equation (4), Casares et al. [71] divide $N_{overlap}$ by 2, so that when the camera comes back from the idle state the bounding boxes can still overlap, even if pixel displacements of the object increase during this duration. In order to analyze the tracking performance when adaptively dropping frames, different scenarios were experimented. When a target moves away from the camera, the pixel displacements, on average, are smaller compared to the case where the target moves parallel to the camera. When the target moves toward the camera, the size of the bounding boxes gets larger, allowing more overlapping between the current and previous bounding boxes of the target. Thus, the most challenging scenario is when the target moves parallel to the camera, in which case the pixel displacement is large and the bounding box size does not change much. Since it is hard to increase or decrease the speed of a remote controlled car in a precise way, simulations for tracking performance analysis were performed. Thus, a worst case scenario in which bounding boxes do not overlap when the camera comes back from the idle state was simulated. After analyzing this scenario in detail below, a solution is instructed to overcome this problem.

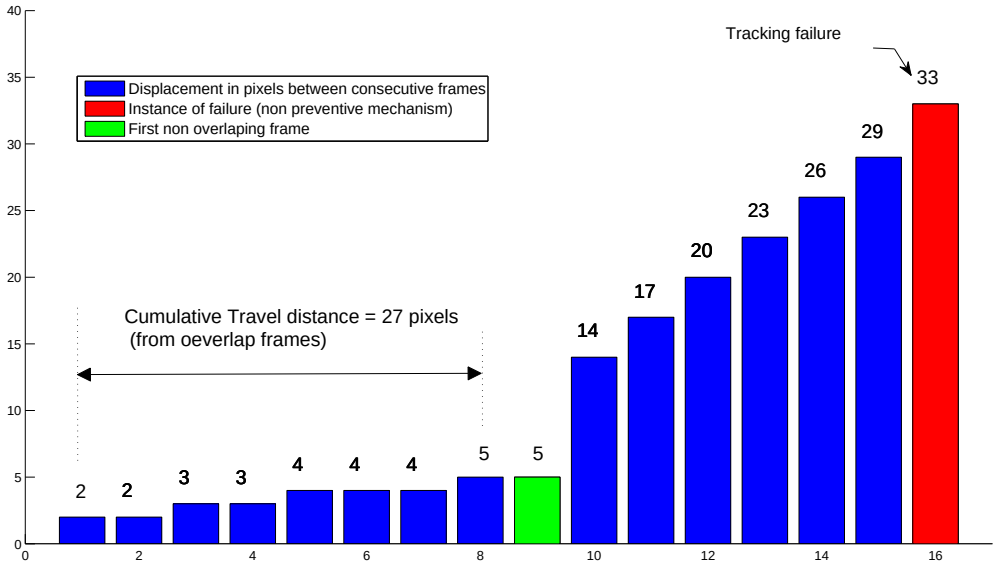


Figure 5.17: Scenario wherein the bounding boxes before and after the idle state do not overlap.

In this scenario, a remote-controlled car travels parallel to the camera. The width of the bounding box of the car is 30 pixels. Figure 5.17 shows the pixel displacements of the car between consecutive frames. The bounding boxes overlap for the first eight frames, i.e., $N_{overlap} = 8$. After nine frames, the camera is sent to idle state, and when it comes back from it, the pixel displacement D_{10} is 14 pixels. As expected, at the end of the idle duration, the pixel displacement is much larger, compared to the displacements at the beginning when the camera is always on. Then, the camera goes to idle state again, and when it comes back, the pixel displacement D_{11} is 17 pixels. Since D_{11} is not greater than $1.25 \times D_{10}$, the algorithm keeps using the same idle state duration. As shown in Figure 5.17, the same situation repeats between frames 11 and 12, 12 and 13, and so on. In other words, since the speed is gradually and very slowly increasing, the algorithm continues to use the same idle duration. However, at frame 16, the pixel displacement D_{16} between frames 15 and 16 becomes 35 pixels. This causes a tracking failure since the bounding boxes do not overlap.

To address the cases of gradual increases in speed, the first pixel displacement is saved, D_{init}, which was calculated after the camera comes back from the first idle state. In Figure 5.17, D_{init} corresponds to D_{10} which is 14 pixels. At the next frame, the algorithm compares the new obtained

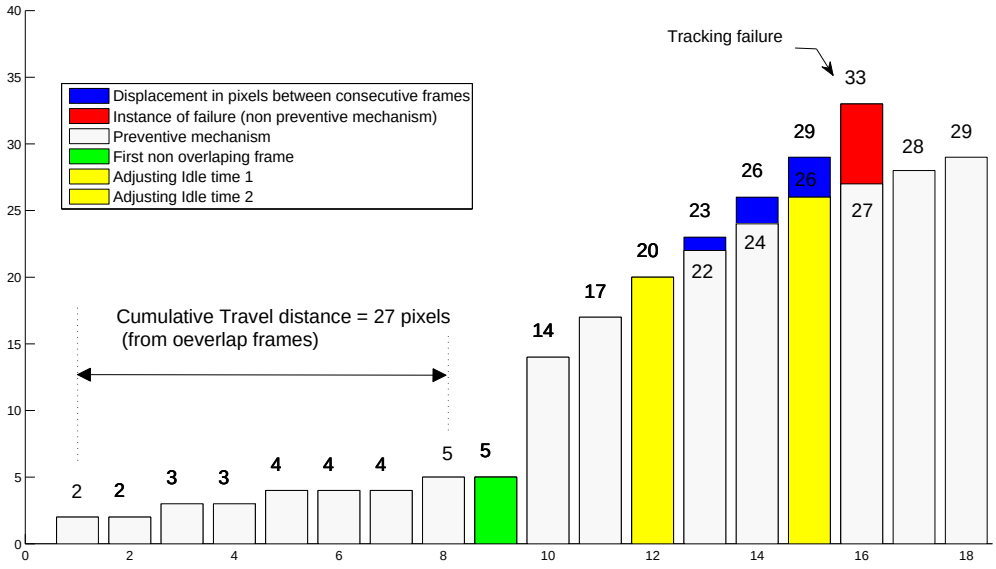


Figure 5.18: Tracking with a preventive mechanism that can handle the gradual increases in speed and resolve the issue seen in Figure 5.17.

displacement with D_{init}. If a significant change is detected in pixel displacement (25% of D_{init}), the algorithm now adapts the idle time accordingly. Figure 5.18 illustrates how these cases are handled and how the issue, shown in Figure 5.17, is resolved. At frame 12, the pixel displacement of the car is $D_{12} = 20$ pixels, which is grater than $1.25 \times D_{init}$. Thus, the camera reduces the idle period, and D_{12} is saved as the new D_{init}. The reduction in idle duration is reflected in the following frames as seen in Figure 5.18. For example, at frame 14, the pixel displacement is now 24 as opposed to 26. At frame 15, the pixel displacement of the car is $D_{15} = 26$ pixels which is grater than $1.25 \times D_{init}$. Therefore, the idle duration is decreased again. At frame 16, the pixel displacement becomes 27 as opposed to 35, and the case of non-overlapping bounding boxes is avoided. As mentioned above, when a target moves away from the camera, the pixel displacements, on average, are smaller compared to the case where the target moves parallel to the camera. This makes it less challenging in terms of overlapping of the bounding boxes. When the target moves toward the camera, the size of the bounding boxes gets larger allowing more overlapping between the current and previous bounding boxes of the target.

5.6 Conclusions

We presented lightweight algorithms and self-adapting methodologies to increase the energy efficiency and battery life of an embedded smart camera node. The proposed methodologies allow us to send the microprocessor of the camera node to idle state even when there are tracked objects in the scene. First, we presented results from a feedback method introduced in chapter 4 for detection and tracking, which provides significant savings in processing time. We took advantage of these savings by sending the microprocessor to idle state at the end of processing a frame. We also presented an adaptive methodology that significantly decreases the energy consumption of the embedded smart camera. The camera can be sent to idle state not only when the scene is empty but also when there are tracked objects in the scene. The amount of time the camera remains in idle state is adaptively changed based on the amount of activity in the scene, and the speed of tracked objects. Instead of continuously capturing and processing every frame, the camera drops frames during idle state while preserving the tracking performance, and thus, system reliability at the same time. This significantly prolongs the battery life. We then presented a combined method that employs the feedback method and the adaptive methodology together, and provides further savings in energy consumption. We presented experimental results showing the gains in processing time as well as the savings in energy consumption and the gain in battery life. In summary, the feedback method provides 48.7% decrease in the processing time of a frame, and 10.44% savings in energy consumption, compared to traditional sequential tracking, when tracking one object. Employing the combination of the proposed feedback algorithm and the proposed adaptive methodology, provides 37% savings in energy consumption when tracking one car. In a scenario where there are always two cars in the scene, the combined method provides 131% gain in battery life. The proposed combined method depends on the tracking results to calculate the idle durations. We presented that, assuming the tracking algorithm performs well when the camera is always on, the presented methodologies do not affect the tracking performance.

Chapter 6

Energy-efficient Feedback Tracking on Embedded Smart Cameras by Hardware-level Optimization

As in the previous chapters, decreasing the processing time and energy consumption on the embedded smart camera is the main goal in this chapter. To achieve this goal, two main operations have been performed at hardware level: (i) the change of the image resolution and (ii) image cropping based on a search region obtained from the tracking stage. To fully understand these two concepts, explaining how the camera grabs a frame is important.

6.1 Frame Capture Operation

On the CITRIC camera platform, there is an interface called Quick Capture Interface or QCI. This interface works in 10-bit Master Parallel mode. It requires a parallel data bus interface, two control signals for frame timing and a pixel clock for basic timing.

The Quick Capture Interface on the CITRIC camera operates in 8-bit YCbCr 4:2:2 mode. Such mode allows the image sensor to provide the line and frame synchronization signals; signals which are also referred to as the Horizontal Reference signal and Vertical Synchronization signal, HREF and VSYNC, respectively. The QCI provides a programmable master clock (MCLK) to interface with the image sensor attached to the camera. Additionally, there is a Pixel Clock (PCLK) derived

from the MCLK. The PCLK is used to perform all operations associated to frame transferring. The CITRIC camera is programmed to operate in YCbCr color space having a luminance channel (Y) and two chrominance channels Cb and Cr. The 8-bit 4:2:2 format samples the captured frame by transferring 16bits per pixel using two clock cycles from the PCLK.

The operation between sensor and the QCI on the CITRIC camera board is defined as the Master mode. This refers to a mode of operation in which the image sensor provides the line and frame synchronization signals, HREF (line valid) and VSYNC (frame valid) as shown in Figure 6.1. In the Intel PXA270 master mode, the line valid and frame valid signals are inputs to the quick capture interface.

The sensor can be programmed for exposure, frame rate, and additional parameters. The programming is done through a separate interface, namely the I2C serial control interface. Once configured, the sensor begins providing data in addition to generating the frame and line synchronization signals. The MCLK signal output for the sensor is programmable. The timing signals VSYNC and HREF, provided by the sensor, activate and reset the quick capture interface that can be configured to provide an interrupt at the end of each line and each frame as shown in Figure 6.1.

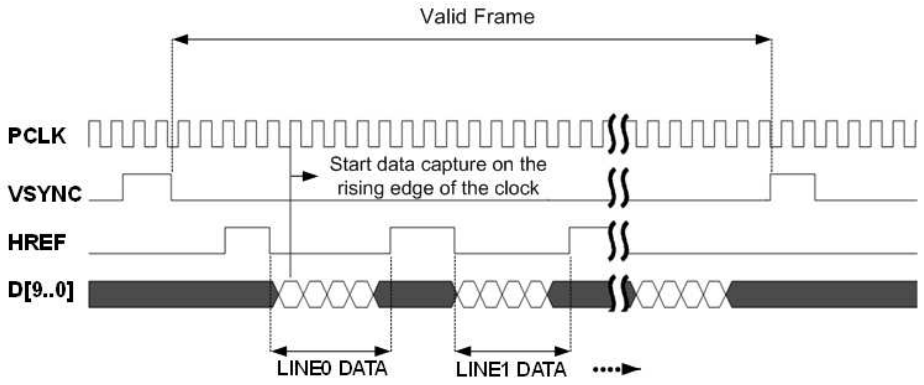


Figure 6.1: Timing diagram for grabbing a frame using the Quick Capture Interface.

6.1.1 CITRIC Middleware Interface

To adaptively modify the shape of the HREF and VSYNC signals, a set of 8-bit registers is altered through the I2C interface of the main microprocessor at the camera board. The image sensor configuration is performed at the Application Program Interface (API) level, where libraries are developed using the Software Development Kit (SDK) provided by the CITRIC camera. A device driver is designed to load the correct values to the registers according to the frame size required by the user.

As previously mentioned, the CITRIC camera runs embedded Linux. The original kernel version running on the CITRIC camera was an optimized and a patched kernel imported from the original Linux kernel 2.6.9. The image sensor of the CITRIC camera is handled by a device driver. The camera drive is obtained by customizing both the "Video-For-Linux-One" driver for the OV9650 image sensor and ARM processor driver, so that the driver can work for the newest OV9655 image sensor. As previously described in Chapter 2, and detailed in the manufacturing manual [76], the image sensor is equipped with two different interfaces as shown in Figure 6.2. The first one, called the Serial Camera Control Bus (SCCB) interface, is used to program the sensor behavior. The second interface, the Digital Video Port interface, provides a connection between the sensor and the quick capture interface to acquire data and control signals, and performs the appropriate data formatting prior to routing the data to memory.

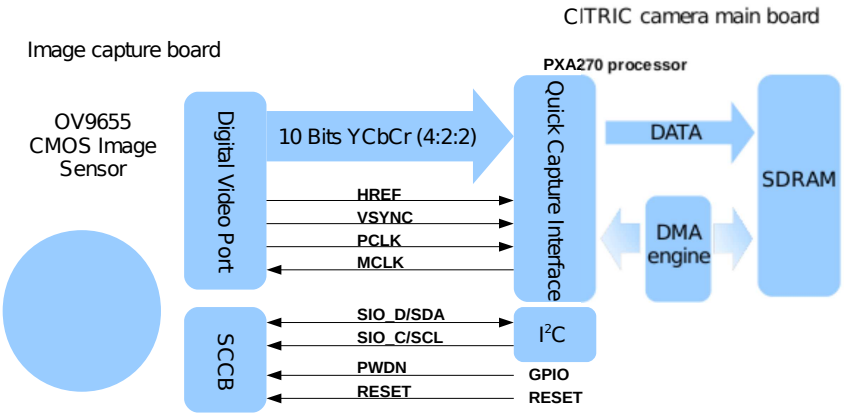


Figure 6.2: Interconnection of OV9655 and the Intel Quick Capture Interface on ARM PXA270.

The software to perform (i) the change of the image resolution, and (ii) image cropping mentioned at the beginning of the section consists of several functions. These functions are used in "live" or "run-time" mode, and some of them are employed to dynamically change the position and the size of the cropped window inside the whole image. The functions for the reconfiguration of the quick capture interface and the control of the Dynamic Memory Allocation engine (DMA) are based on the Video 4 Linux Standard IOCTL (Input/Output-Control), which allows us to collect the right amount of data sent by the image sensor. Additionally, some of these functions are used to clean the frame circular buffer used by the device driver. Since the frame rate of the image sensor can reach 30 f/s, and the frame transferring works in FIFO mode, the circular buffer shown in Figure 6.3 is to be reset to guarantee the grabbing of the latest available frame. In this way, all the video processing tasks are assured to be performed on the frame carrying information of latest location of the object being tracked.

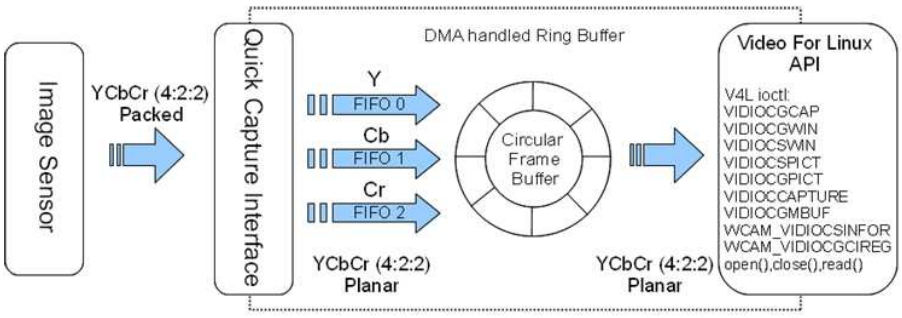


Figure 6.3: Camera Driver Internal architecture.

The functions for reconfiguring the image sensor register set are used at user application interface, and it has been added to the API library of the CITRIC camera SDK. The other functions work at the kernel space, and they have been implemented as part of Embedded Linux OS device drivers. In particular, most of the additions and modifications to handle the image sensor were done on the API IOCTL originally provided by Linux OS. The implemented functions are listed in Figure 6.3 to the right of the frame circular buffer. The tracking algorithm employs these functions to achieve time synchronization capabilities permitting us to perform tracking in "run-time".

The operating system architecture of the CITRIC camera is also presented in Figure 6.4. The striped yellow boxes are the modules that have been modified to dynamically change the size of the cropped window for tracking purposes. These boxes show where most of the work has been performed to accomplish the hardware-level optimization.

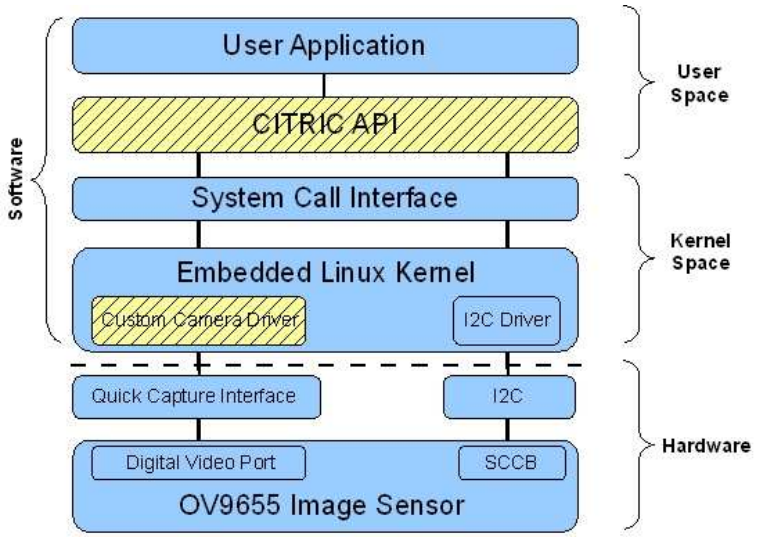


Figure 6.4: Software architecture handling the CITRIC camera board.

The hardware subsystem composed of the image sensor and the quick capture interface (QCI) is highly configurable. The flexible and configurable architecture of the CITRIC camera, which allows us to perform functions at hardware level, provides a reduction in the amount of transferred data. This, in turn, leads to significant savings in energy consumption due to the better use of the memory controller and the memory resources. Additionally, freeing the main microprocessor from the tasks of performing image down-sampling and cropping at software-level also contributes to saving energy. Down- sampling, scaling and cropping operations are accomplished by changing the hardware registers of the OV9655. The acquisition of data from the sensor is initiated by transitions based on the state of the HREF and VSYNC signals (Figure 6.1), which are generated internally as explained in the OV9655 operation manual [76], and described in section 6.1.

6.2 Hardware-Level Image Processing Tasks: Scaling and Cropping

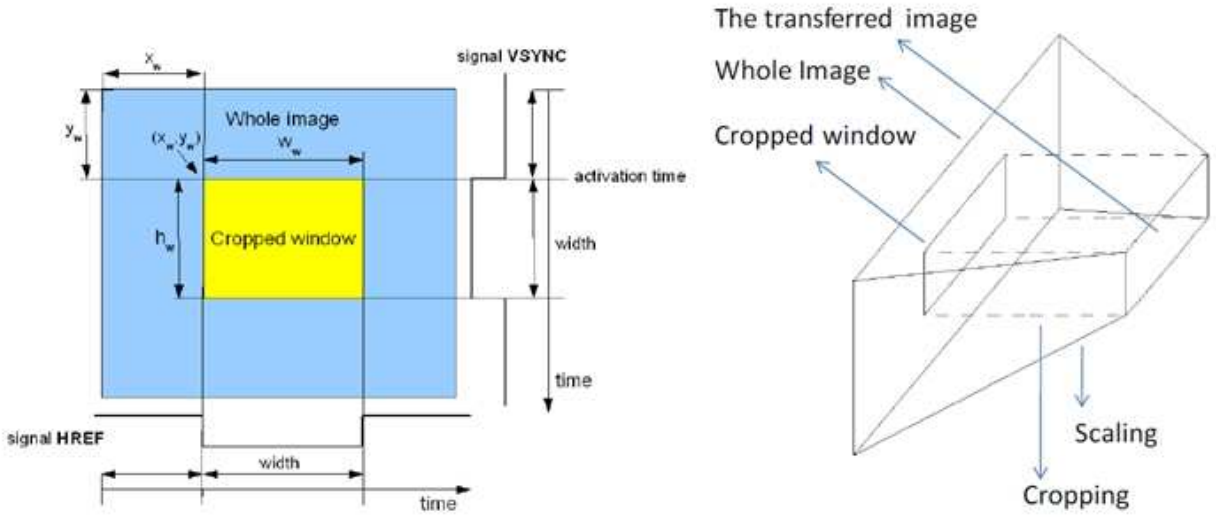


Figure 6.5: Hardware-Level Image processing tasks: Scaling and Cropping

The image cropping is the selection of an area inside the whole image. This area is named "cropped window" and characterized by its position, width, and height. The position is the pixel coordinates of its upper left corner inside the whole image. The synchronization signal VSYNC indicates which sequence of lines has to be captured in a frame. Similarly, the signal HREF indicates which sequence of pixels has to be captured in each line as shown in Figure 6.5.

To perform down-sampling and grab a frame in QVGA resolution, the VSYNC and HREF are set so that the whole information acquired by the sensor is used. Moreover, it is necessary to select the zoom and scaling functionality. To set the horizontal and vertical scale down coefficients, the image sensor register set are accessed and modified. As will be detailed in sections 6.3 and 6.6, hardware-level cropping provides significant savings in energy consumption and increases the battery lifetime of the camera. The localized foreground object detection and tracking algorithm introduced in Casares et al. [70] and Chapter 4 is an application that takes advantage of hardware-level cropping.

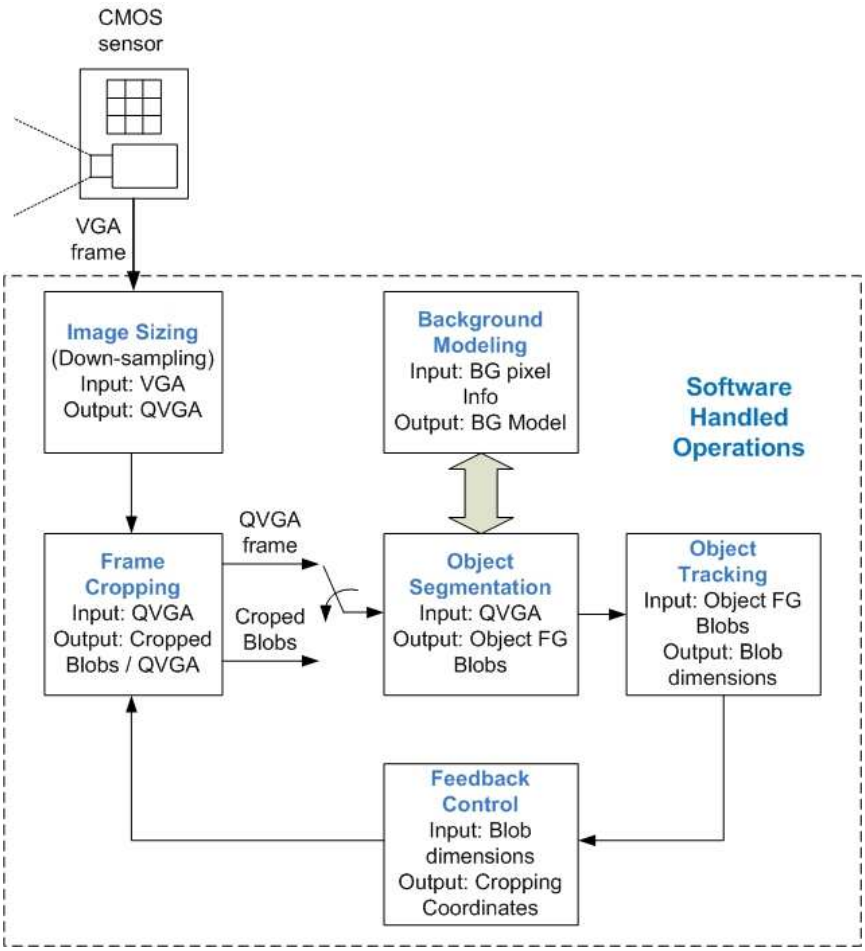


Figure 6.6: Interaction among components used in the Software based-Feedback method (Chapter 4.)

In chapter 4 the concepts of sequential and feedback tracking were introduced. Performing detection and tracking only on specific regions, instead of on the whole frame, was shown to provide significant savings in processing time. Hence, it increases idle state duration of cameras to increase the battery-life. Even though significant savings were reported, the algorithms and methodologies presented in the previous chapters were entirely done by software-level as seen in Figure 6.6. The diagram presented in Figure 6.7, compared to Figure 6.6, demonstrates the goal to be accomplished in this chapter. It also shows the tasks handled by hardware as well as the

subroutines implemented by software.

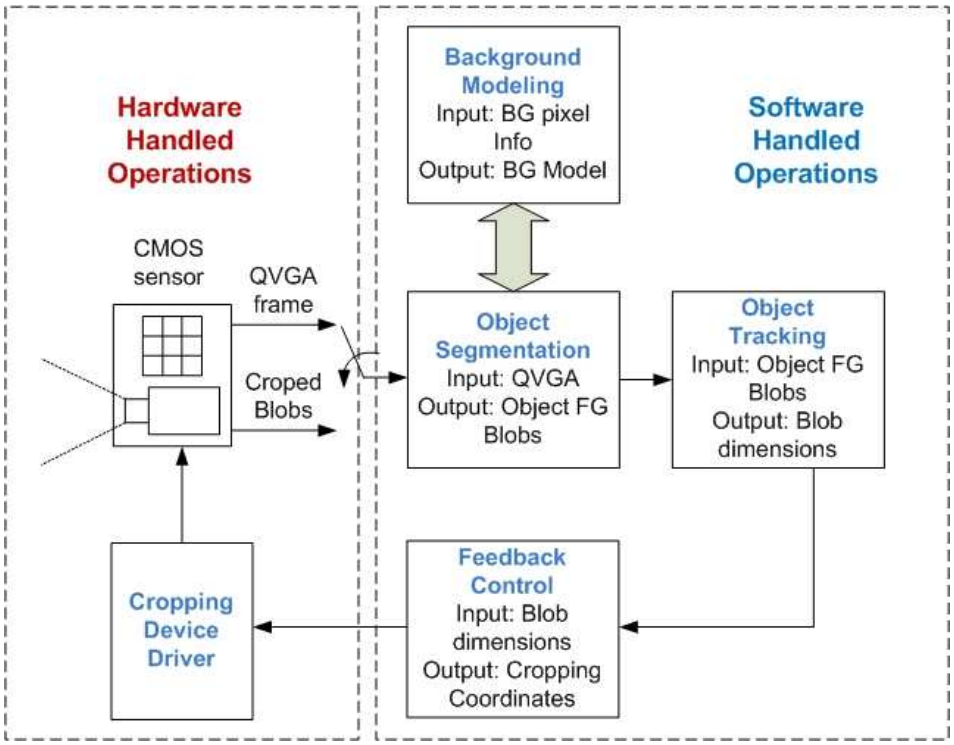


Figure 6.7: Interaction among components used in the Hardware based-Feedback method.

The feedback method [70] explained in Chapter 4 is employed to determine a search region in the following frame. Subsequently, the next image is cropped at hardware-level as described above. After cropping, the detection and tracking are performed on the search areas as seen in Figure 6.8. The experimental results showing the decrease in energy consumption and the increase in battery-life will be presented in Sections 6.3 and 6.6, respectively.

To actually implement the tracking system, the original CITRIC-kernel-2.6.9 has been updated to version 2.6.23, and the Linux device driver for the image sensor has been modified. The kernel of the CITRIC camera was not capable of dynamically changing the size of the cropped regions from frame to frame. Thus, to overcome this issue, the existing device driver of the OV9655, contained in the CITRIC-kernel-2.6.23, has been customized so that it can dynamically crop regions in run-

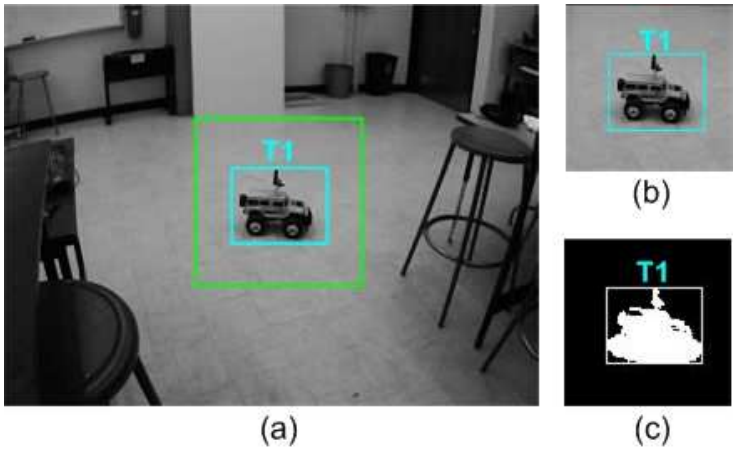


Figure 6.8: Area cropped (a) by software using the API libraries (b) by hardware using the micro-controller of the OV9655, (c) background subtraction output on the cropped region

time.

6.3 Savings in Energy Consumption

In this section, a quantitative comparison is presented, which shows the advantages of performing hardware-level down-sampling and cropping at the micro-controller of the OV9655 sensor for tracking purposes. Rather than processing whole frames and performing these tasks at software-level on the main micro-processor of the camera board, the OV9655 micro-controller will be used.

Before immersing into the analysis of more complex vision tasks, such as object detection and tracking, it is worth presenting the gains of exploiting hardware-level operations even at elemental tasks such as the grabbing of a QVGA frame.

6.3.1 Analysis of grabbing a QVGA frame

Grabbing a frame in QVGA (320×240) resolution is the result of applying down-sampling to VGA images. As mentioned above, this operation was being done at software level on the main ARM processor of the camera board. In this chapter, down-sampling have been performed at hardware-level at the micro-controller of the OV9655 sensor has been performed. Figures 6.9(a) and (b)

show QVGA images captured by the CITRIC camera using software and hardware down-sampling methods, respectively. At hardware-level, neighborhood averaging is used to down-sample. At software–level, instead of averaging, the API library routines drop repetitive information during the down-sampling. Thus, Figure 6.9(a) is slightly sharper compared to Figure 6.9(b).

(a) (b)

Figure 6.9: QVGA images captured by (a) using the API software library down-sampling subroutines and (b) performing hardware–level down-sampling on the micro-controller of the OV9655.

Figure 6.10 shows the operating currents of the camera board while grabbing a QVGA frame. By using (i) only the API software libraries, and (ii) the OV9655 and the quick capture interface at hardware-level. The grabbing of a frame takes 49.8 ms when using the API libraries, while it takes 30.78 ms when employing the hardware-level down-sampling at the micro-controller of the image sensor. This corresponds to 38.2% savings in grabbing time.

The solid and dashed lines in Figure 6.10 show the average current levels when using software-level and hardware-level down-sampling, respectively. As can be seen, a 36.27% reduction in the average operating current is obtained when performing hardware-level down-sampling at the microcontroller of the OV9655 sensor. As shown in Table 6.1, this corresponds to 24.47% decrease in energy consumption. It should be noted that to compare the energy consumption in both scenarios, the main ARM processor has been sent to idle state for 19 ms, so the time window is the same (49.8 ms) for both cases (Figure 6.10).

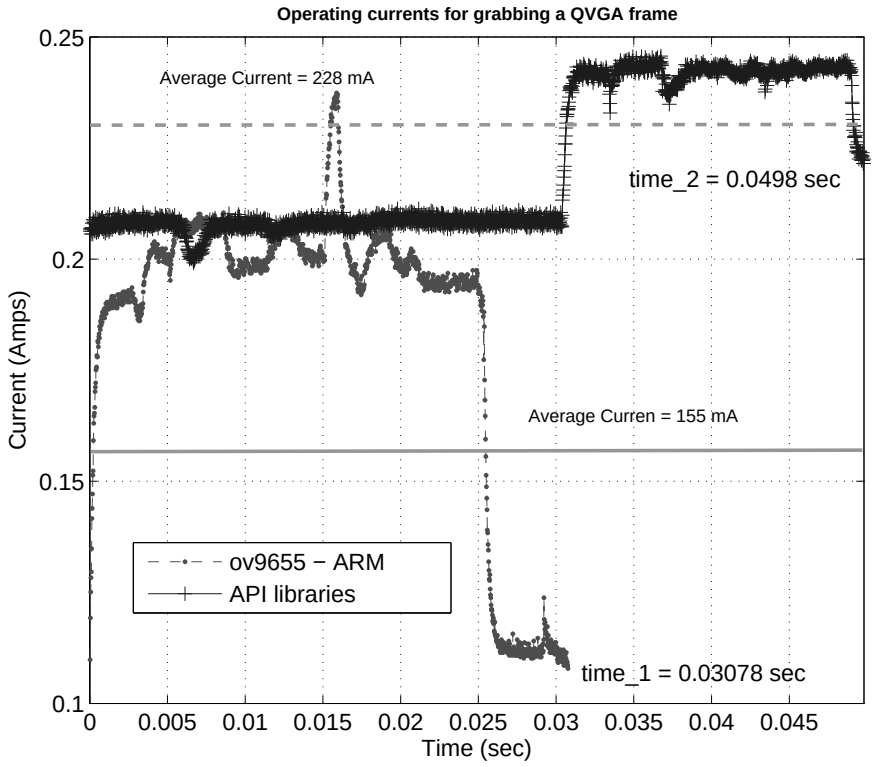


Figure 6.10: Operating currents of the camera board while grabbing a QVGA frame using the API sub-sampling subroutines and using the image micro-controller of the OV9655.

6.3.2 Analysis of hardware-Level image/video processing tasks: Object detection and tracking

In this section, savings in energy consumption are presented when performing hardware-level down-sampling and cropping, while using the feedback method for object detection and tracking (Casares et al. [70]) described in Chapter 4.

As stated in Casares et al. [70], the feedback method provides significant savings in processing

Down-sampling method	Power (W)	Energy (mJ)
Software	1.1655	57.2
Hardware	0.7493	43.2
Gain (%)	35.71%	24.47%

Table 6.1: Energy consumption when grabbing a QVGA frame using the API software libraries versus performing down-sampling at hardware-level.

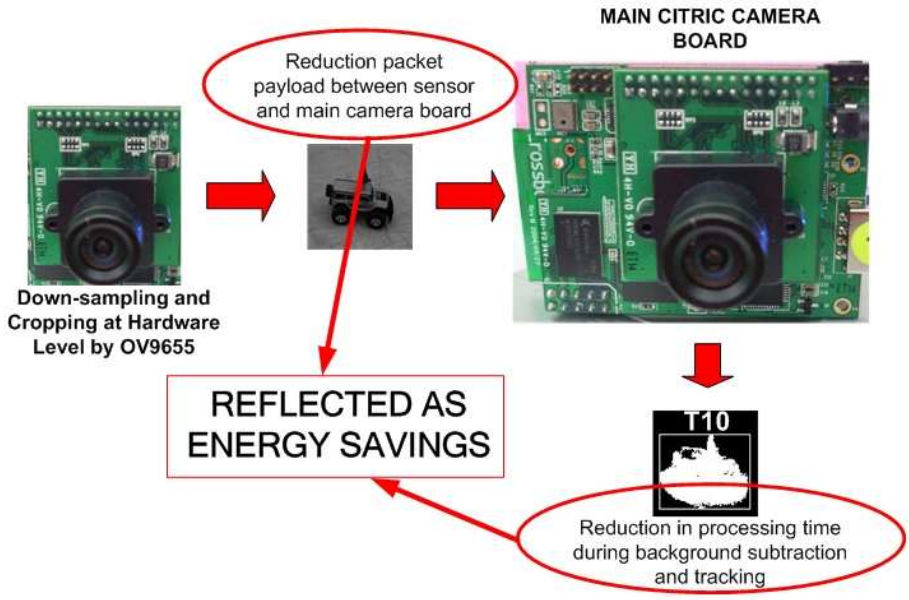


Figure 6.11: Illustration of saving gain by using hardware level operations.

time, and thus allows us to increase idle state durations of cameras to increase the battery-life. As described in chapter 4 section 4.4, in the feedback method, information from the tracking stage is used to determine search regions in the next frame, so that detection and tracking can be performed only in those regions instead of the whole frame. Figure 6.11 presents a diagram to illustrate the process.

Object detection on a QVGA frame

In this section the following scenarios are compared: (i) obtaining QVGA images with software-level down-sampling and performing all processing (down-sampling and foreground object detection) on the main microprocessor of the camera board; (ii) performing down-sampling at hardware-level on the micro-controller of the OV9655 sensor, and performing foreground object detection at the main microprocessor.

Figure 6.13 shows the operating current levels of the camera board when using these two approaches. As seen in this figure, collaborating with the image sensor, and using hardware-level

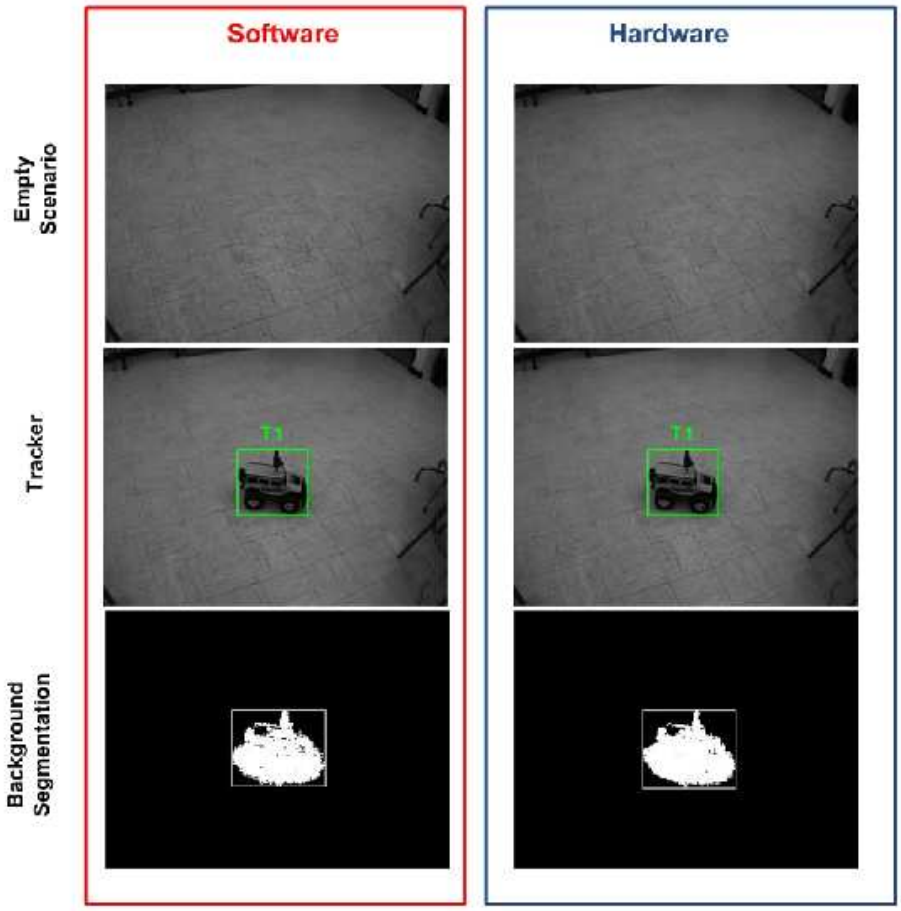


Figure 6.12: Background subtraction output on a frame grabbed by using the API software libraries to down-sample to QVGA resolution (left column), and by using hardware level down-sampling (right column).

operations, provides 43.7% savings in processing time, as compared to the software-level down-sampling relying on the API libraries. In addition, it provides 27.98% savings in energy consumption. As seen in Figure 6.12, the background subtraction output is slightly better when using the hardware-level down-sampling, due to the slight blurring introduced by averaging neighboring pixels, as discussed in section 6.3.1. This provides noise reduction, and thus better segmentation.

Object tracking on a QVGA frame

In this section, savings in energy consumption, when using the feedback method for object detection and tracking, and performing hardware-level cropping, are presented. The aforementioned

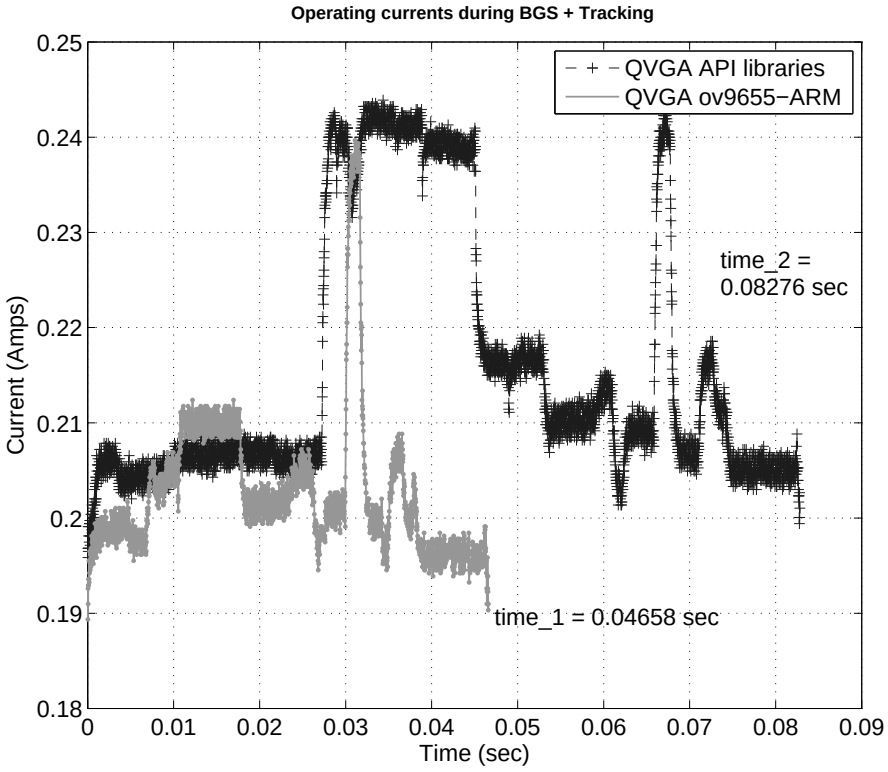


Figure 6.13: Operating currents when grabbing/buffering a frame and performing background segmentation using the API sub-sampling subroutines versus collaborating with the OmniVision OV9655.

scenarios analyzed for QVGA resolution, are now performed in a reduced search region cropped by software or hardware-level operations. The software-based feedback method [70] grabs a frame in VGA resolution, down-samples it, and crops the search regions all by software. On the other hand, the hardware-level method uses the capabilities of the OV9655 to down-sample, and then crop the search regions. Having the search regions, foreground detection and tracking tasks are performed only in those regions at the main micro-processor of the CITRIC camera, as depicted in Figure

	QVGA		
Method	Power (W)	Energy (mJ)	gain (%)
Software-level down-sampling	1.0415	112.5	–
Hardware-level down-sampling	0.751	81.7	27.38%

Table 6.2: Energy consumption when grabbing a QVGA frame at Software versus Hardware-level, and performing detection at the main microprocessor.

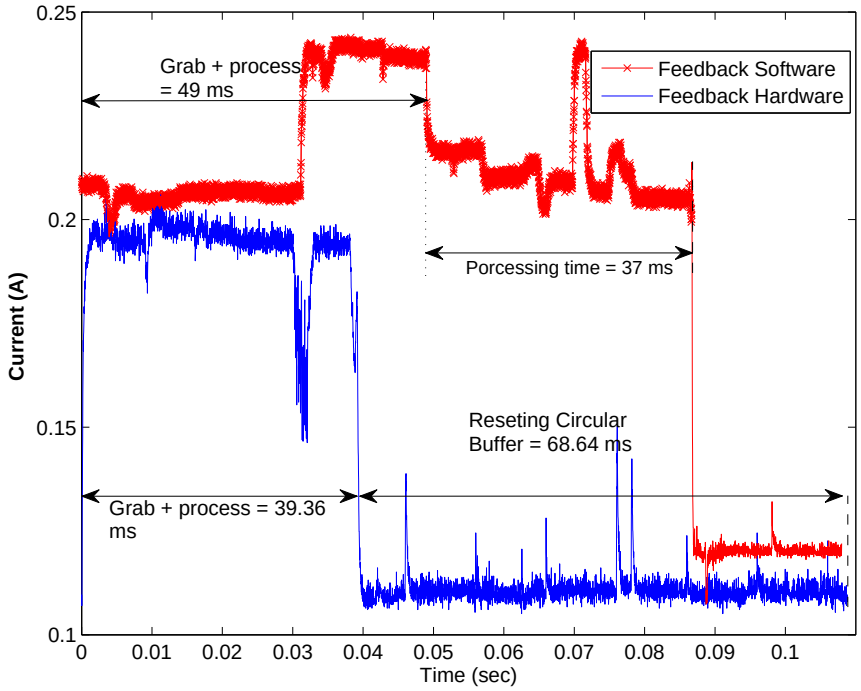


Figure 6.14: Operating currents when (i) obtaining QVGA images with software-level down-sampling, and performing all processing on the main microprocessor ; (ii) performing down-sampling at hardwarelevel on the micro-controller of the OV9655 sensor, and performing foreground object detection and tracking at the main microprocessor.

6.7. However, before presenting the energy consumption analysis during feedback-based tracking, combined with hardware-level cropping, the following two scenarios are firstly compared on a single QVGA size frame. To separately show the contribution of hardware-level down-sampling in terms of savings, we first: (i) obtain QVGA images with software-level down-sampling, and perform all processing (down-sampling, foreground object detection, and tracking) on the main microprocessor of the camera board; (ii) perform down-sampling at hardware-level on the micro-controller of the OV9655 sensor, and foreground object detection and tracking at the main micro-processor. The operating currents of the camera board, while using these approaches, are presented in Figure 6.14. Even though collaborating with the image sensor and hardware-level operations slightly prolongs the processing time per frame by 22 ms, they considerably decrease the energy consumption of the camera by 27.38% as presented in Table 6.2.

To continue the explanation of the proposed algorithm, Figure 6.12 shows the output of the system when detecting and tracking an object. The reader can compare side by the hardware-level approach from this chapter against the software-level introduced in chapter 4.4.

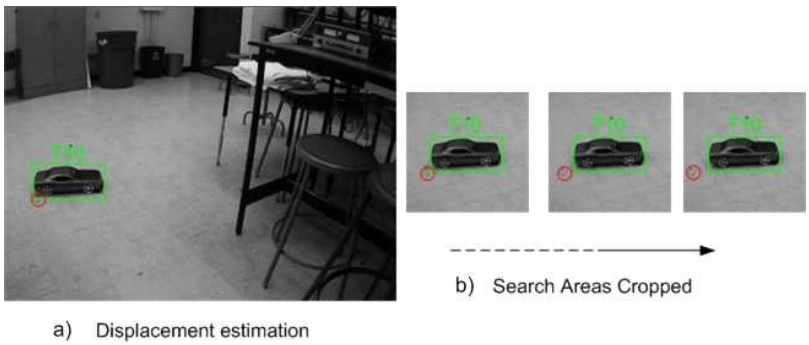


Figure 6.15: (a) Last QVGA frame captured while computing pixel displacement of the tracked object; (b) Search regions cropped at hardware level.

Figure 6.15 shows a sequence of frames in which a remote-controlled car is tracked. Figure 6.15(a) shows a QVGA frame grabbed during the tracking of the remote-controlled car. Whole frames are grabbed until the displacement of the target is computed from two consecutive frames. Then, the location of the target is estimated at the following frame. A search region of size $2w \times 2h$ is formed around this location, where w and h are the width and height of the bounding box in the current frame, respectively. The details can be found in Chapter 4. Then, the following frame is cropped to the search region at hardware level, and the detection and tracking are performed only in the cropped region as depicted in Figure 6.15(b). To show the movement of the car, and the changing cropped window, a small red circular reference point is highlighted on the cropped frame sequence. Figure 6.16 shows the operating current of the camera board when (i) using the feedback method implemented entirely at software level; and (ii) applying hardware-level operations for cropping and down-sampling.

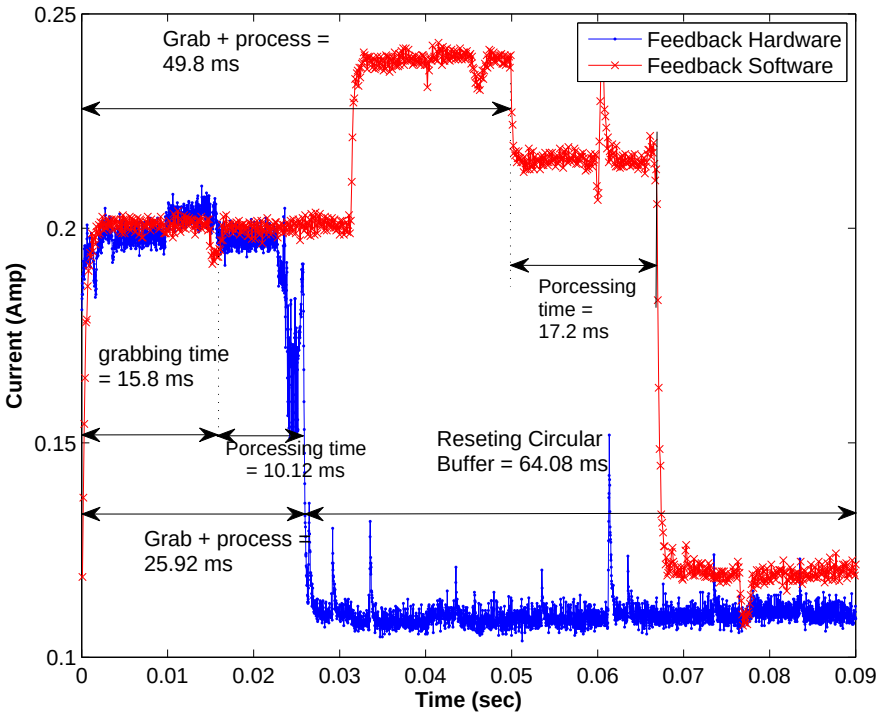
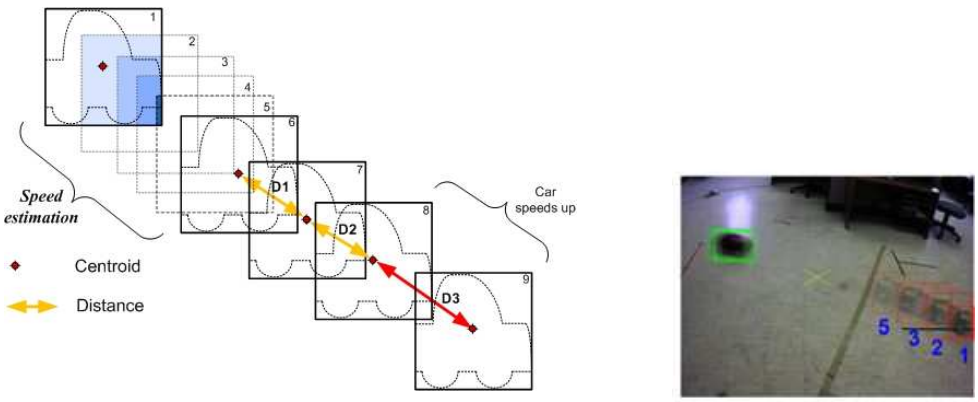


Figure 6.16: Operating currents when performing foreground object detection an tracking on cropped search regions obtained by software versus hardware-level cropping.



(a) (b)

Figure 6.17: (a) Detecting a speed change. (b) Overlapping bounding boxes for a faster car.

Even though the processing time increases by $23ms$ when cropping and processing a search region of 100×100 pixels at hardware level, using the hardware capabilities of the OV9655 provides

	100x100 Search Area		
Method	Power (W)	Energy (mJ)	gain (%)
Sequential software-level	1.0415	112.5	–
Feedback software-level	1	92.23	18.02%
Feedback hardware-level	0.719	66.1	41.24%

Table 6.3: Energy consumption when grabbing and cropping a search region (100x100) at software versus hardware-level and performing detection at the main microprocessor.

28.3% decrease in energy consumption, compared to software-level cropping and processing and 41.24% compared to a Sequential software tracking system. Different scenarios are summarized in Table 6.3 presenting the energy consumption and savings when processing a single frame.

6.4 Longer Tracking Experiment

This section aims to present a detailed analysis of the tracking algorithm over a prolonged period of time. Thus, rather than reporting results at the frame level, the estimation of the energy consumed by the camera is calculated over a longer time interval.

In the following set of experiments, first, a remote-controlled car is tracked continuously for 3 seconds, and the size of the cropped window is changed once every second. The energy consumption during this period of time is measured and reported in Table 6.4. Figure 6.18 shows the operating current of the camera board for different scenarios during 1-second portion of this 3-second experiment. As explained in Section 6.1.1, the circular buffer is reset when performing hardware-level cropping, which slightly increases the processing time of a frame. However, the feedback method combined with hardware-level cropping provides 29.4% and 37% decrease in energy consumption, compared to the software-based feedback method and sequential method, respectively. Table 6.4 summarizes the power and energy consumptions, and savings.

Method	Power (W)	Energy (J)	gain (%)
Sequential software-level	1.1422	3.4273	–
Feedback software-level	1.0203	3.0608	10.7%
Feedback hardware-level	0.7203	2.1609	37%

Table 6.4: Energy Consumption when performing detection and tracking during a 3-second time interval at software versus hardware level.

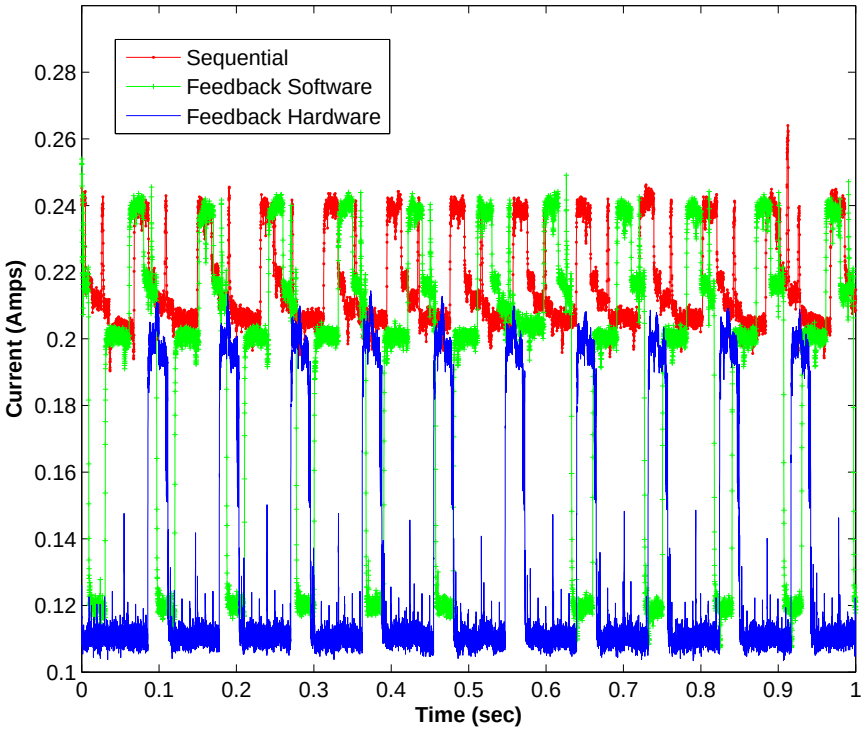


Figure 6.18: Operating currents when performing foreground object detection and tracking during 1-second time interval.

6.5 Outdoor experiments

Figure 6.19 shows an scenario in which a person enters to the FOV of the camera. The camera built the background model of the scene employing the algorithm described in chapter 3. After the foreground detection, the camera assigns a tracker number $T10$ to the person. As illustrated, the the foreground object segmentation and tracking were performed in a reduced cropped region

around the person. The cropping was performed at the hardware level using the logistics from chapter 4. We can also see that the camera processes the whole frame twice a second looking for new object that could have entered to the scene.

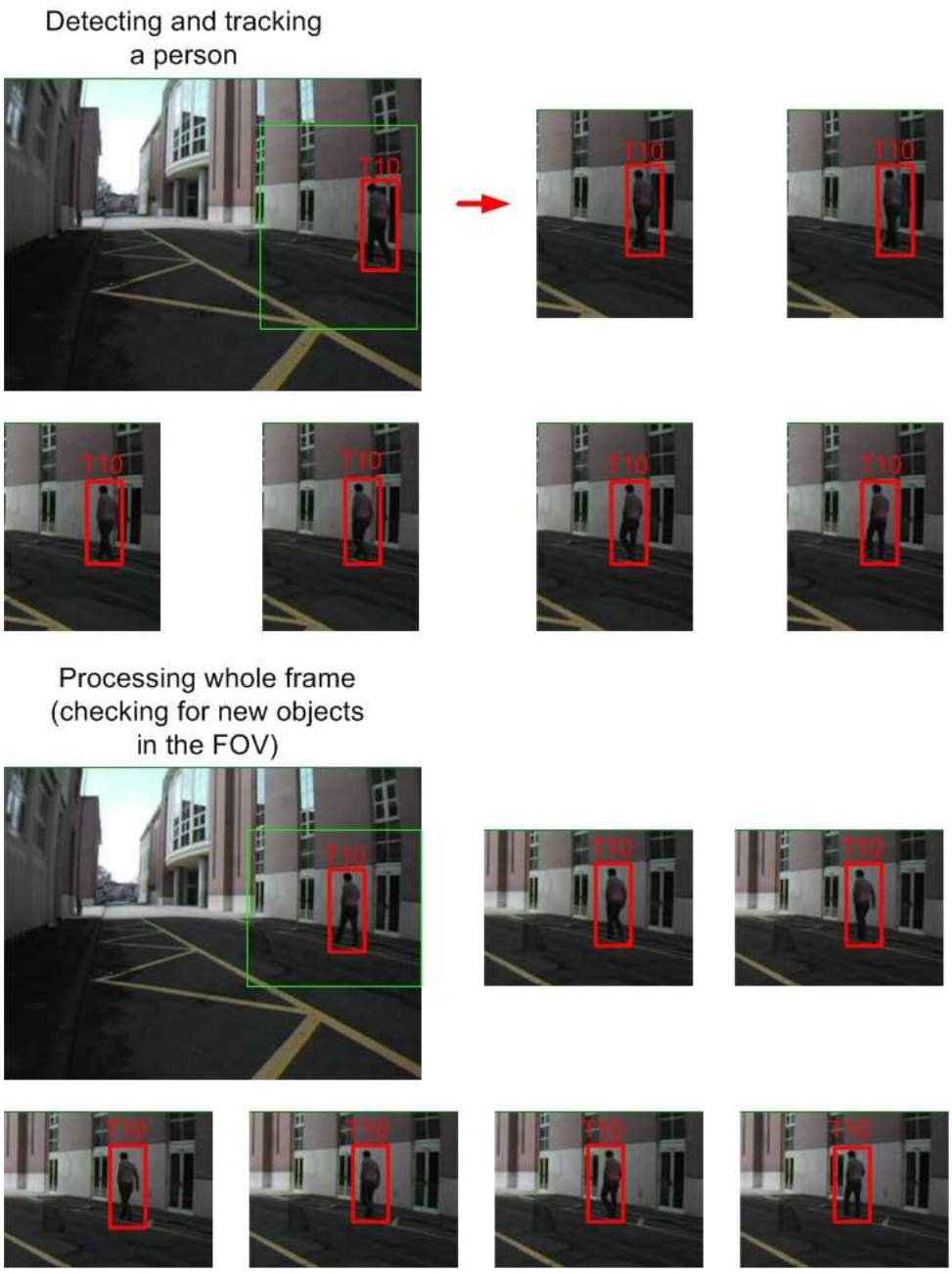
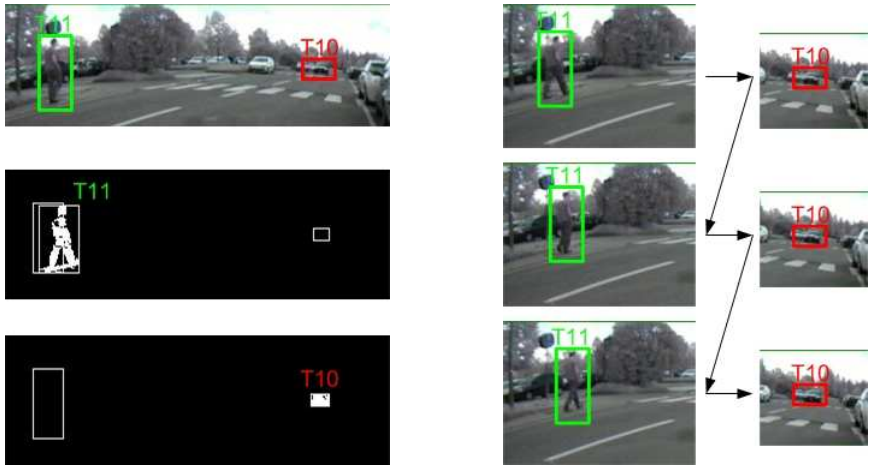


Figure 6.19: Outdoor experiment: Detection and Tracking of a person by employing hardware level operations.

6.5.1 Tracking multiple objects

When there are multiple objects in the scene, it is required to form multiple search regions, and crop multiple windows. In this case, hardware-level cropping can still be performed for one window per frame, and different windows for different objects can be cropped at alternating frames. Figure 6.20 shows a real life scenario in which two objects are being tracked. Figure 6.20(a) shows part of the original QVGA frame illustrating both of the objects (a person and a car) to be tracked. Figure 6.20(b) shows the hardware-level cropping on alternating frames.



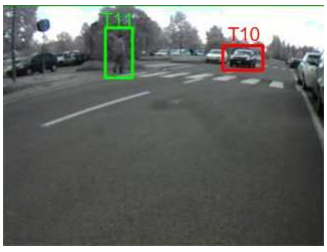
(a) (b)

Figure 6.20: Alternating BGS outputs from two objects being detected and tracked.

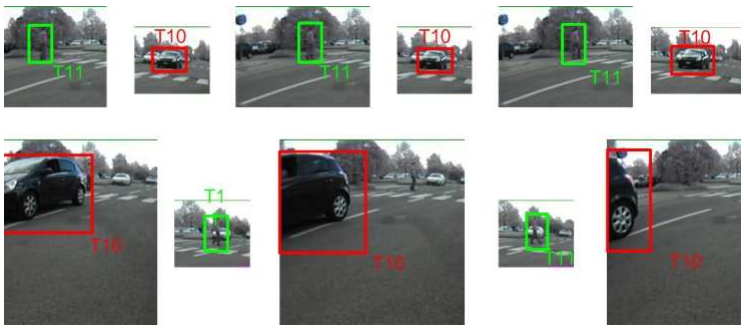
The person in Figure 6.20(b) is labeled $T11$ while the vehicle is assigned a tracker number $T10$. Additionally, in Figure 6.20(a) the alternating background subtraction (BGS) outputs are illustrated. The BGS outputs were obtained at the cropped areas. Thus, a frame alignment with respect to the QVGA background model was required. Details on the building of the QVGA background model are described in chapter 3. The empty white bounding boxes represent where the BGS is going to be performed in the next frame. Hence, it can be seen that there is no background subtraction in the car region when analyzing the cropped frame corresponding to the person, and vice versa. In the BGS output for the person in Figure 6.20(a), it can be seen two bounding boxes are associated with tracker $T11$. Those bounding boxes correspond to the current and previous

instances, t and $t-1$, respectively.

(a) Detecting and tracking a vehicle



(b) Analyzing whole frame (person enters FOV)



(c) Analyzing cropped frames alternatively

Figure 6.21: Alternating tracking of a person and a vehicle on hardware scaled and cropped frame areas.

A sequence of frames from the car-person experiment are illustrated in Figure 6.21 [a-c]. The system tracks the objects alternatively according to the total number of objects in the FOV of the camera. In Figure 6.21(a) the car entered to the FOV of the camera and a tracker $T10$ was created and assigned to it. In Figure 6.21(b) when the camera processed the whole frame, a person is detected and assigned a tracker $T11$. Finally, the alternating hardware cropped frames are presented until the car left the scene, and the system continues tracking the person.

6.6 Increase in Battery-Life

This section is focused on analyzing the gain in battery-life of the camera mote. Thus, the battery-life of the camera has been projected for the following scenarios: (i) Sequential method: performing down-sampling at software-level, and detecting and tracking objects in the whole frame; (ii) Software-level feedback method: performing down-sampling and cropping at software-level, and detecting and tracking objects in smaller search regions; (iii) Hardware-level feedback method: performing down-sampling and cropping at hardware-level by exploiting the image sensor capabilities, and detecting and tracking objects in smaller search regions. The battery characteristic curves provided by the manufacturer of the batteries have been used for the estimation. When there is one car in the scene, the average currents drawn are 0.2162 A, 0.1926 A and 0.1345 A for scenarios (i), (ii) and (iii), respectively. The projected battery lifetimes and energy savings are summarized in Table 6.5. As can be seen, when the feedback method is combined with hardware-level operations (scenario (iii)), the battery life increases to 15.5 hours, and it provides 84.52% and 107.2% increase in battery-life compared to scenarios (i) and (ii), respectively. It should be noted that the projected battery lifetimes are based on the scenario, where there will always be an object to track in the scene, i.e. the scene will never be empty.

Method	Battery Lifetime (hours)	gain (%)
Sequential method	7.48	–
Software-level Feedback Method	8.4	12.3%
Hardware-level Feedback Method	15.5	107.2%

Table 6.5: Battery life projection.

6.7 Conclusion

This chapter has presented two methodologies to increase the energy-efficiency and the battery-life of an embedded smart camera by hardware-level operations when performing object detection and tracking. First, instead of performing down-sampling at software-level at the main microprocessor of the camera board, this operation was performed at hardware-level on the micro-controller of the OV9655 image sensor of a CITRIC camera. Moreover, rather than performing object detection and tracking on the whole frame, the location of the target in the next frame was estimated. A search region around it was formed and the next frame cropped by using the HREF and VSYNC signals at the micro-controller of the OV9655. Detection and tracking was performed only in the cropped search region. It was shown that significant savings in energy consumption and battery-life resulted from reducing the amount of data that is moved from the image sensor to the main memory at each frame. Also, better use of the memory resources, and not occupying the main microprocessor for image down-sampling and cropping tasks significantly prolonged the battery life of the camera node. Experimental results show that, compared to software-level cropping, performing hardware-level cropping when tracking one object provides 84.52% increase in battery-life, prolonging the life of the camera up to 15.5 hours. In addition, hardware level down-sampling and cropping, and performing detection and tracking in cropped regions, provides 41.24% decrease in energy consumption and 107.2% increase in battery-life compared to performing software-level down-sampling and processing the whole frame.

Chapter 7

Conclusions

This thesis has focused on the importance and the benefits of designing lightweight computer vision algorithms suitable for embedded smart cameras. Our research has shown that running well-suited algorithms has a significant impact on the battery life of the embedded platforms. We have presented the gains of designing lightweight algorithms that are well integrated with the camera's architecture, as opposed to using algorithms designed for wall-powered platforms. We have shown that it is feasible to design algorithms that can prolong the battery life time of the embedded smart cameras, without affecting the reliability of the system during surveillance tasks.

Our work spans the whole development process, starting with the design and implementation followed by the simulation and optimization, ending with the testing and performance analysis on actual embedded cameras.

In Chapter 3 , a lightweight salient foreground detection algorithm, which is highly robust against challenging non-static backgrounds has been presented. The memory requirement for the data saved per pixel is very small, which is very important for portability to an embedded smart camera. The number of memory accesses and instructions are adaptive, and are decreased even more depending on the amount of activity in the scene and on a pixel's history. Each pixel is treated differently based on its history, and instead of requiring the same number of memory accesses, and thus, instructions for every pixel, we require less instructions for stable background pixels.

In Chapter 4, we have presented a lightweight feedback-based detection and tracking algo-

rithm to increase the energy efficiency and battery life of an embedded smart camera node. The algorithm provides significant savings in processing time. Experimental results showed the gains in processing time as well as the savings in energy consumption and the gain in battery life. In summary, the proposed algorithm in Chapter 4 provides 48.7% decrease in the processing time of a frame, and 10.44% savings in energy consumption, compared to traditional sequential ways of tracking objects.

In Chapter 5, self-adapting methodologies to increase the energy efficiency and battery life of an embedded smart camera node have been presented. The proposed methodologies allow us to send the microprocessor of the camera node to idle state even when there are tracked objects in the scene. The adaptive methodology significantly decreased the energy consumption of the embedded smart camera used in the experiments. The camera can be sent to idle state not only when the scene is empty but also when there are tracked objects in the FOV of the camera. Additionally, an algorithm called combined method was introduced which provides further savings in energy consumption. Experimental results have been presented showing the gains in processing time as well as the savings in energy consumption and the gain in battery life. Up to 131% gain in battery life has been obtained compared to traditional ways of doing tracking.

In Chapter 6, We have presented two hardware-level methodologies that aim to increase the energy-efficiency and the battery-life of an embedded smart camera. The energy saving are obtained by hardware-level operations when performing object detection and tracking. Instead of performing down-sampling at software-level at the main microprocessor of the camera board, this operation is performed at hardware-level on the micro-controller of the OV9655 image sensor of a CITRIC camera. Moreover, rather than performing object detection and tracking on the whole frame, the location of the target in the next frame is estimated and the object detection and tracking are performed only in the estimated areas. Employing hardware-level operations resulted in an increase in battery life of 107.2% compared to performing software-level down-sampling and processing whole frame.

Bibliography

[1] R.O. Duda and P.E. Hart, “Pattern Classification and Scene Analysis”, New York, Wiley, 1973.

[2] C. H. Anderson, P. J. Burt, and G. S. V. D. Wal, “Change detection and tracking using pyramid transform techniques,” *Proceedings of SPIE Intelligent Robots and Computer Vision*, vol. 579, pp. 72-78, Cambridge, MA, Sept. 16–20, 1985.

[3] P. L. Rosin and T. Ellis, “Image difference threshold strategies and shadow detection,” *Proceedings of British Machine Vision Conference*, pp. 347–356, 1995.

[4] M. Casares and S. Velipasalar, “Light-weight salient foreground detection with adaptive memory requirement,” Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2009.

[5] N. Friedman and S. Russell, “Image segmentation in video sequences: A probabilistic approach,” *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pp. 175–181, 1997.

[6] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, “Pfinder: Real-time tracking of the human body,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780-785, July 1997.

[7] T. Kanade, R. T. Collins, A. J. Lipton, P. Burt and L. Wixson, "Advances in cooperative multi-sensor video surveillance," *Proceedings of DARPA Image Understanding Workshop*, pp. 3–24, Monterey, CA, November 1998.

[8] T. Horprasert, D. Harwood and L. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," *Proceedings of IEEE ICCV Frame-Rate Workshop*, pp. 1–19, 1999.

[9] K. Toyama, J. Krumm, B. Brumitt and B. Meyers, "Wallflower: Principle and practice of background maintenance", *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 1, pp. 255-261, 1999.

[10] F. Oberti, A. Teschioni , C.S. Regazzoni, "ROC curves for performance evaluation of video sequences processing systems for surveillance applications," *Proceedings of the IEEE International Conference on Image Processing*, vol. 2, pp. 949–953, 1999.

[11] N. Oliver, B. Rosario, and A. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 831-834, 2000.

[12] A. Elgammal, D. Harwood and L. Davis, "Non-parametric model for background subtraction," *Proceedings of 6th European Conference on Computer Vision*, pp. 751–767, June/July 2000.

[13] X. Gao, T. E. Boult, F. Coetzee, V. Ramesh, "Error analysis of background adaption," *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pp. 503–510, 2000.

[14] I. Haritaoglu, D. Harwood and L. S. Davis, "W^4: Real-time surveillance of people and their activities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 809-830, August 2000.

[15] J. Rittscher, J. Kato, S. Joga and A. Blake, "A probabilistic background model for tracking," *Proceedings of the European Conference on Computer Vision*, vol. 2, pp. 336350, 2000.

[16] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747–757, August 2000.

[17] W.-C. Feng, W.-C. Feng, and M. L. Baillif, "Panoptes: Scalable low-power video sensor networking technologies," *Proceedings of the eleventh ACM international conference on Multimedia*, pp. 562–571, 2003.

[18] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 142–149, vol. 2. Jun. 2000.

[19] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer, "Multi-camera multi-person tracking for EasyLiving," *Proceedings. Third IEEE International Workshop on Visual Surveillance* , pp. 3–10, Jul. 2000.

[20] L. Lee, R. Romano, and G. Stein, "Monitoring activities from multiple video streams: Establishing a common coordinate frame," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 758–768, Aug. 2000.

[21] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian computer vision system for modeling human interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–834, Aug. 2000.

[22] B. Stenger, V. Ramesh, N. Paragios, F. Coetzee and J. Bouhman, "Topology free hidden markov models: Application to background modeling," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 294-301, 2001.

[23] I. Pavlidis, V. Morellas, P. Tsiamyrtzis and S. Harp, "Urban surveillance systems: from the laboratory to the commercial world," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1478–1497, October 2001.

[24] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for real-time tracking with shadow detection," *Proceedings of the Workshop on Advances in Vision-based Surveillance Systems*, September 2001.

[25] M. Harville, "A framework for high-level feedback to adaptive, per-pixel, mixture-of-gaussian background models," Proceedings of European Conference on Computer Vision, vol. 3, pp. 543-560, 2002.

[26] W. Wolf, B. Ozer, T. Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, pp. 48–53, September 2002.

[27] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "CMUcam: a lowoverhead vision system," *Proceedings of the International Conference on Intelligent Robots and Systems, IROS 2002*, 2002.

[28] A. Elgammal, R. Duraiswami, D. Harwood and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance", *Proceedings of the IEEE*, 90(7), pp.1151–1163, July 2002.

[29] Nam T. Nguyen, S. Venkatesh, G. West and Hung H. Bui, "Multiple camera coordination in a surveillance system," *ACTA Automatica Sinica*, vol. 29 (3), pp. 408-422, 2003.

[30] D. Gay, L. Philip, R. Behren, M. Welsh, E. Brewer, D. Culler, "The nesC language: A holistic approach to networked embedded systems," *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003, pp. 1–11.

[31] T. H. Chalidabhongse, K. Kim, D. Harwood and L. Davis, "A perturbation method for evaluating background subtraction algorithms", *Proceedings of the Joint IEEE International Work-*

shop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, in conjunction with ICCV, Oct. 2003.

[32] K. Kim, T. H. Chalidabhongse, D. Harwood and L. Davis, "Real-time foreground-background segmentation using codebook model," *Real-time Imaging*, vol. 11, no. 3, pp. 172–185, June 2005.

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

[34] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A line in the sand: A wireless sensor network for target detection, classification, and tracking," *Computer Networks (Elsevier)*, vol. 46, no. 5, pp. 605–634, Dec. 2004.

[35] S. Bhandarkar and X. Luo, "Fast and robust background updating for real-time traffic surveillance and monitoring", *Proceedings of the IEEE Workshop on Machine Vision for Intelligent Vehicles*, pp. 55, June 2005.

[36] D.-S. Lee, "Effective Gaussian mixture learning for video background subtraction", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5), pp. 827832, 2005.

[37] X. Luo and S. Bhandarkar, "Real-time and robust background updating for video surveillance and monitoring", *Springer Lecture Notes in Computer Science*, 3656, pp. 1226–1233, 2005.

[38] Y. Sheikh and M. Shah, "Bayesian object detection in dynamic scenes," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 74–79, June 2005.

[39] W.-C. Feng, W.-C. Feng, and M. L. Baillif, "Panoptes: Scalable lowpower video sensor networking technologies," *Proceedings of the eleventh ACM international conference on Multimedia*, 2003, pp. 562–571.

[40] P. Kulkarni, D. Ganesan, and P. Shenoy, "The case for multi-tier camera sensor networks," *Proceeding NOSSDAV'05 Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pp. 141–146, 2005.

[41] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, "Real-time foreground-background segmentation using codebook model," *J. Real-time Imaging* , vol. 11, no. 3, pp. 172–185, Jun. 2005.

[42] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a wireless sensor network platform for detecting rare, random, and ephemeral events," *Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005.* , pp. 497–502, 2005.

[43] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: In situ image sensing and interpretation in wireless sensor networks," *Proceedings of the International Conference on Embedded Networked Sensor Systems*, pp. 192–204, 2005.

[44] D. Lymberopoulos, A. Savvides, "XYZ: a motion-enabled, power aware sensor node platform for distributed sensor network applications," *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, 2005.

[45] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay,J. Hill,M. Welsh,E. Brewer, D. Culler, W. Weber, J. Rabaey, E. Aarts, "Tinyos: An operating system for sensor networks," *Ambient Intelligence*, Springer-Verlag, 2004.

[46] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A second generation low cost embedded color vision system," *Proceedings IEEE Embedded Computer Vision Workshop Conjunction with IEEE Conference Computer Vision and Pattern Recognition*, vol. 3, pp. 136, Jun. 2005.

[47] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay,J. Hill,M. Welsh,E. Brewer, D. Culler, W. Weber, J. Rabaey, E. Aarts, "Tinyos: An operating system for sensor networks," *Ambient Intelligence*, Springer-Verlag, 2004.

[48] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *IEEE Computer*, vol. 39, no. 2, pp. 68–75, Feb. 2006.

[49] A. Shimada, D. Arita, R. Taniguchi, "Dynamic control of adaptive Mixture-of-Gaussians background model", *Proceedings of IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2006.

[50] I. Downes, L. B. Rad, and H. Aghajan, "Development of a mote for wireless image sensor networks," *Proceedings on Cognitive Systems Interactive Sensors*, March. 2006.

[51] S. Fleck, F. Busch, P. Biber, and W. Strasser, "3-D surveillance: A distributed network of smart cameras for real-time tracking and its visualization in 3-D," *Proceedings Conference on Computer Vision and Pattern. Workshop*, Jun. 2006, p. 118.

[52] P. Chalimbaud, "Embedded active vision system based on an FPGA architecture," *EURASIP Journal on Embedded Systems IDOTS*, pp. 26 ,January 2006.

[53] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, "Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance," *International Symposium on Information Processing in Sensor Networks*, 2007, pp. 360–369.

[54] 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. 92827, p. 10, 2007.

[55] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing," *in Proc. ACM/IEEE Int. Conf. Distributed Smart Cameras*, Sep. 2007, pp. 106–116.

[56] A. Rowe, A. Goode, "CMUcam3: an open programmable embedded vision sensor," *Conferences on Intelligent*, 2007.

[57] R. Smith, "SPOTWorld and the Sun SPOT," *Information Processing in Sensor Networks*, 2007.

[58] A. Y. Benbasat and J. A. Paradiso, "A framework for the automated generation of power-efficient classifiers for embedded sensor nodes," *in Proc. Int. Conf. Embedded Networked Sensor Syst.*, 2007, pp. 219–232.

[59] B. Jiang, B. Ravindran, and H. Cho, "Energy efficient sleep scheduling in sensor networks for multiple target tracking," *in Proceedings IEEE International Conference on Distributed Computer Sensor Systems,* Sep. 2008, pp. 498509.

[60] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," *IEEE proceedings*, vol. 96, no. 10, pp. 15651575, Oct. 2008.

[61] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf, "The evolution from single to pervasive smart cameras," *in Proceddings ACM/IEEE International Conference on Distributed Smart Cameras, Sep. 2008,* pp. 1–10.

[62] L. Nachman, J. Huang, "Imote2: Serious computation at the edge," *2008. IWCMC'08*, 2008.

[63] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Leon, E. Lobatonm, M. Meingast, S. Oh, S. Wang, P. Yan, A. Y. Yang, C. Yeo, L.-C. Chang, J. D. Tygar, and S. S. Sastry, "CITRIC: A low-bandwidth wireless camera network platform," *Proceddings ACM/IEEE International Conference on Distributed Smart Cameras*, Sep. 2008, pp. 1–10.

[64] M. Casares, M. C. Vuran and S. Velipasalar, "Design of a Wireless Vision Sensor for Object Tracking in Wireless Vision Sensor Networks," *Proceddings ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), Workshop on Embedded Middleware for Smart Camera and Visual Sensor Networks (eMCAM)*, pp. 1–9, Sept. 2008.

[65] M. Casares and S. Velipasalar,"Light-weight salient foreground detection for embedded smart cameras", *Proceddings ACM/IEEE International Conference on Distributed Smart Cameras*, 2008, pp. 1–7.

[66] Y. Wang, M. Casares, and S. Velipasalar, "Detection of composite events spanning multiple camera views with wireless embedded smart cameras," *Proceddings ACM/IEEE International Conference on Distributed Smart Cameras*, Aug. Sep. 2009, pp. 1–8.

[67] Z.-J. Yu, J.-M. Wei, and H.-T. Liu, "Energy-efficient collaborative target tracking algorithm using cost-reference particle filtering in wireless acoustic sensor networks," *J. China Univ. Posts Telecommun.*, vol. 16, no. 1, pp. 9–15, Feb. 2009.

[68] M. Casares, V. Senem, P. Alvaro, "Light-weight salient foreground detection for embedded smart cameras," *Computer Vision and Image Understanding*, 2010, pp. 1223–1237.

[69] M. Casares and S. Velipasalar, "Light-weight salient foreground detection for embedded smart cameras," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1223–1237, 2010.

[70] M. Casares and S. Velipasalar, "Resource-efficient salient foreground detection for embedded smart cameras," *in Proc. IEEE Int. Conf. AVSS*, Aug. 2010, pp. 369–375.

[71] M. Casares and S. Velipasalar, "An adaptive method for energy efficiency in battery-powered embedded smart cameras," *Proceddings ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, 2010.

[72] Y. Wang, M. Casares, and S. Velipasalar, "Cooperative object tracking and composite event detection with wireless embedded smart cameras," *IEEE Trans. Image Process.*, vol. 19, no. 10, pp. 26142633, Oct. 2010.

[73] M. Casares and S. Velipasalar, “Adaptive Methodologies for Energy-Efficient Object Detection and Tracking With Battery-Powered Embedded Smart Cameras,” *IEEE Transactions on Circuits and Systems for Video Technology*, 2011, pp. 1430–1452.

[74] K. Quast and A. Kaup, “AUTO GMM-SAMT: An automatic object tracking system for video surveillance in traffic scenarios,” *EURASIP Journal on Image Video Processing*, vol. 2011, no. 814285, p. 14, 2011.

[75] Intel PXA27x Processor Familiy Developers Manual, “http://www.balloonboard.org/hardware/300/ds/PXA270- dev-manual.pdf”.

[76] Omnivision Technologies Inc. OV9655 Color CMOS SXGA (1.3 MegaPixel) CAMERACHIP with OmniPixel Technology Datasheet, 2006.

[77] J. Polastre, R. Szewczyk, and D. Culler, “Telos: Enabling ultra- low power wireless research”. *In IPSN/SPOTS, 2005.*

[78] Mats Skogholt Hansen, “Practical Evaluation of IEEE 802.15.4/ ZigBee Medical Sensor Networks”. *NTNU Innovation and Creativity* Norwegian University of Science and Technology. June. 2006.

Vita

Mauricio Casares (M’08) holds a Doctorate degree in Electrical and Computer Engineering from Syracuse University, Syracuse, NY. He received a B.Sc. degree in Electronics and Control Engineering from National Polytechnic University, Quito, Ecuador, in 2005. In 2010, He graduated from the University of Nebraska-Lincoln holding a master degree in Electrical Engineering with a minor in Computer Science. Since 2011, He has been working for Schneider Electric USA, Inc focused primarily on Machine vision, RFID technology, and Control systems. His research has been on wireless embedded smart cameras and lightweight algorithm design for embedded platforms. His current research interests include computer vision, multi-camera systems, control theory, and signal processing.