

DESIGN OF A VISION ENABLED WIRELESS SENSOR NODE

Wijesinghege Yasitha Mahesh Wijesinghe

(188025T)

Thesis submitted in partial fulfillment of the requirements for the degree
Master of Science (Major Component of Research)

Department of Electronic and Telecommunication Engineering

University of Moratuwa
Sri Lanka

March 2020

Declaration

I declare that this is my own work, and this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or institute of higher learning, and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part, in print, electronic, or any other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature:

Date:

The candidate, whose signature appears above, carried out research for the Msc dissertation under my supervision.

Signature:

Date:

Abstract

This thesis describes a novel system architecture and implementation of a wireless visual sensor node. The proposed design of the node can be used to extract traffic information based on the visual description of road. In this research, the real-time performances and the capability to perform at low power consumption meanwhile obtaining accurate results were considered as the essential factors since a large amount of data need to be processed in an embedded level vision system.

At first, a suitable vision algorithm is proposed to harvest the traffic condition on road. The functionality for each section of the algorithm has been performed by using carefully selected available vision techniques and image processing algorithms. The vehicle extraction from the current frame of view and the tracking process of the vehicle are identified as the most important functions in the algorithm. The vehicle extraction from the current frame is carried out by the ViBe algorithm with some modifications in order to acquire promising real time performances and the tracking process is carried out by a light weight but an accurate enough particle filtering technique.

Moreover, the complete system is implemented in the FPSoC hardware system as a hardware and software co-design by considering advantages that can be obtained from different aspects. The performances of the system have been evaluated from many aspects for different standard data available from other research works. The conclusions and suggestions for further development have been presented at the end of this thesis.

Index terms— FPSoC, HW/SW Co-Design, ITS, ViBe, FPGA, WSN, VSN

Acknowledgements

I extend my gratitude to my supervisors, Dr. Jayathu Samarawickrama and Prof. Dileeka Dias of Department of Electronic and Telecommunication Engineering, University of Moratuwa, for their invaluable guidance and the support throughout my research study. I would also like to thank Dr. Ranga Rodrigo, the chairperson of the review panel who shared his knowledge and expertise in many ways.

I appreciate the continuous encouragement and support provided by my workplace, Dialog Research Laboratory, University of Moratuwa.

I would also like to pay my utmost gratitude towards LK domain for providing with funding to present this research work in the ICIIS conference hosted by University of Peradeniya. I believe that their service to the research community in the country is extremely praiseworthy.

I also would extend my appreciation for all of my colleagues in the Department of Electronic and Telecommunication for all the fruitful discussions and moral support given during this study. Moreover, I would like to thank the academic and non-academic staff of Department of Electronic and Telecommunication Engineering, University of Moratuwa for providing their support in numerous ways.

Finally, I thank my university for the education it has provided me throughout these years which made me even stronger to conquer the goals of my life.

W.Y.M Wijesinghe

Contents

Declaration	i
Abstract	ii
Acknowledgements	iii
List of Figures	vi
List of Tables	viii
1 INTRODUCTION	1
1.1 Background to the Research Topic	1
1.2 Aim, Objectives and Scope of the Thesis	6
1.3 Structure of the Thesis	7
2 LITERATURE REVIEW	9
2.1 Summary	22
3 SYSTEM ARCHITECTURE	23
3.1 Selection of Traffic Parameters	23
3.2 Proposed Vision Model for Traffic Estimation	26
3.3 Vehicle Extraction	28
3.4 The ViBe Algorithm	31
3.4.1 Description of the Model and Pixel Classification Method .	31
3.4.2 Initialization of the ViBe Model	32
3.4.3 The Updating Process of the Background Model	33
3.4.4 Identified Drawbacks in the ViBe Algorithm and Modifica- tion Proposed to the Original	34
3.5 Vehicle Tracking	39

4	IMPLEMENTATION	49
4.1	Hardware-Software Partitioning and Optimization of the Algorithm	54
5	RESULTS AND DISCUSSION	62
5.1	Accuracy of the Vision Algorithm	62
5.2	Power Consumption of the Hardware Platform	68
6	CONCLUSION AND FUTURE WORK	70
6.1	Major Contributions	70
6.2	Future Works	71
6.2.1	Increase the Frame Rate of the System	71
6.2.2	Improve the accuracy of Vehicle Tracking	71
6.2.3	Improve the Accuracy of Detecting the Stopped Vehicles in the Scene	72
6.2.4	Enhance the Capability to Count the Number of Vehicles in the Scene	72
6.2.5	Evaluate the Usability of this VSN in the Real world	73
	References	74
	Appendices	81
A	The ViBe Algorithm	82
A.0.1	The Update and Classification of Modified ViBe	82

List of Figures

1.1	The concept of using WWSN in ITS applications.	2
1.2	The basic hardware architecture of a VSN in ITS application. . .	5
2.1	The hardware architecture of VSN in [1]	11
2.2	Example TSI image in left side and block diagram of the algorithm in right side.	13
2.3	Hardware architecture for proposed vision algorithm in [11]	14
2.4	The Proposed algorithm to measure traffic condition.	15
2.5	The steps of the vehicle classification.	16
2.6	Block diagram of the night vision algorithm.	17
2.7	Comparison of performance of the night vision algorithm for dif- ferent SBCs.	18
2.8	Implementation of the parallel functions presented in [19].	20
2.9	HW/SW data flow and interrupt route.	21
3.1	Flow Diagram of the Vision Algorithm.	27
3.2	Illustration of the foreground extraction process.	28
3.3	Gaussian distribution.	30
3.4	Distribution of Mixture of Gaussian.	30
3.5	Representation of the sphere.	32
3.6	The neighborhood pixels at location, “x”.	33
3.7	The updating location of the model.	34
3.8	The cuboid concept.	36
3.9	PCC values for different tolerance values.	37
3.10	Resultant frame for each tolerance value.	37
3.11	The cuboid in actual model and the update process.	38
3.12	Comparison of modified ViBe with MoG and original ViBe.	39
3.13	Accuracy of the MoG, Modified ViBe and the original ViBe in term of PCC.	39
3.14	Basic steps in a tracking algorithm.	40

3.15	Meanshift vector	40
3.16	Illustration of the displacement.	42
3.17	The process of LBP	43
4.1	Overview of a vision system	49
4.2	The architecture of a modern FPSoC.	52
4.3	Ultra 96 Development board	54
4.4	Resources of the Ultra 96 board	54
4.5	Time consumption profile for the execution of one frame.	55
4.6	Types of AXI4 interfaces.	56
4.7	Block diagram of ViBe.	57
4.8	Block diagram of the weight calculation using Bhattacharya coefficient	57
4.9	Integration custom IP with Zynq	58
4.10	Complete flow chart of the algorithm with balanced processing load between PL and PS side.	59
5.1	(a) Early tracking initialization (b) Correct tracking initialization.	63
5.2	Background noise of the patch	63
5.3	Location deviation due to background pixels in the tracking template.	64
5.4	Deviation of the tracking box even though the vehicle is stopped.	65
5.5	Partially occluded situation.	65
5.6	Power consumption of the hardware platform.	66
5.7	The effect of merged blobs.	67
5.8	Static power consumption.	68
5.9	Dynamic power consumption.	69

List of Tables

2.1	Specifications of the SBCs that have been used to compare the performance of the algorithm.	18
3.1	PCC and tolerance values	36
4.1	Comparison of hardware platforms	50
4.2	Specifications of the SBCs that have been used to compare the performance of the algorithm.	60
4.3	Performances and resource utilization for different frame sizes . .	60
4.4	Comparison with other implementation	61
5.1	Accuracy of the average velocity.	63
5.2	The accuracy of the waiting time.	64
5.3	Accuracy of the LO	66
5.4	Accuracy of Traffic Density	67

List of Abbreviations

Abbreviation	Description
WSN	Wireless Sensor Node
RoI	Region of Interest
WVSN	Wireless Visual Sensor Node
ITS	Intelligent Transportation Systems
VSN	Visual Sensor Node
FPU	Floating Point Unit
SIMD	Single Instructions Multiple Data
FPSoC	Field Programmable System on Chip
FoV	Field of View
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
SoC	System on Chip
TSI	Time-Spatial Images
VDL	Virtual Detection Line
LBP	Linear Binary Pattern
APSoC	All Programmable System on Chip
PS	Processing System
HW	Hardware
SW	Software
CPU	Central Processing Unit
SBC	Single Board Computer
MoG	Mixture of Gaussian
SRP	Sparse Random Projection
NCC	Normalized Cross Correlation
GIC	Global Interrupt Controller
PL	Programmable Logic
CNN	Convolutional Neural Network

LO	Lane Occupancy
ViBe	Visual Background extractor
FN	False Negative
FP	False Positive
PCC	Percentage of Correct Classifications
TP	True Positive
TN	True Negative
SAD	Sum of the Average Difference
GPU	Graphic Processing Unit
WT	Waiting Time
AV	Average Velocity
CLB	Combinational Logic Block

Chapter 1

INTRODUCTION

The increase of traffic jam is a burning issue in the modern urban areas and day by day, the problem keeps increasing. This is a demanding problem in terms of health, productivity and pollution for both human society and the environment. This can be manipulated using the concepts in intelligent transportation systems. In order to apply those concepts and implement practical solutions, a measurement method of real time traffic condition on road is highly important. There are many traffic conditions measuring methods, but visual data-based traffic information has higher demand than any other sensing method [2]. Therefore, this thesis proposes an algorithm and hardware implementation for a visual sensor node which can be used in a wireless sensor network implemented in urban areas for real time traffic condition harvesting.

1.1 Background to the Research Topic

Wireless sensor nodes (WSN) act as a layer on the physical world to gather important information. The most popular physical information are usually the scalar parameters such as temperature, pressure, energy consumption in factories/houses and meter readings. Due to the vast range of improvements in embedded level processors, operating systems, communication and cheap visual sensors in previous decades, the feasibility of gathering and integrating visual features to the wireless sensor node which is more information rich, sensible and perspective than other forms of scalar information has been rapidly increased. This also leads to the increase of intelligence of sensor nodes because of the capabilities of detecting the events at a certain level which are created by objects within the region of interest (RoI) of them. Nowadays, most of the researchers all over the world are in search of applicability of wireless visual sensor networks (WVSNs) for various applications which can improve the living standards of the society. The main interested areas are, public, private and commercial security purposes, surveillance systems in military applications, environment monitoring

and incident identifications, quality and automation purposes of manufacturing companies and intelligent transportation systems [2].

Among those application of WVSNs, the intelligent transportation system (ITS) has the potential of making a huge impact on the society and environment in numerous ways. The increase of population in cities, metropolitan areas and requirement for smart cities increase the demand for ITS based solutions. ITS can help for the economy of a country by manipulating the vehicle flow breakdown and the amount of fuel that is burnt out by the automobiles. Besides that, it is also helpful for the reduction of drivers' stress level and the emission of greenhouse gases which can reduce the global warming in return and the reduction of carbon footprint made by humans. The WVSN can be designed to provide reliable and real-time spatio-temporal information about the mobility in densely populated areas as illustrated in Figure 1.1. This knowledge can be used in ITS applications by providing useful information such as traffic information, monitoring of the drivers' misbehavior regarding traffic laws, incident detection, traffic pattern identification in RoI and effective use of space, energy and mobility in urban environment. This description of urban mobility provided by WVSN and strategies and planning techniques in ITS altogether can be used towards the concept of smart cities where adaptive and innovative traffic manipulating mechanisms exist. All of this information can be utilized for traffic light controlling, smart route suggestions for the drivers and pedestrians, effective utilization of all the resources available for better controlling of vehicles in urban area and driving the society towards the concept "smart cities" [2] [3]

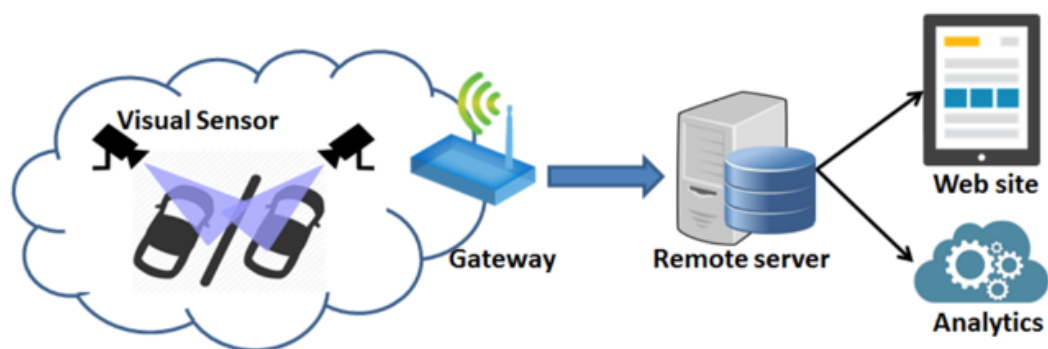


Figure 1.1: The concept of using WVSN in ITS applications.

However, the integration of a sensor node which is having a visual analyzing capability is not an easy task due to many reasons. Transmitting a complete

frame from a sensor node to the central server through wireless communication is an extremely power consuming and cause the loss of data. Besides that, the delay of the transmission and security risk of this kind of approach can lead to the reduction of usefulness of visual sensor node (VSN) in WWSN [2].

On the other hand, implementing the image processing and machine vision model in a sensor node and performing in real time is a challenging task due to the limited resources in VSNs. However, recent improvements in embedded processors, middleware systems and low-cost developments in hardware such as memory chips and CMOS visual sensors motivate the researches to investigate the feasibility of this kind of approach where implementing machine vision algorithm in local sensor nodes and transmitting only the extracted useful top-level information [4]. There are many advantages in implementing the image processing and machine vision algorithms in sensor nodes and transmitting only the useful information [4].

- Usually sensor nodes are having limited power and resources and they are connected via lossy channels. It has turned out that the power consumption for transmitting a single meaningful bit requires more than two thousand times the power consumption for the execution of a single instruction [5]. Therefore, this concept can preserve the power utilization in these devices in a huge fraction.
- The extraction of useful real time information from the local sensor node is more accurate and reliable than collecting all the video frames from many sensor nodes and processing those image frames at a central hub of the network [6].
- It is also beneficial for integrating a VSN with any other heterogeneous network architecture where different sensing and communication capabilities/protocols exist [2].
- This approach influences the intelligence level of the sensor node because the sensor node can be designed so that only the events at a certain threshold need to be informed to the relevant authorities without human intervention. Otherwise, we would have to employ humans for the supervision throughout the day which causes the wastage of money as well as human resources [4].
- Even though, some of the object recognition and event identification can be done accurately by humans than currently available image processing and

computer vision algorithms, some of the calculations can be done accurately and quickly in computer systems. For example, measuring the velocity of an object, calculating the number of objects and identifying their precise locations meanwhile performing the same operation 24/7.

- We cannot expect that the urban areas are having fixed structures in terms of roads, transportation, infrastructure and building facilities. Therefore, sustainable flexibilities such as modification of location can be obtained from the WWSN with respect to the other sensor nodes such as inductive loops [7].
- The WWSN is aware about the location of the occurrence of any event. This is the key point in fog computing with respect to the other cloud-based approaches. This is very useful for quick decision making with the help of efficient collection of data and low latency in data processing [7].

Basically, VSNs consist of three basic components as shown in Figure.1.2. There will be a camera sensor, an embedded level processing system where the image processing and machine vision algorithms are performed and a wireless transceiver for the communication between other nodes and the main gateway. However, due to the vast range of low cost and low power developments in Silicon and CMOS sensors and communication technologies, there are variety of options that are available in selecting a camera sensor and a radio module for communication purposes. The most prominent component in a VSN is the unit where the image processing and machine vision algorithms take place. Most of the times, this multimedia processing unit will be the most power consuming and the costly component existing in the sensor node. It is important to select the most suitable platform for this processing unit in order to perform the required vision algorithm at a required power budget while maintaining the expected performance. However, the performance of a vision algorithm in a hardware platform is not only depending on its clock frequency, the number of cores it contains, the memory speed, the features like direct memory access or parallel processing capability and multiple processing unit availability such as floating point unit (FPU) but also it is important to pay attention to the way the algorithm performs and the selection of most appropriate algorithm to be implemented. The modifications can be done in order to reach the best level of performance and investigation of the limitations can be done with the help of existing technologies.

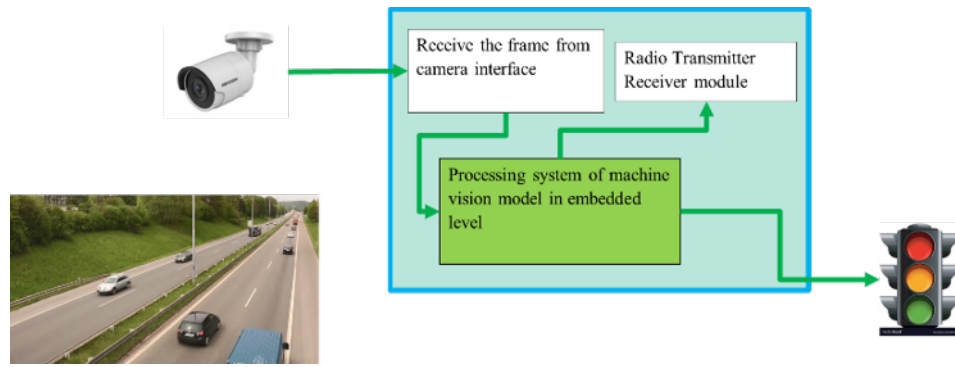


Figure 1.2: The basic hardware architecture of a VSN in ITS application.

Naturally, most of the image processing tasks require higher memory and processing requirements than the scalar signal processing tasks which are typically used in any wireless sensor network. Besides that, most of the operations performed on the images can be categorized as single instruction multiple data (SIMD) type operations. In other words, same instruction needs to be performed to a large amount of data. This means, there is an opportunity to grab the advantages in parallel computing capabilities in order to achieve the expecting performance from a VSN. Therefore, when it comes to hardware design stage of the VSN, it is equally important to consider the effective power conservation capability of the device as well as the parallel performances of them.

1.2 Aim, Objectives and Scope of the Thesis

The aim of the thesis is to design a VSN node to measure real-time but reliable traffic parameters which can describe the traffic condition on road. In order to achieve this aim, following objectives have been set up.

- Development of an appropriate image processing algorithm for resource constrained hardware specially focusing on road traffic estimations.
- Development of vision-enabled, resource-constrained WSN node.
- Optimize the algorithm and hardware in terms of cost, resources, power consumption and functional reliability mainly for the traffic analysis.

These objectives have to be analysed in the current state-of-the-art embedded level image processing, computer vision algorithm and hardware platform where power consumption has been optimized for this kind of machine vision application.

1.3 Structure of the Thesis

The organization of the thesis is described as follows.

- Chapter 1: Introduction

This chapter describes the background of this work and explains what encourage us to carry out this work. The concept of visual sensor node has been presented in this section and narrowed it down to the challenges exist in the research area.

- Chapter 2: Literature Review

A broad literature survey has been carried out about the available visual sensor nodes that have been implemented to measure the traffic conditions on road and the mobility of the urban areas. An informative attention has been paid for the algorithms, the vision techniques and novel available hardware technologies that have been used.

- Chapter 3: Vision Algorithm for Traffic Estimation

A vision model has been proposed and the implemented image processing algorithm has been discussed in this section. The selection of the image processing algorithm has been done with a comparison and modifications that have been made to the existing algorithm and explanations for such modifications have been discussed extensively.

- Chapter 4: Hardware Implementation

This section first describes the selection of field programmable system on chip (FPSoC) by reasoning for implementation of the proposed vision algorithm in the hardware system. Then, based on the time consumption profile, the implementation and optimization of the algorithm has been discussed.

- Chapter 5: Results and Discussion

This section discusses about the performance of the implemented hardware system for the VSN and the accuracy of traffic measurements that can be taken out from the system for several standard data available from other traffic analysis systems that have been used to evaluate the complete system.

- Chapter 6: Conclusion and Future Work

Conclusion of the research work has been discussed and based on the results

obtained from the previous section, few directions for the future improvements have been discussed.

Chapter 2

LITERATURE REVIEW

In recent years, many designs in terms of hardware architectures and image processing algorithms have been proposed to harvest information about the mobility on road using embedded level vision systems. Among those approaches of hardware architectures and computer vision algorithms, only the implementations that can be powered by battery, less costly and can be implemented with less hardware resources utilization are considered.

In the context of embedded level vision systems to extract traffic parameters using image processing and computer vision techniques, there are many considerations that are needed to be taken into account. Since this is an outdoor application of image processing, the effect of weather conditions, static objects such as trees and stalls exist in the field of view (FoV) and the shadows created by those static objects as well as moving objects such as pedestrians and vehicles can be a tremendous challenge. Even in a better weather condition, it cannot be expected to have a constant illumination throughout the day. Apart from that, practical problems like camera jitter has to be taken into account. Besides the algorithm-wise challenges, there are many obstacles that have to be considered when those image processing algorithms are going to be performed in hardware platforms where very limited resources as well as tight power budgets exist. Usually, image processing algorithms deal with large amounts of data and it is required to process these data at a higher speed in a way that, there is no any delay between the result of the system and the real world scenario. The requirement of image data capturing, storing and processing can be a dramatic challenge to the processing units which are running at low frequencies as well as small capacity and low performance of off-chip memory. Yet, this can be somewhat manipulated by using a video stream with a low resolution. However, the frame size of the video stream should be large enough to perform an image processing algorithm to extract useful information about the mobility of vehicles [8].

A broad literature review has been carried out thoroughly in order to identify further challenges in the designing of this sort of an embedded level vision system and how those challenges have been addressed with the help of currently available improved hardware platforms and innovative solutions that have been presented in the computer vision and image processing field while addressing the issues mentioned in above paragraph.

Considering the drawbacks such as inflexibility in installation and less informativeness of the typical moving vehicle detection methods such as passive infrared sensors, magnetometer and microphones, authors in [1] have proposed an embedded level velocity estimation method using image processing techniques. It has been declared that, the use of high-resolution cameras to inspect large area and transmit all the frames to the monitoring center using high bandwidth cable, fiber optics and human resources based all day supervision is very expensive. Besides that, due to the interactive parallel behavior in vision algorithms which can be effectively executed in reconfigurable hardware systems, a low power consuming and cost-effective solution for the small-scale production requirement, field programmable gate array (FPGA) based pure hardware implementation of VSN for traffic surveillance has been proposed. Basically, it examines about the presence and segmentation of vehicles and the spot speed of them. The basic block diagram of the vehicle segmentation algorithm has been illustrated in Figure.2.1.

The vehicle segmentation has been done by using sigma-delta method with some modifications to the algorithm. It has been shown that, having two separate sigma-delta background subtractions can be used to overcome the ghost effect. The main difference between the two background subtraction algorithms is that, the background model of the RoI will be updated based on the selective method where three conditions have been used to classify the pixels that can be used for updating the model and the other one will accept all the incoming pixels to the algorithm as in (2.1). These three conditions in selective method have been defined in the expansion of the $m_{VTS,t}(x, y)$. Since the estimation of spot velocity of the vehicles has been done by performing the geometric transformation on the extracted blobs of the vehicles, this strategy is implemented to extract the blobs of vehicles with low velocity. Besides that, they also have included two other functions to manipulate the highlighted pixels and shadow detection, in order to overcome the issues in sigma-delta method for outdoor applications [9].

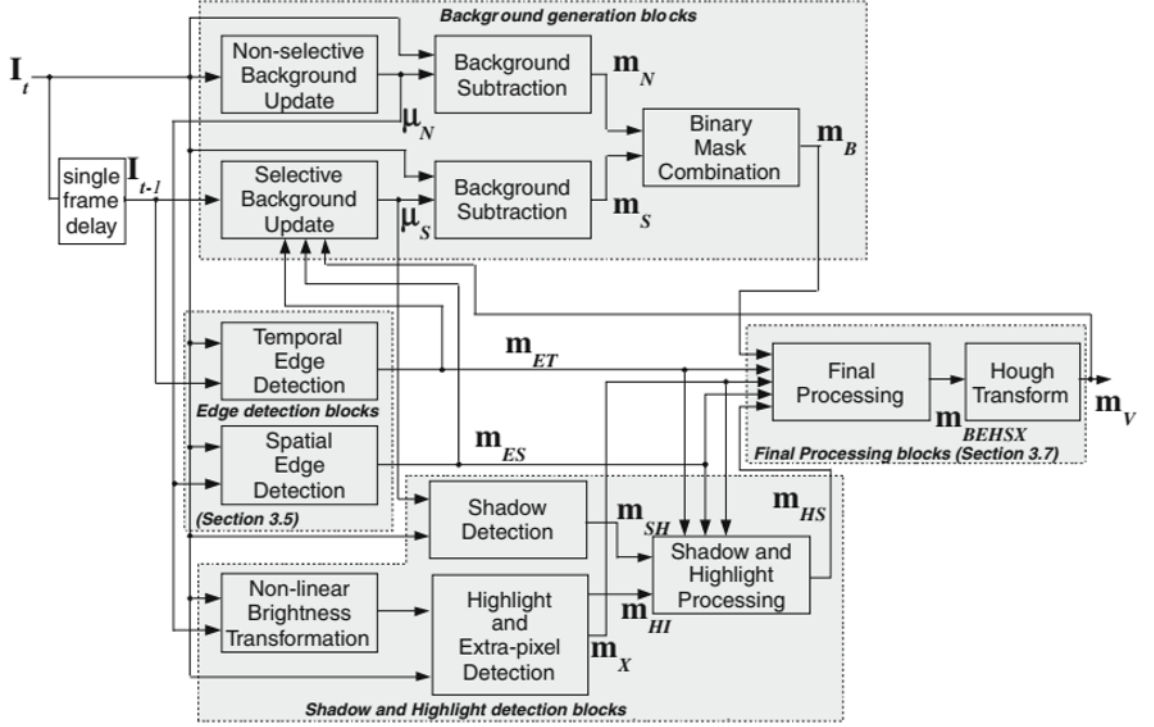


Figure 2.1: The hardware architecture of VSN in [1]

$$\mu_{N,t}(x, y) = \begin{cases} \mu_{N,t-1}(x, y) + \delta_{NI}, & \text{if } I_t(x, y) > \mu_{N,t-1}(x, y) \\ \mu_{N,t-1}(x, y) - \delta_{NI}, & \text{if } I_t(x, y) < \mu_{N,t-1}(x, y) \\ \mu_{N,t-1}(x, y), & \text{otherwise} \end{cases} \quad (2.1)$$

$$\mu_{S,t}(x, y) = \begin{cases} \mu_{S,t-1}(x, y) + \delta_{SI}, & \text{if } I_{t-1}(x, y) > \mu_{S,t-1}(x, y) \text{ and } m_{VTS,t-1}(x, y) = 0 \\ \mu_{S,t-1}(x, y) - \delta_{SI}, & \text{if } I_{t-1}(x, y) < \mu_{S,t-1}(x, y) \text{ and } m_{VTS,t-1}(x, y) = 0 \\ \mu_{S,t-1}(x, y), & \text{otherwise} \end{cases} \quad (2.2)$$

$m_{VTS,t}(x, y)$ can be defined as,

$$m_{VTS,t}(x, y) = m_{V,t}(x, y) \vee m_{ET,t}(x, y) \vee m_{ES,t}(x, y)$$

where, $I_t(x, y)$ is the brightness of a pixel at coordinates (x, y) of the input image at time t , $\mu_{N,t}(x, y)$ is the brightness of a pixel at coordinates (x, y) of the background image which is updated for every pixel at time t , $\mu_{S,t}(x, y)$ is the brightness of a pixel at coordinates (x, y) of the background image which is updated for each selective pixel at time t , $m_{V,t}(x, y)$ is the pixel values belonging to a detected vehicle by non-selective model, $m_{ET,t}(x, y)$ is the pixel values belonging

to the temporal edge and $m_{ES,t}(x, y)$ is the pixel values belonging to the spatial edge.

If a pixel is found to be a background pixel or it is a spatial or a temporal edge pixel, then that pixel will not be used to update the background model. Since stationary objects are also present in the view of the scene, it is possible to categorize the pixels that are belonging to other stationary objects. The spot velocity of the vehicle is calculated using the geographic transformation between 3-D world and 2-D pixel image. Therefore, when the VSN is installed at a certain place, it is necessary to calibrate the VSN according to the focal length of the camera sensor, the height of the camera that is being installed and the camera angle to the road as described in (2.3),(2.4), where, ϕ is the angle to the road, f is the focal length, (X, Y) is the coordinate on the road and h is the height to the camera from road level, (x_{CAM}, y_{CAM}) is the pixel coordinate of the vehicle in the image which is taken by camera.

$$x_{CAM} = f \frac{X}{Y \cos(\phi) + h / \sin(\phi)} \quad (2.3)$$

$$y_{CAM} = f \frac{Y \sin(\phi)}{Y \cos(\phi) + h / \sin(\phi)} \quad (2.4)$$

The complete algorithm is realized in a FPGA by considering the pipeline implementation. However, state machines have also been implemented in order to perform sequential operations such as image acquisition and control of the overall system. They have been able to achieve a higher frame rate which is necessary for the targeted traffic parameter, spot velocity, but a lower resolution frame size which is bounded to a small FoV. Apart from the FPGA implementation, an application specific integrated circuit (ASIC) based implementation has also been investigated in order to reduce the power consumption any further [10]. All of these implementations can handle images with a maximum resolution of 128x128 and a frame rate of 150 fps.

The proposed algorithm in [11] has been implemented in a modern system on chip (SoC), where both hardware and software functions could be effectively performed in a separate section of the same chip. Authors have stated that, due to the random memory access requirement and less parallel behavior in some of the functions in any image processing and vision algorithm such as the region

growing segmentation which is also known as the connected component analysis, it has a less capability for the quick execution of such functions in hardware platforms like FPGA which has a highly parallel behavior. Their basic intention has only been on the detection and counting of the number of vehicles that pass through a specific location. Vehicle detection is basically carried out with time-spatial images (TSI), which are generated using the virtual detection line (VDL) despite there are other methods such as background subtraction methods. The authors have mentioned that the background modeling-based background subtraction methods which can be performed at real time in embedded systems have low adaptability to practical issues such as camera jitter, initializing and reinitializing of the model, sudden illumination changes and shadows. The basic idea of VDL and TSI are illustrated in left side and the right side respectively in the Figure 2.2. The presence of a vehicle is detected in binary format by analyzing the present and previous VDLs. The first step of analyzing is the feature extraction. A CENSUS operation is performed which is also known as the linear binary pattern (LBP) to extract the texture features and SOBEL operator is performed to extract the edge features. Then, the pixel-wise and feature-wise difference in present and previous VDLs is calculated. The final signal is filtered out and the practically generated threshold value has been applied in order to obtain a binary value (TRUE or FALSE) for the presence of a vehicle.

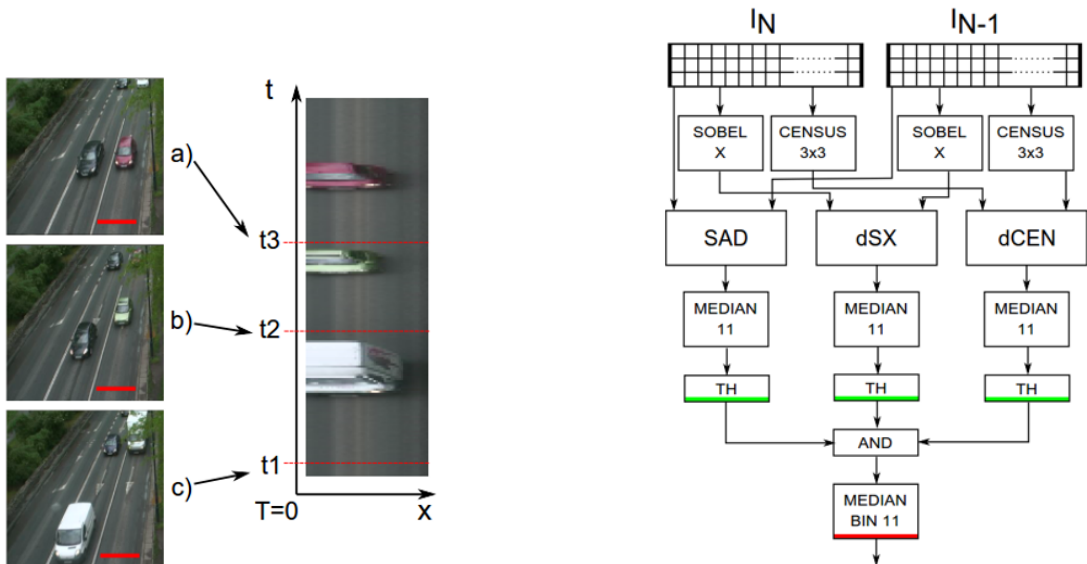


Figure 2.2: Example TSI image in left side and block diagram of the algorithm in right side.

Useful traffic related information using multiple VDLs such as vehicle speed, vehicle type classification according to size and color has been discussed in [12]

by the same authors as an extended research work.

However, it is also noted that the accurate results for these parameters could be obtained when there is a smooth mobility of the vehicle on the road. It is also worth to mention that, this method only utilizes a few number of pixels that are used to generate the TSI, which means, many information that is being sensed from the camera is discarded. This causes the less usefulness in integration of visual sensor node in WSN.

The complete algorithm has been implemented in Zynq all programmable system on chip (APSoC) from Xilinx which has a heterogeneous SoC architecture. This SoC consists of a Dual core A9 ARM processor and FPGA. The reconfigurable section has been utilized for the color conversion and vehicle detection sections of the main algorithm where, SOBEL edge detection, filtering and LBP transform functions exist. Since these functions are performed in nested loops, it has been useful to implement them in the hardware section. Apart from the vehicle detection and image processing functions, the image acquisition part is also performed by the FPGA section even though the processing system (PS) side is running on embedded Linux operating system. This means, the complete design has utilized some additional hardware resources in terms of both reconfigurable resources as well as additional converters and modules such as FMC/DVI as illustrated in Figure.2.3. Instead of adopting this method, a much simpler interface such as USB which is very compatible with Linux could have been used.

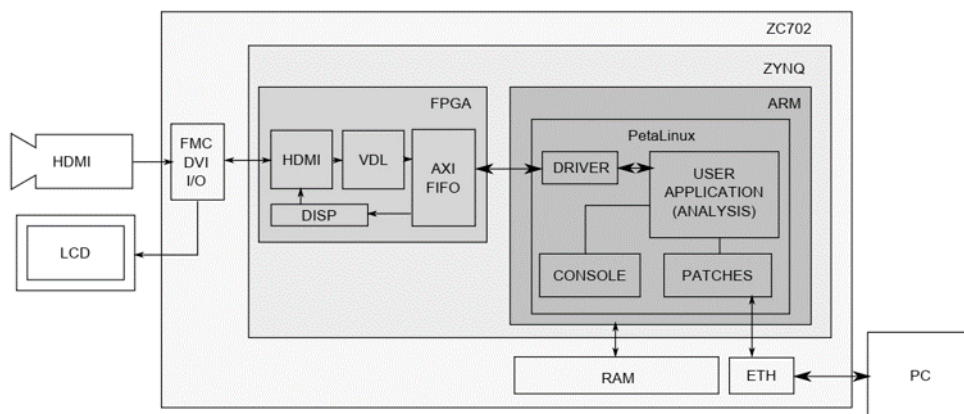


Figure 2.3: Hardware architecture for proposed vision algorithm in [11]

As authors have described, the processing section of the chip performs the day-time, night-time conditions and patch extraction algorithms. The patch ex-

traction algorithm, which segments some data from the main DDR memory, can be effectively performed by the processing system. This can be noticed as a very appropriate separation between hardware (HW) and software (SW) sections on the chip. However, the functions in day-time and night-time condition checking algorithm such as filtering functions, histogram stretching and calculation of SOBEL gradient can be identified as highly parallel algorithm since they are based on nested loops and some mathematical operations. Therefore, that type of algorithms can be implemented in pipelined hardware architecture which can reduce both power consumption and execution time of these functions.

When it comes to processing system (CPUs) based developments to measure the traffic conditions, Raspberry Pi based embedded vision systems have become a trend in the past decade due to several advantages that can be acquired from this series of platforms [14]. The development process of these single board computers (SBCs) is completely based on Linux OS environment. This has led many designs to be based on third party libraries such as OpenCV. Apart from software level advantages, there is a dedicated interface for the CMOS camera integration which can transfer the frame from the sensor to main memory at a higher data transfer rate [13]. The authors in [14] have used this platform to implement the algorithm. As described in Figure 2.2, in the algorithm flow chart, vehicle detection, tracking and speed calculation of the vehicle have been the primary targets of their algorithm.

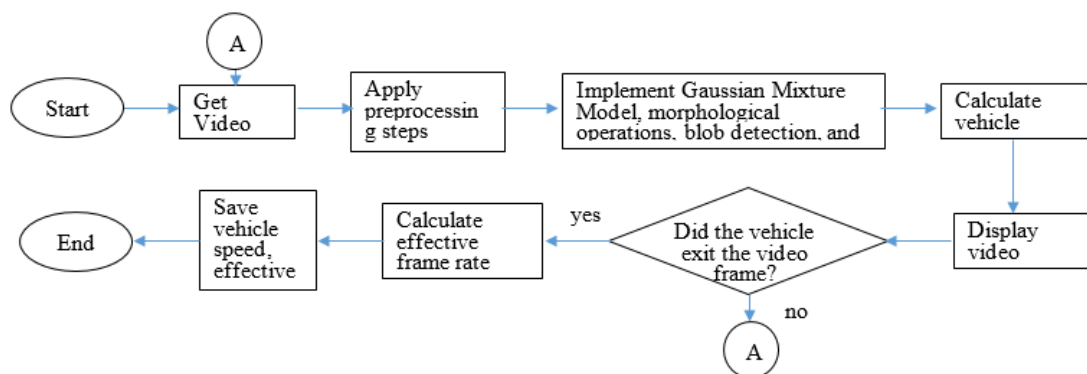


Figure 2.4: The Proposed algorithm to measure traffic condition.

The authors have employed the mixture of Gaussian (MoG) and Kalman filter to detect and track the vehicles. The sparse random projection (SRP) has been utilized to improve the performance in vehicle speed calculation since it can reduce

the amount of data needed to be processed. In general, the SRP is used as a dimensionality reduction method. However, in this case, it has been used as an image compression technique so that the amount of data or in other words, the pixels that are needed to process is much lower than the original raw data. The sparse random matrix has been chosen in a way that, the amount of zero elements are greater than the others. This causes the reduction of the amount of memory requirement to save the sparse random matrix in the main memory. However, the delay caused by the compression of the image using this SRP is not investigated. The morphological algorithm has been applied to the final result of the vehicle detection in order to eliminate the false negative (FN) and false positive (FP) pixels. The complete algorithm has been performed on the video stream with 480x360 resolution. It has shown that, the algorithm can perform with SRP at 7.08 fps and without SRP at 3.29 fps. The use of SRP has caused to speed up the algorithm more than twice than the algorithm without SRP.

A real time and low-cost embedded vehicle counting and classification system has been proposed in [14] so that it can be used to measure the vehicular density of metropolitan areas in order to have proper mobility in those areas. The overall algorithm can be explained as in Figure.2.5.

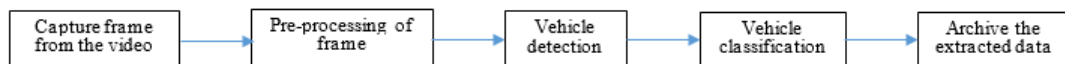


Figure 2.5: The steps of the vehicle classification.

The Gaussian filter which has a 5x5 kernel is used to remove noise in the image. The vehicle detection is carried out using the MoG method since it is widely considered as a better algorithm for background modeling and foreground extraction [15]. The cascade classifier has been employed to classify the vehicles. The window size of the classifier can be changed according to the position of the camera that has been installed in RoI because the size and the angle of the vehicle changes with the camera position. The classifier has been trained with images which have different scales of vehicles as well as different types of vehicles. The algorithm has an accuracy of 97% in average. Although, it is noticed that when two vehicles move close to each other or when they have similar color features, the accuracy of the algorithm decreases. Besides that, better outcomes can be expected from the algorithm only when there are vehicles that can be found within the RoI which are similar to the vehicles that have been used to train the

classifier. A more accurate algorithm comes up with higher computational cost and lack of real-time performance. Even though this algorithm has been targeted to gather traffic information, no functionalities have been integrated to get an idea about the speed of the vehicles.

ODROID-XU4 SBC has been used for the execution of the algorithm because of the convenience of the implementation. This development board is based on Exynos5422 CortexTM-A15 2GHz and CortexTM-A7 Octa core CPUs, Mali-T628 MP6 and 2Gbyte LPDDR3 based memory chip [16]. A proper investigation of functions which have parallel behavior cannot be seen in this kind of a completely CPUs based implementation and since the commonly used vision libraries are implemented considering the use of multi-thread strategy, which is the only option available to improve the performance of the algorithm.

The authours in [17] have used the same development platform to implement the algorithm which can be used to gather traffic information at night time considering the head lights of vehicles. The simple background subtraction method has been avoided since it can malfunction in case of complete or partial unavailability of light. The basic architecture of the algorithm can be illustrated as in Figure.2.6.

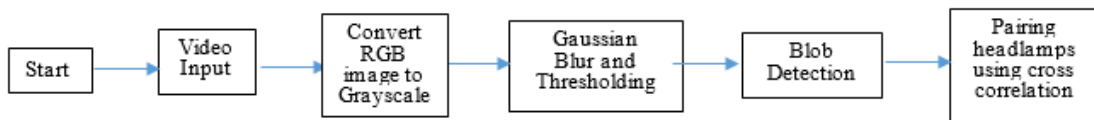


Figure 2.6: Block diagram of the night vision algorithm.

As the first step, the frames acquired from the video input are converted into gray scale images and then, Gaussian blur has been applied to overcome the noise components in the current frame. The headlamps of the vehicles are detected as blobs by applying a threshold value. This threshold value is a predefined value and is not adaptable to the current illumination and environmental conditions. Since, most of the vehicles which causes the traffic such as cars, buses, lorries and trucks are having a pair of headlamps in a symmetric structure, this pair of headlamps classification has been done by using the normalized cross correlation (NCC). This can also be described as NCC template matching for headlamp pair identification.

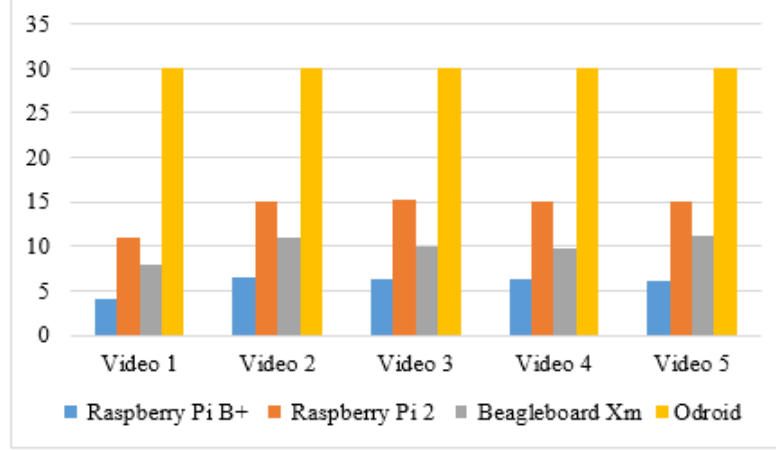


Figure 2.7: Comparison of performance of the night vision algorithm for different SBCs.

$$NCC = \frac{1}{n} \sum_{x,y} \frac{1}{\sigma_f \sigma_t} f(x,y)t(x,y) \quad (2.5)$$

This is the basic methodology that has been followed to identify the number of vehicles at night time. Here, function $f(x,y)$ represents the pixels of the target and function $t(x,y)$ represents the pixels in current template. This night vision algorithm has been implemented in several SBCs and the performance has been investigated.

Table 2.1: Specifications of the SBCs that have been used to compare the performance of the algorithm.

Main component	Raspberry pi B+	Beagle board Xm	Raspberry Pi 2	Odroid XU4
Processor	700MHz ARM11 ARM1176JZF-S core	1GHz ARM Cortex-A8	900MHz quad-core ARMv7 Cortex-A7	2.0 GHz 64-bit quad-core ARM Cortex-A53
Memory	512 MB (DDR1)	512 MB (SD LPDDR)	1 GB(DDR 2)	2 GB (DDR 3)
GPU	Broadcom VideoCore IV, 250 MHz	TMS320C64x+	Broadcom VideoCore IV, 40 MHz	Mali-450 MP3

Both of the implementations discussed above can be used for daytime and nighttime to measure the number of vehicles present on the streets. However,

these kinds of battery powered embedded vision systems are used in urban areas to measure the traffic conditions where streets are lighted using the street lamps.

According to Table 2.1, it can be seen that the performance of the algorithm has been increased with the availability of high-performance hardware resources in the selected CPU based development platforms. The performances of these selected hardware platforms change in terms of operating frequency, number of cores available in CPUs and GPUs, the data transferring speed as well as the capacity of the memory. The main advantage of this kind of only CPUs based hardware platform is the flexibility in implementation. It is mainly because of the availability of vision libraries such as OpenCV which have been designed by considering the multi-threading feature that can be utilized from the multi-core CPUs. It can be noticed that the improvements of the CPU platform-based vision algorithms have strongly depended only on the improvements that can be obtained in the ASIC level. This also causes the designers to stick with the available functionalities in selected hardware and it is not customizable to improve the performance, power as well as the resources utilization in hardware level. Besides that, higher frequency means the higher the power consumption [18], which is a limiting factor for the usability of this sort of hardware platforms in power limited applications such as WSN.

A hardware and software co-design based embedded system for object detection and tracking has been described in [19]. The authors have used an example video stream which contains of vehicles. Besides that, this sort of a tracking algorithm can be effectively used to gather useful information about traffic conditions on road such as average velocity, waiting time and vehicle density in RoI [20]. They have inclined to the heterogeneous hardware platform due to parallel behavior in the detection function which they have used in the algorithm and to obtain the advantages such as branch prediction capabilities in CPUs. They have considered that GPUs obviously are not an option due to their higher power consumption requirement. The object extraction algorithm from the current frame is based on MoG method which is a well-known background modeling method used specifically for outdoor computer vision systems [15]. However, since the original concept of MoG is not so parallel in nature because of the conditional checking of Gaussian parameters for each pixel, the authors in [21] have proposed several methods to perform all the arithmetic operations in a more parallel fashion. These modifications have been done to implement MoG in FPGA section in

pipelined architecture so that it can be performed in real time also by considering the available resources in hardware. Apart from the background modeling, noise filtering and morphological operation, video overlay which adds the color and boundary represented by a rectangle is implemented in hardware section due to the parallel behavior of these functions.

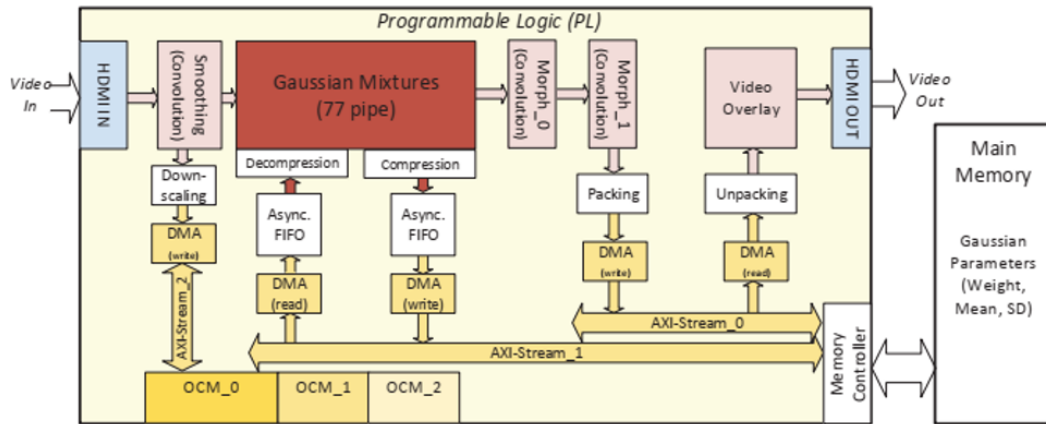


Figure 2.8: Implementation of the parallel functions presented in [19].

They have utilized the external DDR memory to store all the Gaussian parameters namely mean, weight and standard deviation which describe the background image of the video stream. Due to the random memory access requirement, they have used kernel memory map registers to maintain the interface between FPGA and main memory and to obtain high speed frame transaction between CPU and FPGA, the AXI4_Stream interface available in the hardware platform have been utilized. This can be identified as a very effective interface handling since for a custom-made vision algorithm, it is important to choose a proper data transaction and time synchronization between HW and SW sections [22].

After the construction of binary image of the foreground, this image is transferred to the processing system section in order to perform the object labeling, object tracking and trajectory estimation. Since the processing system consists of a dual core processor, the algorithm performed in SW has been partitioned between the two cores so that the maximum throughput that comes from the HW section can be handled in a manner that, any delay between consecutive frames cannot be noticed. The blob labeling has been carried out as a connected component analysis. However, according to the authors in [23], this is also a highly parallel algorithm, but maybe due to the lack of resources in HW section

after the implementation of background modeling algorithm, this function has been performed using available cores in the processing system.

The detected objects are tracked between consecutive frames by considering the histogram similarities. The properties of the tracking objects such as width, height and position have been taken in to account. The Kalman filtering technique has been adopted in order to fine-tune and predict the properties of the objects.

As described in Figure.2.9, the synchronization of the data flow, the direct memory access between HW and SW sections and cache memory handling have been controlled using the interrupts and the share dynamic memory. The interrupts are handled by the global interrupt controller (GIC) and the shared dynamic memory has also been utilized as the buffer between HW section and SW section since the high throughput coming from the hardware section is not bearable to the processing system (PS) section.

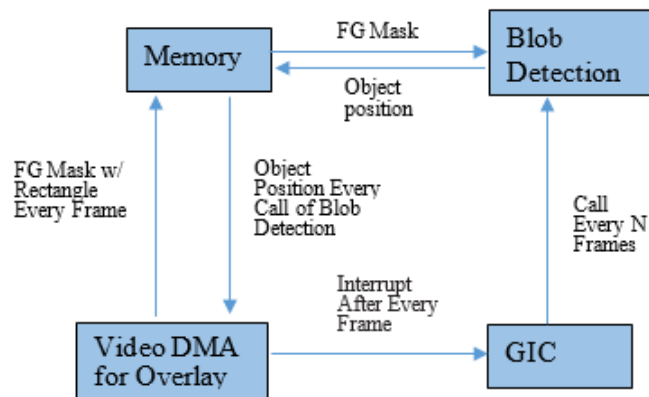


Figure 2.9: HW/SW data flow and interrupt route.

Even though, the FPGA section has been utilized for the MoG based object detection which consists of intensive arithmetic operations, the programmable logic (PL) section has utilized only 16.49% of a power fraction from the total power consumption. When it comes to the hardware utilization in FPGA section, the MoG algorithm has utilized 54% of the hardware resources out of the total hardware resources available in their chosen development platform. In the design, they have effectively used the block memory available in the FPGA section as a local memory storage in order to reduce the frequent dynamic memory access requirement.

2.1 Summary

The achievements, disadvantages and the gaps still exist in the current designs of vision-enabled sensor node fashioned embedded systems that have been implemented by the researches so far are summarized as follow.

- Most of the real time and low power implementations have used reconfigurable devices like FPGA, in order to execute customized image processing functions in pipelined structure.
- The CPU based systems have always stick with available hardware processing units which cannot be adapted to other functionalities as programmable logic designs do.
- Hardware and Software Co-Design based developments have better approaches to execute both sequential and parallel functionalities of the vision algorithm meanwhile taking advantage of using the embedded OS for basic hardware interfaces and software library integration. The partitioning of the algorithm between hardware and software sections is very important to consider.
- Most of the researchers have struggled a lot to separate vehicles from RoI in order to get the number of vehicles exist on the road and to initiate the tracking of the vehicle to get the speed of the vehicle or using the geometrical image transformation.
- As in [19], possibility of modifications that can be done to the existing highly accurate computer vision and image processing algorithm is an important area to investigate in order to execute them in a more parallel behavior so that the real time performance can be expected when the algorithm is performed in hardware.
- The performance and the hardware resources utilization of the system for different frame sizes is not investigated in most of the designs.

Chapter 3

SYSTEM ARCHITECTURE

The traffic estimation or in other words, the information about the mobility condition on road which is based on a video stream of FoV comes up as a result of a computer vision or an image processing algorithm that has been applied on those frames. Therefore, in any vision-based traffic analysis system, the accuracy, performance, or in other words, “how well the traffic condition can be described” depends on the vision algorithm that has been used. Therefore, important traffic parameters which can be implemented in hardware system are investigated in the first place and then, the vision algorithm is implemented so that, the targeted traffic information can be extracted. The feasibility in the expandability of the vision algorithm with available resources in the hardware platform also needs to be considered in a way that, better spatio-temporal description of the traffic condition can be harvested with the higher availability of resources and power budget.

3.1 Selection of Traffic Parameters

The traffic parameters which can be measured using the computer vision techniques have to be considered in the first place. The following traffic parameters have been considered since most of them are more frequently used in existing traffic measurement systems and used in traffic engineering, in order to describe the traffic condition on road.

- Traffic Density

The traffic density is one of the most commonly used parameters. It is defined as a fraction between number of vehicles exist on road with respect to the length of the lane or unit distance. This describes the concentration of the road with respect to a special variation.

$$\text{Traffic Density} = \frac{\text{Number of vehicles on the road}}{\text{Length of the considered lane}} \quad (3.1)$$

In order to measure this parameter using computer vision algorithm, the algorithm should be capable of extracting the number of vehicles exist in current image of the video stream. Besides, it should be capable of eliminating the unnecessary objects exist on the road. In addition to that, the length of the RoI is also essential. This can be manually measured since it is required to be measured only once.

- Average Speed

The average speed is referred to as the speed of a vehicle that has been obtained during a unit time of the journey. This can provide how safe the road is as well as the flexibility of mobility on the road. This is measured according to (3.2).

$$\text{Average Speed} = \frac{\text{The distance of the journey}}{\text{Time to complete the journey}} \quad (3.2)$$

In a computer vision algorithm using this sort of a traffic parameter, it is essential to track the vehicle between consecutive frames, in order to know the time that has been consumed by the vehicle.

- Spot Speed

This is known as the speed of a vehicle (v_i) at a specific point on the road. Usually, this parameter is presented as the time mean spot speed or the instantaneous speed as illustrated in (3.3), where, N is the number of vehicles.

$$U_t = \frac{1}{N} \sum_1^N v_i \quad (3.3)$$

To measure this parameter using vision techniques, it is essential to perform the algorithm at a higher frame speed because otherwise, the algorithm may miss the vehicle or sometimes it may mix up with another vehicle in the RoI. This can be measured using the geometric transformation or an instantaneous pixel analysis technique such as VDL. In the implementations of [10] and [12], this parameter has been taken as an output parameter.

- Waiting Time

The waiting time is an important traffic parameter in case of effective traffic light controlling and lane prioritization. This can measure how long a

vehicle has been waiting in a vehicle queue at a traffic light or in a dead lock situation [24]. This is measured as the time duration of which a vehicle has been stopped at a single point or travelling at a very small velocity during the specified length of journey. At the end of the journey, this can be represented as the summation of the total time the vehicle has been stopped as in (3.4). A vision algorithm with tracking capability is essential in measuring this parameter.

$$\text{Waiting Time} = \sum \left(\begin{array}{l} \text{Time duration of the vehicle that has} \\ \text{been stopped at point 'i'}. \end{array} \right) \quad (3.4)$$

- Lane Occupancy

The lane occupancy which is also referred to as the temporal concentration is another method to measure the concentration on roads. Instead of obtaining the exact number of vehicles exit in the RoI, this is based on how much of area of the road has been utilized by the vehicle in the considered RoI. This parameter has been commonly used in induction loop sensor-based traffic analysis techniques [25] [26], but this can also be used in vision-based systems. The area occupancy is given in (3.5).

$$\text{Area of the Lane Occupancy} = \frac{\sum B_i}{A} \quad (3.5)$$

where, B is the area covered by vehicle and A is the total area of the road. If we can find the area that has been utilized by the vehicle within the RoI, this parameter can be easily found out. The area can be measured using a proper foreground extraction vision algorithm. In addition to that, the interested area of the FoV of VSN can be simply calculated by multiplying the width and the length of the road.

After the consideration of image processing and computer vision requirements which are needed to be performed in order to calculate the above discussed main traffic parameters, average velocity, waiting time and the lane area occupancy are selected as the main expecting outputs from the VSN. Usually, the embedded level object extraction methods are depending on the conventional algorithms such as background subtraction. This kind of traditional algorithms are more often suffered with the occlusion situations when the number of objects is going to be calculated. However, there are many approaches such as optical flow [27]

to eliminate this sort of obstacles to a certain degree but, with higher traffic cognition situations with similar color vehicles, this problem can be increased. On the other hand, the limited hardware resources and power budget may limit the system to perform additional vision algorithms and expect results in real time.

There are rapid improvements based on convolutional neural networks (CNN) and many other neural network architectures in computer vision algorithms in object recognition and classification. Implementation of those algorithms in hardware platforms also has been discussed in [28] [29]. Even though, the accuracy of this kind of CNN based algorithms is higher than the existing conventional algorithms in many aspects, the frame rates of those algorithms cannot be satisfied even in the high-performance platforms. Besides that, the cost and the power requirement of those high-performance hardware platforms can adversely affect the applicability of them in real world application scenarios.

The spot speed of the vehicle is not considered as an output of our VSN since it only describes about the vehicle speed at one specific location in the RoI. When this is compared with the average speed, it is much less descriptive about the traffic condition [30].

3.2 Proposed Vision Model for Traffic Estimation

Considering the selected traffic parameters discussed in the above paragraph, a suitable vision model can be considered so that, it can calculate necessary traffic information required as well as the findings from the literature review. The Figure 3.1 illustrates the overall view of the vision model that has been proposed to implement in VSN hardware platform.

As the first step, since most of the camera output video files are based on RGB color format, the video frame that is being streamed into the system is required to be converted from RGB to gray scale. If we keep using the RGB color frame for the rest of the algorithm, it may cause three times higher memory utilization with respect to the monochrome frame and may consume higher processing power. Then, there should be a method to separate the vehicles in foreground from the background image. This is not only useful for calculating the lane occupancy (LO) but also for the average velocity estimation.

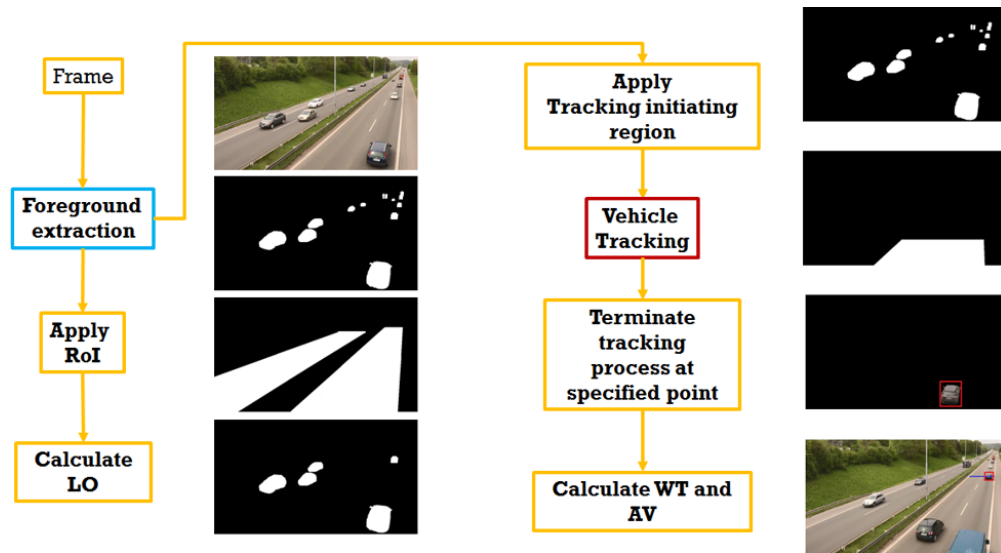


Figure 3.1: Flow Diagram of the Vision Algorithm.

There should also be a method to eliminate the unnecessary detections created by the surrounding objects such as pedestrians and moving trees. Besides, it is also important to filter out the unwanted area on the road and focus on the most important section of road to measure the traffic condition. This may avoid the unnecessary complexity of the algorithm. This can easily be done by using an image of the FoV and manually marking the RoI. Then there will be an image somewhat similar as shown in the Figure 3.1. The area that has been occupied by the vehicle can be calculated using the connected component analysis [23] and considering the road area in RoI.

According to the descriptions of average velocity and the waiting time of a journey in the previous section, there should be two specific locations to calculate these two parameters and it is required to track the vehicles in between the two selected points. Therefore, these two start and end points of the journey are considered as the tracking initializing point and terminating point. These points can be configured in the algorithm just as the RoI was configured.

As can be seen in the vision architecture, the extraction process of the vehicles appearing in the scene and their tracking process are the trickiest functions in the algorithm. Therefore, in detail description of these two functions has been carried out in the following sections.

3.3 Vehicle Extraction

Expected outcome from the functionality of the foreground extraction algorithm is illustrated in Figure 3.2. There are many computer vision and image processing algorithm-based foreground extractions can be found in the literature. Due to the challenges and the requirements in those algorithms to be performed in embedded systems, several foreground-background separation algorithms are considered.



Figure 3.2: Illustration of the foreground extraction process.

Most of the embedded level system based object detection algorithms are based on the background subtraction method. In this method, it is required to have the background image in order to extract the foreground objects in the frame. The basic concept of the background subtraction is based on the inequality as described in (3.6) and (3.7), where, N is the number of consecutive frames which have been considered, I_t is the intensity values of the image at time "t" and I_b is the intensity values of the background image. The threshold value, k , is the parameter which is used to determine the foreground and the background. The final result would be an image which has marked the foreground object area of the frame in white color. The original frame would have to overlay on the final frame of the background image in order to isolate foreground objects from the current frame.

$$\text{If } |I_t - I_b| > k, \text{ foreground} \quad (3.6a)$$

$$\text{If } |I_t - I_b| < k, \text{ background} \quad (3.6b)$$

where,

$$I_b = \frac{1}{N} \sum_{t=1}^N I_t(x, y) \quad (3.7)$$

Even though, some of the designs such as [1] [10] have used static backgrounds, it is certainly not a good approach for an outdoor vision application since its

background may vary due to multiple reasons which in turn will lead to false detection. Therefore, there are many methods to generate dynamic background image through the background modeling technique instead of a static background image. There are some approaches where the background image is updated using the mean filter for each frame incoming to the algorithm. But, the drawback of this kind of an algorithm is that, any sudden change of illumination and the presence of noisy pixel components can largely affect the foreground image.

The running Gaussian model can be identified as a somewhat improved method rather than the simple background subtraction. This method maintains a Gaussian model of the background image and (3.8) and (3.9) describe the parameters of that Gaussian model. Instead of having a constant threshold value, k , as in (3.6), this method calculates the threshold value for each frame at time, t , according to (3.9), where, α is the learning rate. This approach can be seen in many VSN designs which have been developed for different applications due to its simplicity as well as less computational cost. This algorithm has been improved in [1] as described in chapter 2 which has been used to acknowledge the presence of vehicles and to calculate the spot speed of a vehicle using geometric transformation. Even though this algorithm may perform for the indoor vision applications very well, for the outdoor applications, the resultant foreground image may have been largely affected by the false negative (FN) and the false positive (FP) pixels [31].

$$I_{b(t)}(x, y) = \alpha \cdot I_{t-1}(x, y) + (1 - \alpha) \cdot I_{b(t-1)}(x, y) \quad (3.8)$$

$$k_t^2(x, y) = \alpha [I_{t-1}(x, y) - I_{b(t-1)}(x, y)]^2 + (1 - \alpha) \cdot k_{t-1}^2(x, y) \quad (3.9)$$

The foreground extraction using Gaussian distribution based background image is another popular method for most of the object detection requirements in vision algorithms. In this method, the pixel intensity is described by the Gaussian model as described in (3.10). This equation represents how to model a single pixel with multiple Gaussian distributions where, σ is deviation of the pixel value, μ is the mean value of the pixel values, x is the value of the pixels. Figure 3.3 represents the Gaussian distribution of the intensity value, x , of a pixel. Instead of updating the model for each pixel value coming from the camera sensor as in running Gaussian method, this method follows up the fitting method to generate the most suitable Gaussian curve for the past pixel values that have been received.

$$\eta(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\left(\frac{x - \mu}{\sigma}\right)^2\right) \quad (3.10)$$

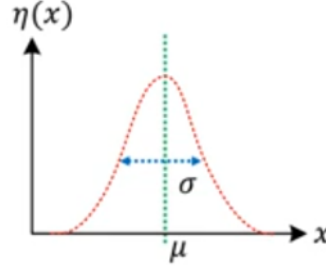


Figure 3.3: Gaussian distribution.

The MoG model can be considered as the extended version of the single Gaussian model which describes the pixel intensity using multiple Gaussian models instead of a single one. This makes the model more robust for many challenging situations such as sudden change of illumination and small vibrations of the camera. The model of the mixture of Gaussian can be described as in (3.11), where, w is the weight factor of each Gaussian distribution. However, the drawback of this method is that the robustness comes up with large number of parameters since all the pixel values are considered to be mutually independent. This leads to a huge memory requirement and a higher complexity in modeling and updating which is based on higher number of iterative methods in the expectation and maximization algorithm.

$$P_r(x) = \sum_{k=1}^k w_k \eta(x; \mu_k, \sigma_k) \quad (3.11)$$

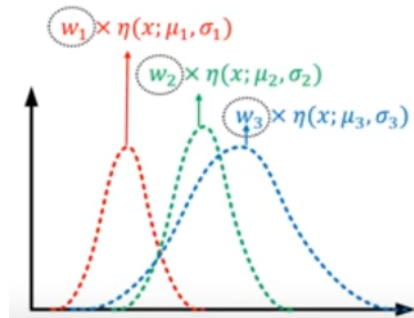


Figure 3.4: Distribution of Mixture of Gaussian.

In [19], some modifications have been done to the MoG in order to perform the algorithm in hardware and software co-design platform with more parallel fashion since the original algorithm is more sequential. Even though, the algorithm performs at 60 fps, the properties of the algorithm such as the number of Gaussian models that have been used and the other valuable parameters and model updating methods which strongly describe the accuracy of the algorithm have not been discussed.

3.4 The ViBe Algorithm

The universal visual background extractor (ViBe) algorithm for foreground separation has been proposed in [32]. This model has a completely different approach with respect to the other background subtraction and Gaussian modeling-based algorithms [33]. This algorithm has shown highly accurate performances towards many practical problems such as camera jitter over other approaches for the foreground extraction. Apart from that, this consists of the quick but accurate model initializing method and time independent model updating solution which is not available in any other background modeling technique [27]. The vision model of this algorithm can be described in three categories of how the model works and describes the background of the scene, how the background model is initiated and how it is adaptable to the upcoming pixels.

3.4.1 Description of the Model and Pixel Classification Method

Instead of a single valued pixel intensity of the background image, the ViBe algorithm describes a pixel intensity at Euclidean location, x , using an array of pixel values as shown in (3.12). Here, N is the maximum number of pixels contains in the array. The term v_i denotes background pixel value which is placed on the i^{th} location by model initializing function or model updating function.

$$M(x) = \{v_1, v_2, \dots, v_N\} \quad (3.12)$$

In order to classify the pixel value at location, “ x ”, of the current frame, a volume of the model which is similar to a sphere of radius, “ R ” and centered at location, x , is considered. If there are j number of pixels that could be found within this sphere, then this pixel at location, x , of the current frame is considered as a background pixel. Otherwise, this location will be considered as a foreground pixel. This sphere is represented in Figure 3.5.

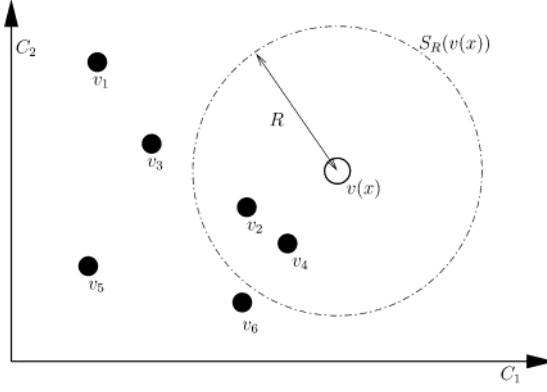


Figure 3.5: Representation of the sphere.

The minimum value for j can be defined by (3.11). The S_R represents the sphere with radius, “R”.

$$j\{S_R(v(x)) \cap \{v_1, v_2, \dots, v_N\}\} \quad (3.13)$$

The accuracy of the model depends on the length of the radius of the sphere and the minimum value of j which is considered to classify the background pixel location from the foreground pixel of the current frame. The sensitivity of the model or in other words, the efficiency of the algorithm to adapt for the variation of the background model may increase with the number of background pixels represented by the model and the number of pixels that are similar to the current pixel value that is required to classify the pixels in current frame as background pixels.

3.4.2 Initialization of the ViBe Model

The model that describes the background in ViBe algorithm is initiated by using the first frame which comes into the algorithm. Since the first frame is not aware of the temporal variation of the background, it is assumed that the neighborhood pixels in the first frame contain similar features of the temporal pixels at the same location. Hence, the initialized model from the first frame can be expressed as in (3.14), where, N_G is the set of neighborhood pixels around pixel, x .

$$M^0(x) = \{v^0(y|y \in N_G(x))\} \quad (3.14)$$

Here, M^0 denotes the initiated model from the first frame. Since the location, x , of the model is described by an array with N number of pixel samples, the array will be filled with the neighboring pixels that exist around the location, x , in the first frame. It is possible to have repeated pixels in the array if the value of N is greater than 8.

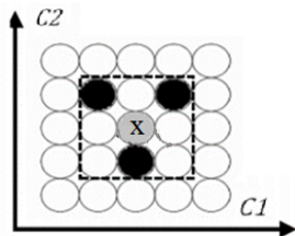


Figure 3.6: The neighborhood pixels at location, “x”.

Figure 3.6 illustrates the neighboring pixels that are being used to fill the initialized model of the ViBe. The main advantage of this kind of a method for initialization is the quick adaptability of the algorithm to the environment. When we use this kind of an approach in embedded systems with respect to other algorithms such as running Gaussian, MoG where many frames are required to initialize the detection algorithm, we have the flexibility to keep the system at sleep mode when there is no requirement for processing, which leads to the power saving of the system.

3.4.3 The Updating Process of the Background Model

The updating process of the ViBe model is based on the conservative method which means only the pixels that are classified as background pixels are used to update the background model. Once the algorithm has classified a pixel location in the current frame as a background pixel, this pixel value is used to update the model. Since, the conservative method is followed, the pixel array where the background pixel is found in the current frame will be updated. The position of the array will be picked up by a random decision and then, the pixel value existing in that particular location will be replaced by the pixel value in the current frame.

Since, the conservative method is followed, this algorithm will not update any of the pixels in the array which exists in a similar position where the foreground pixels exist in the current frame. Therefore, it is essential to keep those pixel arrays of model updated, for the change of illumination and the objects which

are introduced to the scene long time ago. This is done by following the same assumption that the neighboring pixels share the similar temporal feature. This means that, if a particular array of the model is going to be updated because of the similar position of pixel value in the current frame is categorized as a background pixel, then, one of the neighboring pixel arrays out of eight will also be updated using the same pixel value in the current frame as shown in Figure 3.7.

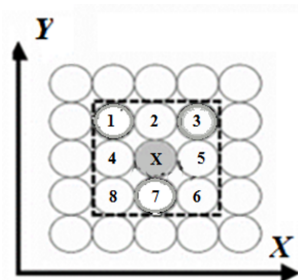


Figure 3.7: The updating location of the model.

The key feature of the updating mechanism of ViBe which differentiates with the other background updating methods is the fact that pixels in the model will not be replaced with other upcoming pixels just because of the time duration a pixel has spent in the model. This makes ViBe a robust descriptor for the background of the scene.

3.4.4 Identified Drawbacks in the ViBe Algorithm and Modification Proposed to the Original

The pseudocode of the ViBe algorithm is given by the authors in [32]. A part of the code has been shown which is important to describe the modification that has been done to the original ViBe algorithm.

In this pseudocode, "samples" variable represents the model of the ViBe which is described in the previous sections. Basically, three main drawbacks in the ViBe algorithm is identified.

1. Since, the algorithm considers a sphere with radius R , the volume or in other words, a higher number of pixels need to be compared. Besides that, the distance also needs to be calculated in order to check whether it is within the sphere.

Algorithm 1: original ViBe

```

int N // No. of samples per pixel
int R // Radius of sphere
int # min // No. of close samples for being part of the background
int  $\psi$  // Amount of random sub-sampling
int width, height // image size
byte image[width][height] //current image
byte samples[width][height][N] // background model
for  $x < width$  do
  for  $y < height$  do
    int count = 0, int index = 0, int dist = 0;
    while ( $count < \#min$ ) && ( $index < N$ ) do
      dist = EuclidDist(image[x][y], samples[x][y][index]);
      if  $dist < R$  then
        count++;
        index++;
        Continued....

```

2. After the classification process of the algorithm, two other memory accesses need to be done in order to update the relevant array which has the similar position to the background pixels and one of the randomly selected neighborhood arrays.
3. Since, this model is placed in the off-chip memory, the power and the time consumption for the random access to this variable can be a bottleneck to the performance of the algorithm in a less resource available hardware platform.

In order to overcome the issues described above, a different model for pixel classification is considered. Instead of a sphere with radius “R”, a cuboid which has a length ‘N’ and a width and a height of 3 is considered. As shown in Figure 3.8, this cuboid is considered to classify the pixels in current frame.

$$\{B_{3 \times 3 \times N} v(x)\} \cap \{v_1, v_2, \dots, v_n\} \quad (3.15)$$

Then, the volume which is represented in the model by the cuboid can be expressed as in (3.15). Since, the searching volume which is tackled by the cuboid is much lower than the sphere, instead of searching for the exactly similar pixel value within the cuboid, the modification was done to the algorithm so that, it

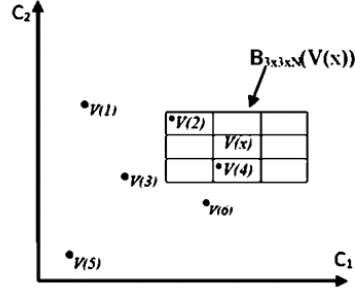


Figure 3.8: The cuboid concept.

Table 3.1: PCC and tolerance values

D value	PCC
1	0.6052
3	0.9671
5	0.9882
7	0.9920
10	0.9931
13	0.9929
16	0.9923
20	0.9903
23	0.9889
26	0.9885
29	0.9879

would consider a pixel located at x in the current frame as a background pixel, if two pixels can be found within the cuboid which are having an absolute difference between current pixel and the pixels in the model which is less than a tolerance value.

A suitable value for the tolerance is obtained by performing this modified algorithm on the dataset which has been used to obtain other constant values of the ViBe algorithm. The modified ViBe algorithm is executed with different tolerance values. Then the percentage of correct classification (PCC) is calculated according to (3.16), where, TP, TN, FP, FN are respectively, true positive, true negative, false positive and false negative pixels. The Figure 3.9 represents the PCC against different tolerance values.

$$PCC = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.16)$$

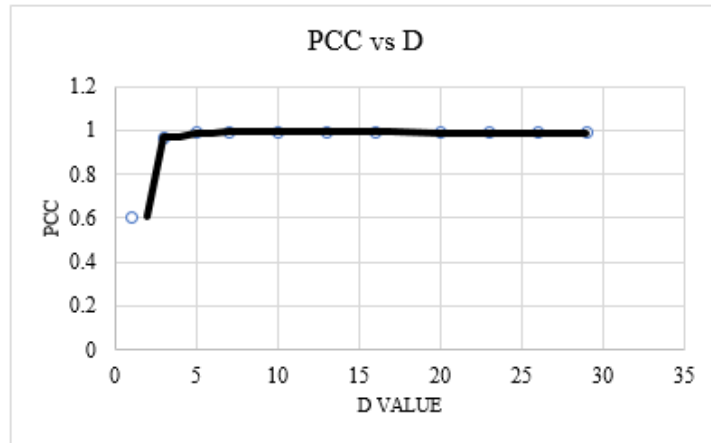


Figure 3.9: PCC values for different tolerance values.

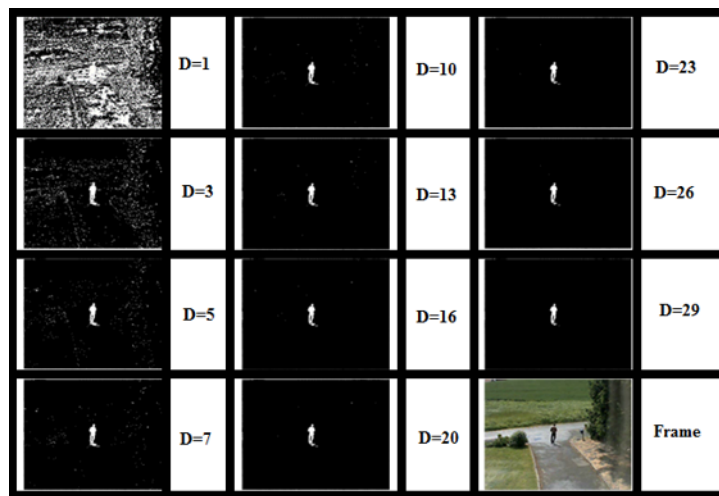


Figure 3.10: Resultant frame for each tolerance value.

The PCC value is taken as an average value for more than 1500 frames. The Figure 3.10 represents the appearance for one frame in the data set. According to the Figure 3.9 and the table, it can be seen that, initially, the accuracy of the algorithm increases when the tolerance value is increasing, but after the tolerance reaches value 10, the accuracy decreases. This is because the space for a pixel to be classified as a background pixel increases with a higher tolerance value.

The pseudocode of the modified ViBe algorithm can be re-written as algorithm 2. The important advantage of this modification is not only the amount of data need to be searched by the pixel classification process is less than the original but also due to the sequential memory access, the random memory access requirement of the algorithm which leads to slow down the performance is avoided. According to pseudocode, this also enables us to place a small fraction of the sample variable,

$temp_sample$, close to the operating processing units. Another advantage is that, we can update both the neighborhood array as well as the array which has the similar location to the current frame using this fractional variable of the original sample variable. This has been demonstrated in Figure 3.11 and in algorithm 2.

Algorithm 2: Modified ViBe

```

uint8_t temp_sample[N][3][WIDTH]; //temp variable memory
for i<HEIGHT do
  for j<WIDTH do
    for n<N do
      temp_sample [n][2][j]= temp_sample [n][1][j];
      temp_sample [n][1][j]= temp_sample [n][0][j];
      temp_sample;
      [n][0][j]=samples[(n*WIDTH*HEIGHT)+((i)*WIDTH)+j];
      // load array back to DDR
      samples[(n*WIDTH*HEIGHT)+((i-2)*WIDTH)+j]=
      temp_sample [n][2][j];
      int count = 0;int index=0;
      while (count < min)&&(index < N) do
        pixel_value=new_frame[WIDTH*i+j];
        Continued....
  
```

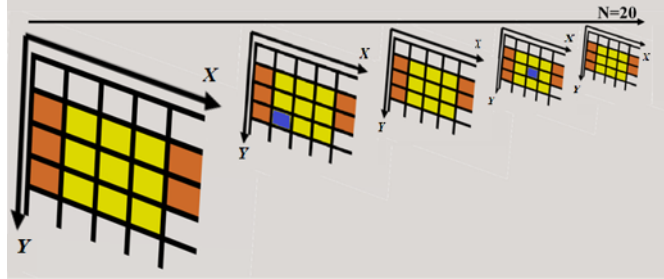


Figure 3.11: The cuboid in actual model and the update process.

The performance of this modified ViBe algorithm is compared with MoG method and the original ViBe using the dataset available in [34] [35]. According to the results, this modification has a lower accuracy in terms of PCC with respect to the original ViBe algorithm yet demonstrates better accuracy than the popular MoG method. This has been illustrated in Figure 3.12. Although, the modification has decreased the accuracy of the original ViBe, the flexibility to improve the efficiency is increased meanwhile holding a greater accuracy than the MoG.

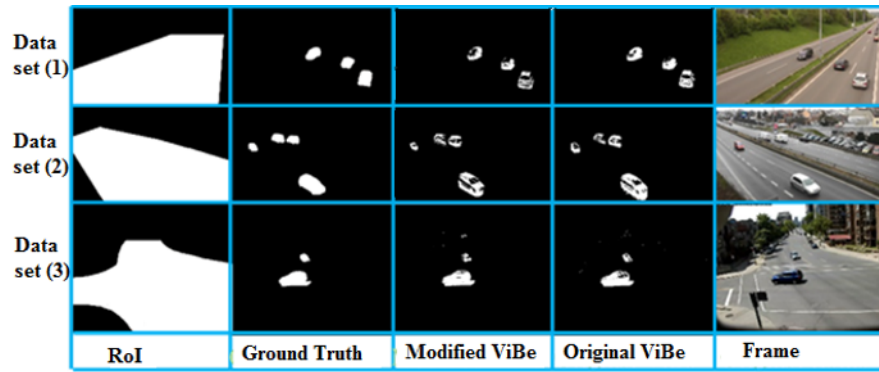


Figure 3.12: Comparison of modified ViBe with MoG and original ViBe.

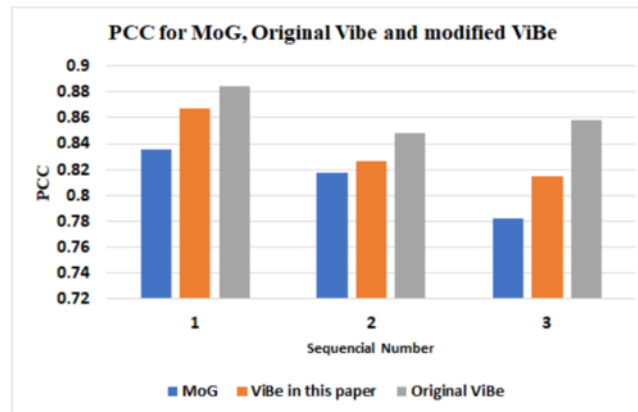


Figure 3.13: Accuracy of the MoG, Modified ViBe and the original ViBe in term of PCC.

3.5 Vehicle Tracking

As explained in the previous sections, the waiting time and the average velocity of the vehicle have been considered to complete the specified journey. The tracking of vehicle to inspect the mobility can be tricky due to the numerous challenges that can be occurred. The sudden change of illumination due to the clouds, the shadows created by the tolls and buildings, same color vehicles and the stationary objects such as trees exist in the RoI are some of the challenges created by outside parties to the tracking algorithm. Besides, appearance changes due to scale variation or rotation of the tracking object and occlusion between tracking object in the scene can be identified as the accuracy and performance related problems.

The basic methodology of the features based automated object tracking algorithms is illustrated in Figure 3.14. As can be seen, the feature extraction and the tracking process of the objects between consecutive frames are the key function-

alities of a tracking algorithm. Therefore, various tracking algorithms are there with different approaches for feature extraction and tracking of the extracted features to overcome the issues mentioned in the previous paragraph.

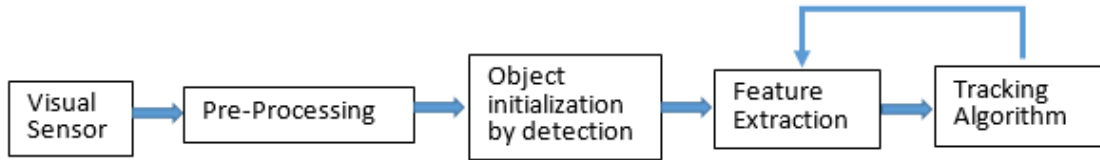


Figure 3.14: Basic steps in a tracking algorithm.

The mean shift algorithm is one of the tracking algorithms which has been implemented in many embedded level vision systems [36]. The tracking function of the algorithm tries to find the new densest location in the current image based on the histogram density function which describes the tracking object. Then, the mean shift vector of the object is calculated as illustrated in Figure 3.15.

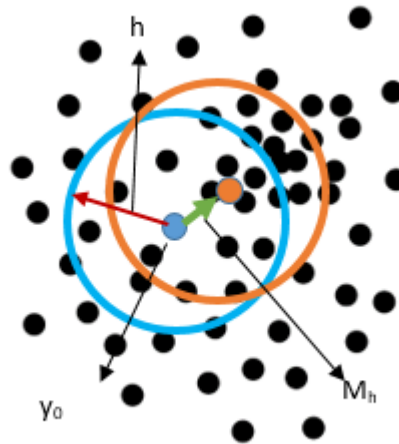


Figure 3.15: Mean shift vector

This mean shift vector can be calculated as in (3.18). The function, “ k ” is the first derivative of the kernel profile since a non-parametric density function is considered as the gradient. The following weight calculating function has been derived from the Bhattacharya coefficient function.

$$w_i(y_0) = \sum_{u=1}^m \delta [S(x_i) - u] \sqrt{\frac{q_u}{\hat{p}_u(y_0)}} , \quad \text{where } 0 \leq w_i \leq 1 \quad (3.17)$$

$$M_h(y_0) = \frac{\sum_{i=1}^n w_i k(|\frac{x-x_i}{h}|^2) x_i}{\sum_{i=1}^n w_i k(|\frac{x-x_i}{h}|^2)} \quad (3.18)$$

In this weight equation, p represents the histogram vector of the candidate for new position of the tracking object and q represents the histogram vector of the template. This summation is calculated from 1 to 255 if the gray scale image is considered and $S(x_i)$ represents the pixel value at x_i location. The authors in [37] have proposed continuously adaptive mean shift (CAMSHIFT), in order to overcome the scale variation problem considering the momentum of the tracking template.

This process is iterated until the value of mean shift vector becomes zero or very close to zero based on (3.18). Even though, the concept of tracking using mean shift is quite simple, it is often reported that the mean shift is not capable to continue the tracking process if the objects are facing an occlusion situation [38] [39]. Besides, it will need more computational power to escape from the saddle place on the distribution curve [40]. Since the iteration process is based on a specific kernel size, k , there is always a maximum object size that can be used in the algorithm. Therefore, the parameters of the algorithm will have to be updated continuously when the position and the angle of the camera changes as well as the size of the frame changes when the range of RoI changes.

The Lucas and Kanade approximation based target tracking using optical flow is another diversely used algorithm. Even though, it is a more computationally complex algorithm, some of the real time implementations in hardware can be found. Optical flow tracking is based on the displacement vectors that can define the translation of each pixel in a region. The fundamental assumption of the optical flow is intensity constancy of tracking object in between two consecutive frames and neighborhood pixels. This can be expressed as in (3.19), where, dt is the time gap between two frames.

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (3.19)$$

The displacement of the interesting point is denoted as u, v . Hence, the new location of the point will be $u + x, v + y$. The above mentions are only valid if the displacement is considerably small. The u, v can be calculated with the help of (3.19).

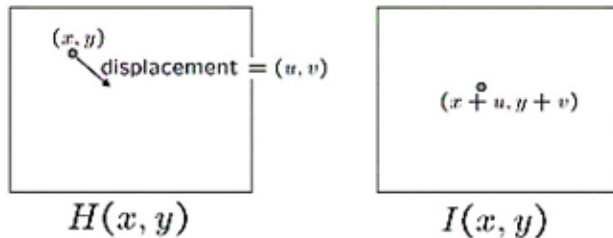


Figure 3.16: Illustration of the displacement.

In the KLT method, it is assumed that the interesting pixel and the eight pixels around it are having the similar intensity between consecutive frames. Using this assumption and the approximation in Taylor series, u, v can be solved as follows.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \quad (3.20)$$

where, f_x, f_y and f_t are $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial t}$ respectively. The left matrix in the right side of the equation is similar to the Harris Corner feature, and according to the concept in [41], the scale variation of the object can be observed by calculating and comparing the distances between the corners. However, since the assumption that the displacement of the feature points is very small, it is essential to make downscaled images from the original image. This calculation of feature point and pyramid of the downscaled images needs to be done for each frame that come into the algorithm which causes to the dramatic complexity in mathematical calculations. In order to perform these complex mathematical operations in embedded system, they have to be performed in higher frequencies which lead to higher power consumption. Another drawback in optical flow is if the illumination of the RoI is suddenly changed and the frame rate of the processing is not enough to bear that illumination change gradually, then the tracking process can miss the tracking object.

Same sort of problems can be seen in the feature-based tracking algorithms. In the hardware implementations of those tracking algorithms, descriptors like SIFT,

ORB, BRIEF and SURF are employed to describe the feature points in tracking object and candidate patch [42] [43]. Besides, most of the times, the feature matching process is carried out by algorithms like brute force. Even though, this kind of feature based tracking algorithms always perform well in case of partially occlusion scale variation, rotational invariant and greatly adaptable for changing of illumination changes. Some of the hardware implementation of this kind of feature based algorithm can be found in [44] [45], but the performances are not in real time. The authors in [46], have proposed vehicle tracking algorithm using LBP. The problem in this sort of fast features is they are not adaptable for scale and rotational variation of the tracking objects. The Figure 3.17 illustrates the process of LBP feature describes.

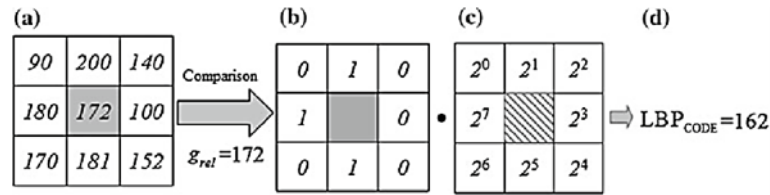


Figure 3.17: The process of LBP

Template matching is a classical method in visual tracking. The main idea is to first detect the object as a template and then, try to find the most similar region to this template in the next frames which would be considered the updated object locations. Several similarity measures have been used like sum of absolute difference (SAD) and NCC [36]. In the (3.5) and (3.5), ' l ' and ' T ' represent the patch and the tracking template of the object.

$$SAD(m, n) = \sum_{x, y \in T} |l(n + x, m + y) - T(x, y)| \quad (3.21)$$

$$NCC(m, n) = \frac{\sum_{x, y \in T} [l(n + x, m + y) - \mu_l] [T(x, y) - \mu_T]}{\sqrt{\sum_{x, y \in T} [l(n + x, m + y) - \mu_l]^2} \sqrt{\sum_{x, y \in T} [T(x, y) - \mu_T]^2}} \quad (3.22)$$

But, this algorithm does not take the dynamic behaviour of the object, scale variation and rotation invariance into account. SAD has less complexity, but

it may be unsuccessful if there are multiple objects with same color [47] [48]. Therefore, dynamic behavior of the tracking also has to be taken into account [47].

The filtering techniques-based object tracking are widely used in many hardware implementations of the visual object tracking systems and in most cases, real time performances have been obtained. The Kalman filtering and the particle filtering are the most commonly used for localization of the object. However, the particle filtering method is more robust over Kalman filtering because it does not depend on a specific model to describe the behavior of the targeting object. Hence, the particle filtering method performs better in case of partial occlusion situations [49]. In most of the real time implementations, the color property of the object has been considered and therefore, scale and rotational variation of the tracking object, in our case, the vehicle tracking can be tolerated by the algorithm.

In this method, the properties of the object such as position, velocity, acceleration and the size modeled statistically throughout the tracking process and in between the consecutive frames [50] [51]. The state of the object, x , and the observation model, y , of the state variable at time, k are expressed as in (3.23),(3.24), where, w_k and v_k are the noise components in the state variable and the observation respectively. Here, it is assumed that the state model of the object follows the Markov property which claims that, the current state of the model only depends on the previous state of the object. Hence, the state of the object at time, k only depends on the state at time, $k - 1$.

$$x_k = f_k(x_{k-1}, w_k) \quad (3.23)$$

$$y_k = h_k(x_k, v_k) \quad (3.24)$$

According to the view point in probability, this state variable and the observation can be expressed as $p(x_k|x_{k-1})$ and $p(y_k|x_k)$. Then, the tracking algorithm is narrowed down to the estimation of the $p(x_k|x_{k-1})$. This is done by the following two steps known as prediction and update step. Based on the total probability and the Markov assumption, the prediction step can be expressed as in (3.25) and then, as soon as the measurement is done, the update equation can be expressed as in (3.26) using the Bayes rule.

$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})dx_{k-1} \quad (3.25)$$

$$p(x_k|y_{1:k}) = \frac{p(y_k|x_k)p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})} \quad (3.26)$$

where,

$$p(y_k|y_{1:k-1}) = \int p(y_k|x_k)p(x_k|y_{1:k-1})dx_k \quad (3.27)$$

The particle filtering is a technique that is used to solve this integration problem using discrete summation of the weighted samples. The state of the object is defined by N number of samples in the state space. One state sample can be denoted as x_k^i , where, i vary from 1 to N .

$$p(x_k|y_k) = \frac{1}{N} \sum_{i=1}^N w_k^i \delta(x_k - x_k^i) \quad (3.28)$$

Based on the importance sampling, it can be shown that, this type of estimation can be done if the particle x_k^i (or in other words, the sample) is selected according to the proposal distribution and weight according to (3.29).

$$w_{k-1} \frac{p(y_k|x_k)p(x_k|x_{k-1})}{q(x_k|x_{1:k-1}, D_k)} \quad (3.29)$$

In most tracking cases, this proposal distribution is taken as the distribution in prediction $p(x_k|x_{k-1}^i)$ [52] [53]. Therefore, (3.30) can be rewritten as,

$$w_k^i \propto w_{k-1}^i p(y_k|x_k^i) \quad (3.30)$$

The pseudo code of the overall particle filtering algorithm is illustrated below. Based on the weight distribution of the particles for current state of the object, the properties of the object for the current frame can be obtained.

The weight of the particle is calculated by considering the histogram similarity between the tracking object and the particle. Many histogram similarity calculating methods can be found in [54]. In [51], the accuracy of correlation, intersection chi-square and Bhattacharyya has been evaluated. Even though, the chi-square method outperforms the Bhattacharyya method, the Bhattacharyya method has been followed due to the less mathematical complexity and is the second best with respect to the chi-square method.

Algorithm 3: Pseudocode of Particle Filtering

-
1. $S_t = 0, \eta = 0$ //initialsetandnumberofparticle
 - for** n **do**
 2. Sample index $j(i)$ from w_{t-1} distribution // based on the previous calculate the new location of the object
 3. Sample x_t^i from $p(x_t|x_{t-1}, v_t)$ using and $x_{t-1}^{j(i)}$ and v_t action // calculate the location using ut and previous position and applying a Gaussian noise
 4. $w_t^i = p(y_t|x_t^i)$ compute importance weight(reweight) // calculate the weight of the new particle
 5. $\eta = \eta + w_t^i$ // get the sum of the all weights to normalised the all weights
 6. $S_t = S_t \cup \langle x_t^j, w_t^j \rangle$ // add each particle in to the particle set
 - for** n **do**
 - └ $w_t^j = w_t^j / \eta$ //normalising the weight
-

$$q = \{q_u\}_{u=1:m} \quad (3.31)$$

$$p = \{p_u\}_{u=1:m} \quad (3.32)$$

In a way that, $\sum_{u=1}^m q_u = 1$ and $\sum_{u=1}^m p_u = 1$.

$$w(n) = 1 - \cos(\phi) = 1 - \frac{p'^T q'}{|p'| |q'|} \quad (3.33)$$

$$q' = (\sqrt{q_1}, \sqrt{q_2}, \dots, \sqrt{q_m}) \quad (3.34)$$

$$p' = (\sqrt{p_1}, \sqrt{p_2}, \dots, \sqrt{p_m}) \quad (3.35)$$

where, q is the histogram vector of the template and p is the histogram vector of the particle patch. The scale adaptability of the vision algorithm is important for a long-range tracking process and in the presence of variety of camera angles. Besides, the tracking process should be adaptable to the dynamic variations of the vehicle movements. This is important since the sudden change of velocity and the acceleration can occur more often unlike the other types of object tracking processes like in humans. These challenges can be achieved in the particle filtering algorithm by defining the state of the object as a vector of static and dynamic

properties of the object [51]. This is illustrated in (3.36) .

$$X = [x \quad y \quad w \quad \rho \quad \dot{x} \quad \dot{y} \quad \dot{w} \quad \dot{\rho}] \quad (3.36)$$

where, x, y represent the location of the object while w and p are the weight and the aspect ratio of the object patch. The rest of the parameters are the dynamic components of the static parts of the object state as described. The aspect ratio of the patch is considered in order to keep the variation between height and the width of the patch in a correlated manner. The state of the object can be defined as follows,

$$X_k = AX_{k-1} + V_{k-1} \quad (3.37)$$

$$A = \begin{bmatrix} I_4 & I_4 \Delta t \\ I_4 & 0 \end{bmatrix}$$

where, v is the noise component in the state variable, I_4 is an identity matrix with four components and Δt is the time gap between two frames. The noise components of each state variable can be denoted as follows.

$$V = (\sigma_x, \sigma_y, \sigma_w, \sigma_\rho, \sigma_{\dot{x}}, \sigma_{\dot{y}}, \sigma_{\dot{w}}, \sigma_{\dot{\rho}})$$

The expansion of (3.37) is illustrated as follows.

$$\text{Dynamic Part} \begin{cases} \dot{x}_k = \dot{\sigma}_x + \dot{x}_{k-1} \\ \dot{y}_k = \dot{\sigma}_y + \dot{y}_{k-1} \\ \dot{w}_k = \dot{\sigma}_w + \dot{w}_{k-1} \\ \dot{\rho}_k = \dot{\sigma}_\rho + \dot{\rho}_{k-1} \end{cases} \quad \text{Static Part} \begin{cases} x_k = x_{k-1} + \sigma_x + \dot{x}_{k-1}(\Delta t) \\ y_k = y_{k-1} + \sigma_y + \dot{y}_{k-1}(\Delta t) \\ w_k = w_{k-1} + \sigma_w + \dot{w}_{k-1}(\Delta t) \\ \rho_k = \rho_{k-1} + \sigma_\rho + \dot{\rho}_{k-1}(\Delta t) \end{cases}$$

As can be seen in above expansion of static and dynamic part of the state variables, the noise components in the dynamic part can influence the variation of the static part. The problem exist with this method is, if the noise component in both dynamic and the static parts is increased, then the range of the algorithm to search the properties of the object per current frame is increased. This causes to mislead and distract the current tracking object from other similar objects in the RoI. Therefore, it is important to assign this random variable to the state of the object in a way that the static components will not be dominated. This can be done using the Gaussian error function as in [51]. This function is always useful in case of multiple variable integration to a function which needs to behave according to a proper Gaussian distribution [55]. The Gaussian error function is defined as in (3.38). The α and β values need to be found for the optimum

results to be obtained from the algorithm. The value, ψ , represents the histogram similarity of the tracking object. Since, we are using the Bhattacharyya coefficient to weigh the particle, it can also be used for the Gaussian error function.

$$\zeta(\psi_k) = \frac{\text{erf}(\alpha(1 - \psi_k) - \beta) + 1}{2} \quad (3.38)$$

$$\sigma_s^k = \zeta(\psi_k) \cdot \sigma_s \quad \sigma_d^k = (1 - \zeta(\psi_k)) \cdot \sigma_d \quad (3.39)$$

The noise component of static and the dynamic variable can be written as in (3.39) in a way that, the noise component in the dynamic part will not propagate to the static part.

Chapter 4

IMPLEMENTATION

Figure 4.1 illustrates the overall outcome of our expectation from the implementation of vision algorithm in hardware platform. The output of the hardware system must be the high-level information which we decided to take out from and this data should be able to transmit through the communication chip. The input to the system will be raw frames from a video resource.

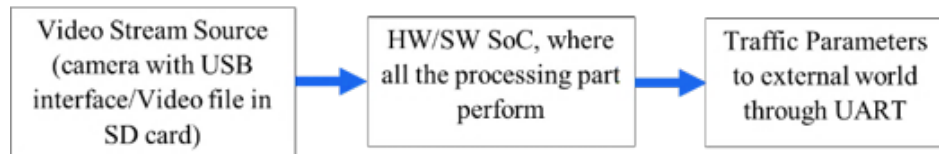


Figure 4.1: Overview of a vision system

There are many different hardware development platforms that have been used to implement the traffic analysis of the vision algorithm. Due to the large amount of data flow which needs to be processed in real-time, the requirement for high dense memory, processing capability and complex interface handling leads to choose high-end IoT edge devices for the implementation of a customized and an application-oriented vision algorithm [56]. Usually, most of the high-end IoT devices have the SBC fashion. Table 4.1 compares the four major types of hardware platforms with examples for each category [57].

There can be many innovative and customized functions in a vision algorithm to solve the real-world scenarios. When it is required to execute these customized functions in a hardware platform, the flexibility to utilize the available hardware resources to synthesize a dedicated hardware device can be beneficial in terms of power, performance and the cost for the hardware resources. Fixed hardware architectures like CPU and GPU do not support the customizability for the given custom functions. However, the flexibility of the already implemented hardware functionality is almost impossible with ASIC but possible in FPGA based designs.

Table 4.1: Comparison of hardware platforms

	Microprocessor	FPGA	ASIC	GPU
Example	ARM Cortex-A9	Virtex Ultrascale 440	Bitfury 16nm	Nvidia Titan X
Flexibility during development	Medium	High	Very High	Low
Flexibility after development ¹	High	High	Low	High
Parallelism	Low	High	High	Medium
Performance ²	Low	Medium	High	Medium
Power consumption	High	Medium	Low	High
Development cost	Low	Medium	High	Low
Production setup cost ³	None	None	High	None
Unit cost ⁴	Medium	High	Low	High
Time-to-market	Low	Medium	High	Medium

¹E.g. to fix bugs, add new functionality when already in production

²For a sufficiently parallel application

³Cost of producing the first chip

The parallel behavior of the hardware platform is very important for the execution of vision algorithm. In most of the vision algorithms, many parallelizable functions can be identified. These parallel functions are poorly performed in CPUs where operations are executed in sequential manner. This situation can be somewhat tolerated by using the multiple cores available in the processing system. But, there is a certain performance level which can be reached and causes to increase the power consumption. The GPUs have many more cores than the CPUs but power requirement also increases rapidly with the number of cores and the operating frequency [58]. According to [7], the FPGAs and ASICs are the best solution for the execution of parallelable functions in customized vision algorithms with low frequency.

When it comes to the development time and cost using each category of the hardware platforms, the CPUs based design always takes less time to develop

with respect to other hardware platforms since there are many more high level programming languages based basic image processing algorithms that have already been implemented and the embedded Linux based operating systems to handle high bandwidth interfaces such as USB, ethernet and display ports. The ASIC based developments take the highest time since there are many verification steps that are involved with the design process. The FPGAs based designs also consume some time to develop because most of the FPGA designs are carried out with hardware description languages such as Verilog and VHDL and therefore, it takes some more time for the development and debugging of the system than the developments done using high level languages.

It is also important that the vision algorithms are not totally built up with completely paralleled functions. There are many algorithms with many functionalities which behave sequentially also can be found such as frame overlay [12]. Even though, the FPGAs are very suitable for the implementation of vision algorithms in number of ways in terms of flexibility in development, adaptability to post modifications, and less power consumption as described above, they are not well suited to perform sequential operations. Usually, the sequential operations are executed in FPGAs using state machines. Since the FPGAs operate at low frequencies (around 100MHz to 200MHz) with respect to CPUs which operate at around couple of Gigahertz, FPGAs cannot perform sequential operations in state machines as high speed microprocessors do. Besides, the implementation of high bandwidth interfaces such as USB, HDMI and display port is a complex process and the use of available reconfigurable resources for a fixed functionality is a waste of hardware resources.

The FPSoC which is having a heterogeneous architecture can be identified as a solution for most of the challenges emerged in totally hardware or totally software-based developments. Basically, FPSoC consists of a reconfigurable section and a processing unit. Figure 4.2 illustrates the architecture of a modern FPSoC [59].

The HPS section of Figure 4.2 refers to as the hard processing system section which is also referred as PS in short. There are many processing units that can be found within this section. There can be many advantages and applicabilities in using FPSoC in numerous ways for IoT related developments where high complex vision algorithms are needed to be performed.

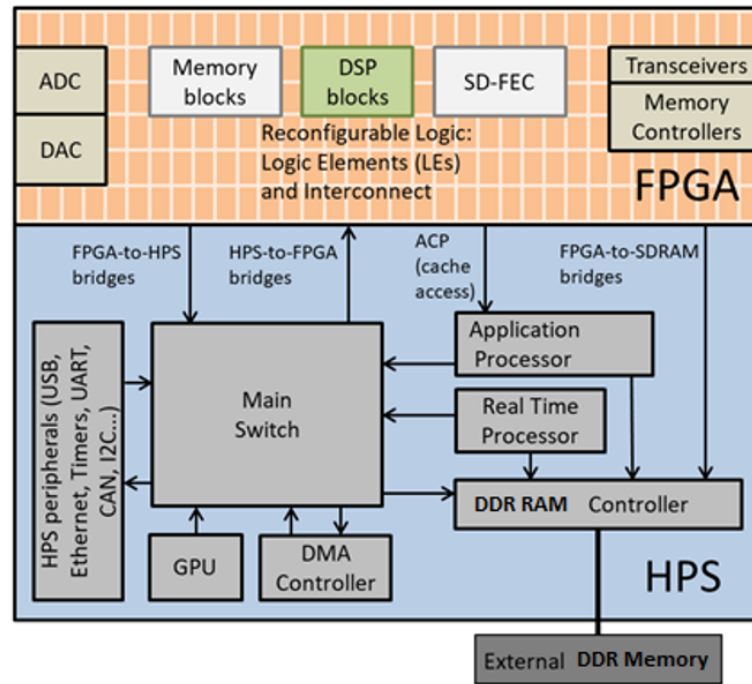


Figure 4.2: The architecture of a modern FPSoC.

- FPSoCs allow the designers to balance the processing load between available processing units and reconfigurable logic elements as a hardware and software co-design.
- The PS section consist of the commonly used hardware interfaces such as UART, I2C, Ethernet etc. This is helpful to preserve the available hardware resources in the PL side and capability interfaces outside hardware such as wireless communication modules and sensors much more easily. Meanwhile, programmable logic can be used for customized protocol implementation.
- There are some hardware implementations using multiple chips which are developed to perform complex algorithms for IoT applications. However, the communication between the chips can be challenging. The FPSoCs have taken this situation into account and a high throughput bridge between the PS side and PL side have been used to establish fast data transferring. Therefore, it is very important to select the appropriate kernel memory mapped data transferring mechanism between PS and PL sides.
- The designer can start the implementation of the algorithm in processing section by making use of the advantages such as embedded Linux based operating system and software libraries where ever necessary. Then, they can

identify the bottleneck functions in the algorithm due to the complex mathematical operations and the large time consumption to process many data using the same instruction. After that, the optimization of the algorithm can be done using available FPGA.

- The parallelable section of the main algorithm can be effectively implemented in the FPGA section in a pipeline structure meanwhile sequential section of the algorithm can be effectively implemented in the processing section. Most of the novel processing system sections in FPSoCs are consisting of multiple cores as well as many types of processing systems such FPU, GPU and real time processing unit. On the other hand, the FPGA section consists of improved DSP modules, high dense block memory such as ultra-memory block and the logic elements and flip flops which operate at higher frequency. These resources can be used for complex applications according to the availability of power and cost [60].
- As discussed above, the time-to-market is much less in CPUs based developments due to the flexibility to use high level languages in the algorithm implementation phase. However, when it comes to the FPGA design process, it gets little complex than the CPUs and in addition to that, it is required to have hardware design experiences for couple of years to develop a well optimized hardware design in FPGA fabric. The capability to synthesize a customized hardware IP using high level language breakthrough this challenge of difficulties in hardware design implementation. This advantage can be used in almost all the FPSoCs available in the market which enables to implement any type of innovative algorithm in FPGA fabric within a less time period and also with less experience in hardware designs. However, the functionality of hardware IPs which are generated using high level synthesis are usually not as optimized as the HDL based hardware functions. But, good performances can be achieved with better understanding of the functionality in optimization required algorithms and the parallelizable behavior of them.

Considering the factors related to the FPSoCs and their suitability for high end IoT platforms, Ultra 96 development board (v1.0) was selected to implement the computer vision algorithm which is a FPSoC based on Xilinx ZYNQ ultrascale MPSoC (Xilinx Zynq UltraScale+ MPSoC ZU3EG SBVA484). The processing unit of this board is based on ARM A53 quadcore processor which operates

at 1.5Ghz. On the other hand, the FPGA consists of 360 DSP modules, 154k combinational logic blocks (CLB), 141k flipflops and can operate at a maximum of 800MHz. The basic hardware architecture and the available resources in Ultra 96 development board is shown in Figure 4.3.

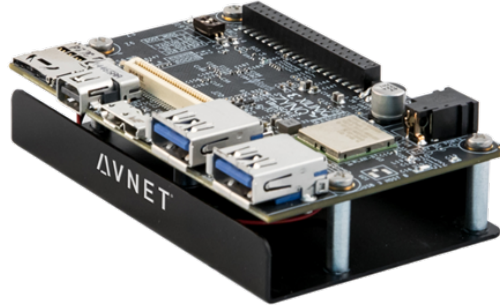


Figure 4.3: Ultra 96 Development board

This development board consists of a display port which is an important feature specially for vision related developments. It also consists of USB3.0 ports which can be used to integrate camera. The UART interfaces can be used to interface with other communication modules which are based on WSN protocols such as 802.15.4.

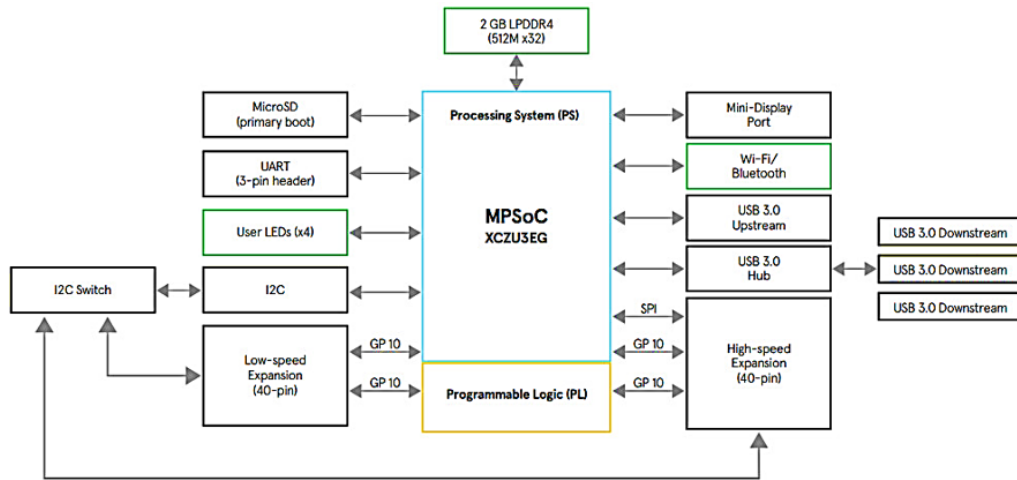


Figure 4.4: Resources of the Ultra 96 board

4.1 Hardware-Software Partitioning and Optimization of the Algorithm

The complete algorithm is implemented in PS section at first. The petalinux has been selected as the operating system over other options such as RTOS and

BareMetal, in order to take advantage of easy configuration of hardware and software components. Xilinx petalinux tools can be used to develop a customized and a suitable petalinux operating system for the given hardware platform. The required library configurations for the Mali GPU such as Gstreamer, FFmpeg, Libav, X11 servers (if a display port is not available), video codec at the kernel configuration stage and the root file system can be configured to include the OpenCV library.

The algorithm is implemented in PS section in a way that it will be performed using only a single thread. The real time performance could not be achieved as expected. The maximum frame rate was 11 fps which means, it takes around 91 milliseconds to completely process one frame.

The modified ViBe algorithm-based foreground extraction, the Bhattacharyya coefficient based weight calculation of the particle, RGB to gray color conversion, file read function to extract the new frame from the video stored in the SD card and connected component analysis with morphological analysis are the main functions in vision algorithm. Therefore, time consumption profiles of them have been considered in order to investigate the bottleneck of the system.

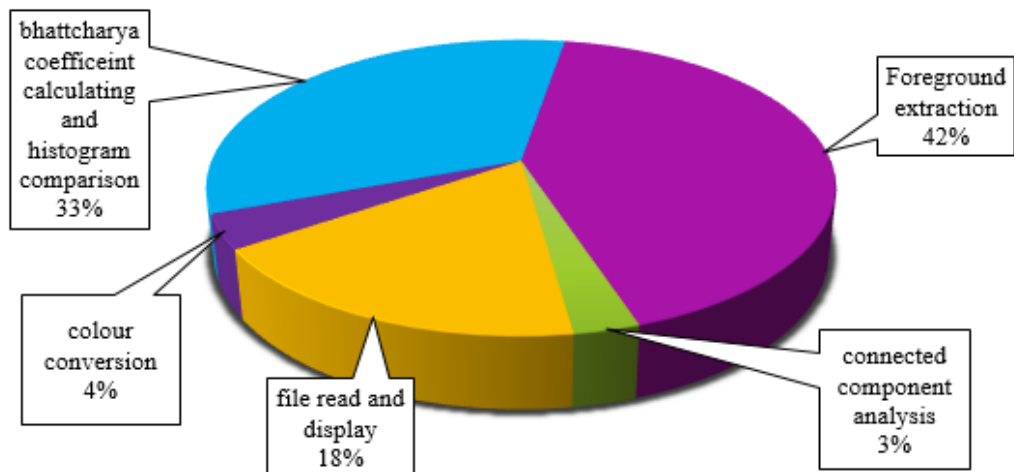


Figure 4.5: Time consumption profile for the execution of one frame.

The Figure 4.5 illustrates the time consumption for each function in the algorithm as a fraction of the total time consumption if we perform this algorithm in single CPU core. The main cause for the higher time consumption for the Bhattachrya coefficient section is, we have used only a single core to evaluate this

200 patches using Bhattacharya coefficient. Based on these results, decision was made to optimize the foreground extraction function and Bhattacharya coefficient calculation for the particle filtering functions using FPGA available in the Ultra 96 board. The custom IPs for the Bhattacharya coefficient calculating functions and the modified ViBe algorithm are generated using Xilinx High Level Synthesis tool associated with the Xilinx Vivado software.

As mentioned in [59], the functions which show more parallelizable behavior perform better in the FPGA hardware. Besides, the data transferring between the PL side and the PS side is the other most prominent aspect for a better performance. The data transferring between PL and PS side is basically based on AXI4 industrial grade bus. Basically, there are three types of AXI interfaces known as AXI-Stream, AXI full memory mapped and AXI-lite as shown in the following Figure 4.6.

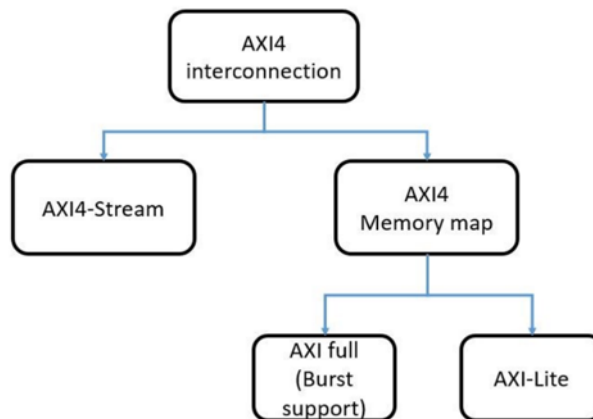


Figure 4.6: Types of AXI4 interfaces.

The AXI memory mapped interfaces are capable of burst transaction between PL and PS sides up to 256 data transferring cycles using one address phase. The AXI-lite is also a memory mapped interface but only single data transaction interface is available per address phase. This is mainly used for controlling the registers and IPs. The AXI Stream does not have an address phase. Hence, the data transaction can be done at a higher speed and in a sequential manner than any other method. If a memory mapped technique has been used, the block memory will be used to create an interface with AXI and the PL side. On the other hand, if the AXI-Stream is used, FIFO will be used to create an

interface [22]. The AXI memory mapped interfaces are frequently used in case of random memory access requirement.

Even though, the AXI-stream provides a high-speed data transmission between PL and PS sides, there are not many functions or algorithms which process address-less incoming byte stream or in other words, sequential access to the data in main memory. However, the modification which was done to the ViBe algorithm enables us to use the AXI-Stream interface. As discussed in section 2, we can store the main model of the background in DDR memory and a fraction of that model can be saved in the block memory which is being used to classify the pixels in current frame and update the model as required. This means, we can just stream the pixel values in the model, pre-generated random numbers and current frame as inputs to the system meanwhile we can get the pixel values of updated model and image which has marked the foreground areas of the current frame. The Bhattacharya coefficient can also be calculated and AXI-Stream interface can be used in the same fashion but it does not require any modifications.

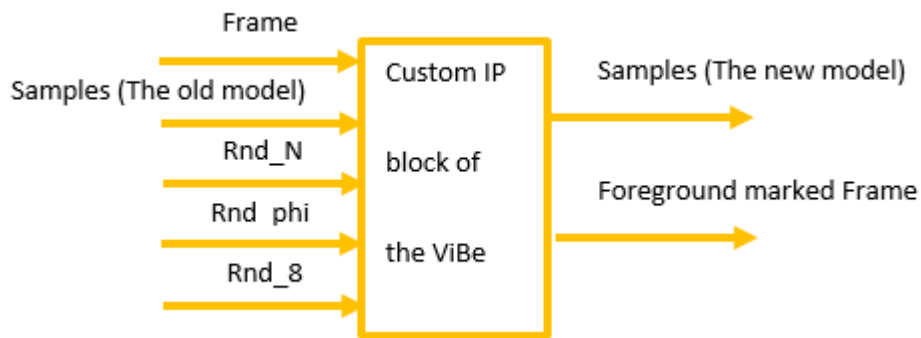


Figure 4.7: Block diagram of ViBe.

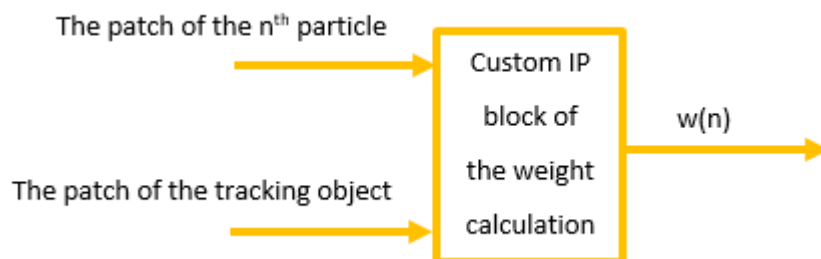


Figure 4.8: Block diagram of the weight calculation using Bhattacharya coefficient .

Figure 4.9 illustrates the general view of integration of the custom hardware IP through AXI-Stream interface and DMA with the processor available in Zynq platform. These IPs are generated with Vivado HLS.

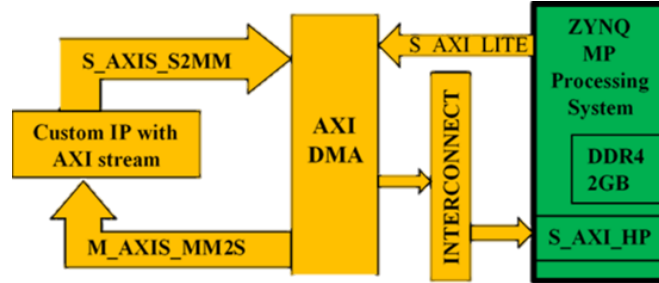


Figure 4.9: Integration custom IP with Zynq .

Then, the final block diagram of the algorithm can be represented as in Figure 4.10 with the balanced processing load between PL and PS side. The hardware functions have been optimized using the “pragma” optimization techniques available in HLS design procedure. Different optimization techniques available in Vivado HLS have been used step-wise as described in Xilinx HLS documentation [61]. The hardware resource utilization and achieved average frame rate for each technique has been used in the design procedure which is tabulated in table 4.2. The performance and the utilization of the hardware has been investigated for different frame sizes and is expressed in Table 4.3. This was evaluated for maximum optimized hardware which has been obtained using pragma techniques as discussed above. This performances and the functionality of the object detection and tracking algorithm is compared in Table 4.4 with currently existing FPSoC hardware implementation.

According to the modified ViBe algorithm, we can implement relevant sample section (*temp_sample* variable in the modified ViBe algorithm) of the ViBe model where multiple access is required to compare the current pixels and update the model. This section can be implemented in the block memory section of the FPGA. Since this block memory is faster than DDR memory, we could achieve higher frame rates as we can see in the Table 4.2. The other most prominent advantage of using FPGA and HPS hybrid approach for this kind of an algorithm is, we can stream the data between main DDR memory and FPGA FIFO without considering the address. This is the major importance of this modified ViBe algorithm when we have pipelined the rest of the algorithm in FPGA.

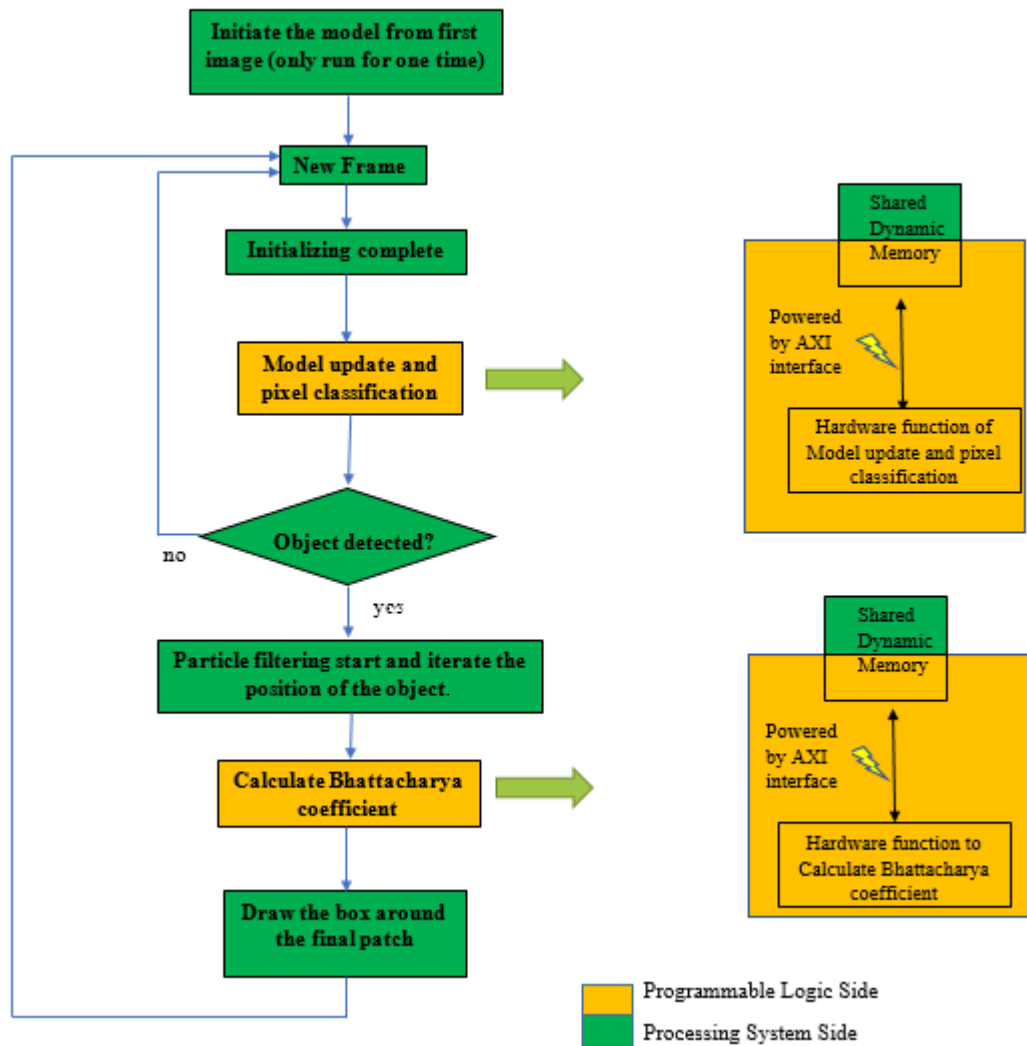


Figure 4.10: Complete flow chart of the algorithm with balanced processing load between PL and PS side.

Table 4.2: Specifications of the SBCs that have been used to compare the performance of the algorithm.

Stage	Avg. Frame Rate	Hardware Utilization			
		LUT	BRAM	FF	DSP
Completely on PS side	11	0	0	0	0
Bhattacharya coefficient in PL side	18	10010	5	13744	12
Implementing temp_sample area in PL	24	34166	64	41387	12
Flatten the inner most loops of ViBe	28	43421	96	52479	12
Single stage Pipeline ($II = 1$)	33	51764	124	67548	12
Unroll the main loops	40	60455	172	83744	16

Table 4.3: Performances and resource utilization for different frame sizes

Frame Size	Avg. Frame Rate	Resource Utilization			
		LUT	BRAM	FF	DSP
299x168	66	58748	94	67486	16
598x336	48	60132	148	78856	16
854x480	40	60455	172	83744	16

Table 4.4: Comparison with other implementation

Ref. no.	Object Detection	Tracking Method	Scale Adaptability	Resolution	Fps	Resources
[62]	Single image generated using thousands of images	Kalman filtering	-	1920x1080	60	Dual core ARM processor with 1GHz+Neon FPU
[63]	Running Gaussian method	Kalman filtering	-	720x480	768 msec per frame	Dual core ARM processor with 1GHz+Neon FPU
[19]	MoG in PL side with modification to the MoG	Kalman filtering	-	1080x768	60	Dual core ARM processor with 1GHz+Neon FPU
[64]	Improved ViBe	-	-	700x450	4.6	Dual core ARM Cortex A9 and Cyclone V FPGA
[65]	Modified ViBe for hardware	Particle filtering	First order dynamic model	854x480	40	ARM processor and FPGA

Chapter 5

RESULTS AND DISCUSSION

In this section, the accuracy of the implemented vision algorithm to extract traffic parameters and the performances in the hardware have been evaluated. In order to evaluate the accuracy of this vision model, a dataset which commonly used to evaluate traffic analysis and widely available has been used.

5.1 Accuracy of the Vision Algorithm

Accuracy of the traffic parameters estimated by the vision algorithm has been evaluated by performing the algorithm for the data available in [35], [66] and [67]. As discussed in the Section 2, the waiting time, average speed and the lane occupancy are the mainly expected traffic parameters from the VSN.

- Average Speed

The average speed of the vehicle has been calculated as the definition explained in (3.2) in the previous section for ten vehicles in total, from different video clips available in [35]. The error of the average speed of the vehicle has been tabulated in table 5.1. As can be seen, there is an average of 4.17% of error that can be seen in every average velocity calculated by the algorithm. There can be two reasons to have an error in the average speed parameter. The first problem exists with the initialization of the tracking process. As explained in section 2, we define the initialization and termination area within the RoI. The algorithm detects the presence of a vehicle based on the number of pixels that have been marked as foreground pixels by the ViBe algorithm. The tracking process starts as soon as there is an area which is bigger than the predefined threshold. This threshold value should be large enough to extract the color feature of the object which is used to track the objects between consecutive frames and distinguishable with other objects existing in the scene.

The second problem can be the background pixels in the tracking template of the vehicle color. As shown in the Figure 5.2, the algorithm makes a boundary

Table 5.1: Accuracy of the average velocity.

Vehicle number	Avg. velocity calculated by the algorithm (km/h)	Avg. velocity calculated manually (km/h)	Error in terms of speed in km/h
1	76.05	78.21	2.76
2	60.15	63.82	5.75
3	57.45	55.14	4.19
4	55.28	60.24	8.23
5	62.01	62.58	0.91
6	69.23	73.65	6.00
7	53.47	54.22	1.38
8	58.07	61.93	6.23
9	62.79	61.88	1.47
10	65.01	68.24	4.73



Figure 5.1: (a) Early tracking initialization (b) Correct tracking initialization.

around the interesting pixels based on the results of the ViBe algorithm. There will be pixels which are related not only to the vehicle but also belonging to the background around the vehicle and the corner of the tracking template since a box shaped template is considered.

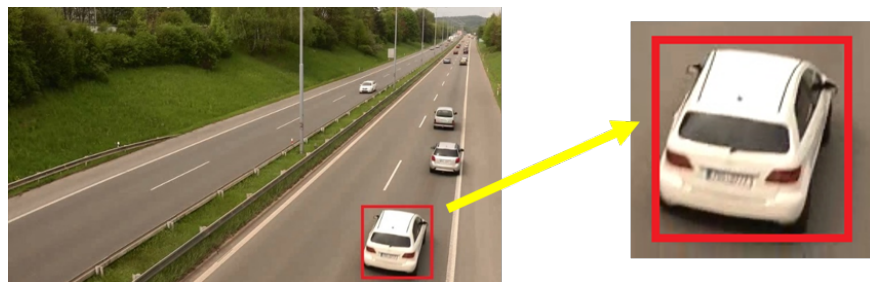


Figure 5.2: Background noise of the patch

This can cause to deviate the tracking process from the actual location of the vehicle since the location of the vehicle is considered to be the middle point of the tracking box as shown in Figure 5.3. Because of this imprecise localization of

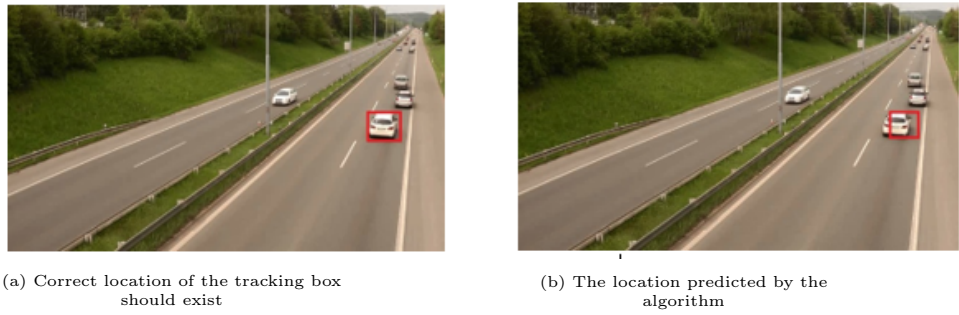


Figure 5.3: Location deviation due to background pixels in the tracking template.

the tracking box, it is possible to terminate the tracking process early or later if middle point of the vehicle crosses the termination line.

- Waiting Time

The waiting time is calculated based on the number of frames which the tracking box has not shifted its middle point with respect to the previous location than a threshold value. The waiting time can be calculated by multiplying the number of frames and the time gap between two frames. The waiting time of three vehicles exist in the data set available in [67] has been calculated.

Table 5.2: The accuracy of the waiting time.

Vehicle number	Waiting time calculated by the algorithm (s)	Waiting time calculated manually (s)	Error of the waiting time (%)
1	35.11	32.57	7.79
2	31.56	33.65	6.21
3	30.25	31.48	3.91

There is an average error of 6.91% in every waiting time which is calculated by the algorithm. This can be mainly caused by the dynamic behavior of the particle filtering algorithm and the lack of particles for a proper estimation of parameters of the object. In the particle filtering algorithm, the noise component, v , will change the existing dimensions and position of the current particles and try to find the best suitable patch and properties of the object in the newest frame. Therefore, due to the noises in object detection as described above and having not enough particles to tackle the properties of the object in current frame can cause change of the position and the size of the tracking box even though, the vehicle is stopped at a certain place such as a traffic light. This is demonstrated in Figure 5.4.



Figure 5.4: Deviation of the tracking box even though the vehicle is stopped.

However, it is also important to mention that the particle filtering algorithm could keep the tracking even though, the vehicle was partially occluded. An example is illustrated in Figure 5.5







Figure 5.5: Partially occluded situation.

- Lane Occupancy

The lane occupancy is calculated as the fraction of the area that vehicles have been occupied from the road. This can be calculated by considering the number of pixels belonging to the vehicles, the area of the ROI and the gradient of the pixels representing the area. This parameter is also calculated for the data available in [35].

Table 5.3: Accuracy of the LO

Data set number	Frame	LO calculated by algorithm (%)	LO calculated manually(%)	Error(%)
1		33.14	36.47	9.13
2		52.35	61.22	14.49
3		21.73	24.36	10.79
4		43.86	48.71	9.96

The pixels that belong to the foreground, in our case, the vehicles are separated by the modified ViBe algorithm. As described in previous sections, although ViBe algorithm performed better than the traditional MoG method, it is possible to have FN and FP in the foreground image which is given by the ViBe as shown in Figure 5.6. This can be somewhat tolerated by the morphological operations but not completely. Therefore, this is the main cause of the 11.09% average error in the lane occupancy reading of the algorithm.

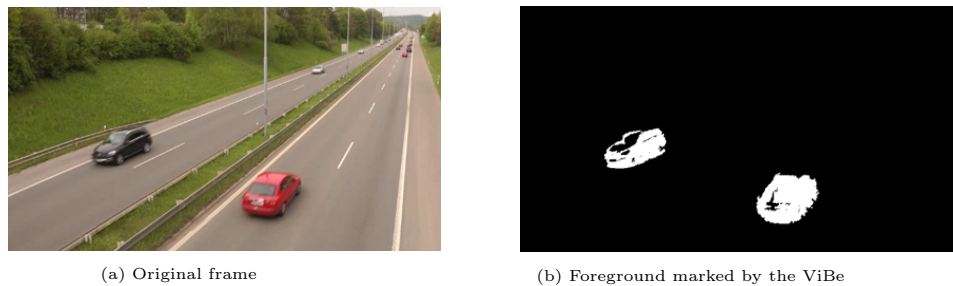





Figure 5.6: Power consumption of the hardware platform.

- Traffic Density

The traffic density is determined as the fraction of number of vehicles present on the road with respect to the area of the lane considered. First, we need to get

the number of vehicles present in the vicinity. This can be achieved by applying connected component analysis to the blobs in the image which come as a result of ViBe algorithm and morphological operations.

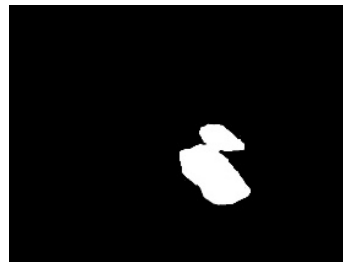
Table 5.4: Accuracy of Traffic Density

Name of the data set	Frame	Traffic Density calculated by algorithm (No. of vehicles/20m) (%)	Traffic Density calculated manually (No. of vehicles/20m) (%)	Error(%)
stmarc		11.11	5.56	5.56
rouen		8.32	8.32	0
sherbrooke		1.06	2.13	1.06

The accuracy of the traffic density strongly depends on the capability of the ViBe algorithm to detect the vehicles individually. If vehicles are too close to each other or partially occluded, the results can be different from the actual scenario in a great fraction. This has been illustrated in Figure 5.7. However, these problems can somewhat be manipulated by changing the camera angle so that we can get the top view of the road as in stmarc and Rouen datasets.



(a) Original frame



(b) Merged blobs

Figure 5.7: The effect of merged blobs.

According to the modified ViBe algorithm, we can implement the relevant sample section (*temp_sample* variable in the modified ViBe algorithm) of the ViBe model where, multiple access is required to compare the current pixels and update the model. This section can be implemented in the block memory section of the FPGA. Since, this block memory is faster than DDR memory, we could achieve higher frame rates as can be seen in Table 4.2. The other most prominent advantage of using FPGA and HPS hybrid approach for this kind of algorithm is, we can stream the data between main DDR memory and FPGA FIFO without considering the address. This is the major importance of this modified ViBe algorithm when we have pipelined the rest of the algorithm in FPGA.

5.2 Power Consumption of the Hardware Platform

The hardware implementation of the algorithm is carried out on the Ultra 96 development board which is considered to be as a high-end IoT device. As described in the hardware implementation section, we could achieve a maximum average frame rate of 40 fps for a data set with 854x480 frame size. The Figure 5.8 shows the static power consumption of the PL and PS sides and Figure 5.9 shows the dynamic power consumption of the PL and PS sides which are estimated by the Vivado software. The estimated total power consumption of the system is 3.435 W.

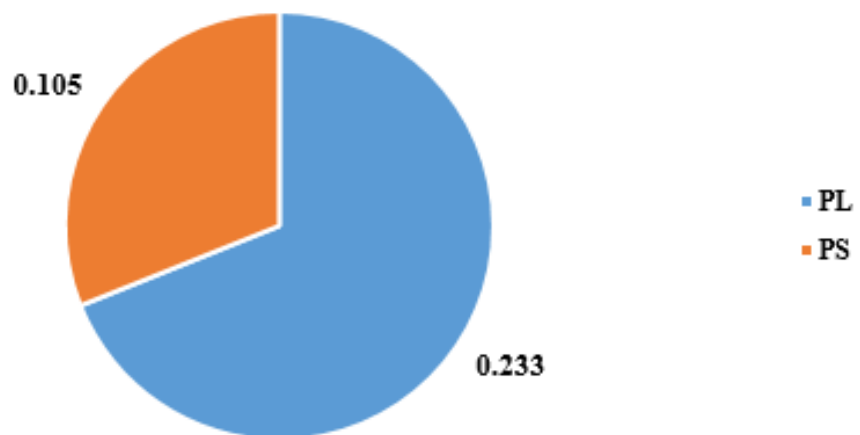


Figure 5.8: Static power consumption.

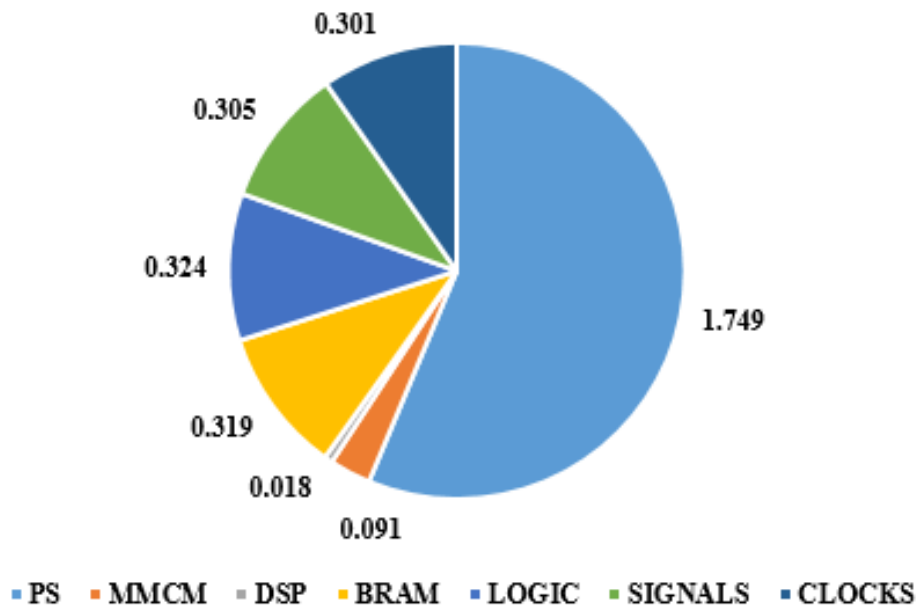


Figure 5.9: Dynamic power consumption.

Chapter 6

CONCLUSION AND FUTURE WORK

This chapter discusses about the major contribution to the research area of VSN based road traffic parameter estimation, key results and suggestions for the future work. The promising results could be obtained for the feasibility of implementing visual information based sensor node to harvest traffic parameters using stationary camera in the FoV.

6.1 Major Contributions

In this thesis, a computer vision algorithm based hardware implementation which can be used in a WSN to estimate traffic parameters has been discussed. As the first step, machine vision model has been proposed to extract the traffic parameters that can be harvested from the FoV. The FPSoC based hardware and software co-design approach has been taken to implement and optimize the vision model in hardware platform. The following contributions have been done based on the identified challenges in the implementation of vision model and challenges still existing in the current state of the art.

1. A less computationally complex modification to the ViBe algorithm is introduced and performance is verified with the original ViBe and the famous MoG background estimation technique.
2. The modified ViBe algorithm is implemented in FPSoC based high-end IoT hardware platform and the capability of using the modified ViBe algorithm in IoT grade devices for object detection is identified.
3. Existing scale adaptable particle filtering algorithm is implemented in the FPSoC and performance and accuracy of the algorithm evaluated.

6.2 Future Works

There are some areas to be improved in the vision algorithm as well as the hardware implementation. Some of the ideas are discussed in the following sections.

6.2.1 Increase the Frame Rate of the System

As discussed in the hardware implementation of the algorithm in Section 4, only the model updating, pixel classification and the weight calculation of the particle filtering algorithm have been implemented in the programmable logic side because of their higher time consumption to perform in the PS side as well as the parallel behavior in the act. Even though, we could achieve a frame rate of 40 fps by transferring these functions to the programmable logic side, it is possible to improve the overall performance little further by implementing some other highly parallel functions such as connected component analysis with morphological analysis and RGB to Gray color conversion functions. This will not only cause the improvement of the functionality of the system, but also reduce the power consumption.

The algorithm is implemented in the hardware using the high level synthesis technique given by the vendor of the chosen hardware platform. However, this method is not the most optimized method to implement hardware functions in the FPGA fabric in terms of hardware utilization as well as the performance. It is often reported that pure HDLs such as Verilog or VHDL based developments outperform the HLS based developments [68]. Therefore, if the algorithm is generalized for a largescale implementation, it is better to implement the complete algorithm in HDLs in order to obtain maximum performance meanwhile preserving the hardware resources.

6.2.2 Improve the accuracy of Vehicle Tracking

As described in Section 3, the vehicle coming into the tracking initialization region is considered to be a square shaped patch. Therefore, it is possible to have pixels which are related to background to be in the extracted patch from the original frame. Therefore, it is possible for the tracking algorithm to be deceived from the actual vehicle which is being tracked by the algorithm. This can somewhat be improved by integrating more features such as edge, texture and corners to the particle algorithm so that there is better description about

the tracking vehicle. In [69], this concept has been proposed and better results have been obtained with respect to algorithms which are based only on the color features.

Besides, the shadows of the stationary objects as well as the shadows created by the vehicles itself existing in the RoI can be a prominent challenge to distinguish the actual description of the color feature of the detected vehicles. The color conversion from RGB to LAB is often used to eliminate this problem [70].

6.2.3 Improve the Accuracy of Detecting the Stopped Vehicles in the Scene

It can be seen that, the vehicles which have come into the scene a long time ago, can disappear from the foreground because the background model keeps updating overtime regardless of the importance of keeping the detected vehicles in the foreground image. This effect can commonly be seen in many conventional background model based foreground-background separation techniques. However, updating the model in the presence of vehicles can be eliminated by considering the motion of the detected vehicle. If a vehicle does not make a move, we can stop updating that region of the model so that the model will not be updated using the pixels in foreground which belongs to the vehicle.

6.2.4 Enhance the Capability to Count the Number of Vehicles in the Scene

The proposed vision model in this thesis strongly depends on the camera angle when it calculates the number of vehicles in the scenery. This is mainly because of the inaccuracy in the background model based object detection method in case of occlusion and false detection created by other moving objects such humans and animals. The convolutional neural network based object detection and classification has shown way better results than the conventional object detection methods. But, the real time performance of this sort of a neural network based algorithm requires high performance hardware due to the large number of floating point operations that are needed to be done. However, recently, a binary number with XOR operation based neural network architecture has been proposed and implemented in FPGA devices and it has been able to reach the similar results by using large number of nodes with respect to the floating point based neural network [71]. This neural network can be performed in the same FPSoC hardware which was selected for this research work and a demonstration can also be found

in [72]. Therefore, BNN network based object detection has a great potential to be implemented in IoT applications.

6.2.5 Evaluate the Usability of this VSN in the Real world

The evaluation of the implemented algorithm in VSN is carried out using the available data sets which are widely used in other traffic analysing vision algorithms. This data set has been taken under constant light condition and using standard cameras which can provide higher quality images. Besides that, traffic condition in this data set is always under a certain level which cannot be true for real situations. Therefore, further developments can be done to the existing system as we can find the limitations when we use it in heavy traffic conditions in the real world.

References

- [1] R. Z. M. Wójcikowski and B. Pankiewicz, “FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network,” *Journal of Signal Processing Systems*, vol. 68, no. 1, pp. 1–18, January 2012.
- [2] Florence, G. H. Yap, and H.-H. Yen, “A survey on sensor coverage and visual data capturing/processing/transmission in wireless visual sensor networks,” *Sensors*, vol. 14, no. 2, pp. 3506–3527, 2014.
- [3] M. Bommers, A. Fazekas, and T. Volkenhoff, “Video based intelligent transportation systems – state of the art and future development,” *6th Transport Research Arena*, vol. 14, pp. 4495–4504, April 2016.
- [4] B. Travly and K. Bicakci, “A survey of visual sensor network platforms,” *Multimedia Tools and Applications*, vol. 60, no. 3, pp. 689–726, July 2014.
- [5] M. Yoshimura, H. Kawai, T. Iyota, and Y. Choi, “FPGA-based image processor for sensor nodes in a sensor network,” *The Open Signal Processing Journal*, vol. 2, pp. 7–13, 2009.
- [6] C. S. Nandyala and H. K. Kim, “From cloud to fog and IoT-based real-time U-healthcare monitoring for smart homes and hospitals,” *Science and Engineering Research Support Society*, vol. 2, no. 2, pp. 187–196, 2016.
- [7] A. Marcus and O. Marques, “An eye on visual sensor networks,” *IEEE Potentials*, vol. 31, no. 2, pp. 38–43, 2012.
- [8] B. Tian, B. T. Morris, M. Tang, Y. Liu, Y. Yao, C. Gou, D. Shen, and S. Tang, “Hierarchical and networked vehicle surveillance in ITS: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 557–580, 2015.

- [9] J. S. Lumentut, F. E. Gunawan, and Diana, "Evaluation of recursive background subtraction algorithms for real-time passenger counting at bus rapid transit system," *Procedia Comput. Sci.*, vol. 59, no. Iccsci, pp. 445–453, 2015.
- [10] M. Wójcikowski, R. Żaglewski, B. Pankiewicz, M. Kłosowski, and S. Szczepański, "Hardware-software implementation of a sensor network for city traffic monitoring using the FPGA- and ASIC-based sensor nodes," *Journal of Signal Processing Systems*, vol. 71, no. 1, 2013.
- [11] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "Hardware-software implementation of vehicle detection and counting using virtual detection lines." Proceedings of 2014 Conference on Design and Architectures for Signal and Image Processing (DASIP), October 2014, pp. 1–8.
- [12] Kryjak, M. Komorkiewicz, and M. Gorgon, "Real-time hardware–software embedded vision system for its smart camera implemented in Zynq SoC," *Journal of Real-Time Image Processing*, vol. 15, no. 1, pp. 123–159, 2016.
- [13] A. Bate. Build a security camera with Raspberry Pi and OpenCV. [Online]. Available: <https://www.raspberrypi.org/blog/raspberry-pi-security-camera-opencv>
- [14] C. C. Paglinawan and A. N. Yumang, "Optimization of vehicle speed calculation on Raspberry Pi using sparse random projection." IEEE HNICEM, 2018.
- [15] S. S. Mohammed, N. M. Tahir, and R. Adnan, "Background modelling and background subtraction performance for object detection." Mallaca City: 6th International Colloquium on Signal Processing and Its Applications (CSPA) IEEE, 2010.
- [16] O. Magazine. (2010) ODROID-XU4. [Online]. Available: <https://magazine.odroid.com/odroid-xu4>
- [17] M. R. A. Rindra Wiska and N. Habibie, "Vehicle traffic monitoring using single camera and embedded systems." Malang, Indonesia: 2016 International Conference on Advanced Computer Science and Information Systems (ICACISIS), Oct 2016.
- [18] J. G. Lee, E. Jung, and W. Shin, "An asymptotic performance/energy analysis and optimization of multi-core architectures," vol. 5408. ICDCN'09, 2008.

- [19] C. Bui, N. Patel, and D. Patel, "A hardware/software co-design approach for real-time object detection and tracking on embedded devices." Southeast-Con, 2018.
- [20] H. A. Rahim, U. U. Sheikh, R. B. Ahmad, and A. S. M. Zain, "Vehicle velocity estimation for traffic surveillance system," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 4, no. 9, pp. 1465–1568, 2010.
- [21] H. Tabkhi, M. Sabbagh, and G. Schirner, "An efficient architecture solution for low-power real-time background subtraction." Proc. 2015 IEEE 26th International Conference on Application-specific Systems, July 2015.
- [22] Xilinx. (2017) AXI reference guide. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
- [23] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," vol. 5747. Proc. SPIE Conf. Med. Imag., 2005, pp. 1965–1976.
- [24] L. Li and F.-Y. Wang, "Approximate vehicle waiting time estimation using adaptive video-based vehicle tracking," vol. 4153. Springer, Berlin, Heidelberg: IWICPAS, 2006.
- [25] M. Chunch and K. R. Rao, "Area occupancy characteristics of heterogeneous traffic," *Transportmetrica*, vol. 2, no. 3, pp. 223–236, January 2006.
- [26] F. L. Hall, *Traffic Flow Theory: State-of-the-Art Report*, 2001.
- [27] B. K. P. Horn and B. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.
- [28] X. Yin, L. Chen, and X. Zhang, "Object detection implementation and optimization on embedded GPU system." BMSB, August 2018.
- [29] M. G. Sarwar Murshed and C. Murphy, "Machine learning at the network edge: A survey." CVPR, June 2019.
- [30] P. T. Adeke, A. E. Zava, and A. A. A., "Spot speed study of vehicular traffic on major highways in makurdi town," *Civil and Environmental Research*, vol. 10, no. 6, pp. 123–131, 2018.

- [31] Y. Xu, J. Dong, Z. B., and D. Xu, "Background modeling methods in video analysis: A review and comparative evaluation," *CAAI Transactions on Intelligence Technology*, vol. 1, pp. 43–60, 2016.
- [32] O. Barnich and M. Van Droogenbroeck, "ViBe: A powerful random technique to estimate the background in video sequences." *Proc. Int. Conf. Acoust. Speech Signal Process.*, 2009, pp. 945–948.
- [33] O. Barnich and M. V. Droogenbroeck, "ViBe: a universal background subtraction algorithm for video sequence," *IEEE Trans. Image Process.*, vol. 20, no. 6, pp. 1709–1724, 2011.
- [34] Telecommunications and I. Laboratory. (2011) Vibe - a powerful technique for background detection and subtraction in video sequences. [Online]. Available: <http://www.telecom.ulg.ac.be/research/vibe/>
- [35] J. Sochor and A. Herout, "Automatic camera calibration for traffic understanding." *Proceedings of BMVC*, 2014, pp. 1–10.
- [36] Al-Hussein and A. El-Shafie, "A survey on hardware implementations of visual object trackers." *CVPR*, November 2017.
- [37] G. R. Bradski, "Computer vision face tracking as a component of a perceptual user interface." *Proc. IEEE Workshop Applications of Computer Vision*, October 1998, pp. 214–219.
- [38] N. M. Atner, "A comparison of mean shift tracking methods." *12th Central European Seminar on Computer Graphics*, 2008, pp. 197–204.
- [39] I. A. Iswanto and B. Li, "Visual object tracking based on mean-shift and particle-Kalman filter," vol. 116. *Procedia Comput. Sci.*, 2017, pp. 587–595.
- [40] A. Miguel and Carreira-Perpinan, "A review of mean-shift algorithms for clustering," *Electrical Engineering and Computer Science, University of California*, pp. 1–28, March 2015.
- [41] J. Shi and C. Tomasi, "Good features to track." *Proc. IEEE Computer Society Con. on CVPR*, 1994, pp. 593–600.

- [42] S. Sinha, J. M. Frahm, P. M., and G. Y., “Feature tracking and matching in video using programmable graphics hardware,” *Mach. Vis. Appl.*, vol. 22, no. 1, pp. 207–217, January 2011.
- [43] W. Fang, Y. Zhang, Y. B., and S. Liu. (2017) FPGA-based ORB feature extraction for real-time visual slam. [Online]. Available: <https://arxiv.org/abs/1710.07312>
- [44] Y. S. Do and Y. J. Jeong, “A new area efficient SURF hardware structure and its application to object tracking.” TENCON IEEE Region 10 Conference Proceedings, 2013.
- [45] S. Yasukawa, H. Okuno, I. K., and Y. T., “Real-time object tracking based on scale-invariant features employing bio-inspired hardware,” *Neural Networks*, vol. 81, pp. 29–38, 2016.
- [46] B. N. Wei, “Developing a smart camera for road traffic surveillance.” MMSP, 2008.
- [47] Alshack. (2010) Template matching. [Online]. Available: <https://aishack.in/tutorials/template-matching/>
- [48] N. Nikolaidis and M. Krinidis, *The essential Guide to Video Processing*. Cambridge University Press, 2009.
- [49] M. Yin, Y. Bo, G. Zhao, and W. Zou, “Adaptive block-fusion multiple feature tracking in a particle filter framework.” Proc. IEEE 3rd Annu. Int. Conf. Cyber Tech. Autom. Control Intell. Syst. (CYBER), May 2013, pp. 400–404.
- [50] K. Nummiaro, E. Koller-Meier, and G. L. J. V., “An adaptive color based particle filter,” *Image Vision Comput.*, vol. 21, no. 1, pp. 99–110, 2003.
- [51] A. D. Bagdanov and A. D. Bimbo, “Improving the robustness of particle filter-based visual trackers using online parameter adaptation.” Conference on Advanced Video and Signal Based Surveillance, 2007, pp. 218–224.
- [52] K. Copsey, “Tutorial on particle filters.” Aston University, Birmingham: Pattern and Information, NCAF January Meeting, July.
- [53] D. Dongsheng, J. Zengru, and L. Chengyuan, “Object tracking algorithm based on particle filter with color and texture feature.” 35th Chinese Control Conference, July 2016.

- [54] P. A. Marin and J. L. Navarro, "Comparative study of histogram distance measures for re identification." CVPR, November 2016.
- [55] S. Direct. Gaussian error function. [Online]. Available: <https://www.sciencedirect.com/topics/engineering/gaussian-error-function>
- [56] M. O. Ojo, S. Giordano, P. G., and I. N. Seitanidis, "A review of low-end middle-end and high-end IoT devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.
- [57] K. Heyse, "Improving the gain and reducing the overhead of dynamic circuit specialisation and micro reconfiguration," Ph.D. dissertation, Belgium, 2015.
- [58] J. G. Lee and E. G. Jung, "An asymptotic performance/energy analysis and optimization of multi core architectures." ICDCN, January 2009, pp. 85–90.
- [59] R. F. Molanes, K. Amarasinghe, R.-A. J., and M. Manic, "Deep learning and reconfigurable platforms in the internet of things:challenges and opportunities in algorithms and hardware," *IEEE Industrial Electronics Magazine*, vol. 12, no. 2, pp. 36–49, 2018.
- [60] Xilinx. (2017) Ultrascale+ MPSoC product tables and product selection guide. [Online]. Available: <https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf>Zynq
- [61] U. Xilinx. (2017) Vivado HLS optimization methodology guide. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1270-vivado-hls-opt-methodology-guide.pdf
- [62] M. Qasaimeh, K. Denolfy, and J. Loy, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels." The 15th IEEE Int. Conf. on Embedded Software and Systems, 2019.
- [63] W. Liu, H. Chen, and L. Ma, "Moving object detection and tracking based on ZYNQ FPGA and ARM SoC." IET Int. Radar Conf., October 2015.
- [64] A. Chen, R. Gupta, A. Borzenko, K. Wang, and M. . Biglari-Abhari, "Accelerating SuperBE with hardware/software co-design," *Journal of Imaging*, vol. 4, no. 10, p. 122, October 2018.

- [65] Y. M. Wijesinghe, J. G. Samarawickrama, and D. Dias, “Hardware and software co-design for modified ViBe algorithm based object detection and particle filtering based object tracking,” Dec 2019.
- [66] J. Sochor, R. Juránek, Špaňhel J., L. Maršík, A. Šíroky, A. Herout, and P. Zemčík, “Comprehensive data set for automatic single camera visual speed measurement,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 1633–1643, May 2019.
- [67] J. P. Jodoin, G. A. Bilodeau, and N. Saunier, “Urban tracker: Multiple object tracking in urban mixed traffic.” Steamboat Springs, Colorado, USA: IEEE Winter conference on Applications of Computer Vision (WACV14), March 2014, pp. 24–26.
- [68] S. Skalicky, C. Wood, M. Łukowiak, and M. Ryan, “High level synthesis: Where are we? a case study on matrix multiplication.” Cancun, Mexico: ReConFig. IEEE, 2013.
- [69] R. Huan, S. Bao, W. C., and Y. Pan, “Anti-occlusion particle filter object-tracking method based on feature fusion,” *IET Image Processing*, pp. 1519–1531, March 2018.
- [70] S. Murali and V. K. Govindan, “Shadow detection and removal from a single image using LAB color space,” *cybernetics and information technologies*, vol. 13, no. 1, 2013.
- [71] Y. Zhou, S. Redkar, and X. Huang, “Deep learning binary neural network on an FPGA.” IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), August 2017, pp. 281–284.
- [72] Charbax. Xilinx Ultra96, FPGA 96boards development board. [Online]. Available: <https://www.youtube.com/watch?v=6NSLcofHlmM&t=212s>
- [73] F. G. H. Yap and H. Yen, “A survey on sensor coverage and visual data capturing/processing/transmission in wireless visual sensor networks,” *Journal of sensors*, vol. 2, no. 14, pp. 3506–3527, February 2014.

Appendices

Appendix A

The ViBe Algorithm

A.0.1 The Update and Classification of Modified ViBe

```
void ViBE_update_classification(
uint8_t foreground_image [WIDTH*HEIGHT] ,
uint8_t new_frame [WIDTH*HEIGHT] ,
uint8_t sample_array [N*WIDTH*HEIGHT] ,
int rd_8 [rndSize] , int rd_phi [rndSize] , int rd_N [rndSize] )
{
uint8_t rde;
static int r_8 [256];
    static int r_phi [256];
    static int r_N [256];
for (int i=0; i < 256; i++){
    r_8 [ i ] = rd_8 [ i ];
    r_N [ i ] = rd_N [ i ];
    r_phi [ i ] = rd_phi [ i ];
}
uint8_t temp_sample [N] [3] [WIDTH];
int back_fore_diff;
for (int i=0; i < HEIGHT ; i++)
{
    for (int j=0; j < WIDTH ; j++)
    {
        for (int n=0; n < N; n++){
            temp_sample [n] [2] [ j ] = temp_sample [n] [1] [ j ];
            temp_sample [n] [1] [ j ] = temp_sample [n] [0] [ j ];
            temp_sample [n] [0] [ j ] =
            sample_array [ ( n*WIDTH*HEIGHT ) + ( i*WIDTH + j ) ];
            if ( i > 2 ) {
```

```

sample_array [(n*WIDTH*HEIGHT)+((i-2)*WIDTH)+j]=
    temp_sample[n][2][j];
}
}
int count =0,index=0;
while((count<noMin) && (index<N))
{
    uint8_t pixel_value=new_frame[WIDTH*i+j];
    for(int bi=0;bi<3;bi++){
        for(int bj=0;bj<3;bj++){
            if((bi)>(HEIGHT)){
                bi--;
            }
            if((bj)>(WIDTH)){
                bj--;
            }
            back_fore_diff=
                temp_sample[index][bi][j+bj-1]-pixel_value;
            if(abs(back_fore_diff)<R){
                break;
            }
            if(abs(back_fore_diff)<R){
break;
            }
        }
    }
    if(back_fore_diff<=R && back_fore_diff>=-R)
    {
        count++;
    }

    index++;
}
if(count>=noMin)
{
    foreground_image[WIDTH*i+j]=0;
}

```

```

int rand= r_phi[rde];
rde=rde+1;
if(rand==0)
{
rand= r_N[rde];
rde=rde+1;
temp_sample[rand][1][j]=
new_frame[WIDTH*i+j];
}
rand= r_phi[rde];
rde=rde+1;
int nx_off_three=1;
if(rand==0)
{
int n_y=j;
int cases= r_8[rde];
rde=rde+1;
switch(cases)
    {
        case 0:
            nx_off_three=2;
            n_y--;
            break;
        case 1:
            nx_off_three=2;
            n_y;
            break;
        case 2:
            nx_off_three=2;
            n_y++;
            break;
        case 3:
            nx_off_three=0;
            break;
        case 4:
            nx_off_three=0;

```

