Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica

Designing an Embedded Traffic Dynamics Meter: A case study

para optar por el título de
Ingeniero en Electrónica

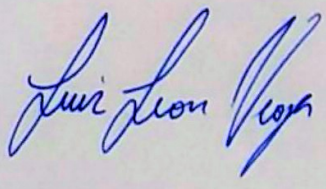
con el grado académico de
Licenciatura

Luis Gerardo León Vega

22 de noviembre de 2018

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.



Luis Gerardo León Vega

Cartago, 22 de noviembre de 2018

Céd.: 7 0233 0194

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado *Designing an Embedded Traffic Dynamics Meter: A case study*, realizado por el señor Luis Gerardo León Vega y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Ing. Marvin Hernández Cisneros
Profesor lector

Ing. Miguel Ángel Hernández Rivera
Profesor lector

Ing. José Miguel Barboza Retana
Profesor asesor

Cartago, 22 de noviembre del 2018

to my dear family, professors and friends, who have helped and supported all this time during my studies.

Acknowledgments

The result of this research has been successfully completed thanks to the support of my mentors: MSc. Jorge Castro-Godínez and MSc. José Miguel Barboza Retana.

Jorge has contributed with several pieces of advice in order to enhance this project and thesis writing. He gave his *approximate grayscale* algorithm which allowed to do a part of the research so as to accelerate image grayscaling process. Also, he suggested using Xilinx SDSoC as a hardware prototyping tool to create and deploy hardware accelerators from C++ code. One of the key contributions given by Jorge was his support to go to Karlsruhe, Germany; especially, with invitation letters, some documentation, and pieces of advice about how to live in Karlsruhe surroundings. Nevertheless, this thesis was read and checked by him, in order to improve thesis writing, bringing a high-quality job. Many thanks to him.

José has contributed with project mentoring, giving pieces of advice about thesis writing, and some tricks to take into consideration during project development. He also contributed with thesis checking and revision, making this work better.

Moreover, a special acknowledgement to Instituto Tecnológico de Costa Rica, which supports economically with accommodation and flight tickets through their student exchange and extension policy. Also, to Intel Costa Rica and their scholarship program, which helps with the accommodation and living expenses.

Finally, a special gratitude to my parents and friends, which supported me emotionally during all this final project.

Luis Gerardo León Vega

Cartago. 22 de noviembre de 2018.

Abstract

Costa Rica is actually facing a severe problem with people transportation caused by traffic congestions, which degrade people life quality due to the time invested during daily commutation, and the stress during the driving. One of the reasons is the incapability of automatic traffic signals to respond to the change in traffic dynamics during the day.

In order to reduce the impact of automatic signalling on traffic congestions, this work proposes to quantify traffic dynamics using average speed, traffic flow, traffic density, and sensor occupancy using Computer Vision. However, there are constraints such as Internet connection unavailability to upload data to the Cloud, and absence of enough energy to power a high-performance computer up in some places. Therefore, the challenge of developing this meter implies to implement the system on a low-power platform.

The application is optimised using Approximate Computing paradigm to tackle this problem, after noticing that it is possible to enhance runtime and power consumption sacrificing results accuracy. After applying approximations, it is possible to improve the application runtime up to 2.55x introducing an error of 11.5% on output results.

Keywords: Computer Vision, Approximate computing, Traffic control, Internet of Things, System performance.

Resumen

En la actualidad, Costa Rica enfrenta un problema severo en el transporte de personas que causa congestiones en el tráfico vehicular y que degrada la calidad de vida de su población debido al tiempo que se invierte durante su traslado de un lugar a otro y el estrés durante la conducción. Una de las razones de este problema es la señalización vial automática, que no responde a la fluctuación de la dinámica del tráfico en el tiempo.

Para reducir el impacto del problema asociado con la señalización automática sobre el congestionamiento vial, esta investigación propone cuantificar la dinámica del tráfico vehicular midiendo la velocidad promedio, el flujo vehicular, la densidad y el tiempo de ocupación promedio de un vehículo bajo el sensor, usando visión por computador. Sin embargo, existen algunas limitantes para llevar a cabo esta tarea, como el acceso a una conexión a Internet estable para subir datos a la nube, la ausencia de suficiente energía en lugares aislados para alimentar equipos de alto rendimiento, que son necesarios para ejecutar algoritmos complejos de análisis de imágenes. Por lo tanto, el reto de este proyecto implica implementar una aplicación de software en un sistema de bajo consumo, pero que sea capaz de ejecutar todos los algoritmos complejos para la medición del tráfico vehicular.

Para afrontar el problema, la aplicación es optimizada mediante el paradigma de la Computación Aproximada, aprovechando la naturaleza inexacta de los algoritmos utilizados en el proyecto y la posibilidad de mejorar el tiempo de ejecución y el consumo de energía sacrificando precisión en los resultados. Después de aplicar aproximaciones, ha sido posible mejorar la aplicación reduciendo en hasta 2.55 veces el tiempo de ejecución al mismo tiempo en que se introdujo un error de hasta 11.5% en los resultados de la cuantificación de la dinámica del tráfico vehicular.

Palabras clave: Visión por computador, Computación aproximada, Control de tráfico, Internet de las cosas, rendimiento de sistemas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Costa Rica, there are problems related with people transportation located in the *Gran Área Metropolitana* (GAM), caused by the high habitational granularity (houses are highly dispersed around work centres), causing people life-quality degradation due to stress and the time invested during commutation. According to the research done by *Programa Estado de la Nación* (PEN) in 2014, 5% of the population moved from San José from 2000 to 2011. At the same time, the population of San José surroundings has increased by 15% [1]. This phenomenon has caused that the traffic flow from the periphery to San José increased drastically, making higher the traffic density (quantity of vehicles per unit of length) and worsening the bottlenecks on GAM roads. In 2014, around of 50.000 -98.000 vehicles were counted commuting amongst Alajuela, San José and Cartago, whose cities are overcrowded due to the job concentration.

The authorities have been sending traffic police personnel to sort the traffic [2] to mitigate traffic jams. So the number of police personnel available required for this task is increasing and the demand for them are not being adjusted properly, limiting road surveillance. One of the reasons for dedicating traffic police personnel to optimise the traffic is because of the incapability of automatic signalling to adapt by themselves to the changes of the traffic dynamics. For that reason, solving this problem requires a tool to free the personnel, actually dedicated to organising the traffic. This tool must measure and quantify traffic dynamics to optimise the GAM traffic flow, modifying the parameters of automatic traffic signs (such as traffic lights, and lane signs), enabling exclusive and reversible lanes. Thus, the project is limited to acquiring the metrics needed for optimising the traffic flow.

An option to acquire the data required for the tool is to use road cameras, available in Costa Rica for research and traffic information. This technique of obtaining data is more affordable than other alternatives, such as installing vehicle counters, inductive sensors, and others.

Hence, this tool also involves the challenge of implementing Artificial Intelligence (AI) and Computer Vision (CV) algorithms on constrained platforms, because of unavailability of Internet access and energy supply in some places, which make it not suitable to use high-

computing hardware. For that reason, the tool has to be implemented on low-power hardware and process all the information in-node, making it autonomous from the Cloud and encouraging to adopt Fog Computing or Edge Computing paradigms [3], uploading information ready to analyse instead sending preprocessed data.

Moreover, the application presents a trade-off related to energy autonomy and computing capacity [4], due to CV and AI algorithms are high-computation consumers, and the measurements should ideally be according to the camera frame rate. To execute the application online, CV and AI algorithms have to be hardware-accelerated

Approximate computing paradigm helps to deal with this challenge, which takes advantage of results precision reduction to speed up and decrease the energy consumption required to perform a task. This fact allows noticing that approximate computing is only applicable to inexact applications. Besides, it makes possible to integrate inexact complex algorithms to IoT devices, which have constrained resources.

This thesis presents the work of designing a meter to quantify the traffic dynamics, especially, for determining *average speed*, *traffic flow*, *traffic density*, and *sensor occupancy*. Those parameters are required in some popular mathematical models which describe traffic dynamics and open the opportunity to analyse a way to optimise traffic flow and reduce congestions. Besides, this work shows the different improvements done for increasing the meter throughput, during its development, based on Approximate computing techniques and multi-threading, with the aim of making the tool more suitable to implement on low-power hardware, facing the challenges presented before.

1.1 Contribution

In addition to proposing the design of a traffic dynamics measurement system, the project aims to contribute in the Approximate Computing field with a new case study, which combines image processing and analysis, and approximate computing.

The authors in [5] and [6] mention that there are promising results that can guide to improving systems performance making the most of the Approximate Computing paradigm. However, more research shall be done to get influential contributions to make possible the application of Approximate Computing (AC) in emerging systems and enhancing the existent devices.

At the end of the project, the main result will be an Approximate Computing application, based on an exact software baseline, improved in throughput by using AC techniques, whose errors will be under the tolerance ranges, and keep output results quality.

1.2 Document structure

This document starts with Background and Related Work, which presents a research of some techniques used for Traffic Dynamics quantification, analysing the most used variables for traffic analysis and modelling, followed by CV techniques for data acquisition, processing and analysis, especially for object detection and tracking. This chapter concludes with Approximate Computing techniques that are useful for this application, describing each of them and their impact on the application performance.

The second chapter is Traffic dynamics meter. It presents all the requirements that have to be met during the meter design, possible challenges, and the selection of the different techniques utilised for the tool development. At the end of this chapter, it is presented a comparison between running the meter on a personal computer (PC) and performing it on an embedded platform.

Optimisation is the third chapter and it takes the results presented in the second chapter conclusion and analyse the problematical modules in order to examine them for optimisations. Then, a list of optimisations is presented showing their impact on the tool performance compared to the baseline developed in Traffic dynamics meter. Optimisation concludes with the global impact after joining all the optimisations on the output results and the application throughput.

The last chapter is Conclusions. It shows the final project observations and mentions the future opportunities to continue working in the traffic dynamics optimisation field.

Chapter 2

Background and Related Work

2.1 Traffic Dynamics Measurement (TDM)

There are efforts to improve people transportation around the world, in order to reduce people stress and make their transportation more comfortable. The problem of traffic jams and congestions are not only found in Costa Rica. Multiple countries have these problems, due to population growth during the last decades and the increase of acquisition power, giving the opportunity to many people to have their own vehicle [7] [8].

The traffic dynamics measurement is studied inside the area of Intelligent Transportation Systems (ITS), which studies widely how to create optimised and advanced autonomous cars, enhancing automatic traffic controls, and how to use the big data analysis to improve the people transportation. In [9], a survey exposes the last tendencies of the ITS using big data.

2.1.1 Big data approach

[9] shows a way to apply Big Data using a multi-layer method depicted in Figure 2.1, summarised in Data Collection Layer, Data Analytics Layer and Application Layer. The first layer is about how the data is collected in order to apply data science on it and compute all the parameters in the third layer. The author proposes a multi-source data collecting approach from multiple sensors in different elements in the road, making it possible to capture data from the road, cars, and personal gadgets. Multiple sources allow the data analysis stage to strengthen its predictions and computations, reducing its vulnerability to noise, and filtering relevant data to the third stage. Basically, the connection between the first and the second layer is made by Internet protocols, where all the computational power relies on a cloud computing grid.

In the third stage, all the data has been analysed and allows to make decisions such as public transportation, traffic flow predictions, and Signal control, being the last one the most relevant for this research. However, the final results could also allow public transport

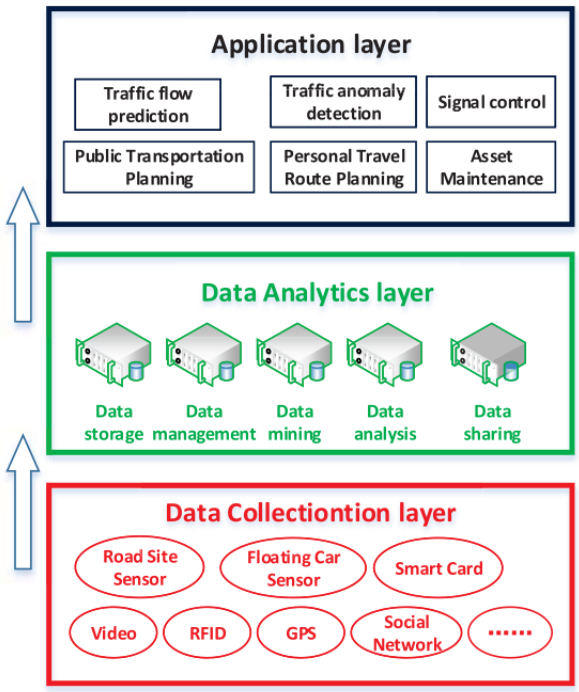


Figure 2.1: Architecture of conducing Big Data analytics in ITS. Reprinted from [9]

optimisation in the future.

Zheng et al. in [10], proposed a non-traditional approach using big data and social data. The proposed method uses mobile phone data, social network posts and transactions in order to collect enough data to analyse using natural language algorithms to extract relevant information about location and events at the location detected, almost at the same time as they are taking place. A number the actual smartphone applications, like Uber [11] and Waze [12], use collective data from people in order to make decisions. For example, they predict which route is faster based on the trip time detected using the Global Positioning System (GPS) data from the phone.

Another strategy cited in [10] and studied [13] in 2018, is the Status Update Message (SUM). Using user post messages on Twitter, it is possible to extract relevant information about traffic making the most of natural language algorithms and cloud computing as shown in Figure 2.2.

The approach presented by [13] is applicable if there are a wide collection of SUM data which contains important information. However, combining other data, such as smartphone GPS and transactions, will increase the range of usable data for computing and mining. Also, using big data with social data work in a similar structure to the stated by [9], and illustrated in Figure 2.1.

Sensors are likely to be used in traffic flow detection, measuring vehicle speeds, vehicle density, traffic flows and trip times. In the case of video sensors, they are also used to identify vehicles for monitoring drivers behaviour, being the cheapest method of collecting data in this category.

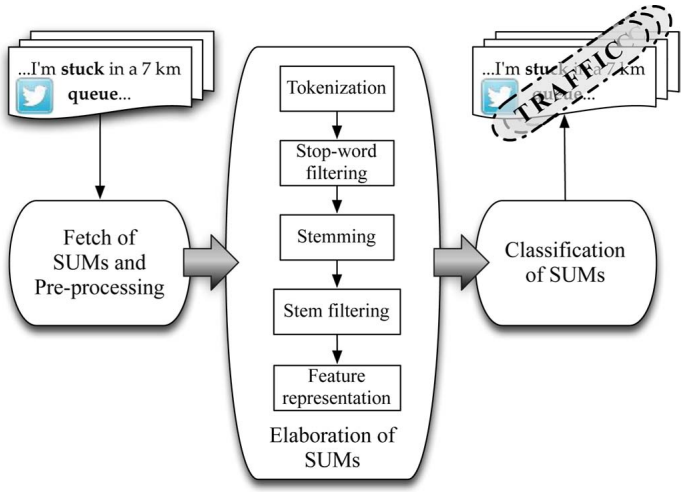


Figure 2.2: Collecting traffic information from SUM. Reprinted from [13]

Furthermore, the author quotes some advanced technologies used for roadside data acquisition, such as ultrasonic and acoustic sensors, magnetometers, light detection and ranging (LIDAR), and video processing and analysis.

In [8], the authors explain how to improve the traffic management using video processing. The system proposed takes into account the existence of Closed Circuit Television (CCTV) camera system on the road to measure the traffic density at an intersection. Furthermore, the system takes into consideration the possibility of having an emergency vehicle, giving it passing priority over the rest of the vehicles, even when there is a high density in one of the roads.

The image processing method used by [8] is basically summarised into a grayscale conversion, and edge detection to make inferences. Using the pixels in black (made by edges), applies a formula to estimate a naive density. Also, they led as future work the management system. Hence, the proposal of [9] compared to [8] is more advanced due to the implementation of Big Data analysis.

Besides, [7] analyses the transportation in India focused on traffic accidents and how to mitigate them using technology. Their solution integrated surveillance, metering the vehicles in case of speeding, lane changing and overtaking as the main cause of traffic collisions. They propose a feedback system to the driver, alerting in case of speeding when it is possible to make a safe overtaking, and lane changing.

For speeding, the author proposed using surveillance cameras to estimate vehicle speeds. This camera, as an author recommendation, should be infrared to enhance performance at night hours.

2.1.2 Out of roadside sensors approach

Other methods of data acquisition for big data analysis are mentioned in [9], such as wide area analysis and floating vehicle data. In the case of wide area analysis, it helps to collect data about traffic tendencies in route planning using photogrammetric processing, sound and video recording and space-based radar. For floating vehicle data, it allows to identify vehicles and acquire precise data of trip variables like car speeds.

2.1.3 Traffic dynamics quantification

In order to study the traffic dynamics for ITS applications properly, it is important to define a set of variables. There are several models which describe the traffic dynamics, such as Speed-flow and Speed-density models [14] and some of them are thought to optimise the traffic flow. Hall in [14] proposes a series of traffic models, whose variables are the most used for traffic modelling.

According to [14] and [9], these variables are: vehicle speeds, traffic flow, vehicle density and trip time (in [14] is called *occupancy*, defined as the time when the sensor can track a vehicle). Some of the formulas to calculate these variables, according to [14] are:

- Vehicle speeds

  There are two ways to calculate the vehicle speed, and both give different results. The first is an average time-based formula:

  $$\bar{u_s} = \frac{D}{\frac{1}{N}\sum_{i=1}^{N} t_i}$$

  Where D is the measurement distance, N is the number of cars sampled and t_i is the time taken by the sampled cars to reach the distance D.

  The second way is an average speed-based, given by:

  $$\bar{u_t} = \frac{1}{N}\sum_{i=1}^{N} u_i$$

  Where u_i is the speed of the vehicle i, given by:

  $$u_i = \frac{dx}{dt} = \frac{x_2 - x_1}{t_2 - t_1}$$

  Where x_i is the vehicle position at time t_i.

  The selection of the speed is likely to the application.

- Flow rates

  The flow rate quantifies how many vehicles are crossing an imaginary line (or limit) per unit of time. It is given by:

  $$q = \frac{N}{T}$$

  Where T is the sampling time.

- Density

  It is given by the number of vehicles per unit of length:

  $$\lambda = \frac{N}{L}$$

  Where L is arbitrarily determined according to the sensor capability.

- Occupancy

  It is the average time which a vehicle lasts to get out of the sensor observation area since it entered. It can be calculated using the following equation:

  $$o = \frac{\sum_{i=1}^{N} \frac{(L_i * d)}{u_i}}{T}$$

  Where L_i gives the length of the car.

The variables explained previously fulfill the requirements of the traffic models quoted in [14], and can cope with a big data collection in order to make inferences. Furthermore, there are advanced algorithms such as *Enhanced Particle Swarm Optimisation* [15], *Closed-Loop Control Theory* [16], *Adaptative algorithms* [17] and *Support Vector Machine-based algorithms* [18], which need those variables to work as is due.

Some of them imply the use of exact methods, especially the proposed by [16] and [18]; however, [15] proposes a probabilistic method to optimise the traffic control, alleviating the congestion at a road intersection. It is possible to model the traffic flow with Monte Carlo techniques, that could be a feasible approach, introduced by Yin in [19].

2.1.4 Open challenges

[9] summarises, based on their analysis, some open challenges for big data and ITS which are:

- Data collection: getting accurate data with advanced techniques will ease data analysis process, obtaining better results. Also, the ways to acquire data should be cleverly invested (reducing costs). Some technologies such as IoT could ease data

transference between the data acquisition and processing layers.

- Data processing: the computational power to process and mine all the data acquired is very expensive and should be mitigated by a kind of preprocessing. This will allow reducing data transmission timings.

- Data opening: open data sources will ease the development of new technologies and techniques in order to enhance transportation. Also, it will lead to better solutions in ITS.

As part of this project, some of the challenges in the area of data collection and data processing will be faced during this research, allowing to reduce the amount of computational power to get important data using image processing, and the available resources in Costa Rica.

2.2 Computer Vision

According to the aim of measuring traffic dynamics for further applications using CV, a brief study of the actual technologies used in similar applications is a must, in order to oversee which techniques are the most popular. Also, this allows developing an application with the most recent algorithms.

One of the closest applications to vehicle detection and tracking is *video surveillance*. Several techniques have been developed in order to detect objects, analyse behaviours and quantify some variables, such as how many people enter to a room, determine if a person is crossing a forbidden barrier, etc.

The intersection between video surveillance and this project is given by the use of cameras in order to detect and analyse the vehicles passing on a predetermined road. Also, the techniques that can be applied to this project are likely to be used in video surveillance, especially in detection and behaviour analysis.

2.2.1 Generalities

As it was mentioned before, one of the applications of Computer Vision is on Video Surveillance. There are two kinds of Video Surveillance, according to Garima Mathur and Mahesh Bundele in [20]: The conventional video surveillance system, which only records video and a human has to analyse unusual situations, and the Inteliggent Video Surveillance (IVS), which has been endowed with intelligence to analyse data and feed some other systems for further actions.

Basically, inside this last area, it is possible to find algorithms which are likely to be used to object detection, classification and tracking, as it is exposed by [20]. Relating this with traffic analysis, those algorithms can be widely used to detect each vehicle and analyse it using object tracking.

Getting inside of these algorithms, the object detection is a technique which allows detecting a predefined object in a picture based on its shape, colours or features. According to Reinhard in [21], object detection looks for classifying objects in a bounding box, often named Region of Interest (ROI), either in *it is the object* or *it is not the object*. However, there are advanced techniques which deal with more kinds of classifications according to a series of features.

Similarly, the object tracking is a process which generally follows to the object detection, and it is applied to videos or a sequence of pictures. When the object is found, it is set as a target and it will be followed in the consequent frames until it is not in the video anymore. In [21], the tracking characteristics are published, which follows a cluster of features in a set of pictures, and locates them in each picture. The principle is the same as for object detection, where a bunch of features are established, and then followed, returning the location in the picture coordinate system or, furthermore, the physical coordinate system, if there is a 3D scenario available.

In the next sections, these algorithms will be analysed in some case studies.

2.2.2 Case studies

The typical case study seen in Computer Vision is the face recognition and tracking. Reinhard in [21], analyse some of the Computer Vision algorithms which have been developed until 2014 related to this case. However, it is applicable to other applications such as pedestrian detection, car detection, etc. Some of the techniques shown by Reinhard, applied to face recognition are the Histogram of oriented Gradients (HoG), and Haar features (shown in Figure 2.3). Both techniques are likely to be optimised with Adaptative Boosting (AdaBoost).

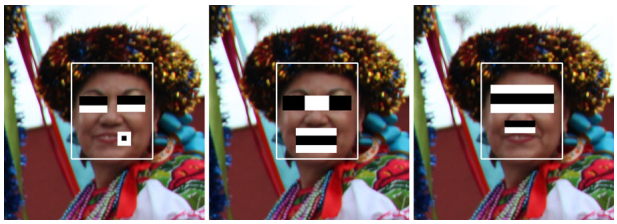


Figure 2.3: Face detection using Haar features. Reprinted from [21]

Mathur also introduces the case study of video surveillance applied to cars and people, focused on object detection and tracking based on background suppresion, and background differencing, respectively. Also, a object detection based on neural networks are mentioned in [20]. For this case, the intrusion detection, behaviour detection and intruder

tracker allow to take advantage of the CV field, where several algorithms can be used, but some of them are discarded due to either this specific application of people and car detection properties or technical weaknesses.

Furthermore, there are projects related to quantification using CV applications, which involve cameras as the main source of data. Forest et al in [22], mentions some of the projects developed and implemented in Italy for metro lines, in order to monitor people behaviours, especially for seeking people who are crossing the safety barriers (yellow lines).

Some of the relevant projects discussed in [22] are Parallel and real-time Advanced Surveillance System With Operator assistance for Revealing Dangerous Situations (PASSWORDS), Video Sensor Object Request Broker Open Architecture for Distributed Services (VISOR BASE) and Visual Inspection and Evaluation of Wide-area Scenes (VIEWS).

All these projects have taken advantage of the cameras because they are low-cost compared to other solutions which integrate different sort of sensors to strengthen them. Hence, the focus of using cameras, based on [22], is accurate due to it is a cheap solution, and the data given by the cameras can be used for other purposes, such as video recording, analysis and so on. Also, their installation is easier compared to other more accurate sensors, which can imply to change some parts of the scenario to analyse. In few words, cameras are a multi-purpose solution suitable in most of the locations without altering the measurement environment.

About the techniques used by these projects, in the case of VIEWS, based on the nature of the problem it can be assumed that they used object detection and classification to accomplish their goal. PASSWORDS uses CV for measuring flow and calculating estimations, very similar to this project aim. Moreover, it measures crowdedness and data motion. VISOR BASE tends to be more complex than the two projects mentioned before. It counts people and tracks them, recognise faces and has a plate number reader. The algorithms of this project are likely to be related to object detection, object tracking and features classification, involving potentially neural networks to make the system more efficient and faster, distributing loads to various servers.

The last case study to be taken into consideration in this section is the *SmartMonitor*. This solution learns some patterns and it is trained to react to some predefined situations, such as crime detection, supervision of ill persons and unauthorised intrusions [23]. According to its author, Visual Content Analysis (VCA) is widely used in the implementation, relying in foreground extraction, candidates extraction, object detection and object tracking.

According to [23], in *SmartMonitor*, Haar-like features and HoG are used for the object detection stage, due to their low overhead. For data acquisition, low-cost and low-resolution cameras were enough to the application and their location determines some useful data, in order to trigger alarms in case of object detection in a restricted area. Also, an object tracker is incorporated into the system, using Mean-Shift tracking instead of *Kalman* tracking which is discussed in [23].

2.2.3 Computer vision techniques

Taking into consideration the case studies mentioned above, and analysing their similarities with the case study stated in this work, it is necessary to analyse some of the CV techniques mentioned in each one of these cases. Hence, they will be analysed in two main fields: object detection, and object tracking.

Object detection

The most common object detection techniques, seen in the already presented case studies, are based on Haar-like and HoG features. These techniques are often called *hand-crafted techniques* due to its way of training and detection. There are other types based on machine learning, such as Support Vector Machine (SVM) and deep neural networks, that are more flexible than feature-based detection at the implementation level, but they require a large amount of training to achieve good results [24].

In the case of HoG, this technique divides the image into blocks called *cells* and computes a histogram of the gradient of each cell. Basically, the histogram summarises the edge direction and uses it as a descriptor (an element which describes a characteristic of the region analysed) [25], illustrated in Figure 2.4. One of the key advantages of this method is the light invariance, being mitigated during the edge detection process used to get the gradient. Also, this method can be combined with Lowe's Scale Invariant Feature Transformation (SIFT) [26] to strengthen the description scale-invariance [27].

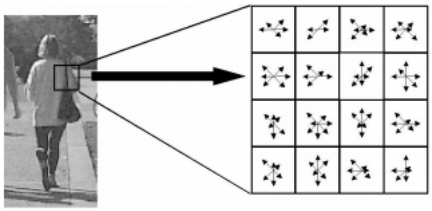


Figure 2.4: HOG features. Reprinted from [25]

Detectors based on HoG features are widely used for human body silhouette detection as it is done in [27]. Besides, the human body silhouette does not change considerably when it is horizontally rotated, HoG can be used with a strong precision. However, the robustness decreases when the object of interest is not geometrically uniform because of its dependence of the form, as can be deduced from [28]. This makes HoG detectors recommended only to certain applications.

On the other hand, the Haar-like features (see Figure 2.5) is one of the strongest non-neural network-based object detection techniques. Their principle is mainly the use of a cascade of weak classifiers to make a strong classifier. This approach was studied by Paul Viola and Michael Jones in [29].

Some pre-built cascade datasets can be found on the Internet, like the published by M.

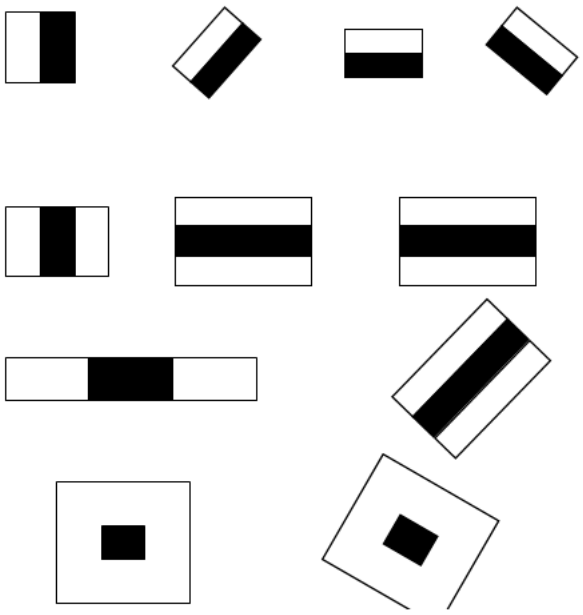


Figure 2.5: Haar-like features. Reprinted from [30]

Oliveira at University of Aveiro [31], used for his research, which contains a large amount of cars pictures, and the file only needs less than one MB to fulfil the cascade classifier.

In the design flow with Haar-like features object detector, the training stage is one of the more problematic stages, due to the time needed to train the cascade classifier, taking from hours to several days to complete. One of the possible reasons of the high computation power required to train a Haar cascade classifier is because of its numeric representation, based on double precision floating point [32], which needs an intensive use of the Floating-Point Unit (FPU). A different approach, but based on the cascade classifiers, is Local binary patterns (LBP) cascade classifier, which uses integer number representations, being faster than Haar-like feature classifiers, and less accurate in results [32].

Furthermore, the moar of object detection are actually based on Machine Learning (ML) techniques. The SVM is one of the first methods which takes advantage of ML [28] and supports supervised and self-supervised learning. It is possible to find object avoidance, route detection and autonomous driving applications [33], that are closely related to the topic of this project.

The challenge that takes into consideration SVM is the constant background changing, and the detector adaptation to different scenarios. In the case of autonomous cars, a robust detection under these conditions is crucial, as [33] explains.

One of the characteristics which make SVM interesting for research is its invariance to several cases: transformation, scale, and rotation. Shigeo in [34] mentions that L1 and L2 (normalization types) are both invariant to those conditions. Also, it is evidenced in the research done by Heisele et al. in [35]. However, there are some tricks to achieve and enhance the object detection invariance, explained in [35], which involves from training

including face rotation to combine features in the inputs, like HoG features to perform the HoG-SVM detector.

The last technique to analyse in this document is Artificial Neural Network (ANN) based detector. This has been studied recently, and some case studies highlight its robustness in object detection and classifying. They tend to have more accuracy in the predictions than SVM, or features-based detectors.

Most of ANNs are composed of two or more layers of cells which define a weight of a defined feature (see Figure 2.6). Lastly, they have been used for pattern recognition, such as audio signals or images (2D signals), achieving promising results that even a human can not do. Some of the most popular ANN used in CV are Convolutional Neural Network (CNN) and Deep Convolutional Neural Network (DCNN) [24].

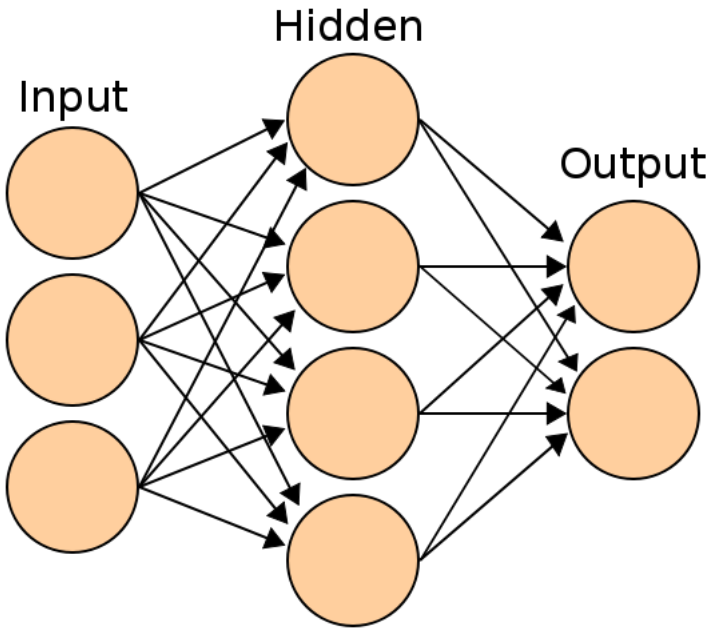


Figure 2.6: Artificial Neural Network (ANN). Reprinted from [36]

According to Aarthi and Harini in [24], object detectors based on ANN give more accurate results than other methods, not matter if there are contrast and light changes. Moreover, the robustness of ANN makes them predict accurately even in geometric distortions. This characteristic of ANN is because of their capability to learn optimal models, which include Shift-Invariance in their feature detectors.

Object tracking

In the case studies analysed before, some techniques of object tracking were mentioned. In this section, they are studied briefly, mentioning some advantages and disadvantages of each one.

The object tracking can be only implemented on a sequence of images, with a start point,

given by the detector. The tracker is able to follow the object in the following images after it has been detected. The differences amongst techniques rely on the image processing stages preceding to the object tracking process.

The first method is **Kalman filter tracker**, that is, in principle, a technique to predict functions using linearization and memory, using the system history. It is widely used for systems which inputs are seriously affected by noise [21]. However, it can be used in other contexts like image object tracking, because of its prediction capability.

To implement a Kalman Filter for object tracking, it is required to define an image feature to follow it. Some of the potential features for tracking are colour, lightness, and contrast [21]. This method needs an image preprocessing stage before the tracker. As it was mentioned briefly in the Case Studies section, background differencing, and background supression are potential preprocessing techniques for this task, where the background is removed from the picture, and replaced by a black background, and the foreground conserves its content. Then, given the contrast between background and foreground, there are potential object candidates, that should be verified by a detector, i.e. a feature-based detector. Afterwards, a new object position is given to the Kalman filter detector input, and its output will give the new position without noise.

However, Kalman filter-based trackers rely on a detector the preceding stages, making them naive trackers, and just predictors.

Another technique is named Mean Shift Tracker, based on the Mean Shift algorithm, and it is a method to locate the maxima given a density function [21]. Similarly to Kalman, it needs an image preprocessing stage to predict the new position of the object given a determined feature such as colour, which is the most common feature used in Mean Shift trackers [37], taking the colour histogram as a density function. Thus, those Mean Shift trackers are only applicable to colour images, and the basic principle implies that the object should be highly contrasted from the background.

The more robust trackers are based on templates. That means that the start point is a template given by a detector, and the tracker will try to find the object of the template in the following images of the sequence [21]. Basically, this principle implies that the detector is a trigger of the object tracker. In the case of multiple object tracking, multiple trackers have to be deployed to fulfil the tracking of each object, assigning to each object its own tracker.

This technique intensively uses the Correlation Theorem, based on convolution in the frequency domain, given by:

$$G = F \odot H^* \tag{2.1}$$

The beginning of this process starts converting the filter kernel, and the image to analyse to the frequency domain. After that, convolves it and multiply both spectra. Then, it takes the correlation peaks (Peak Spectrum Density or PSD) positions, and compares

their values against a threshold. If a peak value is greater than the threshold, the object was found in the specified position. However, to strengthen the detection and tracking, the filter continuously evolves to adapt to potential changes in the object perspective each time that there is an object match, such as Minimum Output Sum of Squared Error (MOSSE) Filter tracker proposed in [38] and illustrated in the picture 2.7.

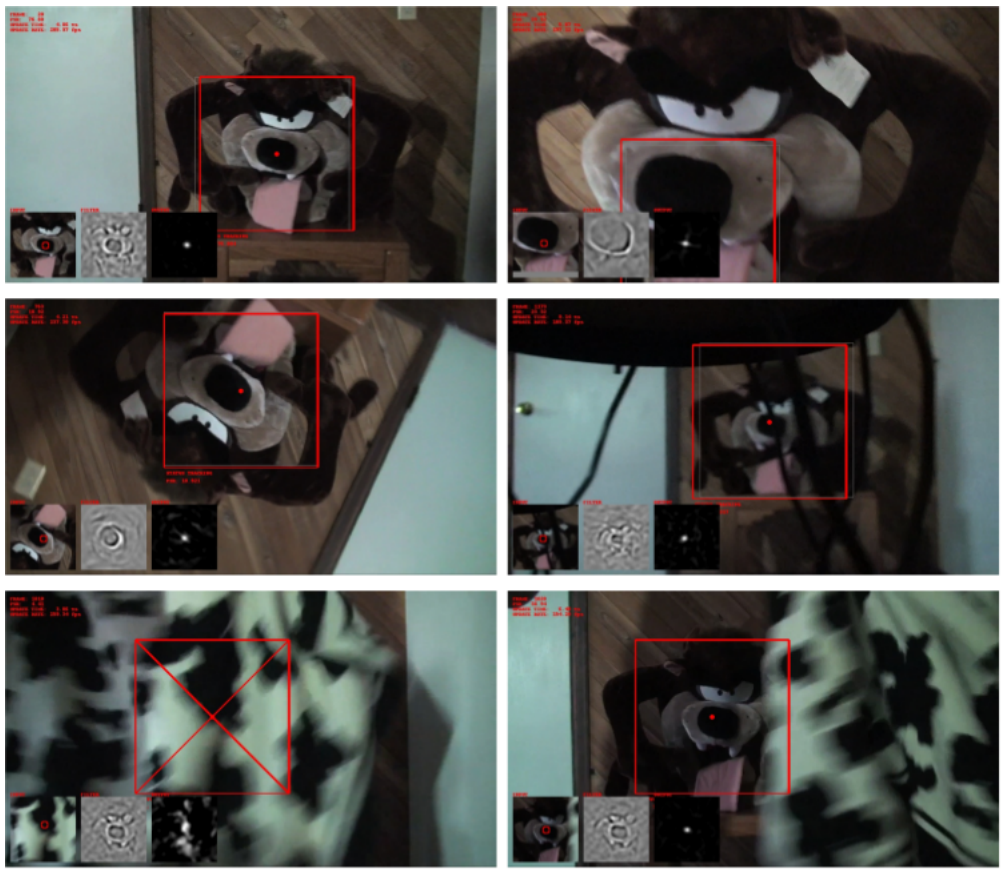


Figure 2.7: MOSSE filter tracker results. Reprinted from [38]

The aim of using convolution theorem and MOSSE filter is to maximise the Signal-to-Noise Ratio (SNR), making a template coupled to the tracked object, and trying to find its *energy* in each picture even if there is distortion on it, as it is commonly used in *coupled filters* in Communication Systems [39].

According to the results presented in [38], the MOSSE filter is very robust in cases of occlusion, scaling, light changes and rotation, making this tracker the best compared to others such as KCF, ASEF and UMACE.

2.2.4 Open challenges

The challenge in object detection is fundamentally getting proper training data for the selected object detection method. Generally, cascade classifiers can save the trained features into a file, like the M. Oliveira cars dataset in [31]. However, finding datasets for neural networks is harder than finding features datasets.

Also, a high accuracy in object detectors is crucial, and sometimes hard to achieve. There are challenges imposed by variances; such as rotation, scaling, geometric transformation, light, and contrast variance. An ideal detector is invariant to the effects mentioned before [28].

The detection speed is one of the key aspects to tackle in future research, especially, for real-time processing [32]. Improving algorithms and making them faster is critical to allow systems to react in short periods.

Object tracking is another part that has to be focused on. Template trackers, even though they are the more robust, work for single objects, especially when the objects tracked have different colours, shapes or variations [21]. An increasing number of tracked objects might cause overhead in an application and should be studied before doing an implementation, evaluating the highest number of cars that the system can manage, and the computational weight of each tracker.

2.3 Approximate computing paradigm

Most of AI applications involve complex algorithms and require therefore high computational power in order to achieve results within the given time. This requirement is often performed by platforms which consume a large amount of power due to the speed required, and the number of computing units [4]. However, there are platforms that do not fulfil this requirement, because they have a constrained power consumption and computing resources, such as IoT and mobile devices.

To deal with the trade-off of computational power and available resources, some applications can be approximated due to their error resilience [5], sacrificing results precision in order to reduce the need of complex arithmetic units, such as FPUs. This paradigm is called Approximate Computing [40].

The main constraint to implement an application using the AC paradigm is to have a resistance to the error in its results. Probabilistic applications are often good candidates to be approximated, such as Machine Learning based on ANN [4] given their weight computation, learning process, and prediction system given by linear algebra.

In spite of having error resilience, some applications can be more suitable to be approximated than others. It depends on the relevance of their results in the following stages and how critical they are. Generally, the approximation is determined by adjusting knobs, which adjust the numeric precision of a given stage in a baseline, evaluating and adjusting at the same time their impact, verifying if results are still suitable for the given application [4]. Shafique et al in [4] present this procedure in their paper, where AC is applied to Machine Learning applications. As they explain, the *quality knobs* (named like that due to the quality change in the results in terms of precision), give control over the accuracy-power trade-off.

Before continuing the analysis of AC and their impact in the application development field, it is required to study their possible implementation ways and some case studies which have put this paradigm into practice.

2.3.1 Methods of implementation

There are two main sectors where the AC can be applied: hardware and software [5]. Depending on the flexibility of the application and the hardware restrictions [6], either software approximation or hardware approximation is more suitable to utilise.

Software approximation

Software approximation involves changes in the algorithm and data types to make use of approximations on an exact platform, but with few computational resources. Some ways to implement are listed below:

- Loop perforation
- Numeric representation
- Approximate formulas or functions

Loop perforation is a technique applied to code loops with the aim of reducing the computation workload and runtime. Besides, it is possible to reduce energy consumption, and enhance performance, as Sidiroglou et al stated in [41].

The main idea of loop perforation is to skip one or more cycles depending on their impact in the results, letting the possibility of dynamically determine whether a cycle can be skipped. To demonstrate that, consider the following exact form of a *for loop* in C++:

```
for(int i = 0; i < b; i++)
{
    // Some instructions
    // ...
}
```

The proposal presented in [41] suggest that a *naive* loop perforation may be:

```
for(int i = 0; i < b; i+=n)
{
    // Some instructions
    // ...
}
```

Where n, with $0 < n < b$, is the number of cycles skipped consecutively.

Furthermore, the work done by Sidiroglou et al reach such focus and go further, indicating that it is possible to dynamically reckon the number of cycles that can be skipped in a row, depending on the program status. Even more, proposes a change at compiler level with *LLVM*, that makes the naive loop perforation automatically.

Numeric representation is another way to approximate calculations in an application, changing the data type to represent a number, i.e. truncating a floating point number and making it integer. This approximation method was used by Lai, Suda & Chandra in [42] to enhance an ANN representation from floating point number representation to fixed-point representation, achieving a reduction in weight storage by up to 36%, and the power required by up to 50%, sacrificing precision in their results.

An example of this change is shown below:

```
// Number represented in floating point
float floatingN = 0.95863445262;

// To fixed-point given by 16 bit signed integer

int16 fixedN = 0.95863445262 * 32768;

// Result: 31412. Now, the number changes to: 0,958618164

// Error:
float error = 0.95863445262 - fixedN/32768;
error /= 0.95863445262;

// Resulting error: 0.002%
```

Fixed-point representation can lead to numeric errors below of 0.01%, considering that the maximum result deviation is given by $\sigma_r = 1/2^{15}$.

This method was tested by Du et al. in [43] on a face recognition application using neural networks, saving energy consumption up to 62.49%.

Approximate formulas or functions is a way to simplify a function computation using a less computational exhausting expression to avoid using complex operators, i.e. using FPUs. Besides, there are some techniques which simplify, and speed up the computation of complex functions. One example is the calculation of integral transformations, such as Discrete Cosine Transform (DCT), and Discrete Fourier Transform (DFT). There have been efforts to parallelise computing fragments using recursion and divide and conquer algorithms, as historically proposed Cooley and Tukey in the Fast Fourier Transform (FFT) [44], with the *Butterfly algorithm*.

Despite the fact of trying to get an exact DFT calculation, reducing the computational power required, there are some alternatives proposed to compute the DFT approximately to ease the computation of the transform, reducing the amount of time needed to get the results. Although they are approximations, applications with error resilience can deal with those inexact results and work well, saving time and energy during their calculations.

A proposal of approximating FFT computation was done by Eldeman, McCorquodale and Toledo in 1999 [45]. They achieved a speed-up of more than a factor of 2, and took advantage of the emerging symmetric multiprocessor, taking into consideration *matrix-vector algorithms* and *fast multipole approach*. Nevertheless, this algorithm has some issues with the computing ratio, making in some cases, that the proposed algorithm runs slower than other traditional algorithms.

Furthermore, research has been performed exploring FFT approximations using fixed-point number representation, done by Freescale Semiconductor in [46] achieving up to a factor of 4 in speed-up and Tom Roberts, Malcolm Slaneyand and Dimitrios P. Bouras, who adapted the *FFT Butterfly algorithm* to fixed-point [47].

There are some code analysis engines that make the machine code generation smarter and data-responsive, i.e; *approximate-aware programming languages* [48]. A simple example of optimisation at code level can be:

```
/*
    Dealing with multiplications
*/

// Multiplying by a 2^n factor
int number1 = 5;

// Operation
int result1 = 5*16;
int result2 = 5 << 4;

// Both results will be 80
```

Despite being an exact example, it can be extrapolated to approximate cases. Consider this other case:

```
/*
    Dealing with multiplications
*/
// Define a const given by a formula
const float factor = 4.05;

// The variable is stored in uint16_t format
uint16_t number1 = 5;
```

```
// Operation
uint16_t result_approx = number1 << 2;
uint16_t result_exact = (uint16_t) number1 * factor;


// Error: 1.23% in the final results
```

In the example presented above, the factor can be approximated to a binary shift, neglecting the fractional part of the constant factor. The error stays near of 1.24% in the values allowed by the *uint16_t* data type.

In applications where this error can be ignored, it is a suitable change that either the compiler can do internally or the programmer can code. This approximation method can avoid the multiple clock cycles needed by the FPU for performing the multiplication [49], and only needs one cycle to perform the multiplication using binary shifts, saving time and speeding the application execution up.

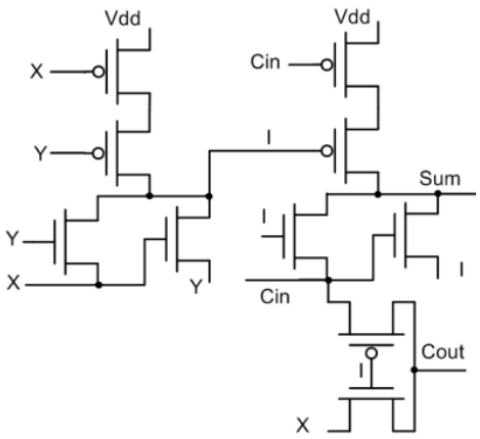
Another classic approach that can be suitable for an approximation of elementary mathematical functions is the Taylor series. Chen and Chichyang proposed in [50] a piece-wise polynomial method for computing elementary functions, taking into account error quantification, and reducing the amount of resources required for implementing lookup tables or dealing with the storage and hardware needed for computations, saving up to 45.2% of the lookup tables using Table Sharing method and the first two terms of Taylor series in floating point number representation.

Hardware approximation

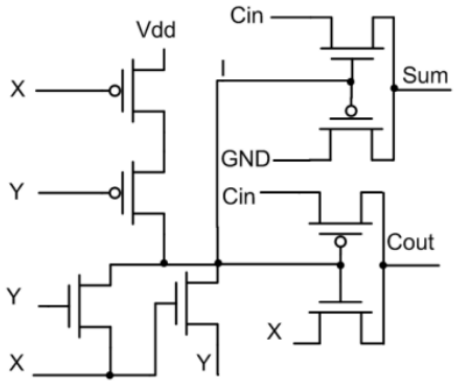
From another perspective, computing approximation can be also done on the hardware site. Modifying from the logic circuits to making accelerators using Hardware Description Languages (HDL). There are some interesting proposals which include changes in the hardware, performing functions using neural networks, accelerating by hardware using non-floating point number representations, approximate Central processing unit (CPU) architectures and memory handling strategies [6].

To study briefly each of them, this section will be divided into the following points:

- Logic circuit approximation
- Memory handling strategies
- Hardware acceleration
- Neural network approximation



(a) Accurate full-adder with 10 transistors. Reprinted from [52]



(b) Approximate XNOR-based Adder 3 (AXA3) with 8 transistors. Reprinted from [53]

Figure 2.8: Comparison between an accurate full adder and an approximate adder

The logic circuit approximation includes attempts to approximate arithmetic operations, such as addition, subtraction and multiplication. In [51], the authors show multiple strategies to design and implement approximate adders and multipliers.

In the case of adders, there are options such as Approximate Mirror Adders (AMA), XOR/XNOR-based adders (AXA), and Lower-part-OR adder (LOA) for implementing full adders [51]. The aim of substituting exact adders with approximate adders is generally to decrease the energy required to power the circuits up in exchange for arithmetic errors, speed or data noise. Some of those strategies are aiming to reduce the number of transistors used to implement the adders (see the case presented in Figure 2.8), bringing down the area and power required.

On the other hand, the multipliers have not had enough attention in order to propose approximate ways to perform multiplications. However, there are alternatives of using partial products, cascade adders, and simplified 2x2 multipliers [51].

The memory handling strategies are fundamentally based on data-storage strategies and hardware techniques. [6] mentions Dynamic Random Access Memory (DRAM) refresh rate as one of the techniques to reduce the amount of power required by keeping the data in the DRAM. To model the error caused by changing the refresh rate, the Last-Access Dependent Bit Flip is used, being a *fine* knob to adjust the energy-accuracy trade-off. An alternative to performing changes on the DRAM refresh rate is presented in [54], saving up to 25% of DRAM power consumption.

Another memory handling strategy is called *load value approximation*, making the most of the memory locality, and predicting values when a cache miss happens instead fetching in the next stage in the memory hierarchy [55], granting to the processor to go on with the next instructions, and minimising the impact of cache misses on the runtime.

Hardware acceleration strategy is based on implementing some functions on hardware ac-

celerators, creating a hybrid Software/Hardware (SW/HW) application. This approach alleviates the computation required in the processor, and sends some functions to HW, freeing the processor to do other tasks. This approach was applied in a case study in biometric security systems presented in [40]. This combination between SW/HW implementation was able to achieve x48 in speed-up respect of an exact application already on HW.

Neural network acceleration is an interesting approach to do acceleration using approximations. This method is essentially training an ANN to learn a function and response like it, letting to change precision in the results with *coarse* adjustment [6]. Implementing this technique can be done either by software or by hardware (using acceleration).

Some research done based on this method adopt the Field-Programmable Gate Array (FPGA) as hardware platforms, generally in combination with soft processors, like Zynq 7000 FPGA-based System on Chip (SoC) [56]. A brief diagram which illustrates the communication between hardware and the soft processor is presented in Figure 2.9.

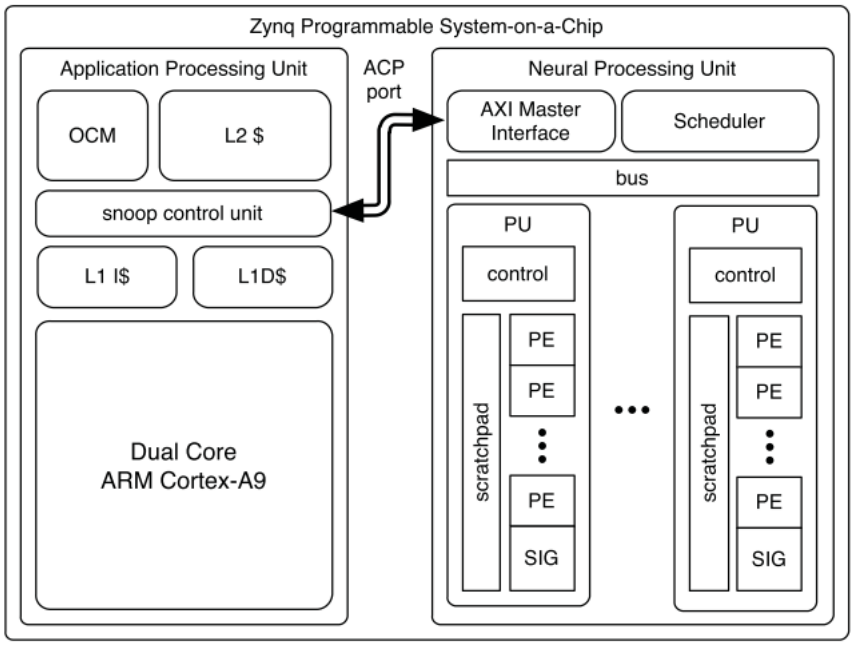


Figure 2.9: Basic system architecture of implementing neural accelerators. Reprinted from [56]

Esmaeilzadeh et al. in [57], stated the term Neural Processing Unit (NPU) to name a unit to accelerate, and approximate functions using neural networks connected to the processor pipeline. The NPU makes an application speed-up up to 2.3x, and up to 3.0x in energy savings according to the authors. They explain that using the NPU is faster than executing the code of a function on the processor, getting better performance with an acceptable accuracy.

Even though the results presented in [57] are a promise for using NPU for approximating and accelerating applications, the results presented in [56] go further, presenting speedups up to x38.1.

The process of implementation of this approach is summarised in Figure 2.10. The training and code generation for the softcore and NPU is made in the *compilation stage*. In this stage, the compiler prepares a set of training data and a set of test data and tries to emulate the behaviour of the target function with an acceptable error.

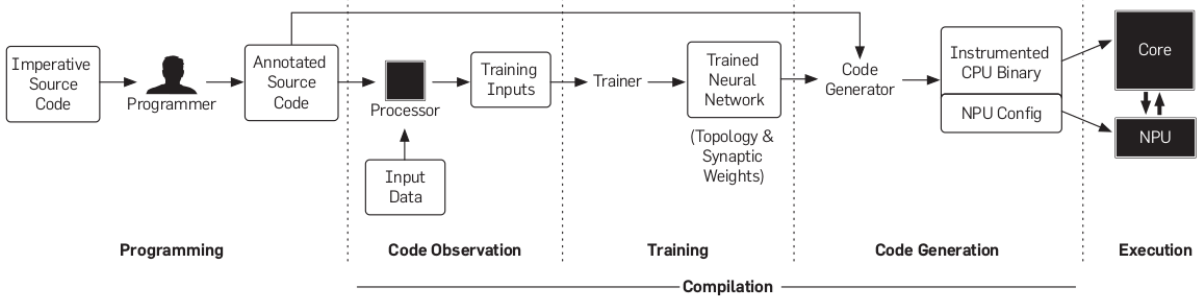


Figure 2.10: From annotated code to the execution. Reprinted from [56]

2.3.2 Case studies

The AC paradigm has contributed to improve some existent applications and allow to incorporate complex algorithms to low-power platforms. In this section, some of those will be studied, and their results discussed.

As examples to show the results of the different methods of approximation, the FFT algorithm [56], JPEG compression [6] and body tracking [55] have been implemented and tested in order to compare the results of an exact baseline against an approximated application. Thus, the scope of AC can be applied to image processing and analysis applications without further complications.

The first case study to analyse is the *Iris Scanning for Biometric Security Systems*, presented by Hashemi et al. in [40]. They were able to implement iris recognition algorithms onto an FPGA-based SoC, applying approximations to the Recurrent Neural Network (RNN) and accelerating taking advantage of the FPGA platform, implementing all the methodology on it.

They took the *kernel size* as an accuracy knob in the focus assessment stage, with the aim of varying the energy required by the convolution between the incoming frame and the kernel. Then, they vary the iris segmentation, changing the scale, thresholds and window size as well. They report having achieved an x48 in speed up in comparison to an exact architecture, making it possible to implement iris recognition for biometric security systems on low-power platforms. Furthermore, this demonstrates the capability to implement complex CV algorithms in resource-limited platforms.

Like the case study of biometric security, Shafique et al. made an analysis of the *error resilience of DNN in hand-written character recognition* in [58]. They implemented approximate arithmetic modules into the DNN for results computation. An important

aspect of their work is the data awareness to deal with the varying behaviour of the approximate modules to distinct data distributions. They explained that if the application had not data-awareness, the benefits gained by approximation can be turned into losses. Thus, the work done in [58] is an example of intelligent data approximation in systems which data is sensible to approximations, and it must decide to approximate reasonably.

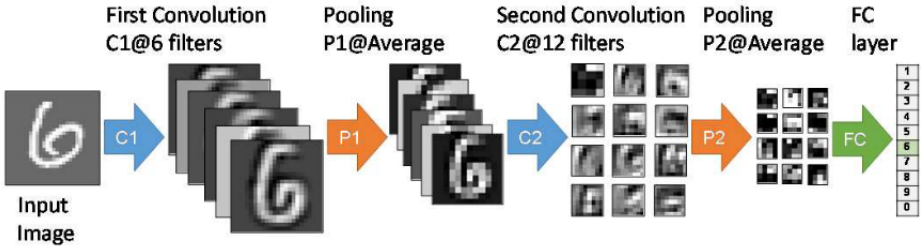


Figure 2.11: Modified LeNet for Handwritten Digit Classification (MNIST dataset). The figure illustrates the results of the different stages inside the DNN. Reprinted from [58]

Figure 2.11 depicts the results of each stage of the DNN used for the experiments in this application case. They based their experiment on the LeNet-5 [59] for hand-written character recognition. Likewise, they tested six different approximate adders, obtaining distinct accuracy results. In spite of approximating, they achieved a prediction accuracy of up to 94% in the best scenario with visible power savings due to the reduction of the adder complexity.

The last case study is presented as an alternative to enhance the security in IoT platforms at the same time of saving resources. The relevance of this case study is summed up in the emerging technologies based on Internet connectivity and the low-power platforms that have been introduced to the market so as to satisfy the applications that have been released. Most of them are resource-limited devices and powered by batteries to increase their mobility. The major challenge in IoT is to implement devices which intelligence enough to process in-node and reduce the dependence of cloud computing processing.

Gao et al. in [60] propose to implement approximations to increase the security in IoT (Internet of Things) systems putting into practice IP (Internet Protocol) watermarking, digital fingerprinting, and lightweight encryptation and save resources that can be used by applications.

To achieve this goal, approximate arithmetic modules have been taken into consideration such as adders and multipliers, based on approximate integer format (both numerical representation and logic circuit approximation). Also, they propose information hiding techniques without putting system security into risk, making use of security bits and approximating the IEEE 754 simple floating point number representation [60]. This process is represented in Figure 2.12.

As Figure 2.12 depicted, the goal with the numeric approximation is to insert security bits into the word, reducing the number of bits available for the number and decreasing the

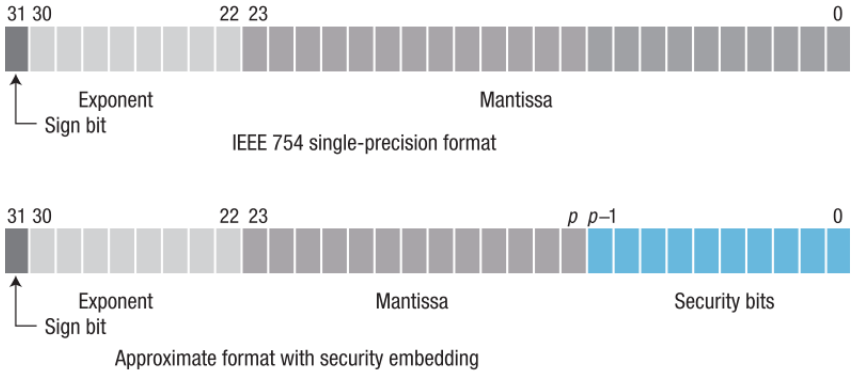


Figure 2.12: IEEE 754 single-precision floating-point format for 32-bit data. The last p bits can be used as security bits to embed information without changing the rest of the 32 – p bits. Reprinted from [60]

precision insignificantly. However, the advantages in security, especially in authentication, might be a good deal [60].

2.3.3 Open challenges

In most of the studied and analysed applications, the promise of application enhancement dealing with the accuracy, and resources consumption trade-off offered by Approximation Computing is viable in Machine Learning and inexact algorithms. However, more work is needed in this emerging field. In [5], the author concludes that significant research should be done to put into practice the promises of this young paradigm, from the programmers to the manufacturers. The key aspect in, for instance, image processing is to spend the resources to get subjectively good enough results for the market.

Shafique et al. in [4] mentions that *open source* is one way to go further and faster with the development of future approximate computing applications, stating a baseline to research on and verify the acquired results. Also, they explained that the focus should be on saving memory because of the memory consumption, which usually takes more than 50% of the total energy spent in the devices. Likewise, the use of FPGA should be more studied, and the implication that could have reconfigurability in the application overhead.

Esmaeilzadeh et al. in [57] explained that programmer effort is mandatory to identify potential approximations and prepare data for training, in the case of NPU-based applications. Besides, the efforts should be done as part of coding efficiency, where the approximable parts could be accelerated and the approximate computing fully exploited.

2.4 Conclusion

Following the case studies and the approximation methods, it is possible to accelerate video processing and analysis applications. Joining the AC paradigm with the CV techniques seen in the second section of this chapter, it is possible to enhance performance due to their inexact basis, making it possible to implement approximations either by software or hardware.

Nevertheless, implementing object detection and tracking in approximate platforms should be studied carefully, measuring the impact of each approximation in the final application results, checking whether the outcome errors are still tolerable or not.

Taking into account the studied research, some modules or functions required by the computer vision analysed before have been implemented and tested, but there are not approximate alternatives to detect objects by *hand-crafted* methods or to track them using *detection-independent* tracking techniques.

Thus, CV techniques can face the open challenges of ITS, because of their low-cost and simple implementation, and it is an opportunity to improve some of those techniques applying approximation computing, contributing to this emergent field.

Chapter 3

Traffic dynamics meter

As it can be noticed from the previous chapter, there are multiple techniques to detect and track objects. However, choosing a technique depends on the properties of the objects of interest and the transformations that they can have during a video.

To analyse them accurately, it is needed to define data acquisition characteristics, phenomena that can affect the image sequence and the variables to quantify through CV.

In this chapter, the project constraints and goals are discussed. Also, several methods of implementation studied according to the restrictions imposed on this project are analysed.

3.1 Design requirements definition

3.1.1 Potential issues and challenges

In Section 2.1 was possible to determine that CCTV was one of the cheapest methods for implementing ITS. Actually, in Costa Rica there is a free and open CCTV network available for research and from which information related to the traffic status in different problematic points of Costa Rican roads. The websites *Camaras Viales CR* [61] and the *Caltrans - California Department of Transportation* website [62] show the hyperlinks to get access to those cameras, which makes possible to extract data for research to improve the driving experience in Costa Rica and save people from getting stuck in traffic.

Taking that into account, the main source of video inputs for this project are roadside cameras. In spite of being free and open, there are some issues with using those cameras, as summarised in Table 3.1.

Another aspect to take into account is the possibility of not finding a proper training dataset, due to the variations of the camera, summarised in Table 3.1, and according to the stated in the Background and Related Work, training can take a large amount of time to complete, being a potential bottleneck during the research. Also, using an external dataset, like the proposed in [31] can bring to accuracy loss in the object detection and

Table 3.1: Potential public camera issues

CV Problem	Origin
Variable video resolution and frame rate	The use of different cameras for video capturing.
Light variance	Outdoor conditions can affect the lightness perceived by the camera due to weather and time.
Different points of focus or perspectives	The objects of interest can have geometric, rotation and scale transformations.
Different road lengths and traffic directions	The ROI varies from a camera to another. Also, there are cameras focused on two or more lanes, most of them with multiple directions.
Latency	The camera Internet connectivity can lead to connection problems.

shall be mitigated in the following stages.

Finally, implementing this application on a low-power platform implies to consider an off-line processing in case of not meeting the online processing requirement due to the time that can be needed to execute the application. For that reason, both scenarios must be considered.

3.1.2 General application design

Figure 3.1 shows a basic perspective which models the potential application in order to measure the traffic dynamics and quantify the relevant variables, that can be useful for further research in automatic traffic control. Taking advantage of the cameras available in Costa Rica, the input data are retrieved from them and will feed the application. Then, the objects of interest (cars) are detected in Detection, which composition will be defined in the next section. Once a car is detected, Tracking is in charge of measure some parameters of the car, such as speed and direction. Then, Measurement takes data and average it to get the final results, available in Output.

Figure 3.1: First perspective for project design

3.1.3 Requirements

With the potential issues met and a general idea of the application, the project requirements can be set properly. They are specified in Table 3.2. These requirements guide

the project design and implementation and help to take the proper decisions in order to develop the project in the right way.

Table 3.2: General application requirements

Concept	**Element**	**Requisite**
Results tolerance	Output results tolerance	All the results are under 20% of error.
Image processing	Image capture	Each frame is retrieved and decoded under 2.2 seconds, to capture cars at 40km/h in a distance lower than 10 meters.
	Preprocessing	The filters and morphology operators must meet: • Capability to be implemented in an ARM architecture. • Have a kernel size at least of 3x3 pixels. • Be separable (parallelizable).
Image analysis	Detection and tracking	The object detection and tracking stages must be: • Transformation invariant or resilient. • Object overlapping resilient (up to 3 objects overlapped is still tolerable). • Possible to implement on an ARM architecture.
	Object classification	The trained stages have to be able to be supervised-trained
Execution modes	Online execution	The application is ideally able to be executed online. This means that all the process should be done in a frame period.
	Off-line execution	If the online execution cannot be met, the application must take up to 5 minutes to process a 1 minute video sequence.

3.2 Technologies and techniques selection

3.2.1 Variables measurement

In the Traffic Dynamics Measurement (TDM) section, a set of variables were defined to quantify the traffic dynamics in terms of *speed*, *flow*, *density* and *occupancy*. Some of the models proposed in [14] use this set of variables to represent several traffic models. According to [14], they are the basis to analyse the traffic dynamics and obtain sufficient data to implement applications to optimise the traffic and feed model simulations with Monte Carlo techniques [19].

Nevertheless, some of those variables can be calculated in different ways which differ from each other. One example of this phenomenon is the speed, which can be calculated either using a constant length and an averaged time of each car or by computing the speed of each car and average the results to get the average speed [14].

Average speed

Considering that the most robust *trackers* are managed individually using templates, it is more implementation-friendly to use a speed-based average, with two space points and taking the time given by the difference of the time when a car is in the second point, and the time when the car was in the first point. The equation for the sampled average speed is given by (3.1).

$$\bar{u_s} = \frac{1}{N} \sum_{i=1}^{N} \bar{u_i} \tag{3.1}$$

where N is the number of cars analysed until the sample time and $\bar{u_i}$ is:

$$\bar{u_i} = \frac{1}{F_N - 1} \sum_{f=2}^{F_N} \frac{||\mathbf{P}_f - \mathbf{P}_{f-1}||}{t_f - t_{f-1}} \tag{3.2}$$

where f is the frame number, F_N is the number of frames until a sample time (defined further), $\mathbf{P}_j$ is the vector returned by transforming the detected point in the frame to the scene coordinate system and t_j is the time point.

To quantify the average speed in this manner, the application requires the scene coordinate system in advance, and it must be personalised for each camera input. The process to do so is mentioned in the following sections. Also, it is required to know the video time to get time points.

Traffic flow

Flow rate can be calculated using the number of vehicles which have crossed a defined imaginary line divided by a sample time. The suggested sample time for high flow rates is usually from *1-5 minutes*, even though a *15 minutes* interval works [14] properly. To compute the flow, therefore, the proper equation is (3.3).

$$q = \frac{N}{T} \tag{3.3}$$

Now, a possible technique that can be used for quantifying the traffic flow is to count the number of cars which have passed over an imaginary line during a time lapse. When the time lapse is reached, the counter is reset to *zero* and the timer restarted. This timer is named *Analysis Timer*.

Density

Traffic density, according to [14], is mainly represented as a longitudinal density given by the number of cars divided by the road length (see equation 3.4). Due to that, it is required to pass the road length value before starting to take new data.

$$\lambda_k = \frac{N_R}{L} \tag{3.4}$$

where the k subindex indicates that is an instantaneous density and N_R is the number of cars inside the ROI (*Region of Interest*).

In order to save resources and make the most of the Analysis Timer, it is possible to reuse it to get samples of the density. However, because of the density time-independence, it is undersampled if this timer is used. For that reason, it is possible to use a faster timer to create a mean value of density. So, the final density equation is given by (3.5), where k is the sample number got using the faster timer and T_k is the number of samples taken until the flow timer ticks.

$$\lambda = \frac{1}{T_k} \sum_{k=1}^{T_k} \lambda_k, T_k \in \mathbb{N} \tag{3.5}$$

Hence, two timers are required, one for sampling time-dependent parameters (Analysis Timer) and the second for sampling real-time parameters (named *Real-time Timer*). This effect also happen on the average speed, where F_N will be determined as the number of frames represented by the period of the Real-time Timer, and final average speed (see equation 3.6) is computed in the same way as density is. It is important to highlight that it is possible to determine average speed and density given by the Real-time Timer with $\bar{u_s}$ and λ_k, respectively; for future trending analysis.

$$\bar{u} = \frac{1}{T_k} \sum_{k=1}^{T_k} \bar{u_s}, T_k \in \mathbb{N} \tag{3.6}$$

Occupancy

To get the occupancy, it is needed to average the time taken when a car is under the focus of the sensor (camera). To do that, also a defined region of interest is required, where there should be two triggers for measuring the time elapsed to cross the region. The average occupancy is given by 3.7.

$$\bar{o} = \frac{1}{N} \sum_{i=1}^{N} \bar{o_i} \tag{3.7}$$

where N is the number of cars sampled until the Analysis Timer tick and $\bar{o_i}$ is:

$$\bar{o_i} = t_2 - t_1 \tag{3.8}$$

where $t1$ and $t2$ are the times triggered by the car i at the beginning of the region and at the end of the region, respectively. A requisite derived from the occupancy is the need of a region defined before executing the application and distinct for each camera.

Variables measurement requisites

Once defined how the traffic dynamic variables can be measured, it is important to summarise the requisites of the Measurement stage as presented in Figure 3.1 for planning the techniques which ease the quantification. Table 3.3 summarises this requisites by each variable.

Requisites enlisted by Table 3.3 help to define the global variables needed in the application, especially, for analysing which ones should be created or reused. The following section develops the CV techniques for object detection and tracking, it is important to highlight the relevance of the requisites 1, 8, 9, 10, 13 and 14. There are external parameters that must be passed to the application and are scene-dependent.

In the case of the requisite 1, a transformation system between the *image coordinate system* and the *scene coordinate system* must exist in order to get the speed in a metric system. For the 8^{th} and 14^{th}, both are related to an *arbitrary ROI*, that must be defined for each camera as same as the *Road length* mentioned in the 9^{th} requisite. It is possible to pass those parameters using a *mask image* to discard the parts of the scene that are not relevant for analysis, a file with the camera settings, which contains the transformation constants (assuming a linear transformation), and the *Road length*.

Table 3.3: Requisites of *Measurement* stage

Variable	Requisites	Symbol
Average speed ($\bar{u}$)	1. Scene coordinate system	
	2. Video time	t
	3. Car counter and results accumulator	N and Acc_N respectively
	4. Average per car	$\bar{u}_i$
Flow (q)	5. An imaginary line for counting cars	
	6. Car counter	N
	7. *Analysis Timer*	T_f
Density (λ)	8. ROI	
	9. Road length (inside the ROI)	L
	10. Car queue size counter	N_R
	11. Entries counter and results accumulator	T_k and Acc_λ respectively
Occupancy ($\bar{o}$)	12. Number of cars sampled until T_f	N
	13. Video time	t (split into t_1 and t_2)
	14. ROI	

For the requisites 10 and 13, they are requisites that can be computed by the CV application. The case of the 10, each car can be treated as an object, and the group of cars inside the ROI can be pushed into a queue. For the 13^{th}, the video might give the frame time [63].

3.2.2 Object detection

Object detection is critical for this application. However, this task is one of the strongest challenges to deal with, especially, when the requirements presented in Table 3.2 pointed out that there can be variances in lightness, scale, rotation, and geometry. Nevertheless, the most challenging part corresponds to the training, which can take a significant amount of time, bringing to the table the possibility of using an external dataset, which can lead to accuracy loss.

The following list summarises the main challenges of this stage:

1. Invariance resilience
2. Different scales
3. Prediction time

4. Features robustness

5. Training time

Perspective resilience

The first and second challenges involve a robust detector, which can deal with rotations, geometric transformations, scaling and light changes due to perspective (see Figure 3.2, which illustrates these variances), weather and time. But, the worst of the potential issues presented Table 3.2 is because of the use of different cameras. There is no available information about the intrinsic and extrinsic parameters in order to reduce the distortion. Therefore, the detector shall deal with all these adverse situations.

Prediction time

Assuming that the same dataset is used for training a HoG, Haar-like features and LBP classifier, it is possible to compare the three methods regarding their prediction time. In [64], the authors developed a study to compare those three methods in a stereo camera application. They found that in execution time, HoG detector performed in prediction time better, followed by Haar and LBP.

In [65], the same classifiers were compared and similar prediction time was obtained both for Haar and LBP classifiers, when used for satellital image detection. Also, they obtained disappointing results for HoG-SVM in the detection of those images.

Sharifara et al. contrasted Haar cascade classifiers and Neural Networks in [66], emphasizing that Haar-like features classifiers produce a lot of false-positives in the results. To mitigate that, use a hybrid classifier combining a Haar classifier with a Neural Network. Besides, the authors mention that the Neural Network tends to have high prediction accuracy; however, there is a trade-off between accuracy and execution time. Hence, the Neural Network-based classifier is a candidate which execution time is hard to compare, due to their variation in the number of layers and quantity of perceptrons in each layer.

Detection accuracy

In the Computer vision techniques section, the robustness to several variances was briefly mentioned for each object detection method. In the case of HoG detectors, they are not robust enough to deal with geometrical variances due to their tendency of basing their features on the silhouette, making them improper to this application.

On the other hand, the cascade classifiers have been tested in face recognition without basing their predictions completely on the object silhouette but based on parts of this as a result of the combination of the weak classifiers that compose the cascade classifier.

(a) Rotation variance - The car rotates because of the road curve

(b) Geometric variance - The car varies its shape when it gets closer to the camera

(c) Scale variance - The car varies its size because of the distance to the camera

(d) Lightness variance - The scene varies its lightness because of the weather, or time

Figure 3.2: Different variances detected over the potential image sources

Due to the feature principle of the LBP, it is possible to have less accuracy in the object classification compared to Haar-like features, which are stronger than LBP features. Furthermore, the possibility of using an external features dataset, in case of failing in the training stage, makes LBP not suitable for this research.

According to several authors, the ANN-based object detectors offer the best prediction. Their accuracy reduces the number of false-positives and can learn complex patterns, and makes these detectors stronger against the other options mentioned above [24] [66] [67]. Hence, the ANN seems to offer the best accuracy followed by Haar-like features cascade classifiers.

Training time

A third challenge is hard to deal with, because of the time dependence [68] of some methods. To train any method, several pictures have been taken in advance. Some of them are illustrated in Figure 3.3.

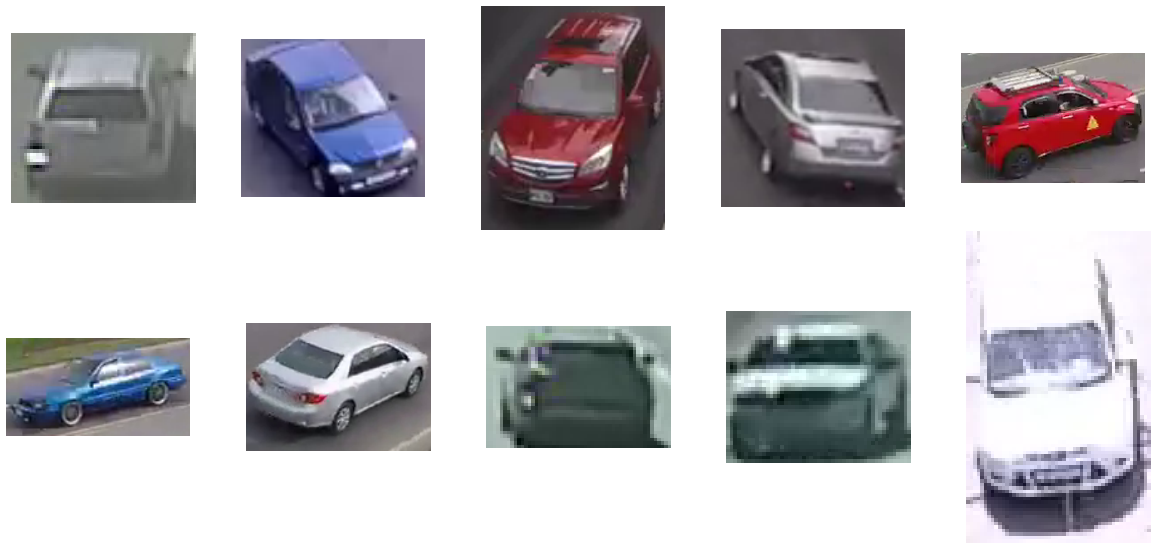


Figure 3.3: Own positive images

As it can be seen, the dataset considers different perspectives, lightness, and scales with the aim of training a robust classifier. In total, the dataset contains 1894 negative images, and 406 positive images (17000 mixing the negative and positive images) with the following properties:

- **Width**: 50px
- **Height**: 50px
- **Minimum hit rate**: 0.999 (if needed)
- **Maximum false alarms**: 0.5 (if needed)

- **Stages**: 10 (if needed)

Now, from the perspective of training each option, LBP is faster than Haar-like features classifier. This is due to the possibility of parallelising more operations compared to Haar because of the data representation, where LBP uses 8-bit integer data, whereas Haar uses floating point representation.

The ANN training process can differ depending on the ANN topology. In the case of a CNN, often used for pattern recognition, they require a large number of images to establish the weights of each layer. Also, the computation power required for training is also expensive, especially, if the accuracy required is high [67].

Albeit the accuracy of ANN and their potential, the Haar-like features cascade classifier is more suitable for the project, due to the easiness of finding pre-trained cascades contrasted to the ANN, which might require a high and intensive training and computation.

Comparison

To conclude the object detection technique selection, all the points mentioned in the different challenges are summed up in Table 3.4. The viable object detection technique, due to its robustness, and the availability of pre-trained datasets, is the Haar-like features-based classifier.

Table 3.4: Object detection techniques comparison

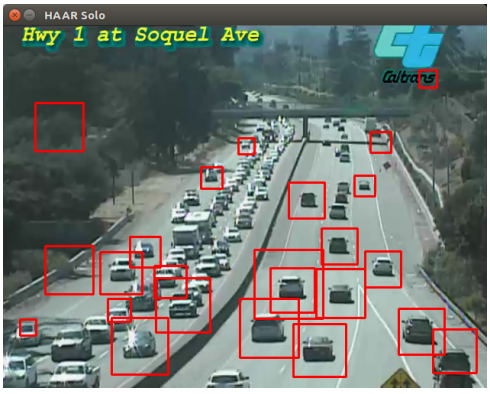
Technique / Criteria	**HoG**	**Haar**	**LBP**	**ANN**
Perspective resilience	Not tolerant to geometric transformations	Strong tolerance to rotation and scaling. Low tolerance to geometric transformations.	Strong tolerance to rotation and scaling. Low tolerance to geometric transformations.	Highly variance tolerant
Prediction execution time	Fast	Slow	Medium	Accuracy dependent
Prediction accuracy	Low	High	Medium	Highest
Training time	N/A	High	Medium	High

Tests

To ensure that the Haar pre-trained dataset works properly for this application, it was tested in distinct video recordings, and their results are shown in Figure 3.4, where the red squares contain the detected objects.

(a) General Cañas highway

(b) Multiplaza Este frontal avenue



(c) Highway 1 at Soquel Ave

Figure 3.4: Tests of OpenCV Haar detector

The results presented in Figure 3.4 indicate that the pre-trained dataset can be used in the application. In Figure 3.4b, and Figure 3.4c, there are false-positives in the picture and must be solved before implementing the pre-trained dataset into the application. In spite of that, it is possible to see in Figure 3.4c the chance of reducing the ROI to take advantage of the lower half of the right lanes.

An aspect not described in Figure 3.4 is that there are true-negatives and noise in the detection. Some objects that are supposed to be detected are sometimes missed, and that could affect critically the application. Thus, in order to improve that, a preprocessing stage is put at the beginning of the detector, which results are presented in Figure 3.5. In spite of detecting fewer cars, the false-positives were almost totally removed. The preprocessor proposed consists of two stages: a Gaussian filter, and an erosion. Both with a 3 pixels squared-kernel size.

Haar cascade classifier training

According to the parameters set in Training time, the Haar cascade classifier training lasted more than 2 months, exceeding the time available to train the stage. Therefore, the pre-trained cascade was used with the preprocessing stage. Henceforward, the object detection stage is defined by a preprocessing stage followed by the cascade detector.

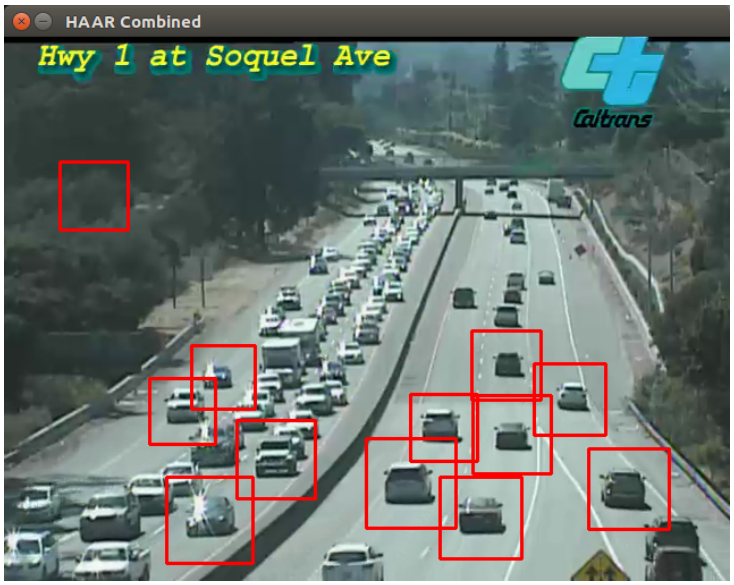


Figure 3.5: Improvement given by preprocessing before the detector

3.2.3 Object tracking

In Object tracking section in Background and related work, there were mentioned two types of object tracking whose key difference was their dependence of the object detector. Also, it was possible to determine that the template-based object trackers were more independent of object detection stages than non-templated-based object trackers, which require a constant refresh to predict and correct the tracked path.

According to the previous section, the object detector is not as strong as it is ideally wanted, and there are object detection misses that can affect an object tracking with a strong dependence of object detection. Due to this fact, the choise that fits to the problem is a template-based object tracker.

Moreover, a static template-based object tracker is not enough given the potential issues presented in Table 3.1, especially, the issues caused by object variance. For that reason, it is needed that the template can evolve according to the frame querying. For all that, seeing the availability of object trackers in the OpenCV library [69], the best option of adaptative-template-based object trackers is MOSSE tracker.

Despite having chosen the object tracker, this only works for single object tracking. Considering that there are multiple cars on road, multiple object trackers have to be deployed and executed. So as to solve this potential issue, multiple trackers objects can be created by employing the tracker class.

On the other hand, it is improper launching multiple trackers for the same object, due to the multiple detection frame-per-frame of the object detector. In consequence of that, it is required a technique to filter whether an object is being tracked. It will be discussed in the following section.

3.2.4 Object management techniques

To enhance application performance, object detection and tracking need to be improved, avoiding stage overhead, single object tracker deployment, and delimiting the occupancy ROI. In the interest to solve further issues, the following techniques are proposed:

ROI delimitation with a mask map

In Table 3.3, it was mentioned the need of passing an arbitrary ROI for quantifying *density* and *occupancy*. The tracking stage is in charge of both those variables, due to the requirement of knowing the position of each frame. This implies that the tracking is defined in a delimited region. Likewise, the bigger the detection region, the more computational resources are required for the detector stage. Therefore, two arbitrary regions must be defined.

However, those regions cannot be regular (described by a rectangle) and have different shapes, being different for each camera. So as to solve that issue, a mask map is proposed to define the different regions:

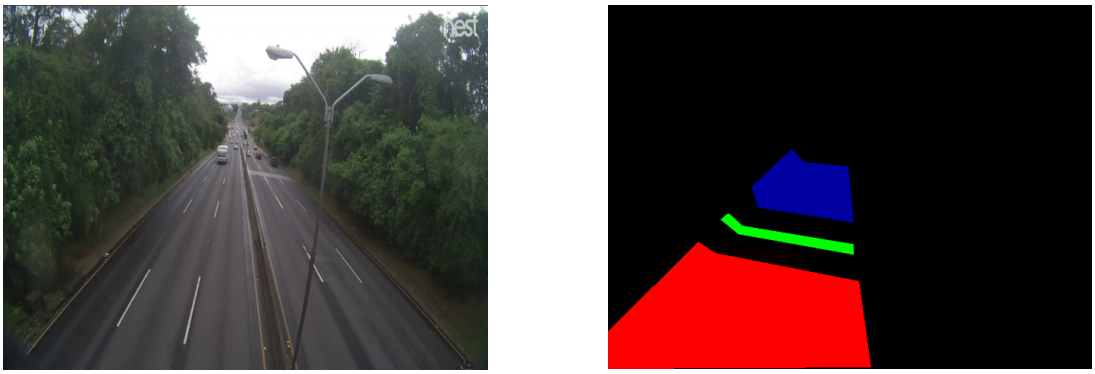
- Detection ROI
- Tracking zone
- Flow counting line
- Destroy zone

The elements listed above can be placed onto a black image same-sized to the input video sequence frame size (see Figure 3.6, where Figure 3.6a shows a frame from the input video sequence and Figure 3.6b shows its mask image). To differentiate them, the map uses the three main colours: red, green, and blue, depicted in Figure 3.6b. Besides, it is needed to define some rules to each zone, in order to increment the *flow counter*, start registering data, and destroy each tracker.

In the case of Detection ROI, it is coloured in blue. The detector can only detect objects inside this zone, even if the object is in the external rectangle where it is possible to make a detection (the Haar classifier in OpenCV expects a ROI defined as a rectangle class [70]).

For Tracking zone, it is limited from the last edge of Detection ROI to Destroy zone. The variables such as **average speed**, **density**, and **occupancy** are computed inside this zone. Also, there is a subregion named *Flow counting line*, and once an object is over it, the car counter for **flow** computation is increased by one.

The last zone is *Destroy zone*, where the tracker of the Object of Interest (OOI) is finally destroyed and the **occupancy** time counter finishes.



(a) A General Cañas camera frame **(b)** Mask map for General Cañas camera

Figure 3.6: The pair frame and mask image which the zones are established

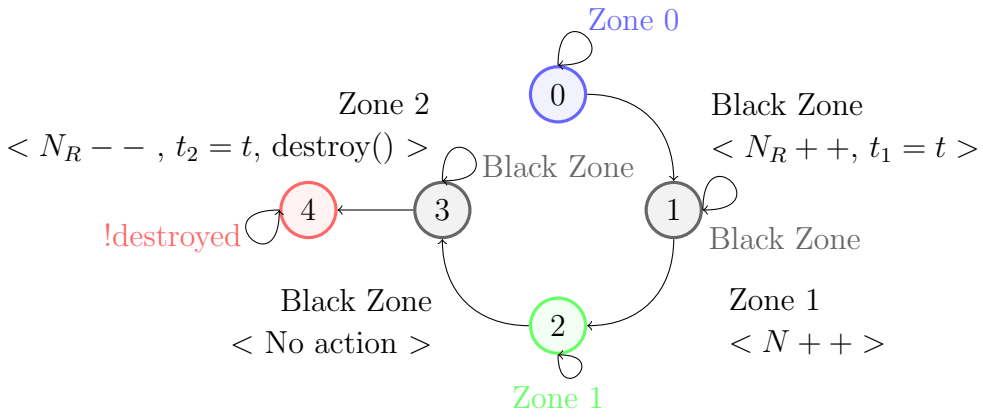


Figure 3.7: FSM-like transitions manager

Now, it is important to manage the zone transitions, which trigger some events. For instance, catch the time to register it in the occupancy start time variable, count a car when it enters to flow ounting line or Destroy zone. The transition manager is a Finite State Machine (FSM-like) programming structure, which states are illustrated in Figure 3.7.

The FSM uses as the main clock the processing period, and it modifies the variables presented in Table 3.3, where N_R is the number of cars tracked, t_1 is the start time, and t_2 is the end time for occupancy. In addition, it is important to highlight that each OOI has one variable which stores its state.

Vehicle queueing

Because each OOI has several variables which help to quantify its behaviour, it must have a data type structure which gathers all of them. This class is called, in the context of this project, *vehicle*.

However, it is important to collect all the OOI in a queue, in order to explore them element-wise and update their trackers. Also, the Measurement stage is allowed to exam-

ine each OOI and quantify the global traffic variables.

Deployed object trackers map ("black map")

Inside Detection zone, an object can be detected multiple times and can cause multiple tracker deployments. To mitigate this problem, the Centre of Mass (CoM) of each OOI in the vehicle queue and a white-filled circle centred on the CoM with a radius proportional to the object size are drawn over a black image (see Figure 3.8).

Figure 3.8: Black map (on the right) based on the frame (on the left) of a vehicle with two false-positives

With the *black map* built from the objects which are already in the vehicle queue, the module computes the CoM of the incoming detected object and then filtered. If the CoM of the incoming object is in a white colour position, the incoming object is discarded, because it is the same object, but moved slightly (in case that the object is in movement). Otherwise, a new vehicle object is created and added to the vehicle queue. Hence, the algorithm is as depicted in Figure 3.9.

An advantage of implementing this filter is that false-positives are filtered, avoiding creating ghost vehicles into the queue. However, the proportion of the object size shall be adjusted properly to minimise true-positives rejection.

3.3 Solution design

In the last section, the main techniques to detect and track objects were selected, but also some special techniques in order to handle multiple objects and optimise the application were crafted. With those tools defined, it is possible to design the application flow, including from the frame retrieving to the results reporting at the end of the application.

Figure 3.10 presents the general flow diagram proposed for traffic dynamics quantification, using CV in order to analyse each video camera, and get the values of the different variables discussed in Variables measurement. The process begins with the opening of a video source, which can be either a camera or a video file and config files. Then, all the global variables are preset with the settings read from config files. That includes setting the cascade classifier up, mask image, detection ROI and transformation parameters to get the scene vectors to estimate average speed.

Afterwards, the loop process starts retrieving the frame and detecting objects from it. The

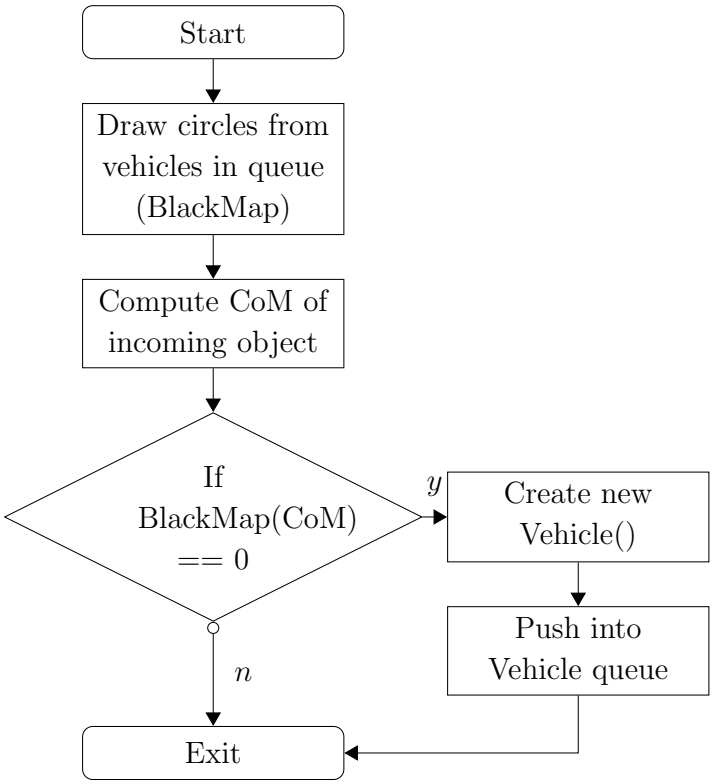


Figure 3.9: Black map filtering flow

objects detected are filtered by mask image and black map. In the case of mask image filtering, all objects outside of the blue zone are discarded, while the rest is passed to the black map filter, verifying that there is no a duplicate in the vehicle queue. Provided the object pass the filters, a new follower (tracker object) is initialised and set to track the new vehicle. During the tracker creation, a new MOSSE template is created to track the *energy* of the object in the following frames. With the tracker object launched, the vehicle object is pushed into the vehicle queue.

The next process is to refresh the FSM of each object already in the queue. This process is done based on trackers updating, computing the CoM and verifying its position against the mask image. Depending whether there is a zone transition or not, the FSM changes its state according to Figure 3.7.

Then, an object can be popped out from the queue by two reasons: the FSM gets state 4, or a timeout occurs. State 4 is given by the transition from the Black zone to Destroy zone. On the other hand, there shall be a timeout setting to avoid that a stuck object in the queue lasts forever, dragging resources unnecessarily. There are two cases where a timeout takes place: it is a false-positive, and it will not get out from the detection zone, and the object does not go near of the average speed detected from the others. In the first case, the timeout is shorter than the other one, because it needs to be configurable in order to not affect the detection of true-positives, while the other case is longer because of the distance of tracking and being a function of average speed. Both timeout followers begin at the same time.

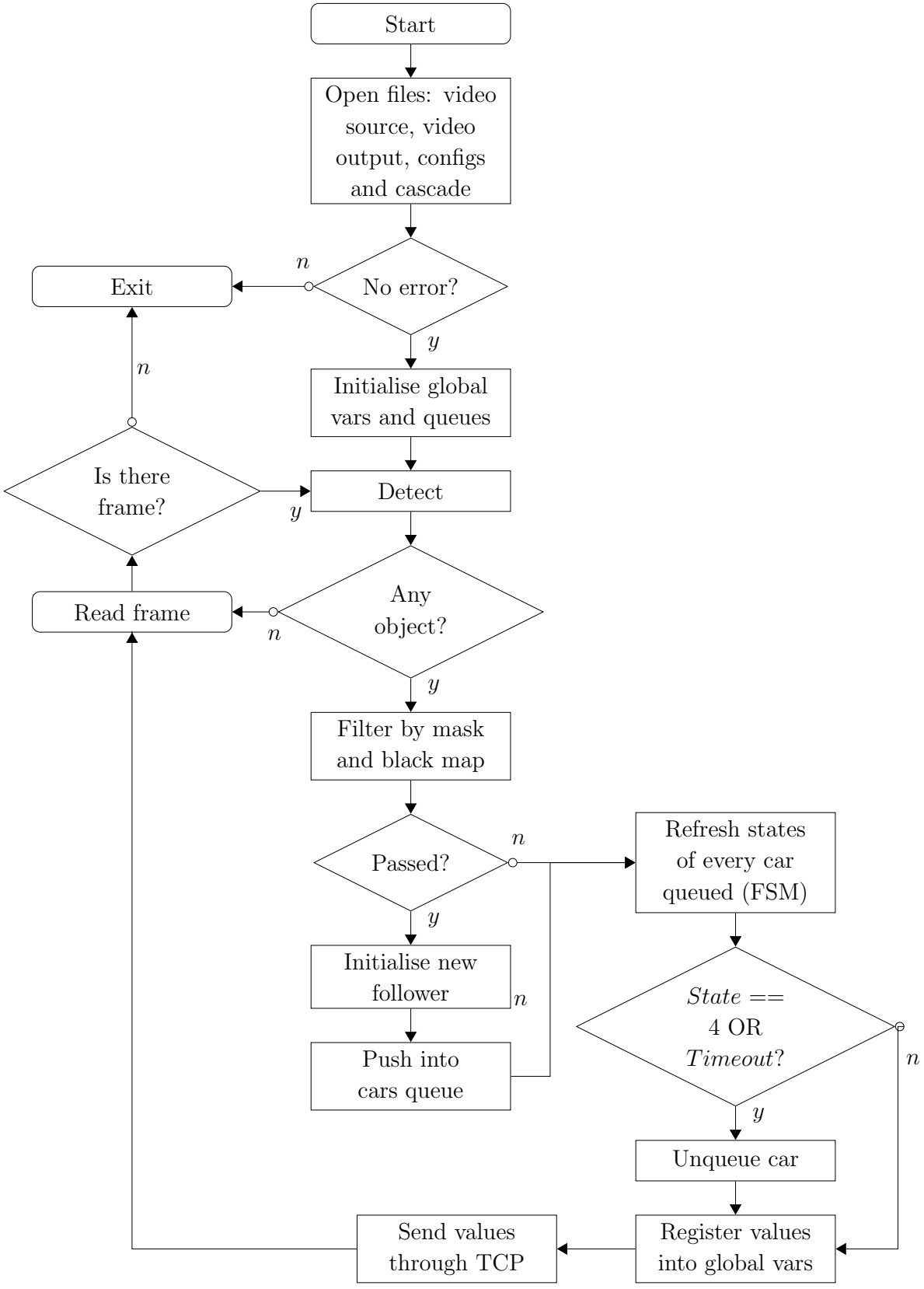


Figure 3.10: General flow diagram for measuring traffic dynamics

Depending if an object is popped out or not, the variables are measured in different ways. In the case of objects which remain in the queue, only speed and FSM state is refreshed. In the other case, the variables t_2, speed and N_R are calculated, allowing to add new samples to Occupancy and Average speed.

At the end of the loop, the results are sent through Transmission Control Protocol (TCP), depending if real-time or analysis timers ticks at the moment. If it is not the case, this process skips sending data.

Consequently of having a flow diagram which depicts the application algorithm, the block diagram presented in Figure 3.11 is a way to implement all the system, taking care of all the aspects, issues and techniques described in sections above.

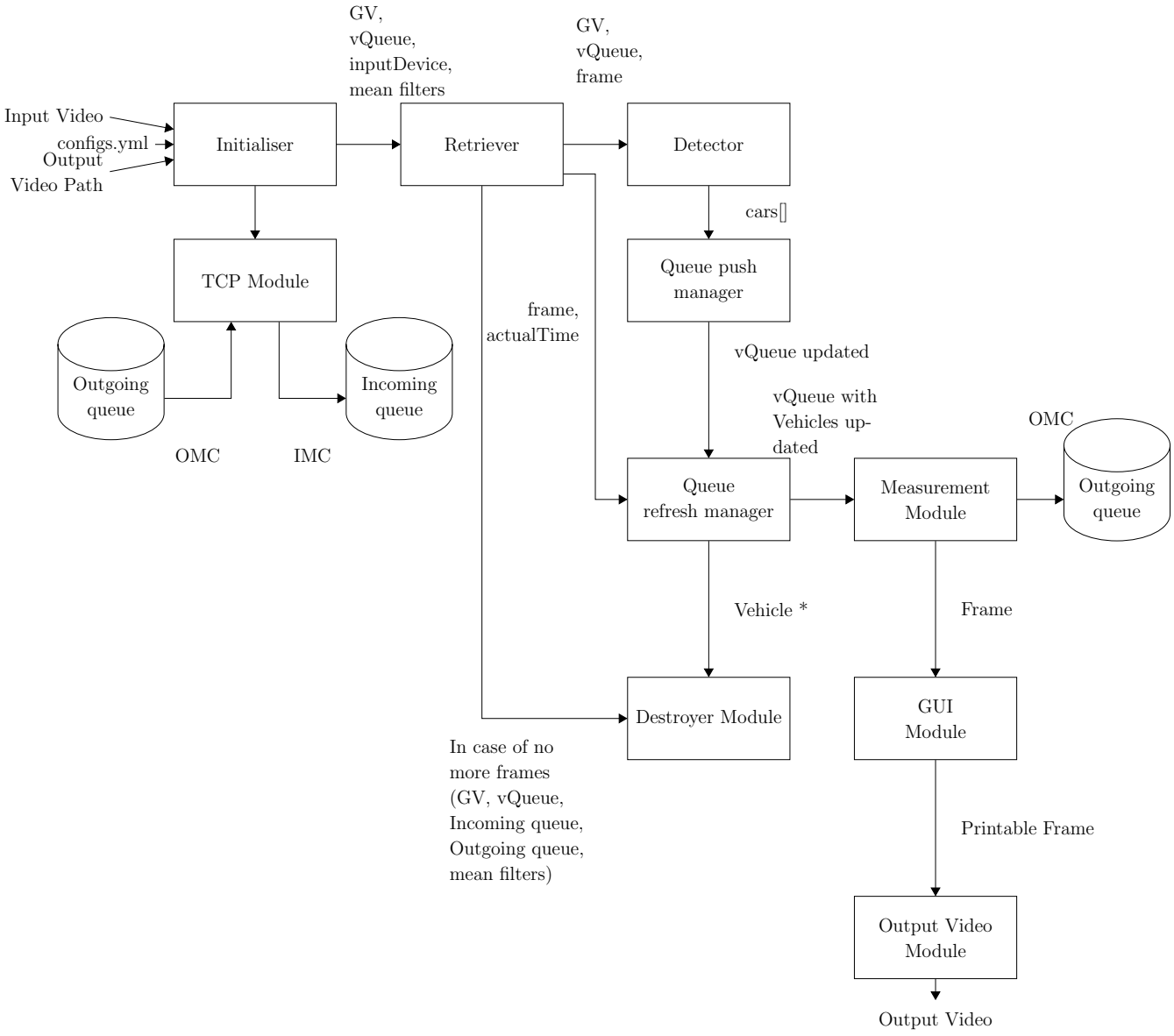


Figure 3.11: Block diagram of Traffic Dynamics Meter

Following Figure 3.11, there are multiple blocks which can be clear with the information presented above; however, to clear this block diagram, all the blocks are described below.

Initialiser

This block is in charge of initialising all parameters needed to the several processes inside the application, especially, the extrinsic parameters such as: coordinate transformation parameters, detection ROI, road length, etc. It receives three parameters:

- Input video: path of the input video in .mp4 format.
- configs.yml: includes extrinsic parameters for the selected input video.
- Output video path (optional): it is the output video assnalysed if it is enabled.

The configs file follows this structure:

- Dataset: pre-trained cascade path.
- Mask image: the path to the mask image.
- Transformation Matrix: 2x2 matrix for transforming from image coordinate system to the scene coordinate system.
- Detection ROI: rectangle-shaped detection ROI for the cascade classifier detector.
- Lane Length (L): length for computing density.
- Minimum Object Size: the minimum object size to be detected by the Haar detector. Objects smaller than this parameters are discarded automatically.
- Kernel Sizes: includes the kernel size for the preprocessor (erosion and Gaussian kernels).
- Enable Gaussian filtering: enables/disables the preprocessor Gaussian filter.
- Enable erosion filtering: enables/disables the preprocessor erosion.
- Black map circle radius: the radius is a proportion of the object size.
- Analysis interval (T_f): it is the sampling time for variables computation.
- Timeouts: it is the predefined timeout for discarding objects in case of stalling caused by either tracking errors or a vehicle whose behaviour is different to the measured in other cases (a stopped car on the road side due to a car problem). This parameter is only required if it is not possible to compute object speed or it could potentially introduce an error in the average speed results.

- Mean filter size: the number of samples for mobile mean filters to reduce the noise during average calculation.

The parameters defined above set the cv::VideoCapture, GV (meter::GlobalVars GV, a hand-made class for storing global vars defined in Global Variables) and mobile mean filters. Also, this module creates a queue for vehicles (meter::Vehicle vQueue, the data structure to handle each object easier defined in Vehicle class). A config file example is shown in Config file example.

Retriever

This module retrieves each frame from the video sequence in 8-bit RGB (Red, Green, Blue) image format. Also, it converts this frame to grayscale format before passing it to Detector and Queue Refresh Manager.

Besides, it retrieves the video sequence time (t) and updates it in GV.

Detector

This module includes two submodules: preprocessor and Haar object detector. The preprocessor enhances the grayscaled image before the detection to reduce noise and highlight borders. Then, the detector is launched and its output is filtered using the mask image to discard objects outside the blue zone and the final results are loaded to an array named $cars[]$. This needs the kernel size and enables parameters for the preprocessor filters.

Queue push manager

Before pushing the object candidates to the vehicle queue ($vQueue$), the black map filter is built from the last detected vehicles enqueued in $vQueue$, using the formula:

$$r = \frac{dw_{tr}}{2} \tag{3.9}$$

where d is the diameter of the object size given by the detection and w_{tr} is a predefined variable in the application, which establishes a proportion of the diameter that tolerates the tracking movement. This proportion is got by try and fail. The centre of the circle is the computed CoM, stored in the data object (meter::Vehicle class).

After representing each vehicle in white circles, the object candidate CoM is computed as follows (take as a reference Figure 3.12):

$$\mathbf{C} = \frac{\mathbf{P}_{obj,br} - \mathbf{P}_{obj,tl}}{2} + \mathbf{P}_{obj,tl} + \mathbf{P}_{ROI,tl} \tag{3.10}$$

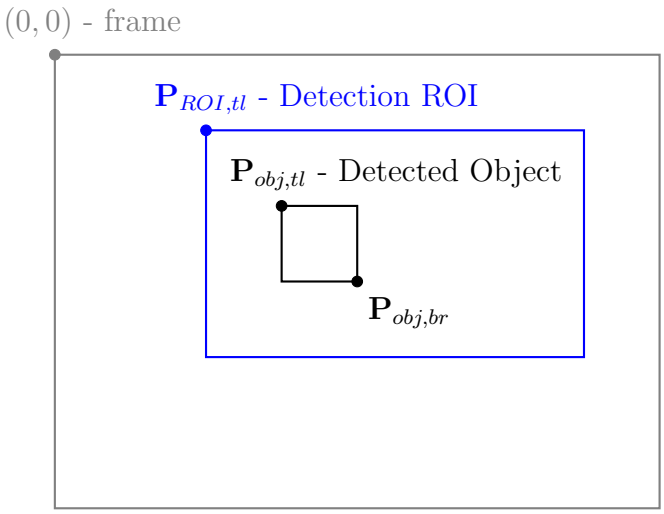


Figure 3.12: Reference points defined for CoM computing

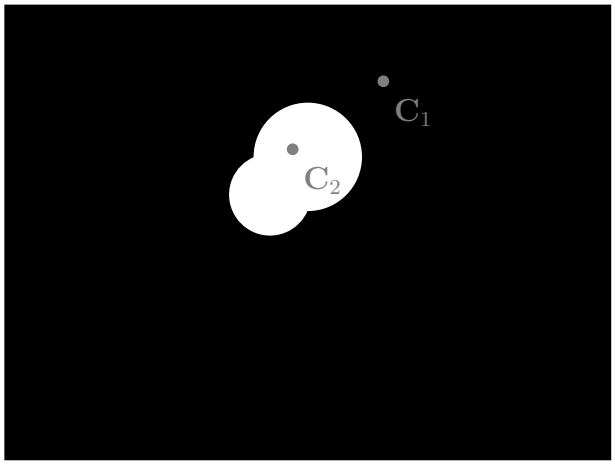


Figure 3.13: Black map filtering. C_1 passes the test, whereas C_2 is discarded

where $\mathbf{P}_{obj,br}$ is the detected right-bottom corner, $\mathbf{P}_{obj,tl}$ is the detected left-top corner and $\mathbf{P}_{ROI,tl}$ is the top-left corner of the detection ROI (to correct the Haar detector output, because the image is cropped to minimise resources). Also, both $\mathbf{P}_{obj,br}$ and $\mathbf{P}_{obj,tl}$ are relative to $\mathbf{P}_{ROI,tl}$.

As it was mentioned before, the condition to discard the object was *if the black map evaluated at* $\boldsymbol{C}$ *gets a white value, the object will be discarded.* If that evaluation $\mathbf{C}$ does not return a white color, a new *meter::Vehicle* class object will be created with the values given by $\mathbf{P}_{obj}$ and $\mathbf{C}$. The condition is graphically depicted in Figure 3.13.

Queue Refresh Manager

This module is in charge of updating the position of each vehicle already pushed in the *vQueue*. It pops out each element from the queue and executes the tracker update. Afterwards, the module computes the speed having the new vehicle position using the

transformation matrix set up by Initialiser. There are two special cases for speed computation: **linear transformation** and **non-linear transformation**.

For the linear transformation, the vectors are computed given by (3.11), whereas in non-linear transformation, it is given by (3.12).

Linear transformation from image to scene coordinate system:

$$\mathbf{x}_{scene} = \Gamma \cdot \mathbf{x}_{image} \tag{3.11}$$

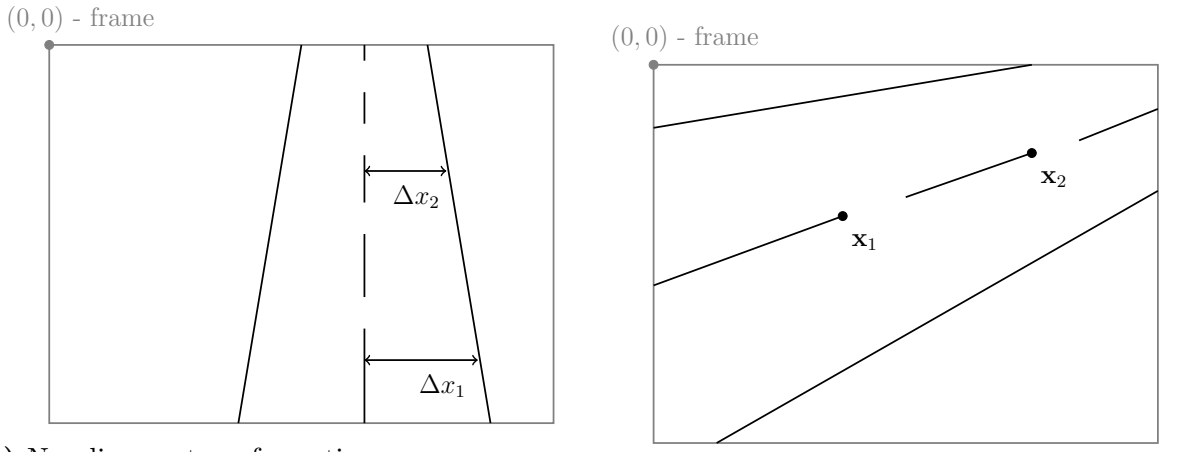
Non-linear transformation from image to scene coordinate system:

$$\mathbf{x}_{scene} = \Gamma \cdot (\Delta x_{image}, y_{image}) \tag{3.12}$$

where Δx_{image} is:

$$\Delta x_{image} = x_{image,i} - x_{image,i-1} \tag{3.13}$$

In the non-linear transformation, the speed in x is computed before transforming. This means that the speed is calculated in the image coordinate system, and then converted to the scene coordinate system. This case is only needed when Δx_{scene} depends of the y_{image} position (see Figure 3.14a). Otherwise, when the coordinates depends of x_{image} and y_{image} simultaneously (like Figure 3.14b), the transformation is linear.



(a) Non-linear transformation case: x_{scene} points does not depends linearly of y_{image} value, but the distance Δx does.

(b) Linear transformation case: x_{image} value depends of both; x_{image} and y_{image}

Figure 3.14: Transformation from image to space coordinate system cases

As Figure 3.14 illustrates, in the non-linear transformation case (Figure 3.14a, where $\Delta x_2 = \Delta x_1$), the x_{scene} value does not depends linearly of y_{image} value; however, the distance between two points at the same height is dependent of y_{image}. For that reason,

Table 3.5: Costa Rican on route signal dimensions

Detail	Urban case	National route
Lane width	3m	3.2m
Dashed lines length	3m	4.5m
Space between dashed lines	5m	7m
Distance between light reflectors	0.9m	4.5m

it is needed a special transformation technique. For linear transformation, the points depends of both image coordinates.

These transformation methods were tested before the implementation and tested qualitatively. Also, the transformation matrices were estimated, due to it was not possible to get real measurements because camera extrinsic parameters are unknown, and the impossibility of calibrating the camera. One of the estimation approaches, using lines of flight, is presented in [71].

For Γ estimation, it was taken into account the Costa Rican road regulation, which determines the lane lines dimensions, and the dashed lines length. In [72], the information about the route is presented in Table 3.5.

With the data presented in Table 3.5, it is possible to get Γ for transforming the image vectors to space coordinate system and calculate the speed frame-per-frame. For the complete information about the Γ matrix for each video used during the project, please refer to Γ matrix computations.

Measurement Module

The measurement module is in charge of computing the global results. Here; $\bar{u}$, q, λ, and $\bar{o}$ are computed. However, those values are only computed until either real-time or analysis timer ticks. For this computation, it is only needed the GV object.

In the case of analysis timer, all the values are completely computed. They are computed in one of the following methods: normal averaging, or mobile mean averaging. In the last case, the implementation for speed is as follows:

```
/*
    Speed
*/
// First, compute the value
double speedAcc = VarsToHandle->AvAnalysisSpeedAcc;
unsigned int speedCnt = VarsToHandle->AvAnalysisSpeedCounter;
double analysisSpeed = speedAcc/speedCnt;
// Mobile mean - insert a new sample
VarsToHandle->AnalysisSpeed->pop();
VarsToHandle->AnalysisSpeed->push(analysisSpeed);
```

```
// Compute the final result
for(int i = 0; i < VarsToHandle->nMean; i++)
{
    double speedAux = VarsToHandle->AnalysisSpeed->front();
    VarsToHandle->AvSpeed += speedAux;
    VarsToHandle->AnalysisSpeed->pop();
    VarsToHandle->AnalysisSpeed->push(speedAux);
}
VarsToHandle->AvSpeed /= VarsToHandle->nMean;
```

The implementation above also applies to the other variables. Besides, there are some important points to highlight:

- Accumulators and counters are in GV variable.
- Mobile mean filtering is optional. It means that the computed value in *analysisSpeed* could be the final result if mobile mean filter is disabled for the specified variable.
- The values are stored in a queue, initialised in Initialiser with zeroes and its pointer is stored in GV.
- The number of samples is set in configs.yml.
- In the case of $\bar{u}$ and λ, the average values are sampled from other average values computed during real-time triggering.
- The accumulators and counters are set to an undefined value (given by -1) at the end.

On the other hand, the real-time measuring subroutine only computes $\bar{u}$ and λ. One of the important things is that the accumulators and counters are not set to zero but only subtracted. This avoids discarding new incoming values.

Both timers do not work with the system time, because the video time is not often equal to the system time in an off-line implementation. For that reason, the actual video time is stored in GV to have control over the real time.

The process finishes sending the values through TCP. However, the communication is done in another thread to avoid bottlenecks during the baseline execution. To send the packages through TCP, therefore, the module inserts a message object into a communication queue named *Outgoing queue*.

TCP Module

This module has two threads inside: one to receive new data and another one to send data. To couple both threads to the baseline thread, there are two queues which contain

received messages and messages to send. Those messages are packed in a new datatype with the characteristics of the message.

To avoid an abuse of resources, each subroutine is executed with a delay of $100ms$, sufficient to deal with the real-time and analysis reports and get instructions.

GUI Module

GUI Module is in charge of adding visual indications to the output video before its writing. Basically, it can put:

- Result reports
- Object state
- Circle the objects
- Overlay the mask image to the video

An example of the full features of GUI Module is presented in Figure 3.15.

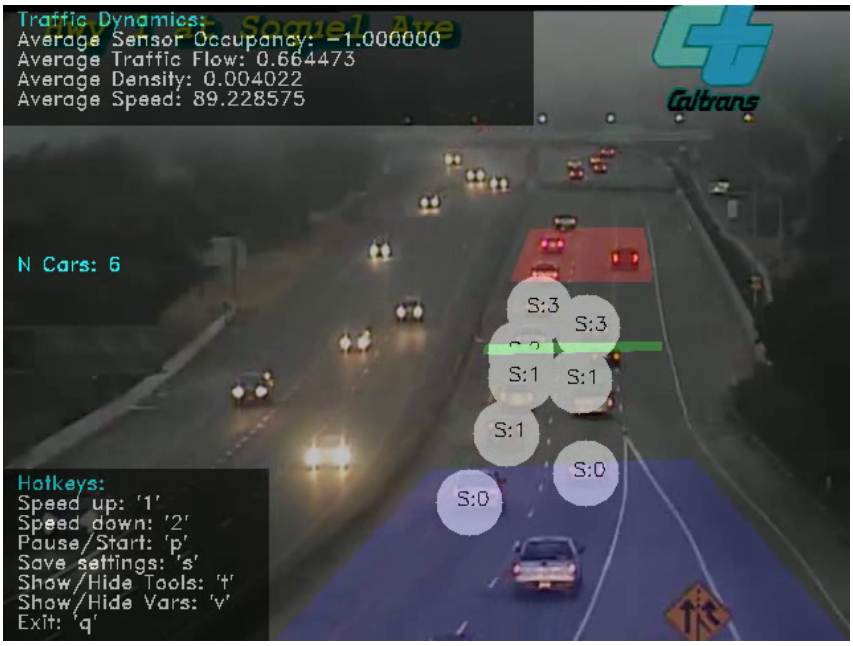


Figure 3.15: Video output with GUI Module activated. Results are shown above and each tracked object is marked by a white circle with their status

This module is only used for debugging and visual demonstrations and makes no sense to enable it during real measurements because of its resource consumption.

Destroyer Module

There are two important memory spaces to release from the *meter::Vehicle*. The first is the tracker class due to its use of dynamic memory to store the adaptative kernel, and the A and B matrices, which are adjusted each tracker update. For more information about the tracker, please look at [69].

The second memory space is the class itself. It is released normally using *free(var)*.

Finally, to see the blocks detailed diagram, please refer to Detailed block diagrams.

3.4 Running the baseline implementation software on PC and SoC

The application performance on a PC (x86-64 architecture) is different from running it on a SoC embedded system (ARM Cortex-A9) [73], due to the different microarchitecture. One of the important differences between those two architectures is mainly the absence of a hardware FPU (Floating Processing Unit) in Zynq-7000 SoC. Hence, the performance when there are operations with floating point numbers might have a decay. Also, there can be an effect due to the different instructions.

3.4.1 Profiling techniques

To compare both architectures, the application is compiled with OpenCV 3.1.0, compatible with both architectures, running Ubuntu 16.04 on PC and Petalinux 2017.4 on a Zedboard. Due to incompatibilities during the development, MOSSE tracker is implemented manually taking as reference the OpenCV 3.4 source code. Also, there is a profiler to analyse each function inside the application, allowing to get average and standard deviation of time on the way. The profiling class implementation is shown in Profiling classes.

To profile the whole application, it is important to know how the application is made. Firstly, there is a profiler which takes the total execution time, being possible to take several samples to strengthen the profiling results. Then, each block is profiled, averaging their execution time in different situations: the number of cars in the detection zone, and the number of cars tracked. Table 3.6 specifies which blocks are affected by those numbers.

Each average will be computed as a weighted average, given by:

$$\bar{a} = \sum_{i=0}^{N_{cars}} \bar{a}_i \cdot w_i \tag{3.14}$$

Table 3.6: Affectation of the several blocks to the number of cars in Detection Zone and in *vQueue*

Block name	**Affected by the number of cars in Detection Zone**	**Affected by the number of cars tracked (N_R)**
Retriever	No	No
Detector	Yes	No
Queue Push Manager	Yes	Yes
Queue Refresh Manager	No	Yes
Measurement Module	No	Yes
Destroyer Module	No	Yes

where w_i is the global weight given by the division between the number of frames with the number of cars i divided by the total frames sampled, and $\bar{a}_i$ is the average given i number of cars, either in Detection Zone or being tracked.

Moreover, the internal blocks illustrated in Detailed block diagrams are also measured to get information about the problematic parts inside each module. Thus, it is possible to get which instructions and functions overuse computational resources.

Finally, each profiling result involves 20 system runs, in order to have an acceptable number of repetitions for statistical analysis.

3.4.2 Traffic Dynamics Meter error quantification

In this section, the software validation is developed, verifying if its results are correct and inside the tolerance ranges. To get theoretical values, they are extracted manually from the videos, and then they are contrasted to the experimental values, given by the total average of the results. One of the reasons to compute a total average is because of the sampling time needed to have the final results. In the application is less than 15 seconds, whereas the recommended is around 1 minute, mentioned in sections before.

The theoretical values are extracted based on Table 3.7.

According to Table 3.7, the case of $\bar{u}_T$ and $\bar{\lambda}_T$ must be calculated from other variables, due to the lack of exact information. However, for practical reasons, they are acceptable for software validation.

Considering the theoretical and experimental data, Table 3.8 presents the results acquired during the comparison, summarising the errors got on the traffic dynamics results. It is important to highlight that the number of cars detected ($N_{E\to T}$) works as a verification variable which helps to explain errors on some of the traffic dynamics results (which are $\bar{u}_{E\to T}$, $\bar{\lambda}_{E\to T}$, $\bar{q}_{E\to T}$, and $\bar{o}_{E\to T}$).

Taking into consideration them, the source of errors has to be explained in order to control

Table 3.7: Theoretical values for results validation

Value	Calculation method
Recording length (T_T)	From video file
Number of cars (N_T)	Manually counted
Distance (L_T)	Based on the number of light reflectors, lane dashed lines, etc.
Occupancy ($\bar{o}_T$)	Average of raw report from application. The application reports the time when a car crosses the two reference lines. Then, the following equation is applied: $\bar{o}_T = \sum_{i=1}^{N_T} \frac{t_{2,i} - t_{1,i}}{N_T}$
Flow ($\bar{q}_T$)	Given by: $\frac{N_T}{T_T}$
Average speed ($\bar{u}_T$)	Given by: $3.6 \cdot \frac{L_T}{\bar{o}_T}$
Density ($\bar{\lambda}_T$)	Given by: $3.6 \cdot \frac{\bar{q}_T}{\bar{u}_T}$

Table 3.8: Exact architecture error quantification referred to theoretical values

Variable	General Cañas	Multiplaza del Este	Soquel Avenue
$N_{E \to T}$	11.11%	0.00%	28.57%
$\bar{u}_{E \to T}$	1.25%	0.27%	7.21%
$\bar{\lambda}_{E \to T}$	4.76%	1.75%	7.85%
$\bar{q}_{E \to T}$	5.13%	6.71%	17.04%
$\bar{o}_{E \to T}$	1.92%	7.81%	5.76%

the errors and minimise them. There are four possible sources of errors because of design decisions: **detector errors**, **tracking errors**, **training errors**, and **extrinsic parameter errors**. Detector errors are mainly caused by true-negatives and false-positives in the object detection. A true-negative implies that a car might not be followed and analysed, affecting parameters which depend on the number of cars ($N_{E \to T}$), such as flow and density. Also, errors during object tracking are inside the group of application errors and happen when a tracker cannot follow an object. In this case, tracking-dependent variables are affected, such as average speed, flow, and occupancy.

Training errors are also linked to errors in object detection, affecting flow and density. These errors can be mitigated with a custom-trained dataset for each camera and scene. However, retraining with a large number of cars is time-expensive. For that reason, from a design perspective, trackers are launched once an object is detected the first time. This minimises the dependence on object detection and reduces the impact of errors caused by generic pre-training.

Despite having object detection and tracking working properly, there are errors caused by external agents, such as errors in extrinsic parameters, camera calibration, camera distortion, and others. These errors are grouped into *extrinsic parameter errors*. Ba-

Table 3.9: Results affectation by different error sources

Variable	Detector errors	Tracking errors	Training errors	Extrinsic parameter errors
N_E	High	Low	Moderate	None
$\bar{u}_E$	Low	High	Very low	High
$\bar{\lambda}_E$	High	Moderate	Moderate	Moderate
$\bar{q}_E$	High	High	Moderate	None
$\bar{o}_E$	Moderate	High	Low	None

Table 3.10: Platforms under test used for profiling

Resource	PC	Zedboard ZYNQ-7000-based
Processor	Intel Core i3 2310M (Dual-core)	Dual-core ARM Cortex-A9
Processor Speed	2.1GHz	866MHz
RAM	8GB	512MB
Storage	HDD 320GB 5400rpm SATA3	8GB SD Class 10
Operating System	Ubuntu 16.04 LTS 64 bit	Petalinux 2017.4
OpenCV version	3.1.0	3.1.0
Threading library	pthread	pthread

sically, the impossibility of having exact extrinsic parameters and measuring the scene properly yields to variable errors. Average speed is one of the most affected by this kind of errors, followed by density, due to its dependence to scene measurements. Having a great transformation matrix, and the camera calibrated might reduce the errors in those variables.

Table 3.9 synthesises error sources described above and their impact to results, based on the design flow.

3.4.3 Traffic Dynamics Meter profiling

The following profiling results are taken without any improvement on the two platforms of interest (see Table 3.10). Also, the profiling tests use 3 videos, previously studied, whose cameras are located in: Soquel Avenue Highway 1, General Cañas Highway, and Multiplaza del Este. Those videos were selected because of their differences in the number of cars and different vehicle behaviour.

General Cañas

This video is characterised by its number of cars, that are fewer compared to the others. However, the average speed on road is over $60km/h$, and there are many true-negatives at detection level, but mitigated by tracking. Because of the errors in Detector, it is possible to have more affectation caused by the number of cars tracked compared to the number of cars in Detection Zone.

The video properties are:

- **Size:** 640x480
- **FPS:** 30 fps
- **Duration:** 42 seconds
- **Minimum object size:** 20x20
- **Maximum number of cars in Detection Zone:** 3
- **Maximum number of cars tracked:** 3
- **Lane length:** 67.2m

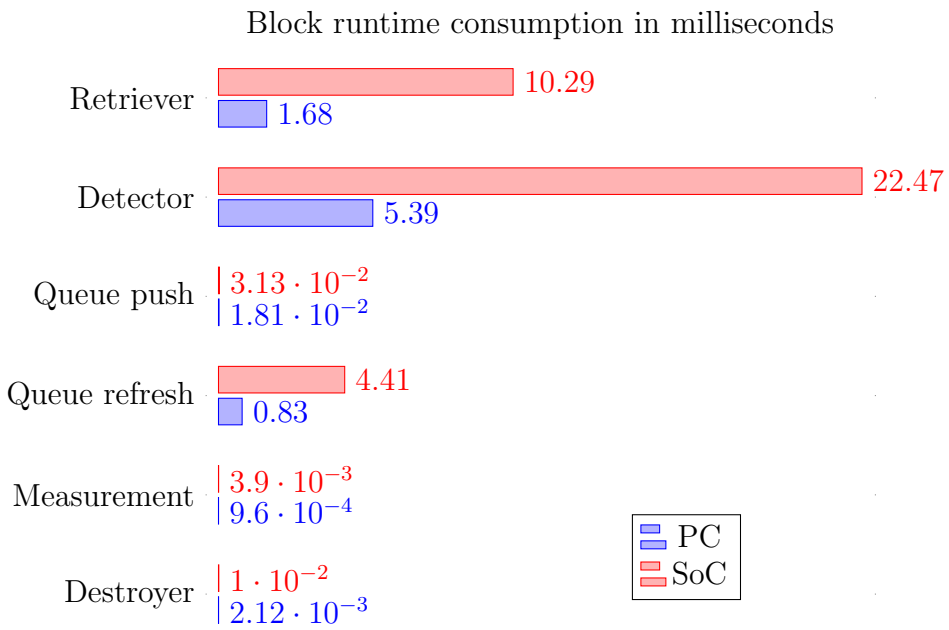


Figure 3.16: Profiling results after running the application on the testing platforms: General Cañas

Figure 3.16 presents the time consumption results for General Cañas video sequence. On PC, it takes around 7.92 milliseconds to process a frame, whereas on SoC, it takes more

than 37 milliseconds. Therefore, it is viable to process the video on a PC online, while on SoC it is suitable.

The blocks which consume more resources in both platforms are: **Retriever**, **Detector**, and **Queue Refresh Manager**.

Multiplaza del Este

This video sequence is the more balanced in the number of cars detected and tracked. Both modules perform better processing this video compared to the others two video sequences.

The video properties are:

- **Size:** 640x480
- **FPS:** 30 fps
- **Duration:** 49 seconds
- **Minimum object size:** 50x50
- **Maximum number of cars in Detection Zone:** 4
- **Maximum number of cars tracked:** 4
- **Lane length:** 78m

Figure 3.17 shows the results got after profiling with Multiplaza del Este video sequence. In this case, the greatest consumption comes from Refresh Manager. That is caused by an increase of the tracked cars compared to *General Cañas* video track (4 and 3 respectively). Also, there is a decrease in the consumption of **Detector**. This can be due to a better filtering, where the possible candidates (falses) are discarded in early stages.

It is important to highlight that the size of the tracked elements affects tracker consumption. In the case of *Multiplaza del Este*, the minimum object size expected is 50x50, whereas in *General Cañas* is 20x20. For that reason, Queue Refresh Manager consumes much more resources in *Multiplaza del Este* than in *General Cañas*.

The runtime for this video sequence keeps quite similar on PC, with 6.89 milliseconds, whereas on SoC, it increased to 43.08 milliseconds. Hence, it is not possible to process the video sequence online on SoC as well, and the application needs to be enhanced.

Soquel Avenue

This is the most challenging video for the application, because this video sequence has the largest area covered by the sensor. That implies that there will be more cars detected and tracked, and it increases drastically the consumption.

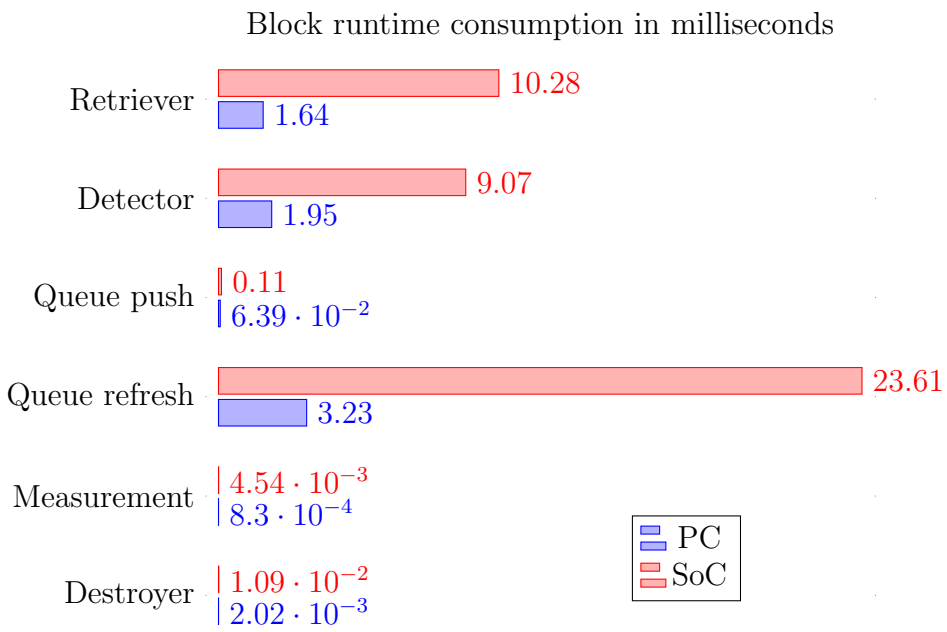


Figure 3.17: Profiling results after running the application on the testing platforms: Multiplaza

The video properties are:

- **Size:** 640x480
- **FPS:** 25 fps
- **Duration:** 84 seconds
- **Minimum object size:** 50x50
- **Maximum number of cars in Detection Zone:** 3
- **Maximum number of cars tracked:** 10
- **Lane length:** 1492m

Figure 3.18 presents the profiling results for *Soquel Avenue*. The highest consuming modules still are **Detector**, **Retriever**, and **Queue Refresh Manager**. Compared to the two other videos, the application has to deal with a large number of vehicles tracked. That implies that the **Queue Refresh Manager** consumes more than the usual. Also, **Detector** increased its consumption, because of the quantity of cars detected in the video, and the negatives which affects the detection, consuming some analysis time in early stages of the detector cascade.

The minimum object size set up for **Detector** is the same as *Multiplaza del Este* video, intensifying the impact in resources consumption.

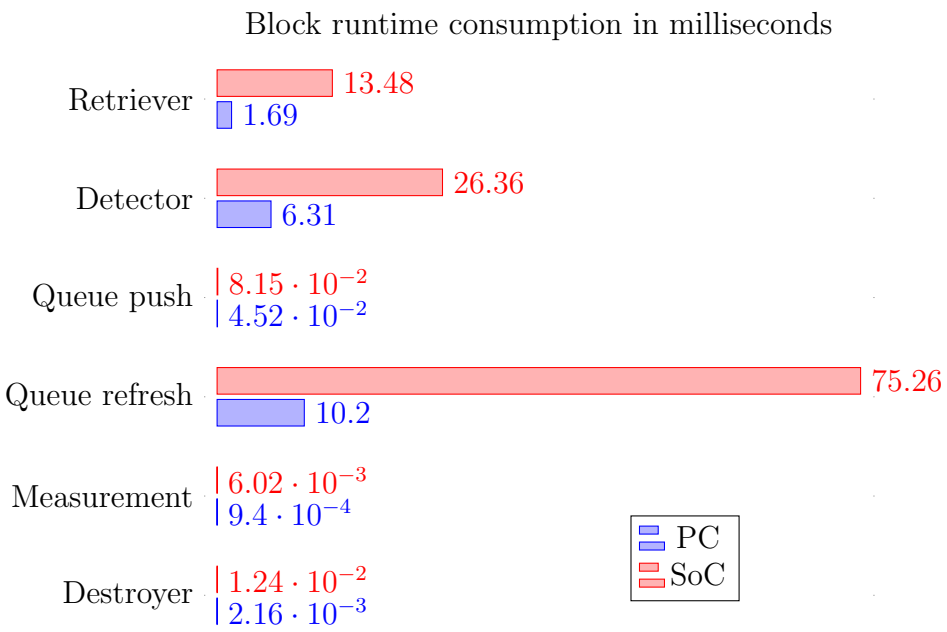


Figure 3.18: Profiling results after running the application on the testing platforms: Soquel Avenue

Despite its consumption, the application is still online runnable on PC with 18 milliseconds per frame. However, running the application using this video should be less possible on a SoC, because the processing time increased to 115 milliseconds (more than 2 times the available time per frame, considering 25 fps).

Review

The most problematic modules, due to their dependence to the number of cars in Detection Zone and tracked, are **Detector** and **Queue Refresh Manager**. However, **Retriever** is the only one which keeps the consumption almost stable and there is no dependence on any variables.

Figure 3.19 demonstrates the relevance of the modules mentioned above, whose results are referred to the software application consumption on the SoC. Therefore, it is crucial to optimise **Detector** and **Queue Refresh Manager**, because of their predominance over the rest of modules in terms of consumption. Likewise, it is important to limit the observation area seen by the camera, so as to reduce the number of cars that can be tracked and detected, and to avoid overuse of resources as it is seen in *Soquel Avenue* video, that is critical compared to the others. This will consequently help to reduce the consumption in both modules.

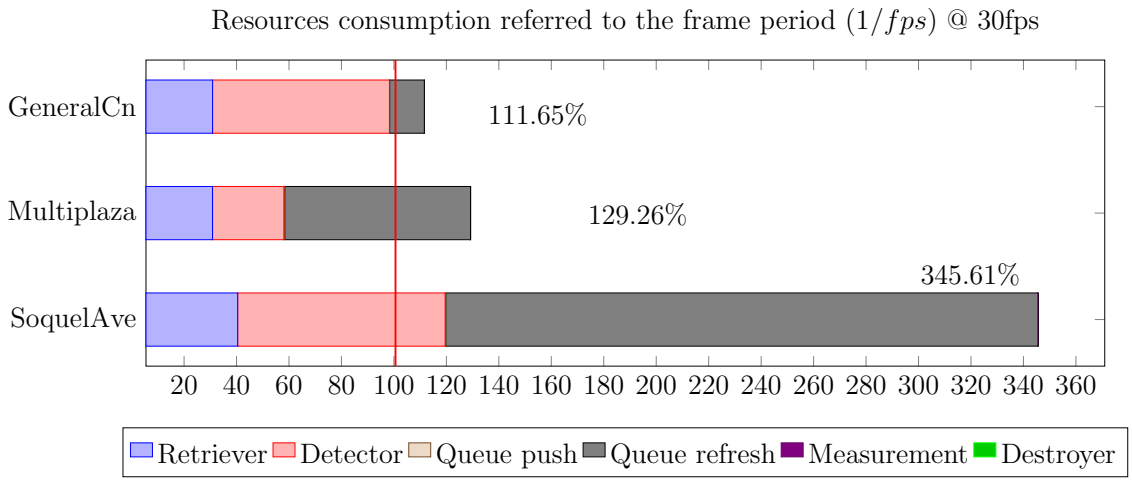


Figure 3.19: Resources consumption per block referred to the frame period available to process each image

3.5 Conclusion

According to the results got from the error quantification and software profiling, the application performs inside the result tolerances, and can be executed online on a PC. However, the application cannot be run online on a Zedboard, that in this case, it is the target board. As a result, the application is not suitable to be implemented on a resource-constrained platform, where the goal of implementing it on a low-power system is not possible yet. In spite of online execution is not mandatory, but it is desirable. With this cleared, the application already accomplished the system requirements.

Some of the problematic modules due to their resource overuse are Detector, Queue Refresh Manager, and Retriever. Those modules shall be optimised so as to enhance the application performance and keep the processing time below the 100% of the frame period ($1/fps$). Based on Figure 3.16, 3.17, and 3.18, **Queue Refresh Manager** represents the module whose consumption increased drastically with the number of cars tracked. To mitigate the effect of N_R on the application performance, it is recommended to keep low the area of interest, limiting the maximum number of cars in the analysis area up to 5.

On the other hand, there have been found error sources which affect the results, causing deviation from the theoretical values. Those errors come from detector and tracker failures, generic pre-trained datasets, and extrinsic values estimation.

Therefore, the conclusion points are:

- Application result errors are under the error tolerance.
- Tracking zone must be reduced to track a maximum of 5 cars.
- Application cannot be run online on Zedboard, and its performance must be enhanced.

- Due to the pre-trained dataset used for the cascade classifier training, there are true-negatives and false-positives which affect the detector performance (affectation greater than 25% in Soquel Avenue).
- Due to the extrinsic parameters estimation, it is possible to have errors during coordinate system transformation, because of transformation matrix approximation.
- The strategy *once detected, tracker launched* (a car only needs to be detected once to be tracked in the following frames) worked properly, reducing the impact of detection failures and training failures.

In the following chapter, the optimisations are studied in order to enhance application performance using software enhancements and approximate computing.

Chapter 4

Optimisation

Traffic dynamics meter section concluded with the need of optimising the software application in order to make it more suitable to be implemented on the target platform, in this case, a Zedboard with a ZYNQ-7000 SoC.

Likewise, the application profiling revealed the problematic modules because of their resource overuse, and its dependence on the number of cars, either in Detection zone or Tracking zone (vehicles tracked). If the analysis area increases, the number of cars in case of a traffic jam will rise to the maximum, and it is possible to have consumption issues in those modules.

Taking into consideration those aspects, this chapter analyses each module instruction-per-instruction, seeing the computational weight of each of them, so as to look for potential optimisations applying either software techniques or approximate computing. Also, it presents a series of results, comparing the evolution of the software application with each optimisation.

4.1 Consumption tendencies

In order to analyse each element in detail, the analysis is divided by each problematic module. Also, each measurement is repeated 10 times to have a stable and realistic average.

4.1.1 Retriever

Retriever module is composed by two instructions (see its block diagram in Retriever block diagram appendix):

- Frame retriever
- Crop

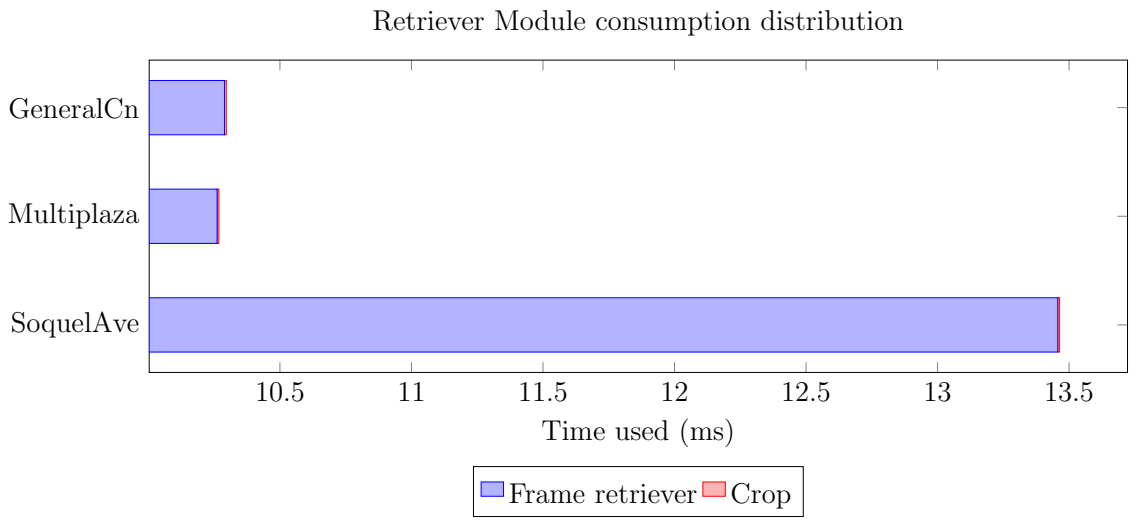


Figure 4.1: Retriever Module Profiling - Without optimisations

The first function (Frame retriever) is in charge to grab and decode each frame from the video. The second module is Crop, and makes the frame ready for Detector, allowing to have only the piece of the frame which belongs to Detection zone.

Figure 4.1 presents the profiling results of Retriever module. According to this figure, Frame retriever consumes more resources in *Soquel Avenue* video sequence than the others. This phenomenon is caused by decoding issues. However, the biggest problem in this module is that video retrieving consumes practically all the resources of this modules. Hence, this module cannot be optimisable by software, and accelerating by hardware implies streaming video amongst soft-core and FPGA.

Also, there is no possibility of affecting this consumption by limiting the number of cars analysed in either Detection or Tracking zone.

4.1.2 Detector

Detector is in charge of identifying whether an object is a vehicle or not. If Detector detects a vehicle, it creates a vehicle object class. It depends only on the number of cars in Detection zone; however, it depends also on the cleanness of each frame. That means if an analysed frame contains false candidates, Detector module has to discard all of them using more than one cascade stage. For this, a noisy frame can make Detector consume more resources than normally it does.

The structure of Detector is:

- To grayscale
- Erode
- Gaussian

- Haar + Equalise Hist
- Mask filter

Frame preprocessor modules are grayscaling, erosion and gaussian blurring. Those modules help to prepare each frame before being analysed by the Haar cascade classifier, reducing noise and highlighting those details useful for feature extraction. Also, running the instruction *DetectMultiScale* requires to equalise the colour histogram to work correctly. After detecting the potential candidates; then, Mask filters the candidates, defined in the past chapter.

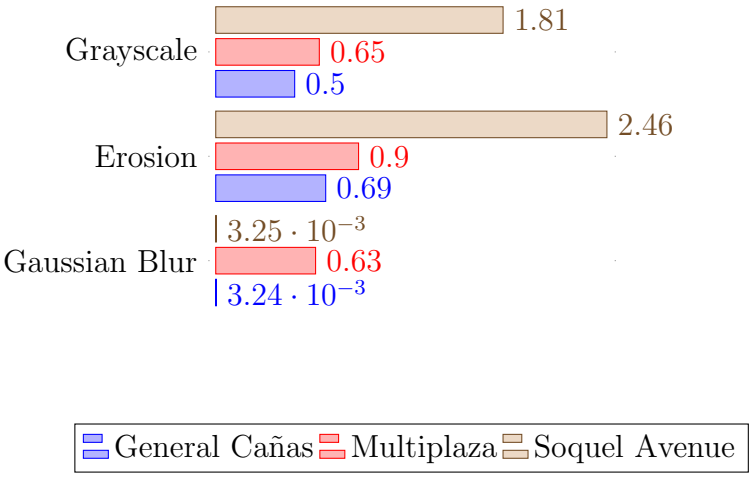


Figure 4.2: Preprocessor stages consumption (in *ms*) for each sample videos, tested on a Zedboard

Figure 4.2 shows the results after profiling the preprocessor module. It is important to highlight the differences that can influence the presented profiling results, especially as an effect of changing Detection ROI (see Table 4.1).

Table 4.1: Video samples detection settings

Video	**Detection ROI size**	**Erosion**	**Gaussian blur**
General Cañas	135x95	Yes, kernel = 3x3	No
Multiplaza	185x95	Yes, kernel = 3x3	Yes, kernel = 3x3
Soquel Avenue	429x124	Yes, kernel = 3x3	No

According to Table 4.1, only one of the videos uses Gaussian blurring, and all of them use a 3x3 kernel to describe both preprocessor stages (erosion and Gaussian blurring). However, the most important difference shown in the table is the Detection ROI size. *General Cañas* video settings set Detection ROI size in 12825 pixels, similarly to *Multiplaza del Este* with 17575 pixels, whereas *Soquel Avenue* ROI size is about 3 times bigger than the others.

In consequence of the different Detection ROI sizes, it is possible to see in Figure 4.2 that preprocessor runtime, just observing the same number of cars for each video, is 3

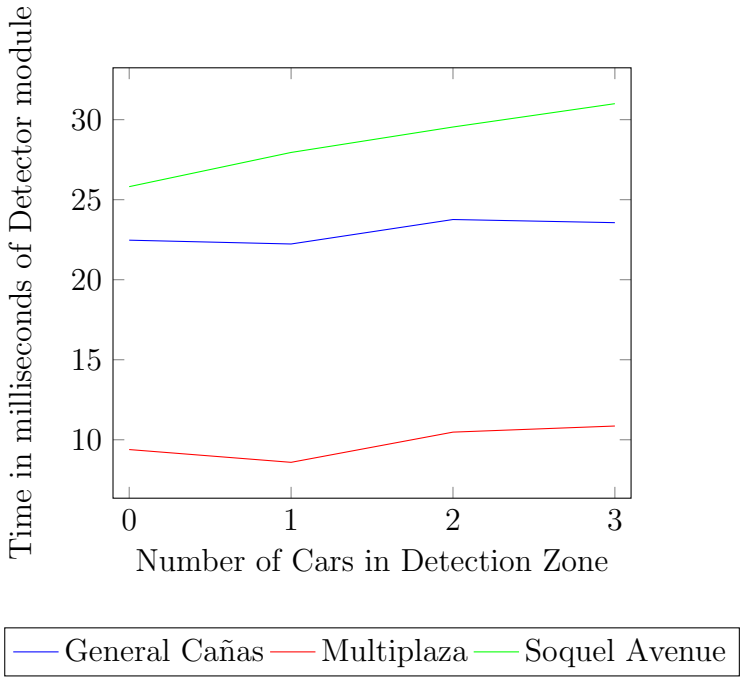


Figure 4.3: Tendency of Detector runtime respect of number of cars in Detection Zone

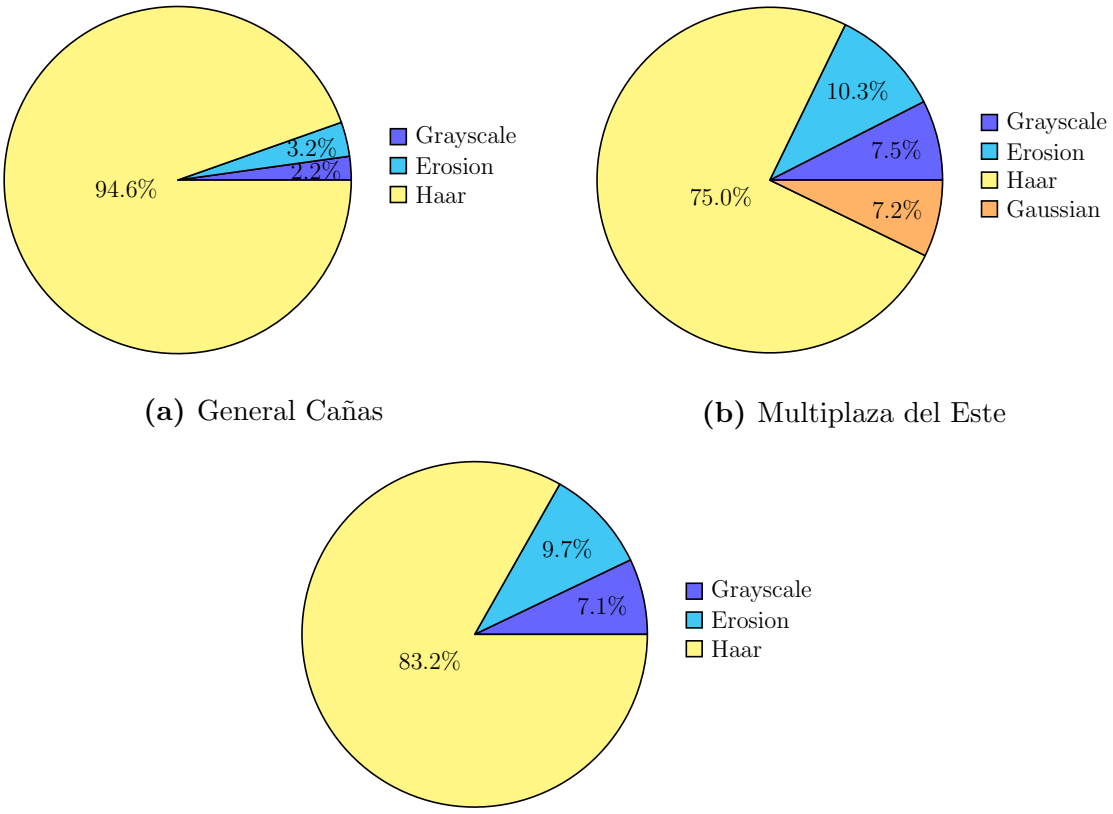
times slower in *Soquel Avenue* than *Multiplaza del Este* and *General Cañas* video samples. Therefore, the size of Detection ROI is directly proportional to the amount of time taken by each preprocessing stage under the same conditions.

On the other hand, considering that the time consumed by preprocessor is almost constant, it is possible to reconstruct the Detector module tendency given the number of cars in Detection ROI.

Figure 4.3 depicts such tendency, showing that is not completely proportional. Nevertheless, there is an increase in time consumption at the same time that the number of cars grows. This is explained by the cascade classifier structure itself, due to an increment in the number of cars implies the use of more cascade stages in the classifier. Now, considering that the image is noisy, the variance because of the increase in the number of cars is not too significant, due to image noise causes that more cascade stages have to be executed to discard an object. Therefore the change in the consumption because of the cascade classifier execution tends to not change proportionally to the number of cars detected.

Figure 4.4 shows Detector module profiling at the function level, whose percentage results are respect to the total Detector module consumption and represents how much of the runtime is consumed by the whole module is used in each function. It is possible to see that the Haar cascade classifier detection consumes more than 75% of the total module time, being approximately 20 ms in *Soquel Avenue* and *General Cañas*. Also, it is possible to see that the preprocessor consumption is about 25% in case that all its stages are enabled.

Besides, the consumption of Haar detection varied from a video sample to another, basi-



(a) General Cañas

(b) Multiplaza del Este

(c) Soquel Avenue

Figure 4.4: Detector module profiling, respect of the total Detector consumption, analysing different video samples. Only *Multiplaza del Este* has Gaussian blurring enabled

cally caused by the noise influence.

4.1.3 Queue Refresh Manager

The software application has some features that have been disabled when profiling. This is because the software under test is the minimum viable. Thus, the Graphical User Interface (GUI), and video output are disabled. In Queue Refresh Manager block diagram in Appendices presents the block diagram of this module. The process is basically described in that block diagram, where the process is a loop which removes from the queue each vehicle from *vQueue*, updates its tracker, analyses the new status, and if the status implies vehicle deletion from *vQueue*, sends the object to Destroyer. If not, push the object into *vQueue* again.

This loop tends to make a consumption of $O(n)$ in this module, where n is the number of cars in *vQueue* and $O(n)$ function represents a linear function whose result is the average runtime required to complete the execution, as Figure 4.5 shows. This figure illustrates the consumption tendency being almost linear. Besides, when the number of cars tracked is zero (*vQueue* empty), there is a residual consumption. This is caused by the pass of

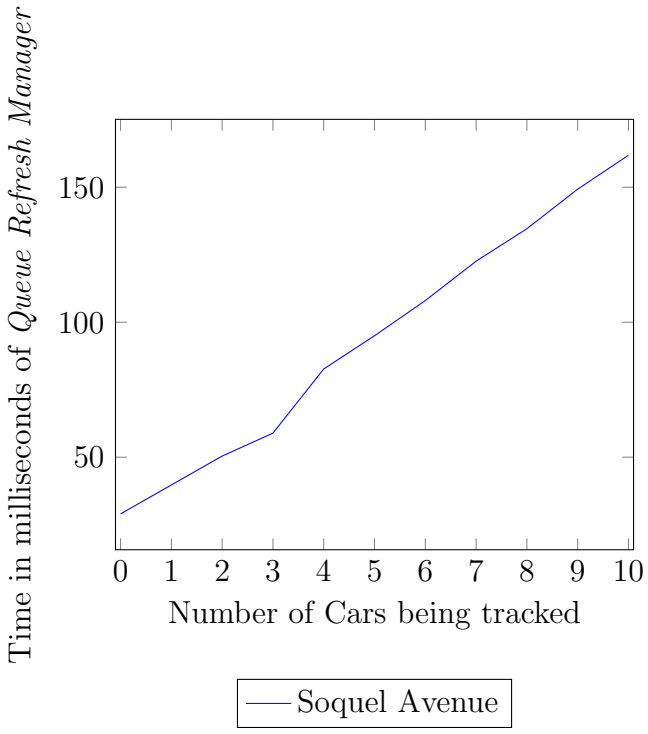


Figure 4.5: Consumption tendency of *Queue Refresh Manager* given by the increase in number of cars tracked

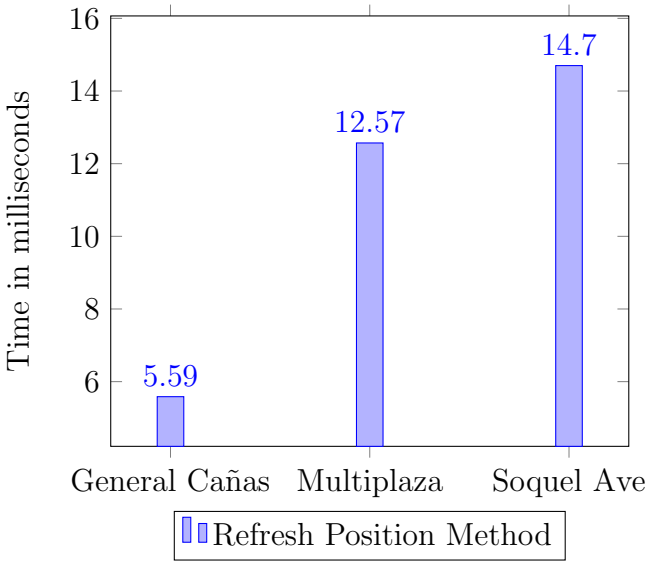


Figure 4.6: Average time consumption in *Refresh Position Method* for one object

the cascade classifier (now deprecated in this module). It is consequently severe and shall be corrected.

On the other hand, State analysis and Show Center of Mass (see Queue Refresh Manager block diagram) have a low impact on computation; hence, they are not taken into account in profiling. Thus, the only block to analyse is *Refresh Position Method*, whose results are shown in Figure 4.6.

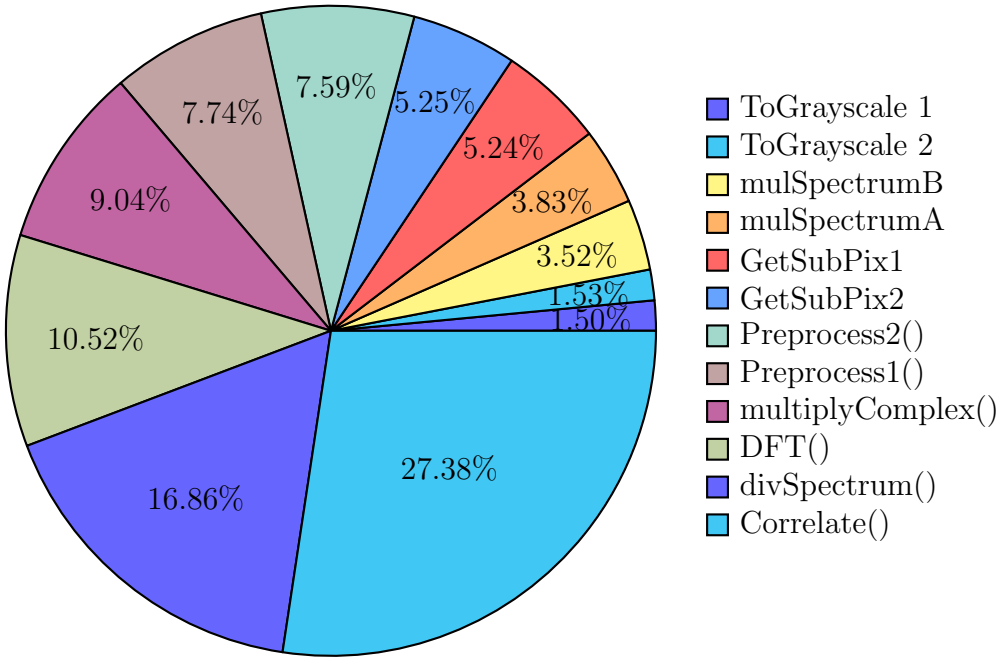


Figure 4.7: Consumption distribution in MOSSE tracker *update* subroutine

The results suggest a change in time consumption due to Minimum Object Size. In the case of *General Cañas*, this video sequence has the lowest minimum object size, whereas the other two video sequences have the same minimum object size. In fact, these parameters affect this module consumption substantialy; however, changing these parameters imply changing the camera position, and it is not possible given those cameras are public and out of the research control.

Another aspect important to analyse is the composition of Refresh Position Method. Basically, it is formed by a MOSSE tracker object, which initialises or updates depending on whether the tracker has been already initialised or not. Nonetheless, most of the impact is due to the Update subroutine.

Figure 4.7 shows the consumption of update subroutine (see the diagram block in MOSSE tracker update subroutine in Appendices). It is possible to see that there are two *grayscaling* functions consuming part of the resources, and they are optimisable. Also, most of the resources are mostly spent by the new filter reconstruction (New filter H*) functions, such as DFT, spectrum division, complex matrix multiplications, and spectrum multiplications. Due to their complexity, it can be optimised, affecting the 50% of Update subroutine. Also, it is suitable to optimise the Correlation subroutine.

It is important to emphasise that the *divSpectrum* is a hand-crafted function inside OpenCV, and it is not optimised, letting the opportunity to fix it.

4.1.4 Review

In order to optimise the software application, it is important to highlight the performance degradation causes. This section summarises the main causes of degradation of each analysed module.

Retriever

Video Retrieve consumes most of the Retriever runtime, which grabs each frame and decodes it. However, an optimisation candidate is entirely by hardware, and it can reach the limits of this research. Also, there are no dependencies with the number of cars, or with extrinsic parameters.

Detector

In this case, more than 3/4 of the consumption is taken by the Haar classifier, followed by the frame preprocessing.

The module dependencies for Haar classifier are:

- **Detection ROI size**: The larger is the Detection ROI, the more is the consumption.
- **Noise**: If a scene contains objects with similar features to the target object, more cascade stages will be occupied to discard them.
- **Number of cars in Detection zone**: although they do not have a significant impact, a target object will consume more stages from the cascade classifier. Hence, more target objects imply more consumption.

The module dependencies for preprocessor is only:

- **Detection ROI size**: The larger is the Detection ROI, the more is the consumption because of the number of pixels to be processed.

Queue Refresh Manager

In this module, there is a significant issue with one of the parameters of its function (cascade classifier), and it shall be corrected to enhance its runtime. Besides, the consumption of Refresh Position Method is $O(n)$, where n is the number of cars to be tracked in *vQueue*. That means that the higher is the number of cars tracked, the more is the runtime consumption.

The module dependences for Refresh Position Method are:

- **Number of cars being tracked**: the more number of cars, the more is the throughput degradation.
- **Minimum object size**: the bigger is the tracked object, more is the processing consumption.

Moreover, MOSSE tracker update subroutine can be optimised, especially because of the existence of non-optimised functions in the Tracker class.

4.2 Improvement strategies

One of the key optimisations proposed by this research is the use of Approximate Computing to improve the application performance, wavering precision in results. However, there are some optimisations purely by software, improving programming structures and polishing code. One example of this kind of optimisation is centralising all grayscaling functions. There are grayscaling before executing Detector and during MOSSE tracker update subroutine. Perfectly, they are suitable to be joined in the beginning after retrieving a frame.

Another sort of optimisations implies implementing functions to hardware without any kind of approximation, using Xilinx OpenCV library [74] in combination with Xilinx SDSoC [75]. This library also allows parallelising an image processing function up to 8 pixels per clock, achieving huge improvements in runtime.

Taking into account the bottlenecks in the past section, and the weak points of each of those modules stated above, it is time to present the optimisations to enhance the application performance and runtime.

4.2.1 Retriever

According to the **grayscaling centralisation** presented during the introduction, it is possible to centralise it in Retriever after having a frame decoded. This will save several grayscaling, because each MOSSE update implies a grayscale, wasting resources unnecessarily. Also, depending on its performance, it is possible to **approximate grayscaling** replacing floating point operations by simpler operators.

4.2.2 Detector

Once detected, tracker launched technique can save some resources skipping some frames, because of its detection error resilience. Basically, if Detector module is well-behaved (it detects regularly target objects) in a video sequence, it is not necessary to launch this module frame-per-frame, because a target object can be detected once, and the tracker

will do the rest of the work. Therefore, the proposed technique to lower the detection impact on the application is **Loop Perforation**. In this case, the skipping element is one or more frames in a row (*Frame Skipping*).

In spite of the error resilience, Frame Skipping shall be responsive to traffic changes given that vehicles can go to different speeds depending on traffic density. The cases are:

- **High traffic density:** Vehicles go to a low speed, and Detector will detect them several times in a series of frames. So, some frames can be skipped.
- **Low traffic density:** Vehicles go fast, and Detector will detect them fewer times than normally does. So, fewer frames can be skipped.

As a result of density changes, Frame Skipping shall be responsive to traffic density. A high density implies that more frames are neglectable, and a low density that fewer frames are skippable in order to continue detecting vehicles correctly.

Preprocessor is optimisable as well. Xilinx OpenCV has functions for *erosion* and *Gaussian blurring*. So, the **preprocessor is suitable to be moved to hardware**.

Those are the possible approximations for Detector. Haar cascade detector is not suitable to be implemented on hardware, because it is not available in Xilinx OpenCV, and approximating it implies rebuild all the code, exceeding the scope of this research. Despite that, as future research, it is a good alternative for boosting application throughput.

4.2.3 Queue Refresh Manager

Most of the application consumption is made by this module. There are several techniques to optimise it:

- Loop Perforation
- Thread optimisation
- Hardware acceleration
- Approximate hardware acceleration

Similar to Detector, **Loop Perforation** is implementable in Queue Refresh Manager. One of the reasons is the error resilience due to its adaptative filter. Frame-per-frame differentiation is not too significant for filter adaptation, because of the minimum changes which occur from one frame to another. Skipping one or two frames in a row in a $30fps$ video sequence implies losing around 66.6 ms of video information.

However, error resilience in the object tracking can vary depending on object variances, such as rotation, geometric transformation, and scaling. If a video sequence presents a

Table 4.2: Difficulty of MOSSE update subroutine hardware implementation

Function	Difficulty	Observations
DFT and IDFT	High	There is a fixed-point proposal for 1D-DFT.
mulSpectrum	Low	It is easy to implement applying complex number theory and algebra.
multiplyComplex	Low	It is easy to implement applying complex number theory and algebra.
divSpectrum	Moderate	Numeric representation must be carefully chosen to avoid overflow and undesired precision loss.

high variance from a frame to another, the error resilience goes down, and Loop Perforation is less viable. Besides, the same phenomenon of traffic density is also applicable to this case.

Analysing carefully the MOSSE update subroutine structure (see the diagram block in MOSSE tracker update subroutine), it is possible to **optimise it by software**. The prediction results are available just after the decision block ($PSR > th$), making possible to compute the new filter (H^*) in another thread, freeing the main thread for another vehicle calculation.

Moreover, focused on the MOSSE update subroutine block diagram, there are possible **hardware acceleration** candidates, but without using floating-point number representation because of its resources consumption (logic blocks) while implementing on an FPGA. Therefore, hardware accelerations of this subroutine shall be approximated using either fixed-point number representation or truncated integer representation.

Table 4.2 mentions the difficulty of implementing each of the functions of MOSSE update subroutine. Also, mentions some important aspects before trying to implement them.

4.2.4 Improvement codification

To ease the analysis, each improvement will be identified by a code as shows Table 4.3.

4.3 Results and global impact

The goal with enhancing the application runtime is to be able to implement the application on a SoC and analyse a video sequence in less than a frame period ($1/fps$). That makes the application suitable to run online directly connected to a video camera or to a video

Table 4.3: Optimisations code table

Module	Optimisation	Technique	Code
Retriever	Centralised grayscaling	Software optimisation	RO-1
	Approximate centralised grayscaling	Approximation	RO-2
Critical fixes	Queue push manager parameter correction	Software optimisation	GO-1
Detector	Frame Skipping	Loop Perforation (Approximation)	DO-1
	Hardware accelerated preprocessor	Hardware acceleration	DO-2
Queue Refresh Manager	Frame skipping	Loop Perforation	TO-1
	Multi-threading	Software optimisation	TO-2
	Filter update accelerator	Hardware acceleration + Approximation	TO-3

stream. Moreover, taking into consideration that implementing the software on a SoC is an important element, the results presented are analysed only for this platform.

In this section, each improvement is analysed individually, explaining how it is implemented in the application, and how it affects application runtime and results. Besides, there are some improvements such as RO-2 and TO-3 that were not implemented because of communication bottlenecks. However, they are analysed in detail and a predicted impact on an ideal implementation.

Likewise, each analysis assumes that each video sequence has a frame rate of $30 fps$ to standardise the measurement; hence, all the profiling results expressed in percentages refer to a period of $33.3 ms$. Also, the analysis sequence is according to the order of when each improvement was implemented.

4.3.1 Retriever

The Retriever optimisation is basically moving all grayscale functions to this module, performing a grayscaling once instead of multiple times in Detector and in Queue Refresh Manager.

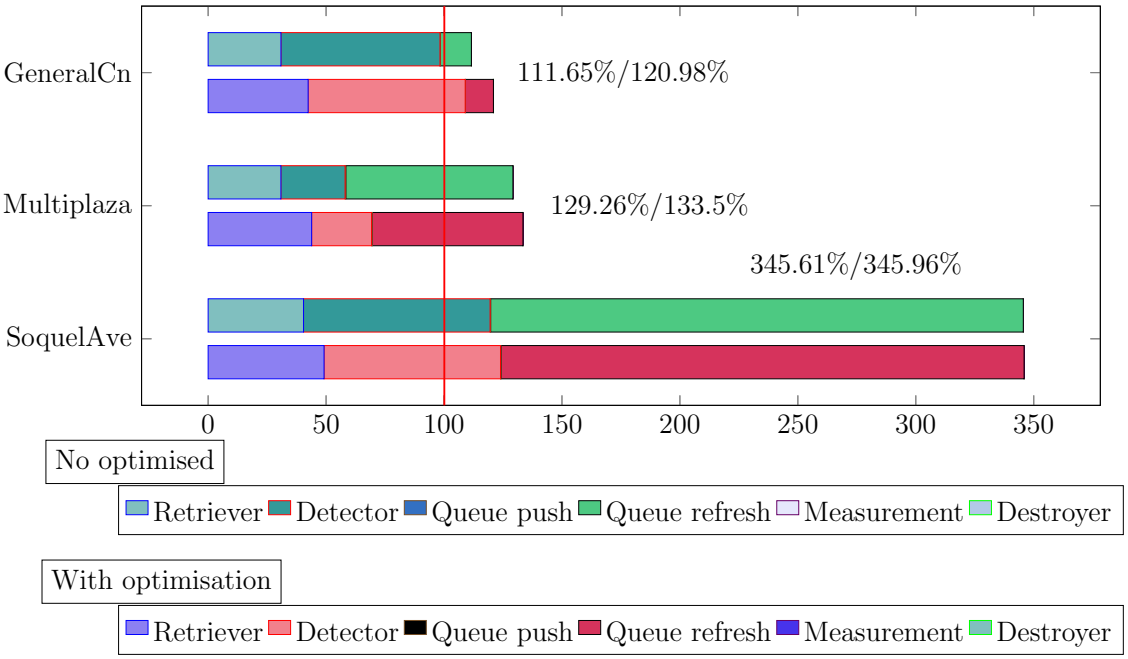


Figure 4.8: Runtime change after applying RO-1 to the no optimised baseline referred to a frame period of 33.3ms

RO-1

Figure 4.8 shows the change caused by implementing RO-1. Centralising the image grayscaling implies to grayscale the complete frame instead frame sections as it is done in MOSSE update subroutine. It makes RO-1 viable only for video sequences whose maximum object size is big enough to have a considerable impact (as it is shown in *SoquelAvenue*). Also, it is also applicable for video sequences which demands a certain number of tracked cars. This can be seen contrasting *General Cañas* (with the lowest maximum object size) with *Multiplaza del Este*. Both videos have a similar number of tracked cars, but their minimum object size is different. For that reason, the impact after implementing RO-1 on the application is different in both videos *General Cañas* and *Multiplaza del Este*, proving the impact of maximum object size on the effects of RO-1 optimisation.

Now, to demonstrate the impact of number of tracked cars on the optimisation effect, *Multiplaza del Este* and *Soquel Avenue* can be compared because they have the same maximum object size parameter, whose number of tracked cars is greater than the others, and this optimisation does not make any difference in runtime results in *Soquel Avenue*.

On the other hand, RO-1 is only viable to be implemented on the application if it analyses *Soquel Avenue* because the number of cars tracked can increase more than the current test conditions, however, RO-1 is not viable in *General Cañas* and *Multiplaza del Este* analyses under the current test conditions.

Finally, RO-1 is taken into account despite worsening the application runtime under test conditions, which are almost ideal in these cases. Nevertheless, RO-1 can be switched

during runtime to enhance application performance during high-density traffic situations, controlling the negative affectation of this improvement on the application performance.

RO-2

Hardware acceleration and approximation of the centralised grayscaling can enhance its runtime. Xilinx OpenCV does not have a grayscaling function, then a new implementation had to be proposed.

The following equation is often used in most of the CV applications [76] to convert from an RGB colour image to a grayscale image:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{4.1}$$

Where R, G, B are the colour channel values in a predefined pixel, and Y is the grayscale value.

The proposed approximation is the following:

```
/*
    Approximate grayscale calculation
    The colour channel values are preloaded in b, g, r
*/
uint8_t r0,b0,g0;
b0 = (uint8_t)(b >> 3);
g0 = (uint8_t)(g >> 4) + (uint8_t)(g >> 1);
r0 = (uint8_t)(r >> 2) + (uint8_t)(r >> 5);
uint8_t Y;
Y = b0+g0+r0;
```

Basically, there are only additions and binary shifts to compute the grayscale image with this approximation, removing floating point number operations and multiplications.

Figure 4.9 compares the grayscaling results after using the equation 4.1 (Figure 4.9b), and the proposed approximation (Figure 4.9c). As it can be noticed, the difference to the human eye is almost insignificant.

Peak Signal-to-noise ratio (PSNR) [77] and Structural Similarity Index (SSIM) [78] are often used to quantify how similar are two images. In the case of PSNR, values greater to 10 means that noise is neglectable in the picture and measures the image quality. On the other hand, values of SSIM close to 1 means that two pictures are similar and their variations are hardly perceivable to the human eye. For Figure 4.9, SSIM is 0.994% out of 1%, which indicates there is no any perceptual difference between the grayscale generated by OpenCV and the approximated pictures. PSNR is 31.5149; therefore, noise is neglectable in the approximated picture.

(a) Original image before grayscaling **(b)** Grayscale image converted by OpenCV **(c)** Grayscale image using approximation

Figure 4.9: Grayscale conversion using OpenCV and approximation

Besides, the approximate grayscaling was implemented on software and hardware to measure the advantage of using hardware acceleration and approximate computing. Also, to have a software base to compare in similar conditions, a naive implementation using equation 4.1 was developed.

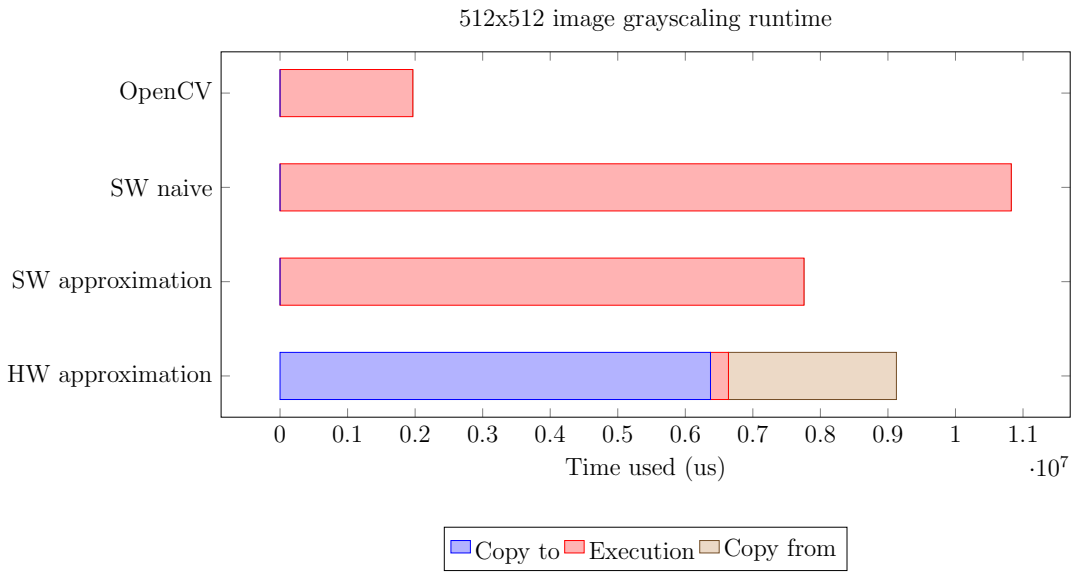


Figure 4.10: Comparison amongst different implementations of image grayscaling

Figure 4.10 presents the results got after implementing several ways of image grayscaling. The chart shows the predominance of OpenCV implementation compared to other implementations. OpenCV grayscaling uses Single Instruction Multiple Data (SIMD) instructions to parallelise pixel processing [76]. OpenCV is followed by hardware implementation, with half of the OpenCV performance, achieving image grayscaling twice faster than a naive implementation (based on pixel-wise operations). This hardware implementation was made using Xilinx OpenCV and an 8-bit parallelisation, given by Xilinx OpenCV data types.

In spite of the promising results, hardware acceleration is not as good as OpenCV im-

plementation, and RO-2 was consequently discarded as an optimisation. In the figure, communication degrades accelerator performance, taking almost the much more time than execution. The hardware implementation analysed for Figure 4.10 uses DMA Simple as the communication memory handler. However, the net speed of the accelerator is around 200us, being much faster than OpenCV implementation, and making this approximation suitable in case that all the application is moved to hardware.

It is also interesting to see the software approximation, which was faster than the software naive implementation, which demonstrates that approximate computing can accelerate image processing functions.

4.3.2 Critical fixes

GO-1

GO-1 is a critical improvement that must be analysed before continuing with other enhancements. It was noticed after implementing Retrieve enhancements, profiling each module and analysing functions at code level. For that reason, it is presented after *Retriever* optimisations. Therefore, before continuing with Detector and Queue Refresh Manager optimisation, it is important to see the impact of removing cascade parameter in Queue Refresh Manager module, because of its consumption in the tracking process.

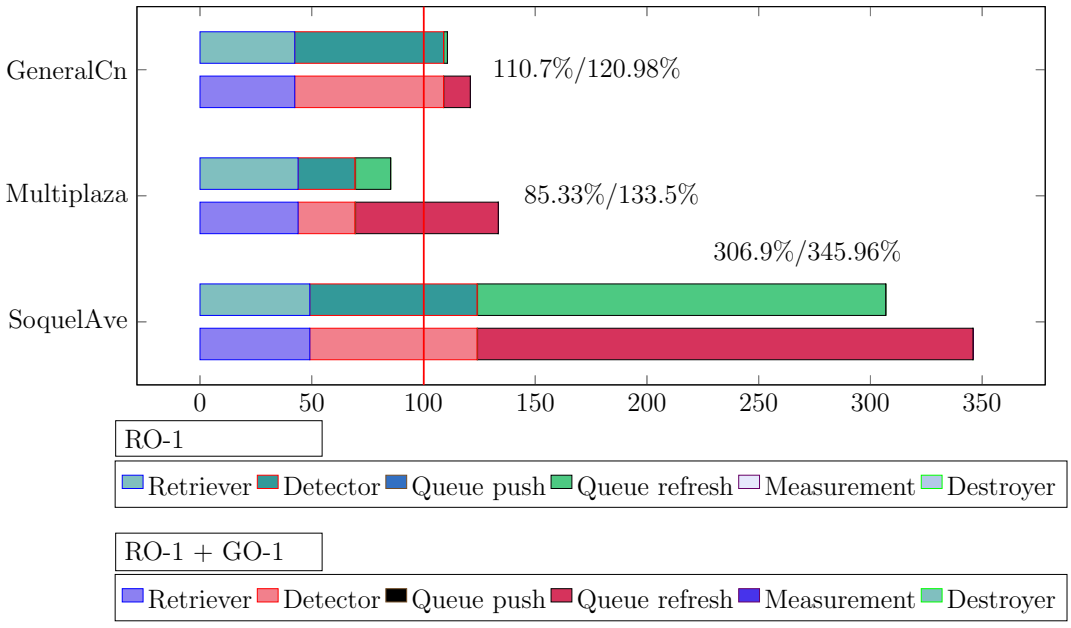


Figure 4.11: Runtime change after enhancing the application with GO-1 RO-1 compared to improving with only RO-1, referred to a frame period of 33.3ms

The results presented in Figure 4.11 shows how severe was the affectation given by cascade classifier parameter passed by value, comparing the application runtime after applying GO-1 and RO-1 improvements, and referred to the results after improving with RO-1 (without fixing this issue) only.

It is important to highlight that GO-1 + RO-1 results are taken as the reference (named *baseline*) to the next optimisations because it represents the application profiling in normal conditions and applying good programming practices, and it can work as an application baseline for analysis. So, when GO-1 is shown as a reference, it includes RO-1 and GO-1 simultaneously.

On the other hand, the profiling results already presented in *Retriever* continue having the same impact shown above, where RO-1 has a negative impact on *General Cañas* and *Multiplaza* analysis runtimes, whereas it does not have a significant impact on the *Soquel Avenue* case analysis.

4.3.3 Detector

One of the strongest methods to lower the global computational cost of an application is applying Loop Perforation. However not all the times, it is possible, and for that reason, it only affects the average consumption. Even if the global impact suggests that the program can run online, if the application cannot execute online without the optimisation, there will be a lag when the frames able to be skipped are low. After applying Frame Skipping to Detector module, the global consumption lowered considerably when it was suitable. Also, there is a little contribution of hardware acceleration in image preprocessing, but not as impacting as Frame Skipping.

DO-1 and TO-1

Frame Skipping for DO-1 is traffic density-responsive implemented to guarantee system adaptation to the different traffic circumstances. To acquire the maximum number of frames that can be skipped, and the equation of the number of frames in function of traffic density, test and fail method was used, seeing when results were not tolerable.

The equation is a linear model described by two points. These points are the minimum density which allows skipping one frame, and the maximum number of frames skippable in a high-density traffic scenario. The equation receives the traffic density each loop cycle, and it computes the number of frames allowed to jump without degrading Traffic Dynamics results above the tolerable error. Then, it starts to skip frames, until the number of frames skipped meets the allowed value given by the equation. After that, a new frame is retrieved and analysed, repeating the skipping process again.

Table 4.4: Frame skipping parameters used for Detector approximation (DO-1)

Video	λ_{min}	λ_{max}	F_{max}
General Cañas	0.001	0.05	3
Multiplaza del Este	0.008	0.05	4
Soquel Ave	0.003	0.05	3

Table 4.5: Frame skipping parameters used for Queue Refresh Manager approximation (TO-1)

Video	λ_{min}	λ_{max}	F_{max}
General Cañas	0.001	0.05	1
Multiplaza del Este	0.008	0.05	1
Soquel Ave	0.003	0.05	4

Table 4.4 and Table 4.5 shows the Frame Skipping parameters used for sample video sequences. The parameters are the minimum density to jump only one frame (λ_{min}), and the maximum number of frames to jump (F_{max}) at maximum density (λ_{max}). Those parameters are got using *trial and error method*; then, those values were verified testing their effectiveness in the application qualitatively, observing how many cars were skipped and not well processed. One of the reasons to make those parameters different for each video sequence is the variances which happen during the video sequence, especially, when the object changes its shape.

Both implementations DO-1 and TO-1 are implemented simultaneously to save time and make the most of code structures. Nonetheless, the equation for each one is different, given by:

$$f(\lambda) = \frac{F_{max} - 1}{\lambda_{max} - \lambda_{min}} \cdot \lambda + \frac{\lambda_{max} - F_{max}}{\lambda_{max} - \lambda_{min}} \tag{4.2}$$

The software application computes equation (4.2) at the beginning of the software application, to save resources and avoid recomputing the same number each iteration unnecessarily. Also, Figure 4.12 shows modules runtime contribution with Frame Skipping, after applying the parameters and equation mentioned above. It indicates that Detector only was launched 78% of the total number of frames in *General Cañas* analysis, 53% in *Multiplaza del Este*, and 59% in *Soquel Avenue*. That means that the detection process does not analyse all frames, and some of them were skipped. On the other hand, Queue Refresh Manager analyses in 100% of the total frames in *General Cañas* and *Multiplaza del Este* analyses, so there was no any frame skipped for object tracking in those videos, whereas in *Soquel Avenue*, Queue Refresh Manager module executes 58% of the times. The percentage of the number of frames skipped is given by $100\% - p$, where p is the runtime contribution of the module.

Figure 4.13 illustrates the profiling results after adding DO-1 and TO-1 optimisations (explicitly RO-1 + GO-1 + DO-1 + TO-1) to GO-1 baseline (which includes GO-1 and RO-1). The change between the baseline and these improvements are huge, making two of the sample video sequences now suitable to online execution, even though with the lag risk mentioned above.

DO-1 and TO-1 improvements impacts relevantly *Soquel Avenue* video sequence, reducing the time consumed in 150% thanks to launching Detector 59% and Queue Refresh Man-

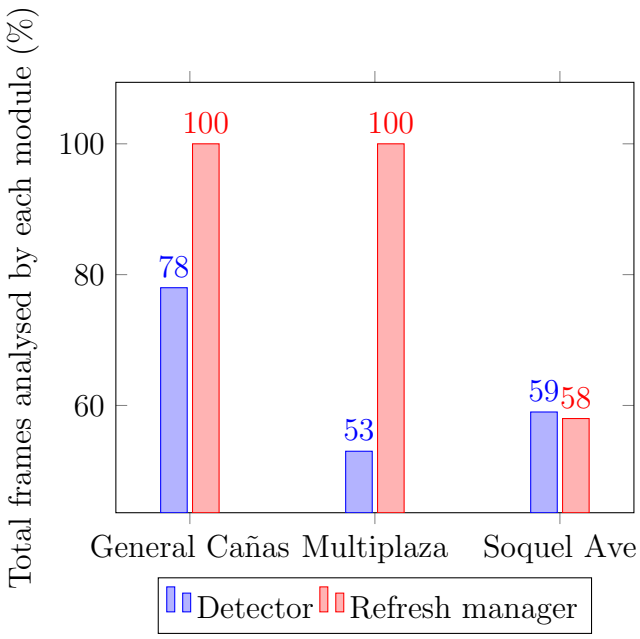


Figure 4.12: Total frames analysed for object detection and tracking out of the total frames available in each video after applying Frame Skipping (DO-1 and TO-1)

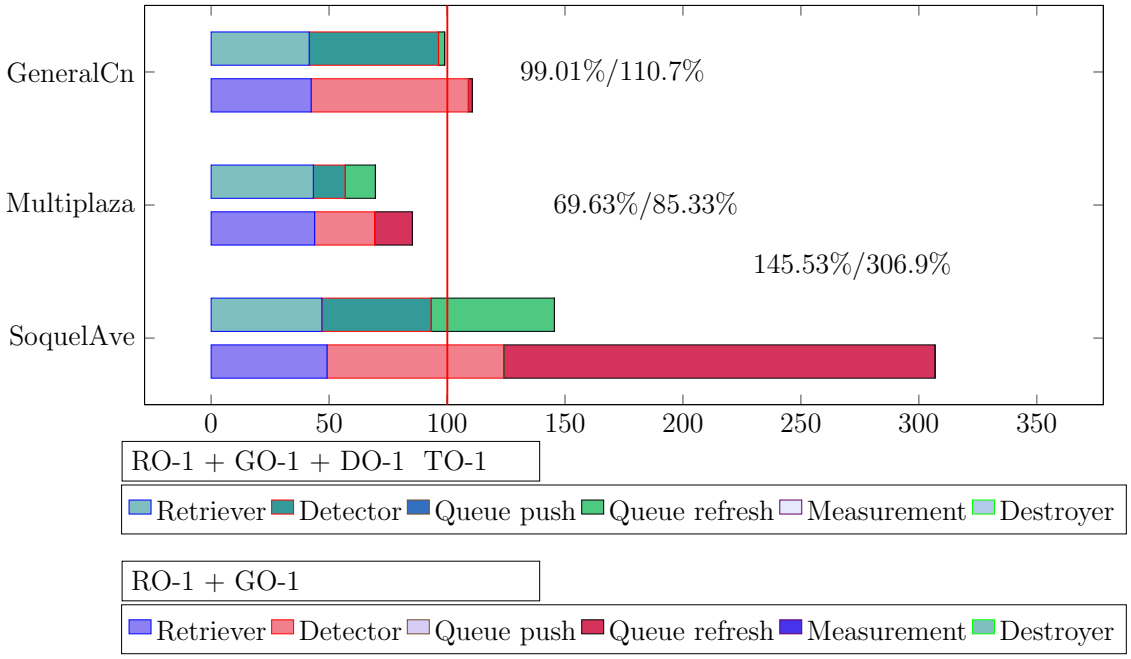


Figure 4.13: Runtime change after enhancing the application with adding DO-1 and TO-1 compared to baseline, referred to a frame period of 33.3ms

ager 58% of the total application cycles or video frames (almost 1 frame skipped every 2 frames). One of the reasons for this huge impact is the use of TO-1 and DO-1 simultaneously, whereas the other two video sequences (*General Cañas* and *Multiplaza del Este*) do not support TO-1 because their object transformation variance (the objects tends to change fast and MOSSE filter cannot adjust to adapt itself to follow those changes), so

application only run with DO-1 enhancement in *General Cañas* and *Multiplaza del Este* analyses.

Despite not activating TO-1, *General Cañas* and *Multiplaza del Este* have a decrease in their consumption. *General Cañas* needs 99% of frame period for its execution almost skipping 1 frame out of 4 frames, skipping 22% of the total frames (see Figure 4.12), whereas *Multiplaza del Este* needs only 69.63% of frame period (executing Detector only 53% of the chances). Thus, the achievement of using loop perforation as an AC technique is very effective and reached the goal of allowing to run the application online. Nevertheless, it shall be used carefully if the approximation conditions are not met, because of lag risk, caused by adverse situations where traffic density does not allow skipping frames.

DO-2

In spite of having a 99% of frame period usage in *General Cañas* video sequence, it is still adventurous to accept such runtime. For instance, if the traffic dynamics changes to a low-density traffic situation, Frame Skipping will lower its skipping rate, due to a low-density scenario implies that cars can go fast and it reduces the chance to analyse a car properly, and the runtime will increase as an immediate effect, because the application will deploy Detector and Queue Refresh Manager for more frames. This will raise the period usage and cause response lag, agglutinating a series of frames pending to process, provoking analysis losses and system errors.

In order to get this risk mitigated, DO-2 proposes accelerating the Detector preprocessor using hardware, with Xilinx OpenCV. This accelerator is constituted by:

- Erosion (3x3 kernel)
- Gaussian filter (3x3 kernel)

The kernel size using hardware acceleration remains constant, with a 3x3 size due to all the videos use this kernel size. Also, there is no possibility to enable or disable preprocessor components, caused by the increment in accelerator consumption, where the communication will spend the time earned by running accelerator processing.

Figure 4.14 depicts the comparison between an implementation made by software against an implementation accelerated by hardware. The speedup achieved by hardware acceleration, according to the figure, is about 2 times faster than the software implementation. However, analysing results acquired carefully, the output results generated by hardware acceleration are not the same to the given by the software, and they are approximated.

According to PSNR and SSIM quantification, hardware accelerator results are 28.12 and 0.955 respectively. Hence, there is a 95% of similarity between the hardware preprocessed and the software preprocessed images. That means if a human has to compare both pictures, some differences will pass unnoticed during a fast sight. On the other hand, the

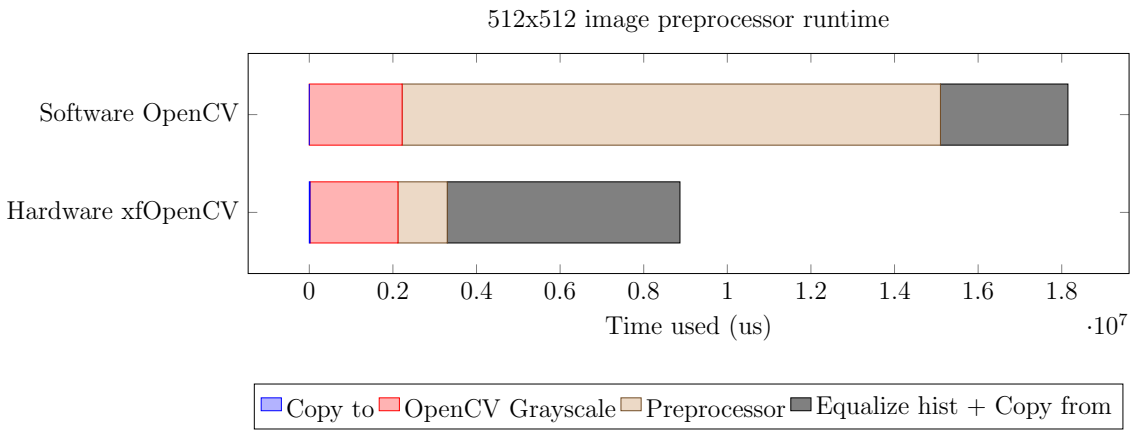


Figure 4.14: Comparison amongst different implementations for Detector preprocessor

output image got from hardware is 28.12 times greater than noise, making it adequate for analysis, which requires that image signal has to be ten times greater than noise. Taking into account that, final traffic dynamics results will be affected after implementing DO-2, because the similarity is not 100%, and it can change potentially some image details needed for detection and tracking.

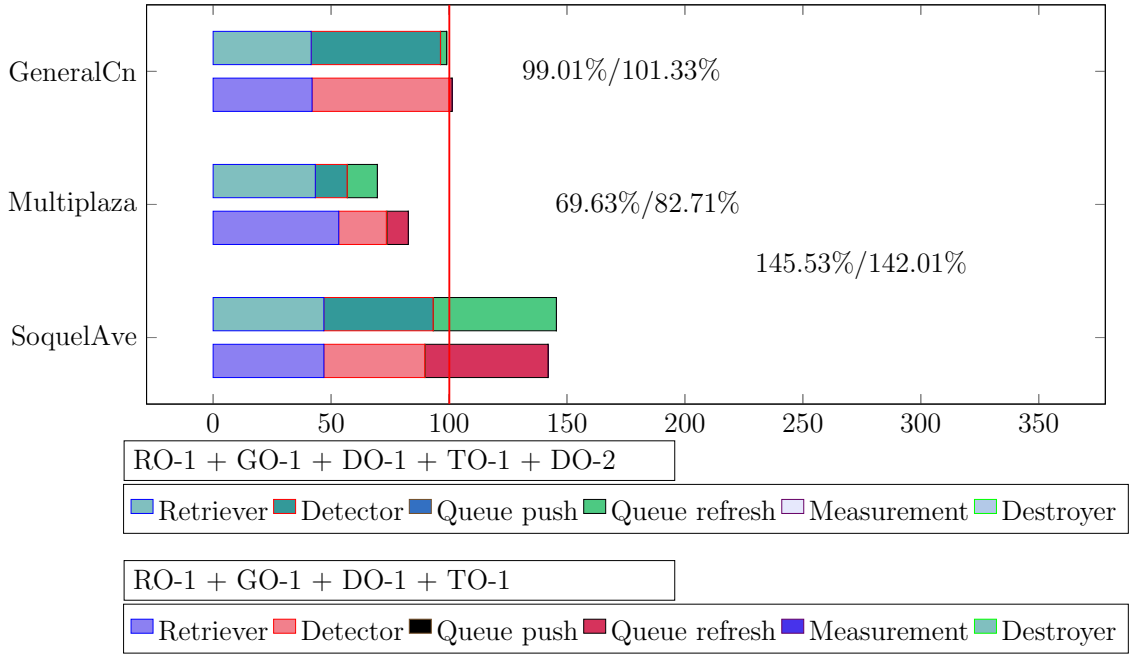


Figure 4.15: Runtime change after enhancing the application with adding DO-2 compared to Frame Skipping (RO-1 + GO-1 + DO-1 + TO-1), referred to a frame period of 33.3ms

Figure 4.15 illustrates the profiling results acquired after implementing all the improvements presented above (RO-1 + GO-1 + DO-1 + TO-1 + DO-2), except RO-2 due to it was not suitable because of its resources consumption. According to the picture, the performance during *General Cañas* and *Multiplaza del Este* analyses with adding DO-2 is less than without enabling it. Whereas, the performance improved in *Soquel Avenue*.

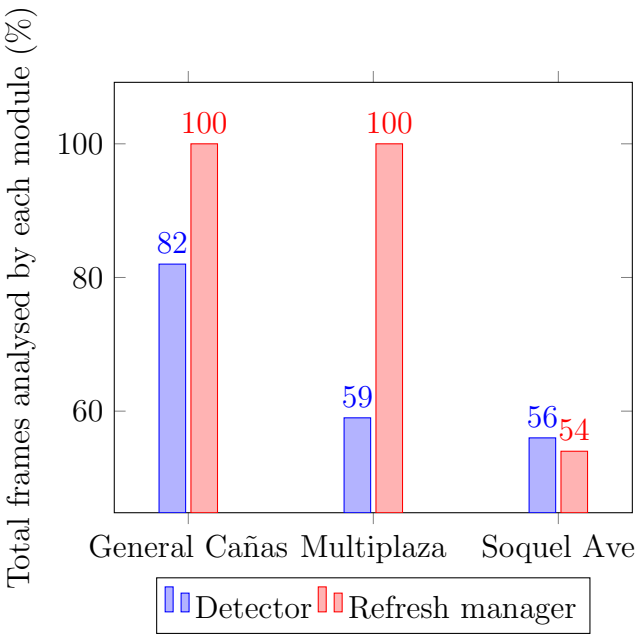


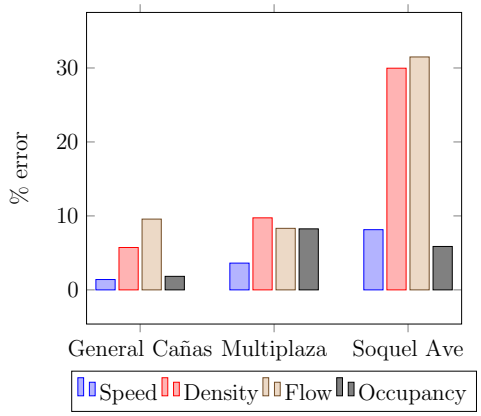
Figure 4.16: Total frames analysed for object detection and tracking out of the total frames available in each video after adding DO-2 to the baseline

This performance change is due to a change in the number of frames skipped, as it is shown in Figure 4.16.

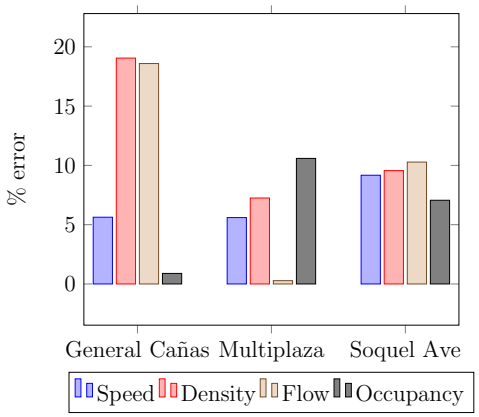
In the case of *General Cañas*, 18% of the total frames were skipped after adding DO-2 (RO-1 + GO-1 + DO-1 + TO-1 + DO-2), whereas without DO-2, the number of frames skipped were 22% (Figure 4.12). The same phenomenon happened to *Multiplaza del Este*, with 41% of frame skipped after adding DO-2 against 47% without having DO-2 activated (Figure 4.12). Also, it happened to *Soquel Avenue*, which number of frames skipped was reduced in both DO-1 and TO-1 (reminding that TO-1 was only suitable for *Soquel Avenue* analysis). Nonetheless, the performance in *Soquel Avenue* is better than the baseline given by DO-1, which explains that there is an enhancement after applying DO-2, but in the other two videos, it is not possible to see this enhancement because of the performance degradation caused by the change in the number of frames skipped.

That means that after adding DO-2 on the baseline, the changes caused by the hardware (SSIM = 0.995) approximation do not allow Frame Skipping jumping the same or a greater number of frames than with only having DO-1 enhancement. Therefore, the reduction of the number of frames skipped provoked a rise in the consumption, degrading the performance in *General Cañas* and *Multiplaza del Este* analyses more than the optimisation can offer.

Going deeper, the number of frames skipped changed because a variation in traffic density, suggesting a change in the final traffic dynamics results. Therefore, after implementing Frame Skipping (RO-1 + GO-1 + DO-1 + TO-1) and DO-2, it is expected to have a change on them due to the approximations in both improvements. Figure 4.17 summarises the change in traffic dynamics results after implementing those improvements, because DO-1,



(a) Error quantification after implementing Frame Skipping (RO-1 + GO-1 + DO-1 + TO-1)



(b) Error quantification after implementing Frame Skipping and Preprocessor (RO-1 + GO-1 + DO-1 + TO-1 + DO-2)

Figure 4.17: Different errors on traffic dynamics results detected over the potential image sources

TO-1, and DO-2 introduce errors to traffic dynamics results due to their approximations. Also, car detection rate (more true-positives were detected) improved in DO-2 respect of Frame Skipping only (Figure 4.17a), enhancing the traffic dynamics results when both approximations (frame skipping and preprocessing) were implemented together (Figure 4.17b). This enhancement in the car detection rate is due to the enabling of erosion in *Soquel Avenue* because it was disabled before implementing DO-2 and it is not possible to disable it after performing the preprocessor on hardware (on the FPGA), due to technical reasons.

Also, taking into consideration that result errors should be under 20%, the final results when implementing DO-1 and DO-2 are still acceptable. Therefore, approximations did not degrade the results considerably under the design goal specifications of Table 3.2.

Finally, despite DO-2 degrades the results, it will be kept for future work, because of the aim of implementing the whole Detector module to hardware in order to reduce its runtime consumption, assuming that it will be reduce if it is moved to hardware.

4.3.4 Queue Refresh Manager

One of *Queue Refresh Manager* improvements has been already analysed in the section above; thus, the missing improvements to analyse are TO-2 and TO-3. In the case of TO-2, it is fully tested on the software application, whereas TO-3 was proposed and tested. However, a complete proposal has been developed for future work, highlighting the most important results acquired after implementing a part of the functions of MOSSE filter update subroutine.

TO-2

Part of the MOSSE filter update subroutine can be moved to another thread, allowing to the main thread continue processing other vehicles without being stuck precomputing new coupled filters. Hence, this improvement implies *Multi-threading* processing using the POSIX threading library (pthread) [79].

Due to the number of filters to compute, and the hint of enhancing performance without creating too many threads to compute each of them, a single thread with a filter queue was designed. The data type defined for this queue is personalised, being possible to store matrices values and pointers (see Appendices - MOSSE filter object for TO-2).

To compute a filter updated, it is need first A and B values, because H^* is given by (4.3). Then, A and B are evolvable variables given by (4.4) and (4.5), where η is the learning rate. Then A_{new} and B_{new} are the values given by the target object spectrum (template) G and the incoming image F [69], according to (4.6) and (4.7).

$$H^* = \frac{A}{B} \tag{4.3}$$

$$A_i = A_{i-1} \cdot (1 - \eta) + A_{new}\eta \tag{4.4}$$

$$B_i = B_{i-1} \cdot (1 - \eta) + B_{new}\eta \tag{4.5}$$

$$A_{new} = G \odot F \tag{4.6}$$

$$B_{new} = F \odot F \tag{4.7}$$

G and η are constants, whereas F is given by converting the tracked object to the frequency domain. A, B and H are variables which belong to each tracker, that means that for each tracker, those values are distinct.

Moreover, the program flow must consider the thread termination before executing Queue Refresh Manager again, because it can cause a conflict between new incoming values and values which are being calculated. Also, with implementing TO-2, there is no affectation to final results, because of the absence of approximate computing elements in this optimisation.

Figure 4.18 shows the enhance after applying TO-2 on GO-1 baseline. This improvement only affects Queue Refresh Manager. The change in consumption is considerable only parallelising at thread level, especially in *Soquel Avenue* video sequence, reducing the consumption by 92.5%, followed by *Multiplaza del Este* with 7%. The change in *General Cañas* is not too significant due to the few cars followed in that video.

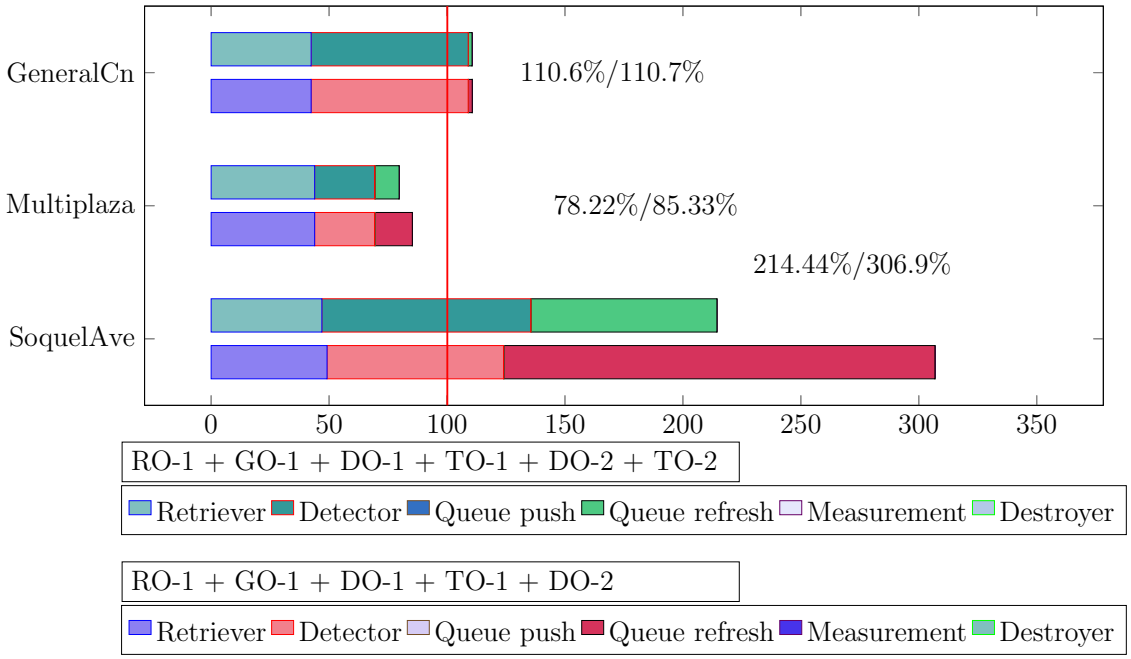


Figure 4.18: Runtime change after enhancing the application with adding TO-1 compared to baseline, referred to a frame period of 33.3ms

TO-3

The last optimisation is to implement MOSSE filter update subroutine using a hardware accelerator; however, it was not entirely implemented on the application because its complexity and it is left for future work. This accelerator is also approximated to save FPGA resources, because of floating point number representation expensiveness. The accelerator is composed by:

- mulSpectrum
- multiplyComplex
- multiplyByRealScalar
- addition

The MOSSE update subroutine has its components placed in cascade, where each of the components works twice. Also, the accelerator was implemented using two architectures. The first is a pixel-wise architecture, which process one pixel per clock (*HW approximation 1px*), and the other using hardware parallelisation, processing eight pixels per clock (*HW approximation 8px*). The contrast of software and hardware implementations gives the results presented in Figure 4.19.

The approximation made for this accelerator is basically changing numeric representation from floating point to scaled integer representation, in order to avoid using the software FPU library and save FPGA resources.

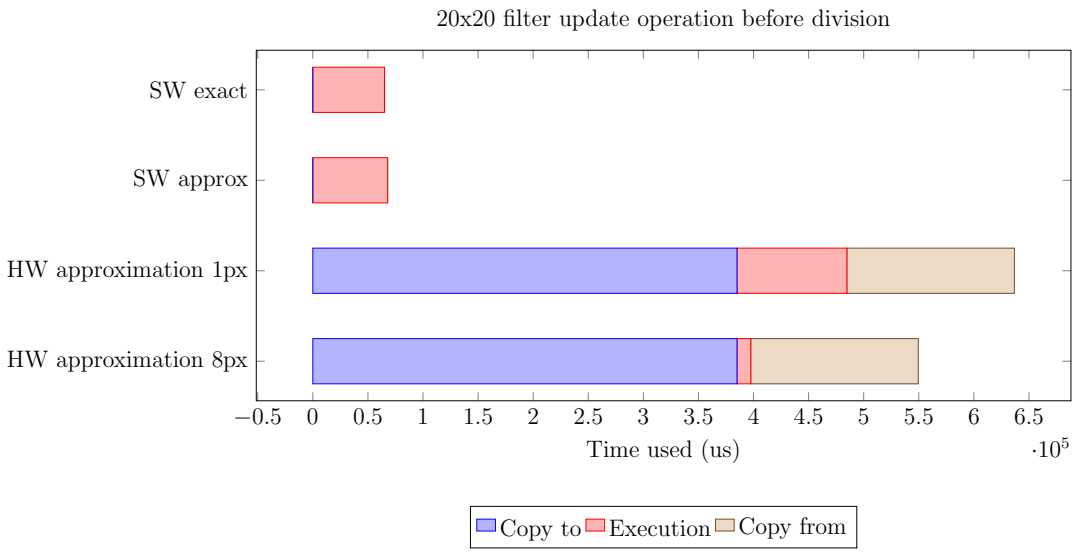


Figure 4.19: Comparison amongst different implementations for spectrum operations

Figure 4.19 presents that *SW exact* and *SW approx* does not have an important difference and almost take the same time to process. Therefore, there is no difference in using approximation and exact operations. However, it cannot be discarded that using approximation is not a strong technique to enhance performance, because of the matrix size used in the example, and it was not technically possible to increase the size because of floating point representation did not fit in the FPGA, used for spectrums division. Moreover, hardware acceleration of approximated part was successfully run, giving almost 5x of speed up compared to its software counterpart, shown by the Execution bar (Figure 4.19 red bar).

The same phenomenon related to communication, seen in preprocessor acceleration in DO-2 (in Figure 4.14 is seen as the extension of time in *Equalise hist + Copy from*), and grayscaling in RO-2 (in Figure 4.10), also happened here. All the runtime earned with running the accelerator was spent by data transference. Therefore, implementing approximate accelerators is not suitable with more than two data transferences (one at the input and another at the output), and the size of the data transferred to the accelerators needs special attention.

In spite of being not suitable because its runtime ($550usvs70us$ in 4.19), results are still promising for the typical MOSSE values, with less than 15% of error in a range of data from 0.001 to 1 at the spectrum division output, where the numerator is always less than the denominator, and a range from 1 to 1000000 at the input, representing numbers with 32-bit integer variables. Now, the range from 0.001 to 1 is given by the spectrums division at the end of MOSSE update function, being the only part which had floating-point number representation. Therefore, implementing a MOSSE update function hardware accelerator is viable due to the results mentioned before, that can be suitable for this application, but the runtime achieved by hardware ($550us$) contrasted to its software counterpart ($70us$) makes it no possible to implement as an improvement

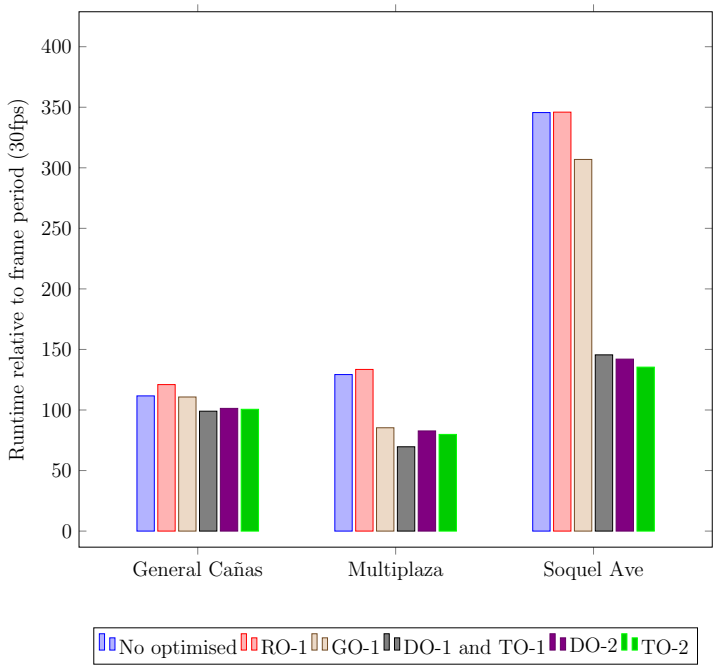


Figure 4.20: Runtime after implementing improvements @ frame period = $33.3ms$

in this case, because it will degrade the application runtime.

4.3.5 Global impact

Until here, all the optimisations proposed at the beginning of this chapter have been analysed in detail, measuring their error deviations and contributions to lower the general runtime.

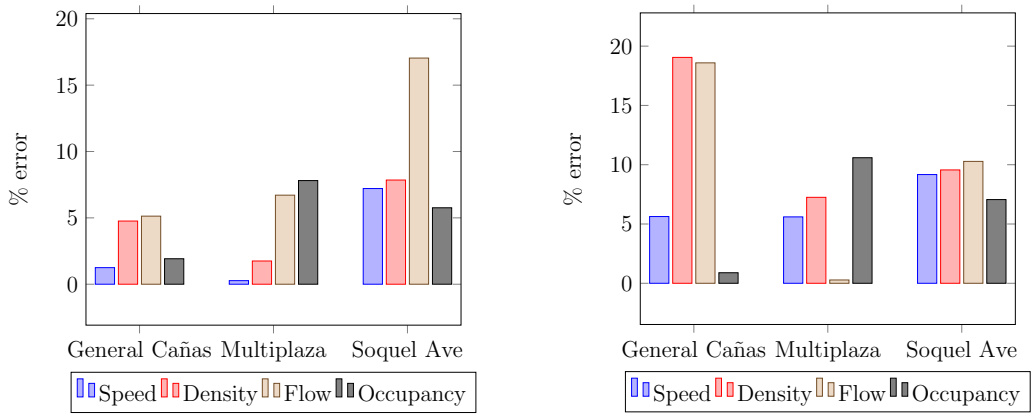
Figure 4.20 shows the final results after implementing the already seen improvements. The results go on after adding each improvement until TO-2, being this last one the final application runtime. The improvement DO-2 can be removed in order to enhance the application, because of its runtime degrading. Nevertheless, this block enables all Detector preprocessor stages, increasing consumption, and it was implemented in order to enhance the final application results (traffic measurement).

In the case of *Multiplaza del Este*, the goal of being able to run the application online was met, but not in *Soquel Avenue* and *General Cañas* cases. Some solutions to deal with its runtime budget are, mainly:

- Increase the frame period (reduce frames per seconds to $25fps$) to $40ms$.
- Reduce Detection ROI size.
- Reduce Tracking zone size.

Those solutions are proposed based on several phenomena seen during *General Cañas* and *Multiplaza del Este* video analyses, which with less area, their runtime budget was more

acceptable, and made them suitable to run online. In spite of not being able to compute *Soquel Avenue* video online, it is possible to run it off-line, speeding the application up 2.55 times (see Table 6.7), which needs less than 5 minutes to analyse 1 minute of video, considering that the runtime is 135.41% at $30fps$, which means that in a 1 minute video, the application needs 1.36 minutes to process all frames, accomplishing the *Execution modes requirement* of Table 3.2. Likewise, *Soquel Avenue* analysis demonstrates that the application can achieve more speedup using approximations, meeting some conditions such as the number of cars tracked, and frame skipping capabilities (especially F_{max}), due to after applying Frame Skipping, it helped to reduce the runtime in almost the half of the time, whereas in the other video analyses, the improvement was less than the half.



(a) Final result errors quantification without any improvement

(b) Final result errors quantification after improving

Figure 4.21: Final error quantification with and without improvements

Another aspect to analyse is the error introduced by approximations. Figure 4.21 shows the final error quantification (Figure 4.21b) against the original error without any improvement (Figure 4.21a). With approximation, result errors increased drastically, but inside the tolerable ranges (under 20%). Therefore, the approximations do not affect the viability of application improvements, and all of them can be placed in a final software application.

To conclude, some improvements were not suitable to be implemented, because of communication bottlenecks. One example was *approximate grayscaling*. OpenCV was dominant over hardware acceleration; however, the effective processing time of hardware grayscaling was much lower than its software counterparts, as Figure 4.10 shows. Then, to make the most of hardware acceleration, it is preferred to implement all the system on hardware instead creating hybrids SW-HW, because of data transference delays as it occurred during this work (see Figure 4.10 and Figure 4.19), due to the number of elements to transfer (columns x rows) stored in each image. Thus, as future work, there is an opportunity to improve this application totally by hardware, taking advantage of approximate computing to avoid floating point operations and enhancing throughput.

Results used in this chapter are also available in Architectures comparison table.

Chapter 5

Conclusions

5.1 Conclusions

The implementation of the software application with all the requirements exposed in Table 3.2 was successfully finished and tested on a Zedboard. This application is capable of receiving a video sequence, analysing it, and measuring the traffic dynamics based on vehicle detection and tracking, giving average speed, traffic density, traffic flow and sensor occupancy as final results, with an acceptable accuracy. Moreover, the CV techniques implemented were carefully analysed, tested and selected in order to have better results. Thanks to that, MOSSE tracking technique is very promising because of its adaptative principle, while helps with object detection weaknesses.

Besides, improving this application using Approximate Computing paradigm guided to interesting results. The application performance was boosted up sacrificing results accuracy but under the 20% of tolerance established for this application. Some of the Approximate Computing techniques used to enhance were mainly: *loop perforation* and *numeric representation simplification*. However, there are some software optimisations and improvements which were not suitable to implement because of their runtimes, such as *approximate hardware acceleration*, and *function approximation*. Besides, after applying successfully the approximations, Approximate Computing effectively enhances the application performance, making most of the platform resources, especially for FPGA-based SoC.

A complete performance and error analysis determined each of the application optimisations. This analysis demonstrates, step-by-step, how effective are Approximate Computing techniques. With this, it was expected to have more throughput improvement after applying hardware acceleration, but communication bottlenecks degrade all the earns obtained by passing some of the functions to hardware. For this case study, the hardware accelerated part is Detector preprocessor, because of its performance, using mainly Xilinx OpenCV, which have promising codes to implement hardware CV applications.

Finally, it was possible to get up to 2.55x of speed up using approximations and software

improvements (in the *Soquel Avenue* case analysis), introducing an error of up to 11.5%. Nonetheless, the speed up can be higher if hardware has more modules implemented in, such as *Detector* and *Tracking*, which substantially drain computational resources. However, the *project aim was successfully accomplished*, delivering an application which receives a video sequence, and transmits the results through TCP, optimised using Approximate Computing. Also, this works demonstrates that using cameras for measuring traffic dynamics is totally possible, but it requires a good knowledge of the scene conditions to have better results.

5.2 Future work

This work left some improvements to future work. However, the contributions are more important in this case. After finishing the state of art analysis, there are opportunities to develop solutions in Approximate Computing. To make this field more relevant, it requires more contributions in order to extend this field with novel solutions, and optimise actual applications. There are a few of proposals written below.

5.2.1 Going on with the traffic dynamics responsiveness

At the end of this research, implementing a Traffic Dynamics Meter was possible only using Computer Vision techniques. This study advises that it is viable to implement a complete grid of responsive automatic traffic signalling, by taking most of the cameras, which are one of the most affordable techniques to get traffic data. Hence, the next step is to design the traffic dynamics analyser and the automatic traffic signals control.

5.2.2 Hardware-implemented baseline

Detectors

Although the object detection technique used in this research was Haar cascade classifiers, LBP-based classifiers are a hand-crafted detection technique that is suitable to be implemented by hardware. It would be useful to have it, and viable because of its numeric representation (integers).

MOSSE hardware implementation

According to state of the art, there are not attempts to implement MOSSE-based trackers accelerators. For that reason, there is an opportunity to implement hardware-accelerated trackers to have a robust adaptative tracker in hardware-based CV applications.

Approximate complex functions in hardware

In Xilinx OpenCV library, there are missing important functions, such as DFT, IDFT, detectors, and trackers. Therefore, this fields requires more contributions in order to ease hardware-based accelerators development for CV applications.

Bibliography

[1] L. Loría, *VIGESIMO PRIMER INFORME ESTADO DE LA NACIÓN EN DESARROLLO HUMANO SOSTENIBLE*, P. E. de la Nación, Ed. [Online]. Available: http://www.estadonacion.or.cr/21/assets/pen-21-2015-baja.pdf.

[2] *Tránsito necesita duplicar oficiales para control de vías.* [Online]. Available: http://www.estadonacion.or.cr/21/assets/pen-21-2015-baja.pdf.

[3] N. K. Giang, R. Lea, M. Blackstock, and V. C. M. Leung, "Fog at the edge: Experiences building an edge computing platform", in *2018 IEEE International Conference on Edge Computing (EDGE)*, Jul. 2018, pp. 9–16. DOI: 10.1109/EDGE.2018.00009.

[4] M. Shafique, R. Hafiz, M. U. Javed, S. Abbas, L. Sekanina, Z. Vasicek, and V. Mrazek, "Adaptive and Energy-Efficient Architectures for Machine Learning: Challenges, Opportunities, and Research Roadmap", *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 627–632, 2017, ISSN: 21593477. DOI: 10.1109/ISVLSI.2017.124. [Online]. Available: http://ieeexplore.ieee.org/document/7987592/.

[5] Q. Xu, N. S. Kim, and T. Mytkowicz, "Approximate Computing: A Survey", *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016, ISSN: 2168-2356. DOI: 10.1109/MDAT.2015.2505723. [Online]. Available: http://ieeexplore.ieee.org/ielx7/6221038/7386744/07348659.pdf?tp=%7B%5C&%7Darnumber=7348659%7B%5C&%7Disnumber=7386744%7B%5C%%7D5Cnhttp://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7348659%7B%5C&%7Dfilter%7B%5C%%7D3DAND%7B%5C%%7D28p%7B%5C_%7DIS%7B%5C_%7DNumber%7B%5C%%7D3A7386744%7B%5C%%7D29.

[6] M. Wyse, "Modeling Approximate Computing Techniques", [Online]. Available: https://homes.cs.washington.edu/%7B~%7Dwysem/publications/wysem-msreport.pdf.

[7] S. Mishra, A. Bavane, R. Khairnar, A. Nannaware, A. Patil, S. Shinde, and A. Professor, "Engineering and Technology (A High Impact Factor", *International Journal of Innovative Research in Science*, vol. 7, no. 5, 2018. DOI: 10.15680/IJIRSET.2018.0705170. [Online]. Available: www.ijirset.com.

[8] M. Kadaieaswaran, V. Arunprasath, and M. Karthika, "Big Data Solution for Improving Traffic Management System with Video Processing", Tech. Rep., 2017, p. 4606. [Online]. Available: http://ijesc.org/.

[9] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, "Big Data Analytics in Intelligent Transportation Systems: A Survey", *IEEE Transactions on Intelligent Transportation Systems*, 2018, ISSN: 15249050. DOI: 10.1109/TITS.2018.2815678.

[10] X. Zheng, W. Chen, P. Wang, D. Shen, S. Chen, X. Wang, Q. Zhang, L. Yang, W. Chen, D. Shen, S. Chen, and X. Wang, "Big Data for Social Transportation", *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, vol. 17, no. 3, 2016. DOI: 10.1109/TITS.2015.2480157. [Online]. Available: http://ieeexplore.ieee.org..

[11] *Uber - Earn Money by Driving or Get a Ride Now*. [Online]. Available: https://www.uber.com/.

[12] *Free driving directions, traffic reports & gps navigation app by waze*. [Online]. Available: https://www.waze.com/.

[13] E. D'Andrea, P. Ducange, B. Lazzerini, and F. Marcelloni, "Real-Time Detection of Traffic from Twitter Stream Analysis", *IEEE Transactions on Intelligent Transportation Systems*, 2015, ISSN: 15249050. DOI: 10.1109/TITS.2015.2404431.

[14] F. Hall, "Traffic stream characteristics", Jan. 1992. [Online]. Available: https://www.fhwa.dot.gov/publications/research/operations/tft/chap2.pdf.

[15] H. S. E. Chuo, M. K. Tan, A. C. H. Chong, R. K. Y. Chin, and K. T. K. Teo, "Evolvable traffic signal control for intersection congestion alleviation with enhanced particle swarm optimisation", in *Proceedings - 2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems, I2CACIS 2017*, 2017, ISBN: 9781538608463. DOI: 10.1109/I2CACIS.2017.8239039.

[16] L. Bohang, Y. Xiaoyong, Q. Li, and S. Huang, "An Improved Method for Traffic Control Relying on Close-Loop Control theory", Tech. Rep.

[17] W. Cheng, X. Liu, and W. Zhang, "An Optimal Adaptive Traffic Signal Control Algorithm for Intersections Group", Tech. Rep.

[18] L. Ximin and L. Shoufeng, "Vehicular Traffic Flow Dispersion Prediction of Coordinated Signal Control Based on Support Vector Regression *", Tech. Rep.

[19] J. Yin, "The traffic capacity model M/G/1 simulated by the Monte Carlo algorithm", in *2014 International Conference on Information Science, Electronics and Electrical Engineering*, 2014, ISBN: 978-1-4799-3197-2. DOI: 10.1109/InfoSEEE.2014.6948116.

[20] G. Mathur and M. Bundele, "Research on Intelligent Video Surveillance techniques for suspicious activity detection critical review", *2016 International Conference on Recent Advances and Innovations in Engineering, ICRAIE 2016*, vol. 2016, pp. 1–8, 2017. DOI: 10.1109/ICRAIE.2016.7939467.

[21] R. Klette, *Concise Computer Vision*. 2014, ISBN: 978-1-4471-6319-0. DOI: 10.1007/978-1-4471-6320-6. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-6320-6.

[22] G. L. Foresti, C. Micheloni, C. Piciarelli, and L. Snidaro, "Visual sensor technology for advanced surveillance systems: Historical view, technological aspects and research activities in Italy", *Sensors*, vol. 9, no. 4, pp. 2252–2270, 2009, ISSN: 14248220. DOI: 10.3390/s90402252.

[23] D. Frejlichowski, K. Gościewska, P. Forczmański, and R. Hofman, ""SmartMonitor"- An intelligent security system for the protection of individuals and small properties with the possibility of home automation", *Sensors (Switzerland)*, vol. 14, no. 6, pp. 9922–9948, 2014, ISSN: 14248220. DOI: 10.3390/s140609922.

[24] R. Aarthi and S. Harini, "A survey of deep convolutional neural network applications in image processing", *International Journal of Pure and Applied Mathematics*, vol. 118, no. 7, pp. 185–190, 2018, ISSN: 1314-3395.

[25] B. Başa, "Implementation of hog edge detection algorithm onfpga's", *Procedia - Social and Behavioral Sciences*, vol. 174, pp. 1567–1575, 2015, International Conference on New Horizons in Education, INTE 2014, 25-27 June 2014, Paris, France, ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2015.01.806. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042815008587.

[26] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, Oct. 2005, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.188.

[27] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, Jun. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[28] S. Stamatiadis and A. Argyros, "Efficient scale and rotation invariant object detection based on hogs and evolutionary optimization techniques", vol. 7431, Jul. 2012.

[29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, Dec. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[30] L. Dinalankara, "Face detection & face recognition using open computer vision classifies", Aug. 2017. [Online]. Available: https://www.researchgate.net/publication/318900718_Face_Detection_Face_Recognition_Using_Open_Computer_Vision_Classifies.

[31] M. Oliveira and V. Santos, *Automatic detection of cars in real roads using haar-like features*, Jul. 2008. [Online]. Available: https://www.researchgate.net/publication/267863282_Automatic_Detection_of_Cars_in_Real_Roads_using_Haar-like_Features.

[32] M. Heikkila and M. Pietikainen, "A texture-based method for modeling the background and detecting moving objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 657–662, Apr. 2006, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2006.68.

[33] S. Zhou, J. Gong, G. Xiong, H. Chen, and K. Iagnemma, "Road detection using support vector machine based on online learning and evaluation", in *2010 IEEE Intelligent Vehicles Symposium*, Jun. 2010, pp. 256–261. DOI: 10.1109/IVS.2010.5548086.

[34] S. Abe, "On invariance of support vector machines", 2013. [Online]. Available: http://www2.kobe-u.ac.jp/~abe/pdf/ideal2003.pdf.

[35] B. Heisele, P. Ho, and T. Poggio, "Face recognition with support vector machines: Global versus component-based approach", vol. 2, 688–694 vol.2, Feb. 2001.

[36] *Artificial neural network*, 2011. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Artificial_neural_network.svg.

[37] G. Médioni, "Emerging Topics in Computer Version", p. 661, 2005, ISSN: 978-0131013667. DOI: 10.1002/ijc.26087. [Online]. Available: https://books.google.com.co/books?id=qV90QgAACAAJ.

[38] D. Bolme, J. R. Beveridge, B. a. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2544–2550, 2010, ISSN: 1063-6919. DOI: 10.1109/CVPR.2010.5539960. arXiv: 1404.7584. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5539960.

[39] S. Haykin, *Communication systems*, 5th. Wiley Publishing, 2009, ISBN: 0471697907.

[40] S. Hashemi, H. Tann, F. Buttafuoco, and S. Reda, "Approximate Computing for Biometric Security Systems : A Case Study on Iris Scanning", pp. 319–324, 2018.

[41] S. Sidiroglou-douskos, S. Misailovic, H. Hoffmann, and S. Sidiroglou, "perforation Citation Accessed Citable Link Detailed Terms Managing Performance vs . Accuracy Trade-offs With Loop Perforation", 2013.

[42] L. Lai, N. Suda, and V. Chandra, "Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations", no. Icml, 2017. arXiv: 1703.03073. [Online]. Available: http://arxiv.org/abs/1703.03073.

[43] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators", *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 201–206, 2014, ISSN: 02780070. DOI: 10.1109/ASPDAC.2014.6742890.

[44] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", *Mathematics of Computation*, vol. 19, no. 90, p. 297, 1965, ISSN: 00255718. DOI: 10.2307/2003354. [Online]. Available: http://www.jstor.org/stable/2003354?origin=crossref.

[45] A. Edelman, P. McCorquodale, and S. Toledo, "The future fast Fourier transform?", *SIAM J. Sci. Comp.*, vol. 20, no. 3, pp. 1094–1114, 1999.

[46] Freescale Semiconductor, "Complex Fixed-Point Fast Fourier Transform Optimization for AltiVec™", pp. 1–21, 2004. [Online]. Available: http://notes-application.abcelectronique.com/314/314-66510.pdf.

[47] M. S. T. Roberts and P. D. Bouras, "Fixed-point Fast Fourier Trans- form", 2003. [Online]. Available: http://www.dbouras.eu.

[48] N. S. K. Qiang Xu Todd Mytkowicz, "Approximate computing: A survey", *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[49] Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual", *Intel Technology Journal*, vol. 09, no. 03, pp. 1–660, 2005, ISSN: 15222594. DOI: 10.1535/itj.0903.05.

[50] C. Chen, "High-order Taylor series approximation for efficient computation of elementary functions", *IET Computers & Digital Techniques*, vol. 9, no. 6, pp. 328–335, 2015, ISSN: 1751-8601. DOI: 10.1049/iet-cdt.2014.0158. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-cdt.2014.0158.

[51] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design", *Proceedings - 2013 18th IEEE European Test Symposium, ETS 2013*, 2013, ISSN: 1530-1877. DOI: 10.1109/ETS.2013.6569370.

[52] J. Lin, Y. Hwang, M. Sheu, and C. Ho, "A novel high-speed and energy efficient 10-transistor full adder design", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 5, pp. 1050–1059, May 2007, ISSN: 1549-8328. DOI: 10.1109/TCSI.2007.895509.

[53] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing", *Proceedings of the IEEE Conference on Nanotechnology*, pp. 690–693, 2013, ISSN: 19449399. DOI: 10.1109/NANO.2013.6720793.

[54] T. Moscibroda and B. G. Zorn, "Flikker : Saving DRAM Refresh-power through Critical Data Partitioning", *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pp. 213–224, 2011, ISSN: 0163-5964. DOI: 10.1145/1950365.1950391.

[55] J. San, M. Mario, B. Natalie, and E. Jerger, "Load Value Approximation", pp. 127–139, 2014. DOI: 10.1109/MICRO.2014.22. [Online]. Available: http://ieeexplore.ieee.org/document/7011383/.

[56] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: Approximate computing on programmable SoCs via neural acceleration", *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, pp. 603–614, 2015, ISSN: 1530-0897. DOI: 10.1109/HPCA.2015.7056066. arXiv: 1608.03665.

[57] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General Purpose Approximate Programs", *Proceedings of th 2012 25th Annual IEEE/ACM Intenational Symposium on Microarchitecture*, vol. 33, no. 3, pp. 16–27, 2012, ISSN: 1072-4451. DOI: 10.1109/MICRO.2012.48. [Online]. Available: http://dl.acm.org/citation.cfm?id=2457519.

[58] M. S. et al, "Cross-layer approximate computing: From logic to architectures", in *Design Automation Conference (DAC), 53nd ACM/EDAC/IEEE*, IEEE, Austin, TX, USA: IEEE, 2016.

[59] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, "Advances in neural information processing systems 2", in, D. S. Touretzky, Ed., San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. Handwritten Digit Recognition with a Back-propagation Network, pp. 396–404, ISBN: 1-55860-100-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=109230.109279.

[60] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu, "Approximate computing for low power and security in the internet of things", *Computer*, vol. 50, no. 6, pp. 27–34, 2017, ISSN: 00189162. DOI: 10.1109/MC.2017.176. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85020940495%7B%5C&%7Ddoi=10.1109%7B%5C%%7D2FMC.2017.176%7B%5C&%7DpartnerID=40%7B%5C&%7Dmd5=.

[61] *Camaras Viales*. [Online]. Available: https://www.camarasvialescr.com/.

[62] *Live Traffic Cameras*. [Online]. Available: http://www.dot.ca.gov/video/.

[63] *cv::VideoCapture Class Reference*. [Online]. Available: https://docs.opencv.org/3.1.0/d8/dfe/classcv_1_1VideoCapture.html.

[64] D. Neumann, T. Langner, F. Ulbrich, D. Spitta, and D. Goehring, "Online vehicle detection using Haar-like, LBP and HOG feature based image classifiers with stereo vision preselection", *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 773–778, 2017. DOI: 10.1109/IVS.2017.7995810.

[65] J. Cruz, E. Shiguemori, and L. Guimarães, "A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery", *Int. Sci. J. Comp. Int. Sci*, pp. 121–1363, 2015. DOI: 10.6062/jcis.2015.06.01.0101. [Online]. Available: http://epacis.net/jcis/PDF%7B%5C_%7DJCIS/JCIS-0101.pdf%7B%5C%%7D0Ahttp://epacis.net/jcis/PDF%7B%5C_%7DJCIS/JCIS11-art.96.pdf%7B%5C%%7D0Ahttp://epacis.net.

[66] A. Sharifara, M. S. Mohd Rahim, and Y. Anisi, "A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection", *Proceedings - 2014 International Symposium on Biometrics and Security Technologies, ISBAST 2014*, pp. 73–78, 2015. DOI: 10.1109/ISBAST.2014.7013097.

[67] D. T. Nguyen, T. D. Pham, N. R. Baek, and K. R. Park, "Combining deep and handcrafted image features for presentation attack detection in face recognition systems using visible-light camera sensors", *Sensors (Switzerland)*, vol. 18, no. 3, 2018, ISSN: 14248220. DOI: 10.3390/s18030699.

[68] T. Ball, *Train Your Own OpenCV Haar Classifier*, Jul. 2013. [Online]. Available: https://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html.

[69] O. C. V. Library, Ed., *Tracking API*. [Online]. Available: https://docs.opencv.org/3.4/d9/df8/group__tracking.html.

[70] O. C. V. Library, Ed., *Cascade Classification*. [Online]. Available: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html.

[71] L. A. F. Fernandes, "Um método projetivo para cálculo de dimensões de caixas em tempo real", p. 126, 2006. [Online]. Available: http://hdl.handle.net/10183/6958.

[72] SIECA, "Manual Centroamericano de Dispositivos Uniformes para el Control del Tránsito", p. 434, 2000. [Online]. Available: https://www.csv.go.cr/documents/10179/10903/Manual+Centroamericano+de+Dispositivos+Uniformes+para+el+Control+de+Tr%7B%5C'%7Ba%7D%7Dnsito.pdf/e0765c16-b565-4fa2-bfdf-811949eeb71f.

[73] Xilinx, Ed., *Zynq-7000 SoC Product Advantages*. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[74] Xilinx, Ed., *Xilinx OpenCV User Guide*, Jul. 2017. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1233-xilinx-opencv-user-guide.pdf.

[75] Xilinx, Ed., *SDSoC Environment UserGuide*, Jan. 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1027-sdsoc-user-guide.pdf.

[76] OpenCV, Ed., *OpenCV: Color Conversions*. [Online]. Available: https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html.

[77] Matworks, Ed., *Compute peak signal-to-noise ratio (PSNR) between images*. [Online]. Available: https://la.mathworks.com/help/vision/ref/psnr.html.

[78] Matworks, Ed., *Structural Similarity Index (SSIM) for measuring image quality*. [Online]. Available: https://la.mathworks.com/help/images/ref/ssim.html.

[79] TutorialsPoint, Ed., *C++ Multithreading*. [Online]. Available: https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm.

[80] S. of Delaware, *Marking*, 2011. [Online]. Available: http://regulations.delaware.gov/register/july2011/final/mutcd/Part3.html.

Chapter 6

Appendices

6.1 Config file example: configs.yml

```
%YAML:1.0
---
% Paths
inputVideoPath: "videos/GeneralCn_afternoon.mp4"
outputVideoPath: "videos/GeneralCn_afternoon_output.wmv"
datasetPath: "data/cascade.xml"
maskPath: "data/GeneralCn.png"
% Transformation matrix
TransformationMatrix: !!opencv-matrix
   rows: 2
   cols: 2
   dt: f
   data: [ 4.87888008e-02, 2.54519999e-01, 4.79119986e-01,
       -9.19047296e-02 ]
% Detection configs and extrinsic data
DetectionROI: [ 190, 190, 135, 95 ]
ObservationArea: 6.72e+01
DeltaX_Transform: 0
minObjectSize: [ 20, 20 ]
minOverSize: 3
% Preprocessor configs
erodeKernel: 3
gaussianKernel: 3
erode: 1
gaussian: 0
% Device ID for server service
deviceUID: 2
```

```
RouteDirection: 0
% Timeouts
zoneROITime: 10.0
zone0Time: 6.0
% Mean filter
nMean: 5
% Timer
analysisInterval: 3.00000000e+03
% Approximation configs
d1_detector: 1.00000000e-03
d1_tracker: 8.00000000e-03
d2_detector: 5.00000000e-02
d2_tracker: 5.00000000e-02
F2_detector: 2
F2_tracker: 0
```

Actually, this file is used for the *General Cañas* video sequence and its information is real and usable.

6.2 Data type definitions

6.2.1 Global Variables

This data type envolves important parameters needed in different modules and to storage the global results.

```
/*
    Global Variables data type definition
    - Used globally for important parameters encapsulation -
*/
typedef struct GlobalVars{
    // Occupancy
    double timeAccOccupancy = 0;
    unsigned long numberOfOccupancy = 0;
    double Occupancy = -1;
    // Number of cars
    unsigned int numberOfCars = 0;
    // Flow
    unsigned int FlowCounter = 0;
    unsigned int FlowCounterTotal = 0;
    double Flow = 0;
    // Speed
    double AvSpeed = -1;
    double AvSpeedAcc = 0;
    unsigned int AvSpeedCounter = 0;
    double AvAnalysisSpeedAcc = 0;
    unsigned int AvAnalysisSpeedCounter = 0;
    // Density
    double AvDensity = -1;
    double AvAnalysisDensityAcc = 0;
    unsigned int AvAnalysisDensityCounter = 0;
    // Gross number of cars
    unsigned int totalCars = 0;
    // Video info
    double actualtime = 0;
    int framesPerSecond = 30;
    // Timing
    double AvRealtimeClock = 0;
    double AvAnalysisClock = 0;
    // Mean measurement
    std::queue<double> * AnalysisSpeed;
    std::queue<double> * AnalysisDensity;
```

```
std::queue<double> * AnalysisFlow;
std::queue<double> * AnalysisOccupancy;
/*
    Settings
*/
// Mask location
std::string maskPath = "";
// Min SIze
cv::Size minObjectSize;
float minOverSize;
// Detection ROI
cv::Rect DetectionROI;
float zoneROITime;
float zone0Time;
// Enable filters
bool erode = true;
bool gaussian = false;
// Filters deepness
int erodeKernel = 3;
int gaussianKernel = 3;
// Transformation matrix
cv::Mat TransformationMatrix;
// Observation area
float ObservationArea = -1; // m2
bool DeltaX_Transform = false;
// Sampling
int nMean = 1;
double analysisInterval = 3000;
/*
    Communications
*/
std::queue<outgoingMsgStr*> * outgoingQueue;
std::queue<incomingMsgStr*> * incomingQueue;
/*
    Route info
*/
int deviceUID = -1;
int RouteDirection = -1;
/*
    Benchmarking
*/
#ifdef TIME_PROFILING
MainProfile * appProfile;
```

```
        #endif

        /*
            Approximation
        */
        #ifdef USING_APPROXIMATION
            float frameSkipperScope[2] = {0,0};
            float frameSkipperIntersect[2] = {0,0};
            int frameSkipper_F2[2] = {0,0};
        #endif
        #if defined(USING_OPTIMISED_MOSSE) ||
        defined(USING_OPTIMISED)
            std::queue<meter::approximate::filterUpdater_t*> * _
            updaterQueue = NULL;
        #endif

    } Vars;
```

6.2.2 Vehicle class

Vehicle class helps to manage each object of interest easier and more compact. This data type contains the pointer to a tracker, which helps to track the object frame-per-frame and to a FSM class, which manages the state transition of the OoI.

```
/*
    Vehicle class
    - Contains important info about vehicles
    // (objects of interested detected and tracked) -
*/
class Vehicle {
    cv::Rect2d ROI = cv::Rect2d(0,0,1,1);
    cv::Point CenterOfMass = cv::Point(0,0);
    float Speed = 0;
    int ActualZone = 0;
    double lastTimeDetected = 0;
    cv::Ptr<cv::Tracker> Tracker;
    float Area = 0;
    bool firstTracked = false;
    FSM<Vehicle> * StateMachine;
    public:
        // Get Pointers for destroyer
        FSM<Vehicle> * getFSM(){return StateMachine;}
        cv::Ptr<cv::Tracker> getTracker(){return Tracker;}
        // Constructor
        Vehicle();
        // Get State
        int getState(){return StateMachine->getState();}
        // Get CenterOfMass and ROI
        cv::Point getCenterOfMass(){return CenterOfMass;}
        cv::Rect2d getROI(){return ROI;}
        // Initialize state machine
        void initFSM(){StateMachine = new FSM<Vehicle>(this);}
        // Change State
        bool changeState(int transition, Vehicle object,
        GlobalVars * vars){return StateMachine->changeState(
            transition, vars);}
        // Refresh
        bool refreshPosition(double actualTime,
        cv::Mat & frame, cv::Mat & mask,
        cv::CascadeClassifier detector, GlobalVars * vars);
        // Create
        bool begin(cv::Rect2d gROI, cv::Point CoM,
```

```
            float detectedArea, GlobalVars * vars);
            // Public vars
            double StartTime ;
            double EndTime ;
            double SpawmTime;
};
```

6.2.3 FSM class

FSM class helps to ease the Ooi management in order to get different values. The subroutine *changeState()* is defined completely in a distinct file, because of its complexity.

```
/*
    State machine
    - Helps to manage the state transition of OoI -
*/
template <class T>
class FSM {
    int State; // Initial State
    T * ObjectV;
public:
    // Get State
    int getState(){return State;}
    // Change State
    bool changeState(int, GlobalVars *);
    // Set State
    void setState(int fixstate){State = fixstate;}
    // Constructor
    FSM (T * VehicObj);
};
```

6.2.4 Profiling classes

To get a detailed application profile, it is needed to implement an internal profiler that can average and calculate the standard deviation of time values got when a subroutine is executed. With these classes, it was possible to get the results and see the evolution after applying improvements.

```
/*
    SubProfile: manage an specific number of cars combination
*/
class SubProfile{
    // Label

    // First and second moment
    double firstMoment = 0, secondMoment = 0;
    // Sample counting
    unsigned long samples = 0;
    // Standard Dev
    float stdDev = 0;
    // Points
    std::chrono::high_resolution_clock::time_point startPoint;
    std::chrono::high_resolution_clock::time_point endPoint;
    // Functions
    public:
        std::string label; // Example: main
        SubProfile * next;
        SubProfile(std::string);
        void addSample(double);
        void setStart();
        void setEnd(std::chrono::high_resolution_clock::_
        time_point endTime);
        void close(double &mean, double &variance,
        std::string &name);
};

/*
    Profilers - are used for a entire module
*/
class Profile{

    // Main profile
    SubProfile * module;
    // Subprofiles - list
    SubProfile * submodulestop;
```

```
    SubProfile * submodulesbottom;
    // Methods
    public:
        // Label
        std::string ModuleName;
        int nCar_ROI, nCar_Zone0;
        unsigned long iterations;
        Profile * next;
        Profile(std::string name, int detection, int ROI);
        // Subprofiles
        SubProfile * LookSubModule(std::string name);
        SubProfile * CreateSubProfile(std::string name);
        SubProfile * GetTop(){return submodulestop;}
        // Profile ops
        void addSample(double sample){module->addSample(sample);
        iterations++;}
        void setStart(){module->setStart();}
        void setEnd(
            std::chrono::high_resolution_clock::time_point _
            endTime)
            {module->setEnd(endTime); iterations++;}
        void close(double &mean, double &variance,
        std::string &name);
};


/*
    General profile
*/
class MainProfile
{
    // Label
    std::string appName = "main";
    // Main profile
    SubProfile * app;
    // Profiles - list
    Profile * modulestop;
    Profile * modulesbottom;
    // Methods
    public:
        MainProfile();
        // Profile ops
        Profile * LookProfileByDetection(std::string name,
```

```
            int detection);
            Profile * LookProfileByROI(std::string name, int ROI);
            Profile * LookProfile(std::string name);
            Profile * CreateProfile(std::string name,
            int detection, int ROI);
            Profile * GetTop();
            // General ops
            void addSample(double sample){app->addSample(sample);}
            void setStart(){app->setStart();}
            void setEnd(
                std::chrono::high_resolution_clock::_
                time_point endTime)
                {app->setEnd(endTime);}
            void close(double &mean, double &variance,
            std::string &name);
    };
```

6.2.5 TCP message classes

The messages sent and got through TCP are encapsulated into these classes to ease their transmission.

```
typedef struct incomingMsgStr{
    char message[MAXDATASIZE];
} incMsg;

typedef struct outgoingMsgStr{
    // String mode (using message buffer) or
    // making the string from variables
    bool strMode = false;
    // Realtime mode or analysis mode
    bool rtMode = false;
    // String buffer
    char message[MAXDATASIZE];
    // Numeric values
    float speed = -1;
    float density = -1;
    float flow = -1;
    float occupancy = -1;
    int uid = -1;
    int direction = -1;

} outMsg;
```

6.2.6 MOSSE filter object for TO-2

This data structure groups the elements needed to update coupled filters for MOSSE tracking.

```
typedef struct filterUpdater_t{
    // Filter computation values - Modificable
    cv::Mat * A;
    cv::Mat * B;
    cv::Mat * H;
    cv::Mat * G;
    // Filter constants
    cv::Mat F;          // Generate by DFT
    cv::Mat A_new;
    cv::Mat B_new;
    double rate=0;
    } FilterUpdater_t;
```

6.3 Detailed block diagrams

6.3.1 Retriever block diagram

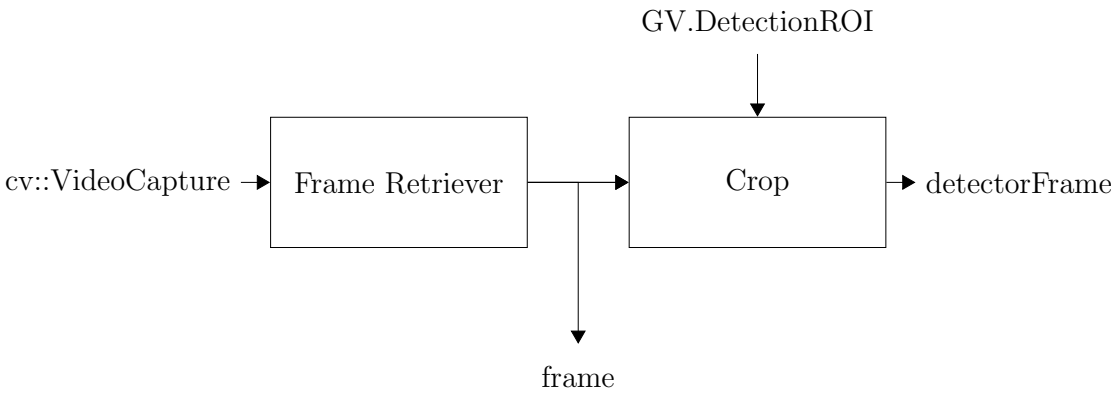


Figure 6.1: *Retriever* block diagram based on Figure 3.11

6.3.2 Detector block diagram

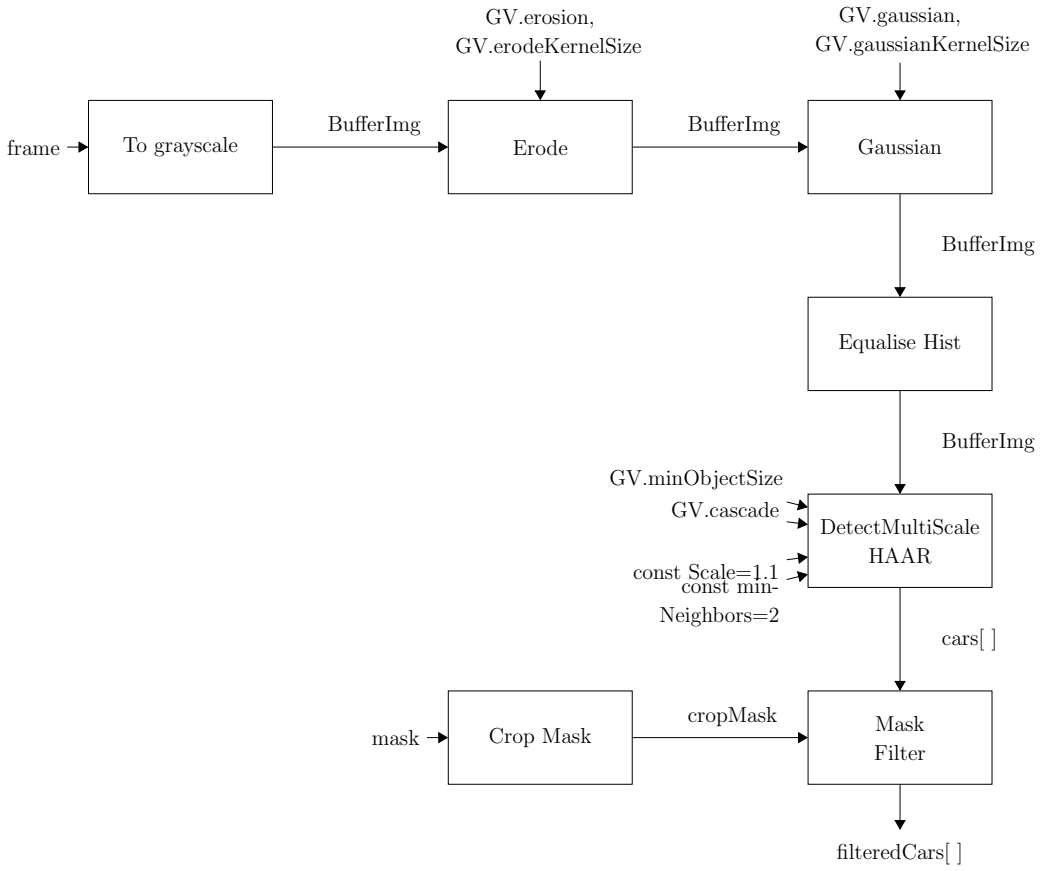


Figure 6.2: *Detector* block diagram based on Figure 3.11

6.3.3 Queue push manager block diagram

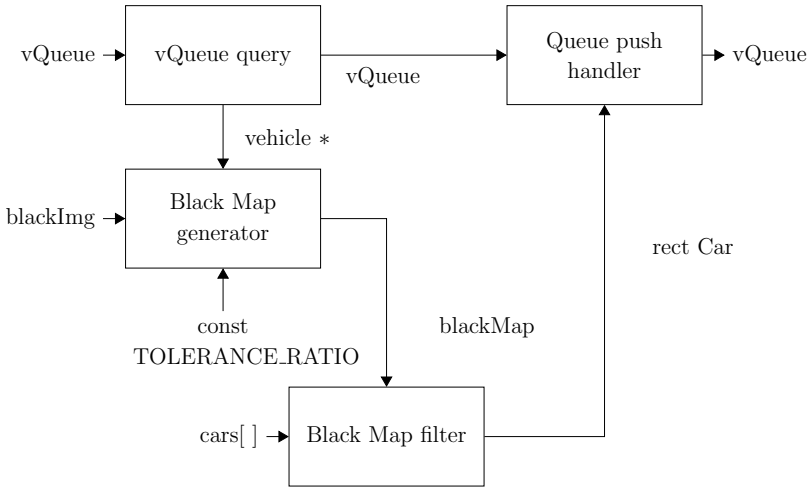


Figure 6.3: *Queue push manager* block diagram based on Figure 3.11

6.3.4 Queue refresh manager block diagram

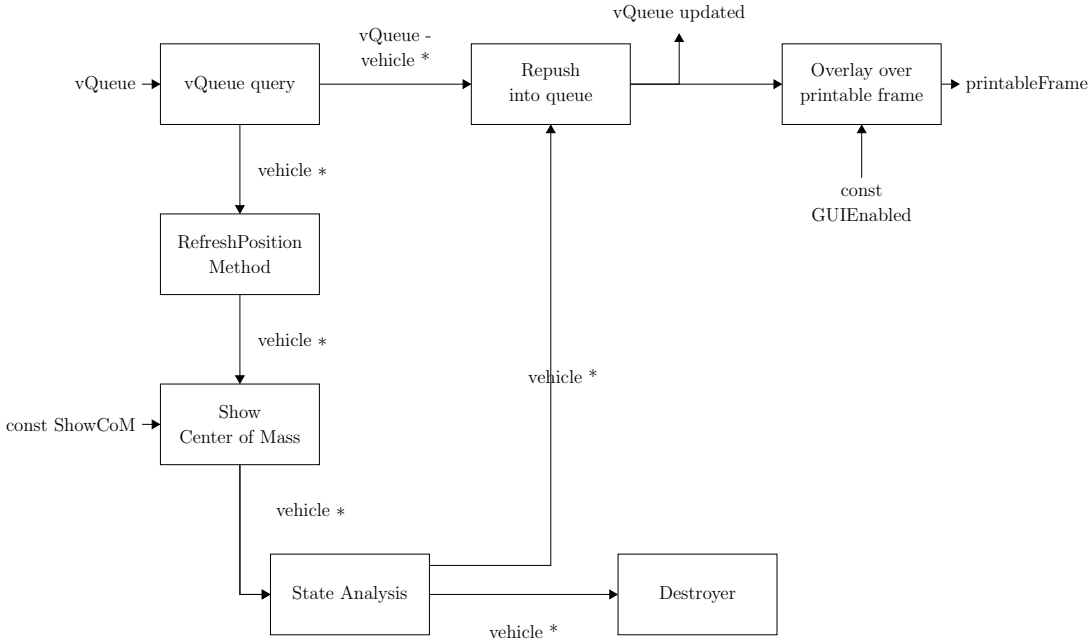


Figure 6.4: *Queue refresh manager* block diagram based on Figure 3.11

6.3.5 GUI module block diagram

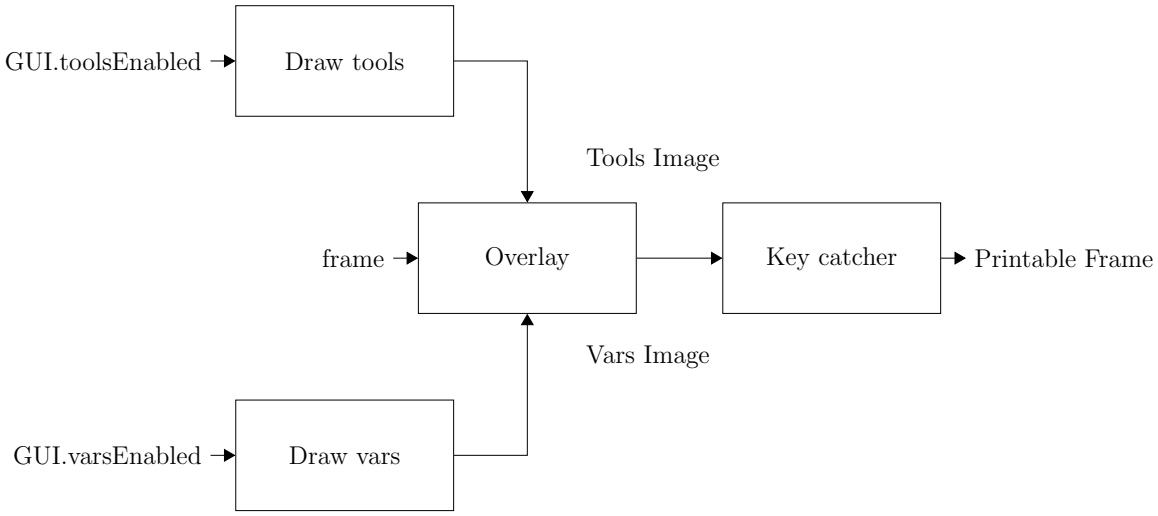


Figure 6.5: *GUI module* block diagram based on Figure 3.11

6.3.6 MOSSE tracker update subroutine

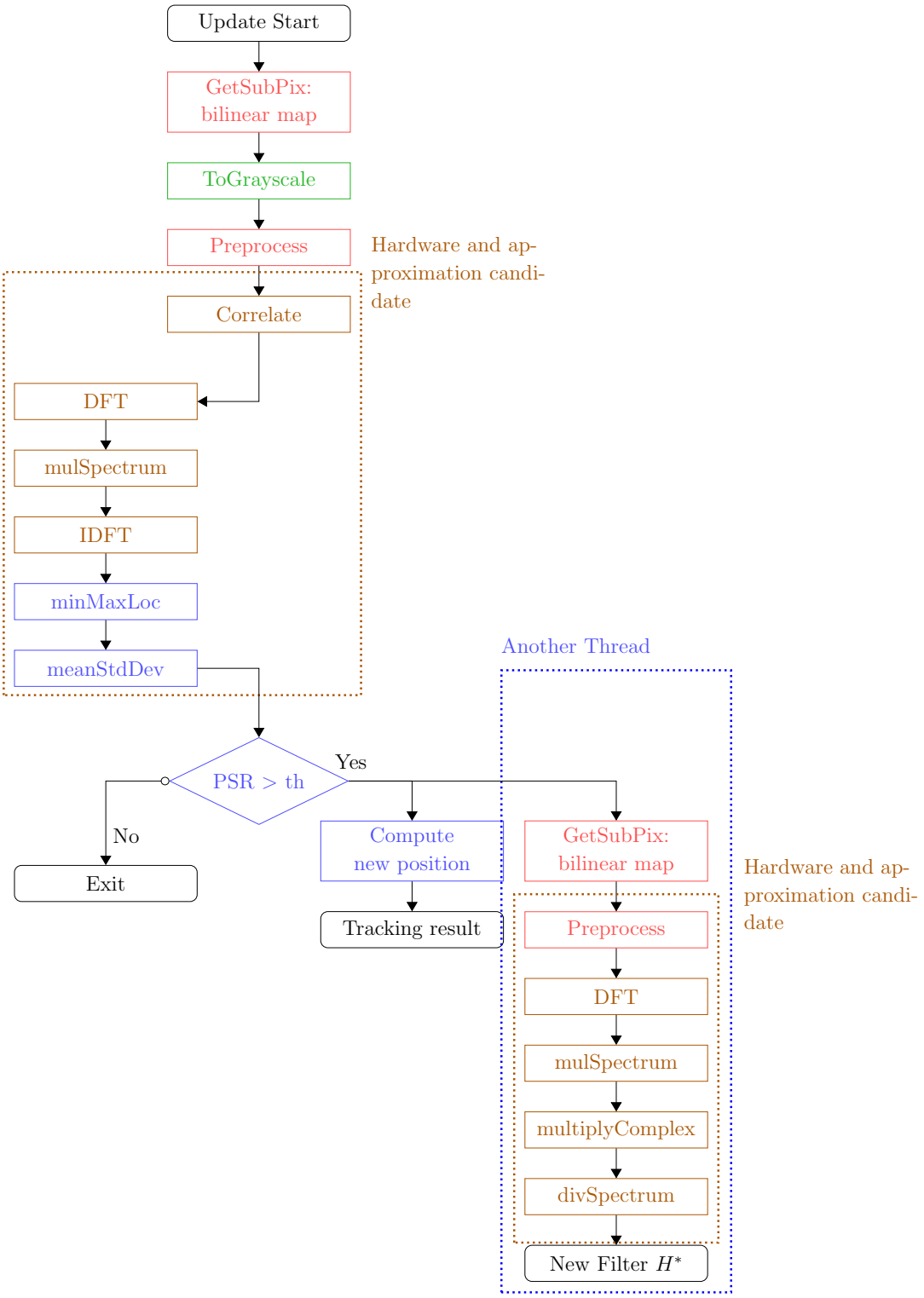


Figure 6.6: *Queue refresh manager* block diagram based on Figure 3.11

6.4 Γ matrix computations

6.4.1 General considerations

Before starting to compute the Γ matrix for each video sequence, it is important to define which properties can be found in each video. These hints are, for example, lane width, dashed lines, light reflectors, etc.

Costa Rican roads

COSEVI is in charge of Costa Rican road regulation. According to its website, Costa Rican roads are regulated in the same way to the different Central American countries [72].

In Table 3.5 is summarised all the information about road signaling.

USA (United States of America) roads

To take an estimation of *Soquel Avenue* video sequence, an official document from *State of Delaware* was used in order to get the road dimensions, available in [80].

Table 6.1: Soquel Avenue on route signal dimensions

Detail	National route
Lane width	3.66m
Dashed lines length	3.05m
Space between dashed lines	9.14m
Distance between light reflectors	N/A

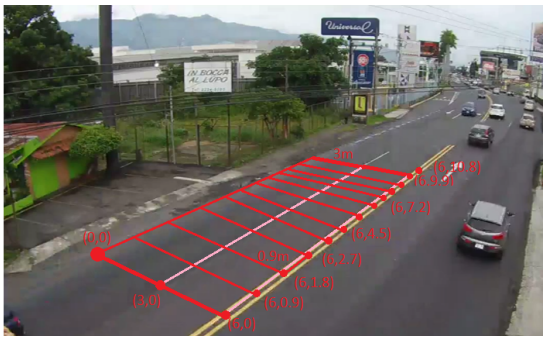
Table 6.1 summarises all the important information about USA roads, especially, for *Soquel Avenue* modelling.

6.4.2 Road characterisation

The strategy to characterise each road is to define points referred to a reference point ($x = 0$,$y = 0$). This estimation does not consider changes in z axis, therefore, this estimation can have a gross error applied to non-well-behaved roads. In this case, all the video sequences are treated as uniform in z for experimental purposes.

Figure 6.7 shows the road marking for each sample video used in this research. Based on road regulations, it is possible to set distance points referred to an arbitrary origin. This allows making pairs of coordinates (image and scene coordinates) to compute a linear equation using *regression*.

(a) General Cañas (in m)



(b) Multiplaza del Este (in m)

(c) Soquel Avenue (in ft)

Figure 6.7: Road signal marking to get a linear function to describe image coordinate system to scene coordinate system

Multiplaza del Este

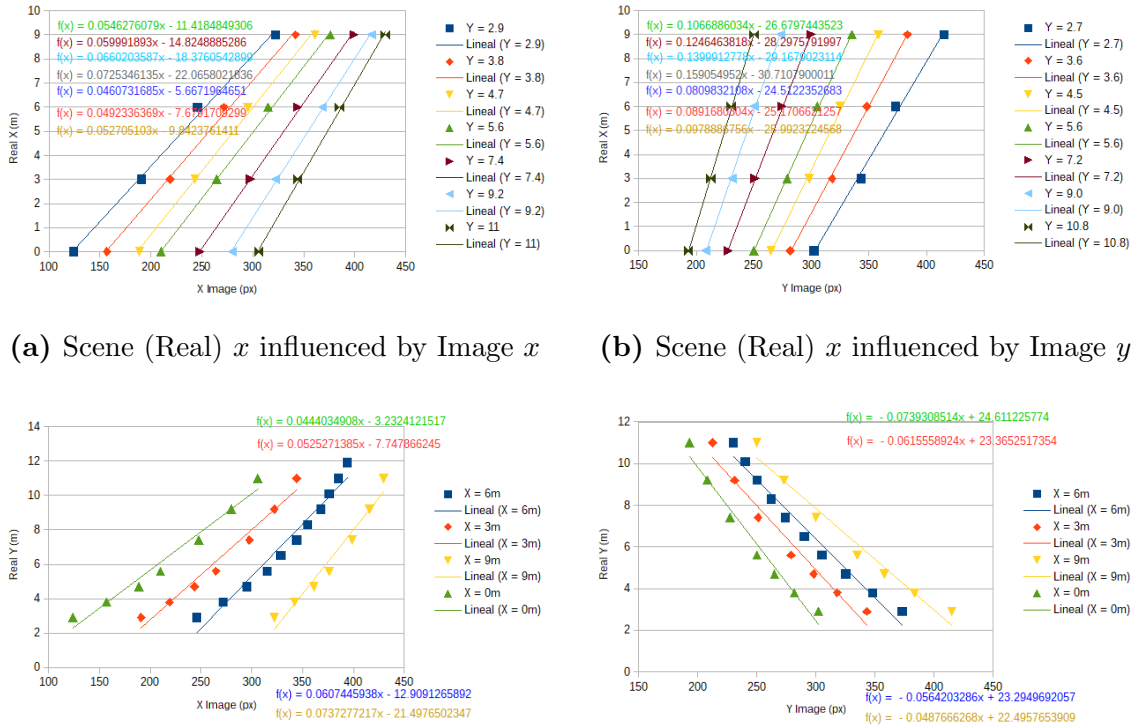
Multiplaza del Este video is one of the well-behaved sequences because of its elements dependence. All the transformation matrix terms are different to infinite. On the other hand, it is possible to compute an average scope with a neglectable standard deviation.

Figure 6.8 shows all the tendency lines needed to calculate the transformation matrix coeficients. To build each chart, it is necessary to fix one of the variables constant (x_{scene} or y_{scene}), and plot the variable of interest vs either x_{image} or y_{image}. This matrix has the following structure:

$$\Gamma = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}$$

Where the first digit of the subindex is the scene variable (1 for x, and 2 for y), and the second digit is for image variable (1 for x, and 2 for y). Therefore, T_{12} is x_{scene} component affected by y_{image}.

Now, the intersection value of each regression tendency line is ignored because of its cancellation when speed is computed. Thus, each T_{uv} represents an average scope value. Taking it into account, the Γ matrix for *Multiplaza del Este* is giving by:



(a) Scene (Real) x influenced by Image x

(b) Scene (Real) x influenced by Image y

(c) Scene (Real) y influenced by Image x

(d) Scene (Real) y influenced by Image y

Figure 6.8: Dispersion charts and regression lines for measuring the affectation scopes - Multiplaza del Este

$$\Gamma_{ME} = \begin{bmatrix} 0.0573 & 0.1141 \\ 0.0485 & -0.0677 \end{bmatrix}$$

Where the subindex ME means *Multiplaza del Este*.

The lane length, according to the characterisation, is $L = 7.8m$. It must be less than the analysed in Γ calculation because L only occupies *Tracking Zone*.

General Cañas

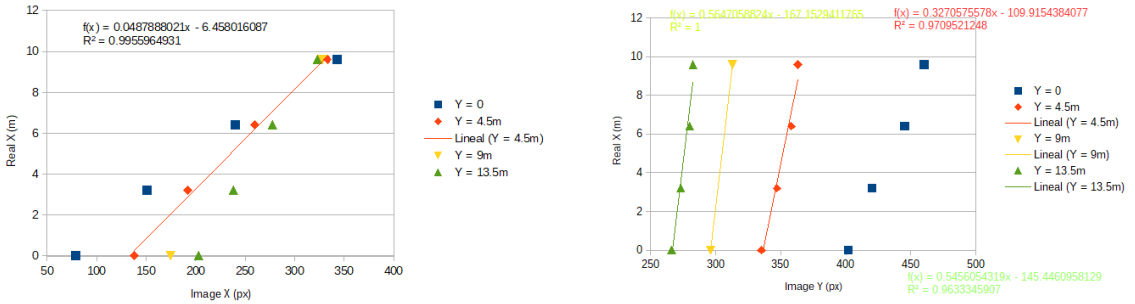
General Cañas Γ is computed as *Multiplaza del Este* Γ is. However, *General Cañas* contains some values that can be greater than *Multiplaza* due to the camera focus, but Γ is still calculable. The Γ matrix for this video sequence, based on Figure 6.8, is given by:

$$\Gamma_{GC} = \begin{bmatrix} 0.0488 & 0.4791 \\ 0.2545 & -0.0919 \end{bmatrix}$$

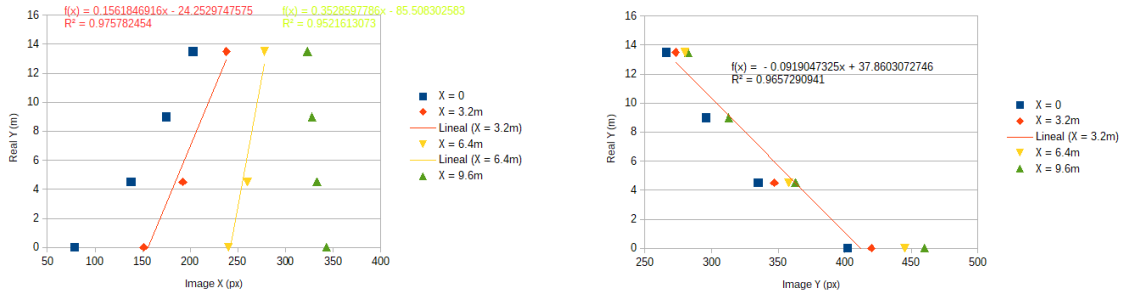
T_{12} and T_{21} values are greater than Γ_{ME}, because of the scope. This difference can be noticed when comparing Figure 6.8 and Figure 6.9

Where the subindex GC means *General Cañas*.

The lane length, according to the characterisation is $L = 67.2m$.



(a) Scene (Real) x influenced by Image x **(b)** Scene (Real) x influenced by Image y



(c) Scene (Real) y influenced by Image x **(d)** Scene (Real) y influenced by Image y

Figure 6.9: Dispersion charts and regression lines for measuring the affectation scopes - General Cañas

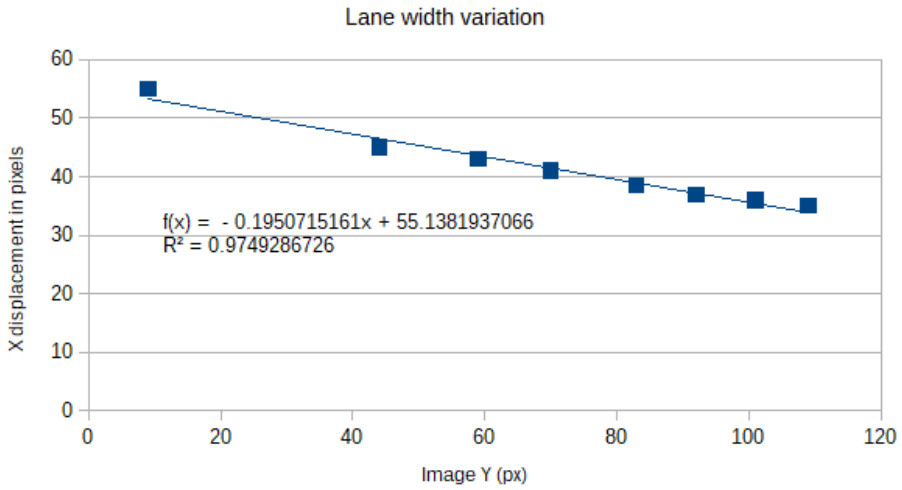
Soquel Avenue

Soquel Avenue video is computed slightly different to the other videos because there is a vanishing point in the vertical-center position of the route under analysis. This causes an indefinition of T_{12}, where the x_{scene} component is infinite, and useless for transforming from *image* coordinate system to *scene* coordinate system.

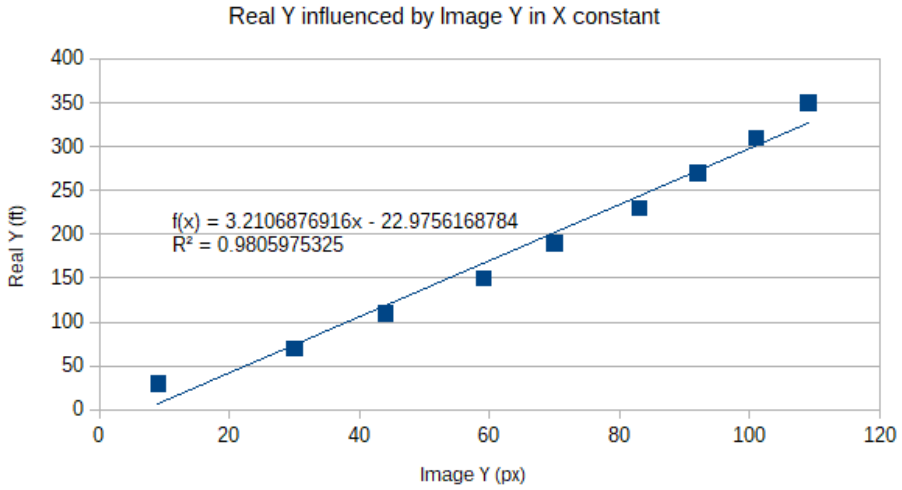
In order to get a way to compute the transformation, instead transforming x_{scene} from x_{image} points, the difference Δx_{scene} is calculated using Δx_{image}. In principle, this implies transforming the *displacement vector* instead a *position vector*. The demonstration for *Modified* Γ *matrix* is shown in Modified Γ demonstration.

Figure 6.10 shows the necessary equations for computing modified Γ matrix. There is no influence of x_{image} on y_{scene}, because the lane is almost vertical, and its scope tends to 0.

In the case of T_{22}, it is computed in the same way of the other videos, and T_{11} and T_{12} are calculated following the demonstration presented in Modified Γ demonstration. Finally,



(a) Δx_{image} tendency respect of y_{scene} for a constant Δx_{scene}



(b) Scene (Real) y influenced by Image y

Figure 6.10: Dispersion charts and regression lines for measuring the affectation scopes - Soquel Avenue

the Γ matrix for *Soquel Avenue* is:

$$\Gamma_{SA} = \begin{bmatrix} 0.2716 & 0.0435 \\ 0 & 3.21 \end{bmatrix}$$

Where the subindex SA means *Soquel Avenue* and the lane length L is 1412 m.

6.4.3 Modified Γ demonstration

Suppose that the equation which describes the displacement in x_{image} (Δx_{image}) in terms of y_{image}, and evaluated in $\Delta x_{scene} = L$, is given by:

$$\Delta x_{image} = a_{12} \cdot y_{image} + b \tag{6.1}$$

When a car enters to the ROI, $y_{image} = 0$. So:

$$\Delta x_{image} = b \tag{6.2}$$

Transforming Δx_{image} to Δx_{scene} implies a linear relation, hence:

$$\Delta x_{scene} = \frac{L}{b} \cdot \Delta x_{image} \tag{6.3}$$

This cancels b and gives the right Δx_{scene}. Therefore, the relation between both displacements is $\frac{b}{L}$.

Now, extrapolating (6.1) to any displacement in Δx_{scene}, the equation to transform from Δx_{scene} to Δx_{image}, and taking into consideration y_{image} influence, is given by:

$$\Delta x_{image} = a_{12} \cdot y_{image} + \frac{b}{L} \cdot \Delta x_{scene} \tag{6.4}$$

Therefore, getting Δx_{scene} from (6.4):

$$\Delta x_{scene} = \frac{(\Delta x_{image} - a_{12} \cdot y_{image}) \cdot L}{b} \tag{6.5}$$

For practical purposes, equation 6.5 was tested successfully, giving results with errors under 20%.

6.5 Architectures comparison table

This section shows the tests proposed in the thesis proposal to prove Objective 3 achievement.

6.5.1 Required tests

Repetitive tests

They are applied to get the average runtime of each module using the same video sequence. The output results are compared between each iteration. The following data is mandatory:

- Average runtime with its variance
- Repetitions number.
- Result errors.

However, *Result errors* does not generate relevant data, because the analysis is done with the same video; so, it will be skipped here, but highlighted in *Global tests*.

These tests allow analysing each module in order to detect the problematic modules in each experiment (video), so as to optimise them.

Global tests

They are acquired after repetitive tests and take into account all the video sequences. The items to evaluate are:

- Average runtime per experiment.
- Exact architecture mean squared error and maximum error.
- Approximate architecture mean squared error and maximum error.
- Mean squared error and maximum error amongst architectures.
- Experiment details.

To fulfill the experiment, there shall be 3 experiments at least, with more than 10 repetitions.

6.5.2 Results

Repetitive tests

General Cañas Video

Table 6.2: Repetitive test results - General Cañas video

Module	Number of cars analysed	Iterations	Mean (s)	Std. Dev. (s)	Contribution
Retriever	-	25316	0.0153547	1.93287E-05	0.0153547
	Total Iterations	**25316**		**Average runtime**	**0.0153547**
Detector	0	19920	0.023117	0.000923	0.018190
	1	828	0.025018	0.000007	0.000818
	2	48	0.027965	0.000003	0.000053
	Total Iterations	**20796**		**Average runtime**	**0.019061**
Queue push manager	0	19920	2.32117E-06	2.77585E-13	1.826E-06
	1	828	0.000354703	1.56659E-08	1.160E-05
	2	48	0.000326254	0.000000014	6.186E-07
	Total iterations	**20796**		**Average runtime**	**0.000014**
Queue refresh manager	0	21724	0.000003	0.000005	0.000002
	1	3364	0.003922	0.000022	0.000401
	2	228	0.006096	0.000000	0.000047
	Total iterations	**25316**		**Average runtime**	**0.000450**
Measurement	0	21724	3.424E-06	1.706E-12	2.938E-06
	1	3364	3.739E-06	6.232E-13	4.969E-07
	2	228	4.274E-06	3.176E-12	3.849E-08
	Total iterations	**25316**		**Average runtime**	**3.474E-06**
Destroyer	-	25316	2.44034E-06	7.75796E-13	2.44034E-06
	Total iterations	**25316**		**Average runtime**	**2.44034E-06**

Multiplaza del Este Video

Table 6.3: Repetitive test results - Multiplaza del Este video

Module	Number of cars analysed	Iterations	Mean (s)	Std. Dev. (s)	Contribution
Retriever	-	29876	0.019406	0.000966	0.019406
	Total iterations	**29876**		**Average runtime**	**0.019406**
Detector	0	8176	0.009013	0.000002	0.002467
	1	1876	0.011190	0.000002	0.000703
	2	7572	0.009860	0.000002	0.002499
	3	52	0.013258	0.000001	0.000023
	Total iterations	**17676**		**Average runtime**	**0.005691**
Queue push manager	0	8176	0.000002	9.223E-13	0.000001
	1	1876	0.000345	2.043E-08	0.000022
	2	7572	0.000294	2.795E-08	0.000074
	3	52	0.000336	1.491E-08	0.000001
	Total iterations	**17676**		**Average runtime**	**0.000097**
Queue refresh manager	0	17988	2.91E-06	2.180E-12	1.75E-06
	1	8960	0.016403	6.451E-05	0.001990
	2	2928	0.020803	6.176E-05	0.001081
	Total iterations	**29876**		**Average runtime**	**0.003073**
Measurement	0	17988	3.47E-06	3.438E-12	2.09E-06
	1	8960	3.88E-06	1.202E-12	1.16E-06
	2	2928	3.97E-06	1.084E-12	3.89E-07
	Total iterations	**29876**		**Average runtime**	**3.64E-06**
Destroyer	-	29876	2.67794E-06	7.97917E-13	2.678E-06
	Total iterations	**29876**		**Average runtime**	**2.678E-06**

Soquel Avenue Video

Table 6.4: Repetitive test results - Soquel Avenue video

Module	Number of cars analysed	Iterations	Mean (s)	Std. Dev. (s)	Contribution
Retriever	-	29876	0.019406	0.000966	0.019406
	Total iterations	**29876**		**Average runtime**	**0.019406**
Detector	0	11154	0.029681	1.96E-05	0.011321
	1	4410	0.032253	1.95E-05	0.004864
	2	816	0.034496	1.43E-05	0.000963
	3	48	0.041340	3.15E-05	0.000068
	Total iterations	**16428**		**Average runtime**	**0.017215**
Queue push manager	0	11154	2.35E-06	3.71E-14	8.96E-07
	1	4410	3.50E-04	1.39E-08	5.28E-05
	2	816	4.20E-04	1.19E-08	1.17E-05
	3	48	4.48E-04	6.98E-09	7.36E-07
	Total iterations	**16428**		**Average runtime**	**6.61E-05**
Queue refresh manager	0	840	2.98E-06	7.38E-13	8.57E-08
	1	1044	0.030870	0.000306	0.000360
	2	2040	0.037752	0.000360	0.001183
	3	3444	0.041418	0.000417	0.002429
	4	3432	0.048069	0.000325	0.003201
	5	2418	0.059303	0.000484	0.003184
	6	1380	0.061577	0.000230	0.001925
	7	366	0.077101	0.000433	0.000705
	8	504	0.083924	0.000087	0.001088
	9	348	0.095339	0.000467	0.000887
	10	78	0.109376	0.000677	0.000236
	Total iterations	**15894**		**Average runtime**	**0.015199**
Measurement	0	876	4.14E-02	4.17E-04	1.24E-03
	1	1662	4.00E-06	4.65E-12	2.28E-07
	2	3276	4.31E-06	3.73E-12	4.83E-07
	3	6696	4.33E-06	2.81E-12	9.91E-07
	4	6510	4.75E-06	3.24E-12	1.06E-06
	5	4710	4.94E-06	4.14E-12	7.96E-07
	6	2730	5.05E-06	4.32E-12	4.72E-07
	7	864	4.96E-06	1.23E-12	1.47E-07
	8	1068	5.43E-06	4.43E-12	1.98E-07
	9	696	5.62E-06	1.89E-12	1.34E-07
	10	156	6.17E-06	2.99E-12	3.29E-08
	Total iterations	**29244**		**Average runtime**	**0.001245**
Destroyer	-	29244	2.66549E-06	3.13355E-12	2.66549E-06
	Total iterations	**29244**		**Average runtime**	**2.66549E-06**

Runtime architecture comparison

To understand better the improvement achieved by applying Approximate Computing on the baseline, the following tables show the runtime consumption by each module in *milliseconds*.

Table 6.5: Architectures runtime comparison - General Cañas video

General Cañas	**Architecture**		
Module	**Exact**	**Approx.**	**Improvement**
Retrieve	10.289	13.977	0.736
Detector	22.475	19.061	1.179
Queue push manager	0.031	0.014	2.228
Queue refresh manager	4.408	0.448	9.850
Measurement	0.004	0.003	1.130
Destroyer	0.010	0.006	1.697
Total	**37.217**	**33.509**	**1.111**

Table 6.6: Architectures runtime comparison - Multiplaza del Este

Multiplaza del Este	**Architecture**		
Module	**Exact**	**Approx.**	**Improvement**
Retrieve	10.284	17.746	0.579
Detector	9.071	5.691	1.594
Queue push manager	0.109	0.097	1.125
Queue refresh manager	23.607	3.071	7.687
Measurement	0.005	0.004	1.249
Destroyer	0.011	0.006	1.730
Total	**43.087**	**26.616**	**1.619**

Table 6.7: Architectures runtime comparison - Soquel Avenue

Soquel Avenue	**Architecture**		
Module	**Exact**	**Approx.**	**Improvement**
Retrieve	13.480	15.640	0.862
Detector	26.362	14.252	1.850
Queue push manager	0.082	0.066	1.233
Queue refresh manager	75.262	15.199	4.952
Measurement	0.006	0.005	1.295
Destroyer	0.012	0.007	1.693
Total	**115.204**	**45.169**	**2.550**

Conclusion

Table 6.2, Table 6.3 and, Table 6.4 present final architecture results with Approximate Computing approximations and software optimisations. Problematic modules, such as *Retriever*, *Detector*, and *Queue refresh manager* have still an important impact on runtime, but *Detector*, and *Queue refresh manager* can be much more optimised implementing their functions by hardware. For future research, these modules are potential candidates to optimise and they might have a 5x in speedup according to results presented during this document.

To get the improvement after implementing optimisations on the application, Table 6.5, Table 6.6, and Table 6.7 show the frame processing runtime between architectures. The maximum speedup achieved was 2.5%, whereas the minimum was 1.1%. Seeing the total time to process each frame, and establishing 166.5 ms as the maximum (condition 5 minutes to process one-minute video), the final application meets with *off-line execution* requirement. But, in the case of *Multiplaza del Este*, the improvement is enough to run *on-line*.

Global tests

Experiment details

The experiment details are summarised as follows:

Table 6.8: Experiment details

Detail	General Cañas	Multiplaza del Este	Soquel Avenue
Frame size	640x480	640x480	640x480
Frames per second	30fps	30fps	25fps
Duration	42 s	49 s	84 s
Max. object size	20x20	50x50	50x50
Max. Number of cars in Detection Zone (exact)	3	4	3
Max. Number of cars tracked (exact)	3	4	10
Max. Number of cars in Detection Zone (approx)	2	3	3
Max. Number of cars tracked (approx)	2	2	10
Lane length	67.2m	78m	1492m

Despite the difference between *number of cars* in the exact and approximate architecture, the additional car runtime affectation is neglectable.

Table 6.9: Average runtime per experiment

Average runtime	General Cañas	Multiplaza del Este	Soquel Avenue
Total exact arch.	51.8s	70.8s	165.4s
Total approx. arch.	42.4s	39.8s	110.1s
Relative exact arch.	111.65%	129.26%	281.08%
Relative approx. arch.	100.53%	79.85%	112.95%

Only *Multiplaza del Este* passed the On-line requisite. However, all the videos passed Off-line requirement, because all consume less than 500% of the frame period.

Theoretical evaluation

These tests evaluate each architecture against theoretical values to verify its effectiveness against the real world results. In each test, the maximum error obtained during results analysis, and the RMSE (Root Mean Squared Error) is studied to verify that the error is

under 20%. RMSE helps to know the quality of each result and projects typical errors in each architecture.

Table 6.10: Exact architecture evaluated against theoretical values

Description	Speed	Density	Flow	Occupancy
MSE	34.173	1.05E-05	0.021	0.124
RMSE	5.846	3.24E-03	0.146	0.352
Max Range	80 km/h	0.05 veh/m	1 veh/s	5 s/veh
Margin	7.31%	6.48%	14.55%	7.04%
Max. Error	-7.21%	-7.85%	-17.04%	-7.81%

The *exact architecture* (Table 6.10) demonstrates being inside 20% tolerance. The most critical error is -17.04%, which indicates that the flow is being underdetected by this architecture (the real value is higher than obtained by the software). The same phenomenon occurs for the rest of values. The most problematic experiment (giving errors in Speed, Density and Flow) is Soquel Avenue, followed by General Cañas.

Table 6.11: Approximate architecture evaluated against theoretical values

Description	Speed	Density	Flow	Occupancy
MSE	69.389	4.57E-05	0.014	0.190
RMSE	8.330	6.76E-03	0.117	0.436
Max Range	80 km/h	0.05 veh/m	1 veh/s	5 s/veh
Margin	10.41%	13.52%	11.66%	8.71%
Max. Error	-9.17%	-19.05%	-18.59%	-10.59%

In the case of *approximate architecture* (Table 6.11), the result errors continue under 20% tolerance. The highest typical error given by RMSE is 13.52%, being lower than *exact architecture*, showing that the improvements given by approximation do not affect drastically the final results. However, the highest error obtained is -19.05%, repeating the phenomenon seen in *exact architecture*.

Architecture evaluation

This test compares both architectures developed during this research in the same way to the tests presented before.

The difference between architectures is demonstrated by Table 6.12. Even though they are based on the same algorithm, the preprocessor introduces an enhancement to approximate architecture, improving results respect an exact architecture. Therefore, despite having different results, the results presented in Table 6.10 and 6.11 does not allow comparing the architectures from those results; hence, a complete analysis (already presented in Table 6.12) has to be done in order to have a comparison.

Table 6.12: Approximate architecture evaluated against exact architecture

Description	Speed	Density	Flow	Occupancy
MSE	12.713	1.92E-05	0.00732	0.00724
RMSE	3.565	4.38E-03	0.086	0.085
Max. Range	80 km/h	0.05 veh/m	1 veh/s	5 s/veh
Margin	4.46%	8.77%	8.55%	1.70%
Max. Error	-5.89%	15.00%	14.19%	3.01%

Conclusion

Both architectures meet the requirements shown by Table 3.2. Also, with this comparison, the problematic modules were discovered, and the third objective was successfully accomplished by completing the comparison charts.

Besides, the results presented in the body of this document shows a detailed version of this analysis.

6.6 Requirements table

In this section, the project application is verified against the requirements table presented in Table 3.2, in order to prove research goal completeness.

Table 6.13: Requirements achievement

Concept	Element	Requisite	Achivement
Results tolerance	Output results tolerance	All the results are under 20%	Yes. Highest: 19.05% - Achieved in Table 6.11
Image processing	Image capture	Each frame is retrieved and decoded under 2.2 seconds, to capture cars at 40km/h in a distance lower than 10 meters	Yes .Highest: 17.75 ms - Achieved in Table 6.6
	Preprocessor	Capability to be implemented in an ARM architecture	Yes. Check it in [74]
		Have a kernel size at least of 3x3 pixels	
		Be separable (parallelisable)	
Image analysis	Det. and Track.	Transformation invariant or resilient	Yes. See 3.2-3.3
		Avoid object overlapping	
		Possibility to be implemented in an ARM architecture	
	Object classification	The trained stages have to be able to be supervised-trained	Yes. See 3.2.2
Execution modes	On-line	The application is ideally able to be executed on-line. That means that all the process should be done in a frame period	Yes. See 4.3.4
	Off-line	If the on-line execution can not be met, the application must take up to 5 minutes to process a 1 minute video sequence	

In Table 6.13, the requirements presented in Table 3.2 have been checked, showing that the project accomplishes with all the requirements defined in the thesis proposal.