Using Existing Surveillance Infrastructure To Monitor Pedestrians on Pedestrian Bridges: A Proof Of Concept

by

Hardy Fraser Van Der Merwe

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electrical & Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. MJ. Booysen

Co-supervisor: Dr. SJ. Andersen

December 2016

The financial assistance of the South African National Roads Agency .Ltd (SANRAL) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the SANRAL

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2016 .....

Abstract

Using Existing Surveillance Infrastructure To Monitor Pedestrians on Pedestrian Bridges: A Proof Of Concept

HF. Van Der Merwe

Department of Electrical & Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (E&E)

September 2016

Due to the high number of pedestrians involved in crashes along the freeways in South Africa, there is a need for authorities to monitor pedestrians using the freeways in order to gain valuable mobility information. As South Africa is a developing country, existing surveillance infrastructure must be used to its fullest extent.

This work presents a proof of concept that existing surveillance infrastructure can be used to implement a pedestrian monitoring system. The system will monitor pedestrians using pedestrian bridges along the freeways, and will collect pedestrian mobility information, such as the direction and relative speed of movement. The system thus aims to contribute to pedestrian safety in South Africa.

A cascade classifier is trained with surveillance footage collected from SANRAL using AdaBoost and the use of approximated image scales. It is shown that the detectors need to be trained specifically for a scene/camera using footage from that scene/camera in order to compete with modern pedestrian detection applications. The Integral Channel Features (ICF) detection method, which incorporated HOG and colour channels, is used in this thesis as it is easily implemented and achieved state-of-the-art performance. A graphical user interface is created to run the training and detection phases as this removes the user form the complicated back-end.

The detected pedestrians are tracked using parametric tracking methods such as a Kalman filter, and supplementary track management functions. The tracked pedestrians are automatically counted and compared to pedestrian counts done by hand. The results show that the automatic pedestrian counting system achieved an accuracy of more than 90%. A user friendly verification user interface is created to allow for the meaningful representation the pedestrian mobility results.

A further analysis of the pedestrian mobility information collected, shows that the peak hour pedestrian traffic in the morning is between 06:30-07:30, and in the afternoon between 17:00-18:00. The data also shows that pedestrian traffic in the morning is highest towards the CDB, and in the afternoon is away from the CBD. It can also be seen that pedestrians move on average 10% faster towards the CBD.

The results is proof that the pedestrian monitoring system using existing low-resolution surveillance infrastructure on pedestrian bridges can be used to obtain state-of-the-art performance. Thus it demonstrates that the system achieves the goals set, and that the application of such a system is to be kept in mind during future infrastructure installations and upgrades.

Uittreksel

Die Gebruik Van Bestaande Sekuriteit Infrastruktuur Om Voedgangers Op Voedganger-brue Waar Te Neem: 'n Bewys Van Konsep

("Using Existing Surveillance Infrastructure To Monitor Pedestrians on Pedestrian Bridges: A Proof Of Concept")

HF. Van Der Merwe

Departement Elektries & Elektronies Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (E&E)

September 2016

As gevolg van die hoÃ« aantal voetgangers betrokke in ongelukke langs die snelweÃ« in Suid Afrika, is dit noodsaaklik vir die betrokke owerhede om voetgangers te monitor en so belangrike bewegingsinligting te genereer. Suid Afrika 'n ontwikkelende land en dus moet die bestaande veiligheidsinfrastruktuur ten volle benut word.

Hierdie tesis dien as 'n bewys dat bestaande infrastruktuur gebruik kan word vir die implementering van 'n voetganger-waarneemingsisteem. Die stelsel sal voetgangers monitor wat die voetganger brue langs die snelweÃ« gebruik, en sal bewegingsinformasie soos, rigting en relatiewe spoed van beweging, genereer. Die stelsel beoog dus om by te dra tot die veiligheid in Suid Afrika.

'n Kaskade klasifiseerder is opgelei met bestaande veiligheidsbeeldmateriaal wat van SANRAL verkry is, deur gebruik te maak van AdaBoost en geskatte beeld-skale. Daar word aangetoon dat 'n klasifiseerder spesifiek opgelei moet word vir die kamera wat gebruik gaan word. Die *Integral Channel Features* klasifiseerdingsmetode is gebruik, wat HOG en kleur kanale inkorporeer, omdat dit maklik implimenteerbaar is, en goeie resultate behaal. 'n Grafiese gebruikerskoppelvlak is ontwikkel om die opleidings- en klassifiseringsfases uit te voer sodat die gebruiker nie bloot te stel is aan die ingewikkelde program-agterkant nie.

Die waargenome voetgangers word gevolg deur parametriese volgmetodes soos 'n Kalman filter en aanvullende spoor-besturendsfunksies. Die gevolgde voetgangers is

automaties getel en vergelyk met voetgangers wat met die hand getel is. Die resultate dui daarop dat die automatiese voetganger telling 'n akkuraatheid van meer as 90% behaal. 'n Gebruikers vriendelike grafiese koppelvlak is onwikkel om voorsiening te maak vir betekenisvolle voetganger bewegings resultate.

Verdere ontleding van die voetganger bewegings resultate wat ingesamel is, toon dat die spitstyd voetgangerverkeer in die oggend tussen 06:30-07:30 is, en in die middag tussen 17:00-18:00. Die data toon ook dat die meeste voetgangersverkeer soggens in die rigting van die middestad is, en in die middag weg van die middestad is. Die resultate toon ook dat voetgangers gemiddeld 10% vinniger beweeg in die rigting van die middestad.

Die resultate dien dus as bewys dat 'n voetganger moniteringstelsel wat gebruik maak van bestaande lae-resolusie veiligheids infrastruktuur op voetganger brue, gebruik kan word om kompeterende resultate te behaal. Dit demonstreer dat die stelsel die doelwitte behaal wat gestel was, en dat die implimentering van 'n voetganger waarnemingsstelsel iets is wat ingedagte gehou moet word tydens toekomstige infrastruktuur installasies en opgraderings.

Acknowledgements

I would like to acknowledge the support of my thesis supervisors, Dr. MJ Booysen, and Dr. SJ Andersen for their patience and assistance over the two years. I would like to acknowledge my parents for their moral support and frozen food packages, without which I would have been sad and hungry. I would like to acknowledge my flatmates for sticking with me through messy times. Lastly I would like to thank God for his guidance and allowing me the privilege to do my thesis.

Contents

List of Figures

List of Tables

Nomenclature

Variables

Tp	True Positive
Tn	True Negative
Fp	False Positive
Fn	False Negative
PHF	Peak Hour Factor

Acronyms

ITS	Intelligent Transport Systems
ROI	Region Of Interest
TMC	Transport Management Center
NMS	Non-Maximum Suppression
SANRAL	South African National Roads Agency (Limited)
PTZ	Pan-Tilt-Zoom
ICF	Integrated Channel Features
SVM	State Vector Machine
ROC	Receiver Operating Characteristics
PR	Precision-Recall
CBD	Central Business District
L2R	Left To Right
R2L	Right To Left
FPPI	False Positives Per Image
HOG	Histogram of Orientated Gradients

Chapter 1

Introduction: Computer Vision For Pedestrian Detection

1.1 Pedestrian Activity On South African Roads

A report released by [1] shows that a total of 356 pedestrians were involved in crashes along highways from 2010 to 2014. The most accidents occur on a Friday with the peak times around 7:00am and 7:00pm. The survey also shows that the number of people using pedestrian bridges exceeds by far the number of pedestrians that cross the freeway at grade in the same area. Approximately 8700 pedestrians cross freeways and approximately 14000 pedestrians cross via pedestrian bridges on a typical weekday.

Pedestrian safety is a key challenge in South Africa as a significant number of pedestrians use the highways illegally each day. This is partly due to rural pedestrians commuting to and from public transport by walking along the highway. While walking on the freeways is illegal, some pedestrians regard it as their only option. This could be a result of not having enough time to walk to a pedestrian bridge and it could also be due to safety concerns, as pedestrian bridges in high-risk areas are known to be dangerous, particularly at night.

Current pedestrian-survey techniques require operators to count pedestrians manually, which is time consuming. This is done to help understand the nature of pedestrian activity in an area and to help advise with the planning of new pedestrian-bridge locations. These surveys are only carried out once every 6 months and thus do not allow for a rapid response to changes in pedestrian activities.

1.1.1 Current Pedestrian Monitoring

There is a need in South Africa to better understand the nature and number of pedestrians who use freeways or pedestrian bridges daily. In Cape Town along the freeway alone, there are about 237 surveillance cameras streaming directly to the Transport Management Center (TMC). Over 16 pedestrian bridges are also monitored by static

surveillance cameras. These cameras form part of the ITS infrastructure for the Cape Town Freeway Management Systems Project, which is funded by SANRAL, the Western Cape Provincial Government, and the City of Cape Town. This means that there is an opportunity to collect data on pedestrian mobility.

Pedestrian statistics are at present collected from the SANRAL surveillance footage archive, and require employees to count every pedestrian manually. This is extremely time consuming and monotonous work, and resources could be allocated more effectively.

Modern camera systems could automate the counting process, but this would require specialized hardware and software such as high-end thermal cameras, specialized image processors, and the supporting infrastructure.

Figure 1.1 shows the pedestrian crossings per pedestrian bridge on a average weekday as counted manually in the pedestrian survey in [1].

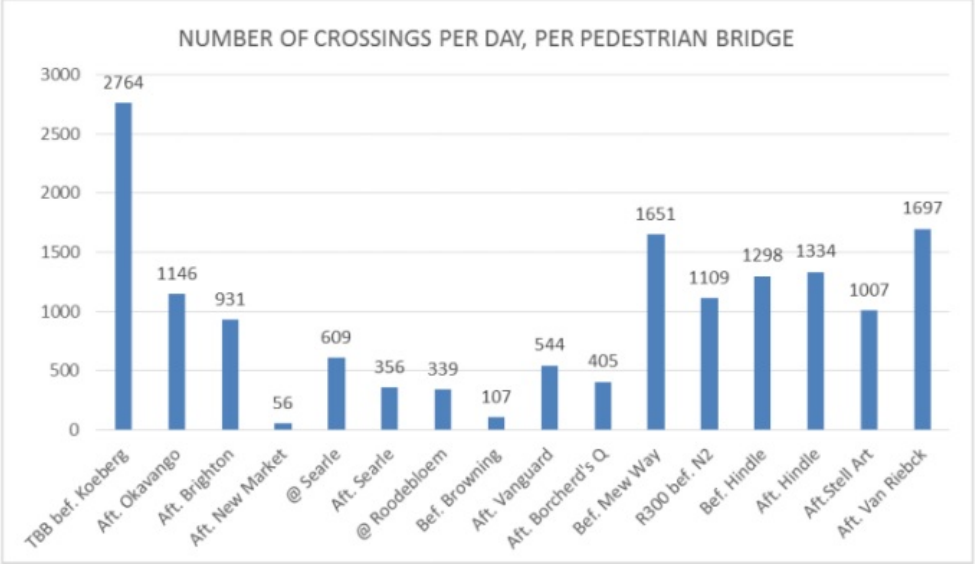


Figure 25: Number of crossing per day, per pedestrian bridge

Figure 1.1: Number of pedestrian crossings on an average weekday [1]

From figure 1.1 it can be seen that the bridge at Koeberg has the highest number of pedestrian crossings with 2764 crossings, and the bridge at Van Riebeeck (on the R300) has the second-highest number of pedestrian crossings with 1697 crossings. There are at minimum 14152 pedestrians crossing pedestrian bridges on the freeway in the greater Cape Town area.

This means that a proof of concept of a system for collecting pedestrian movement data can be done using existing surveillance cameras monitoring pedestrian bridges,

as a vast number of pedestrians cross the freeway via pedestrian bridges, representing valuable information.

1.2 Environmental and Physical Challenges

The implementation of computer-vision techniques to monitor pedestrians for safety purposes faces many challenges in a developing country such as South Africa.

It is currently not feasible to accurately monitor and count pedestrians walking on the highways or pedestrian bridges. Some of the reasons for this include:

- the cost of implementing state-of-the-art pedestrian monitoring systems
- the cost of maintaining the monitoring systems as well as training operators to use the system
- the complexity and cost of integrating the monitoring system with existing surveillance systems
- the cost of tailoring the pedestrian-monitoring systems to South African needs

As a low-cost alternative, existing surveillance infrastructure can be considered for the purpose of detecting and counting pedestrians. The main challenges are those imposed by the technical and operational limitations of the existing surveillance infrastructure.

1.2.1 Technological and Operational Challenges.

Using existing surveillance infrastructure for pedestrian monitoring is a developing country's solution to computer-vision implementations. The existing camera infrastructure poses technical challenges. Key challenges include:

- Surveillance cameras monitoring the highway can barely see pedestrians in their default zoomed-out position.
- It is difficult to obtain long uninterrupted footage from the PTZ (Pan-Tilt-Zoom) cameras monitoring the highway, as operators constantly move the camera to inspect incidents.
- A significant portion of static cameras viewing pedestrian bridges have a sub-optimal view of the pedestrians. This is due to visual obstructions from the environment or because the camera is oriented in such a way as to make it hard to distinguish between pedestrians at peak times.
- The low-resolution nature of archive footage makes the implementation of accurate pedestrian-monitoring systems challenging.

- The cost of upgrading infrastructure with state-of-the-art pedestrian-monitoring technology is a prohibitive.

The use of existing low-resolution cameras allows for a reduction in the cost of infrastructure upgrades as well as incorporating the existing maintenance cost of the surveillance cameras.

1.3 Proposed Automatic Pedestrian Counting System

This thesis introduces a method with which to automate the gathering of pedestrian-movement information on pedestrian bridges using state-of-the-art image processing and computer learning techniques.

These techniques are then applied to SANRAL's existing surveillance infrastructure, specifically on cameras monitoring pedestrian bridges along the freeways. The proposed solution will use historical archived surveillance footage, and thus real-time implementations will not be applicable.

The system can be expanded by adding behaviour recognition to be used in more complex safety and security applications such as cable monitoring, as it would be able to detect thieves or suspicious behaviour.

The purpose of this project is to aid in the pedestrian-safety effort in South Africa, and to research the feasibility of using image-processing techniques on existing surveillance cameras to count pedestrians and provide supplementary movement information. The project also aims to automatize the pedestrian-survey process of manually counting pedestrians over two weeks. The project also aims to create a seamless data flow from source video collection to the count histogram output results for reviewing, and to realize this with the aid of graphical user interfaces.

The results will serve as a proof of concept that existing pedestrian bridge-surveillance infrastructure can be used to monitor pedestrian activity under ideal conditions with the end goal of expanding the system to freeway monitoring.

1.4 Thesis Statements And Hypotheses

Thesis statement 1:
Pedestrians on the pedestrian bridges can be detected with high accuracy using existing surveillance cameras.

A robust rigid-body pedestrian detection system is designed and implemented to detect multiple distinct pedestrians crossing the pedestrian bridges using existing surveillance infrastructure. This work incorporates state-of-the-art pedestrian-detection techniques.

Hypothesis 1.1:
The existing surveillance camera has a resolution adequate to perform pedestrian detection.

Hypothesis 1.2:
There is an existing surveillance camera with the correct orientation to allow for pedestrian detection that can be used as a best-case scenario.

Hypothesis 1.3:
The detector is able to distinguish between partially occluded pedestrians moving in a group.

Hypothesis 1.4:
The detector is able to detect multiple distinct pedestrians throughout a day with a hit rate of more than 80%.

Hypothesis 1.5:
The detector needs to be trained specifically for each scene in order to obtain a hit rate of more than 70%.

Hypothesis 1.6:
False detections can be suppressed to a visually acceptable rate.

Hypothesis 1.7:
The training and detection parameter values do not deviate significantly from those normally employed for pedestrian detection.

Thesis statement 2:
Pedestrians on the pedestrian bridges can be tracked with a high degree of certainty from their detections.

A pedestrian-tracking system is developed utilizing a parametric tracking model and the pedestrian detections from the detection system. The system assigns tracks to detections to link detections from previous frames together and provide information about pedestrian movement.

Hypothesis 2.1:
Pedestrians can be identified and tracked during peak pedestrian activity and throughout the day with a hit rate of more than 80%.

Hypothesis 2.2:
Kalman filter parameters can be optimally tuned to achieve a hit rate of more than 80%.

Hypothesis 2.3:
Further rejection of false/noisy detections from the detection stage can be achieved with the tracking algorithm.

Hypothesis 2.4:
Visual verification, and thus quantitative evaluation, are adequate to evaluate the tracking results of subsets of footage to tune the required tracking parameters.

Thesis statement 3:
The tracking results can be used to count pedestrians crossing pedestrian bridges, and to provide useful pedestrian-mobility information.

A counting algorithm is developed that uses the tracking results to count pedestrians and give supplementary movement information, such as the direction of movement, the relative pedestrian speeds, and the time of crossing.

Hypothesis 3.1:
Information about the direction of movement can be reliably obtained from the tracking results.

Hypothesis 3.2:
Information about the relative pedestrian speeds can be obtained from the tracking results.

Hypothesis 3.3:
The time of crossing and total number of pedestrian crossings can be obtained from the tracking stage.

Hypothesis 3.4:
The counting results can be represented in a meaningful way using a graphical user interface.

1.5 Research Objectives

Research Objective 1:
To develop a practical system that improves the safety of pedestrians by giving the authorities added information about pedestrian movement

Research Objective 2:
To develop a proof of concept solution that uses existing infrastructure on pedestrian bridges under ideal conditions to gather pedestrian information

Research Objective 3:
To develop a system that automates the pedestrian survey process employed by SANRAL, where pedestrians using pedestrian bridges are currently counted manually

Research Objective 4:
To develop a system with a graphical user interface that is easy to use and provides black-box capabilities.

Research Objective 5:
To develop a system that can show whether patterns in pedestrian mobility occur

Research Objective 6:
To develop a system that allows the user to use custom counting lines and ROIs

Research Objective 7:
To develop a system that allows the user to export the counting results to different formats to enable the analysis of the counting results by external programs other than MATLAB

1.6 Contribution

The main contributions of this thesis is the automatic processing of low resolution surveillance footage to report on the number of pedestrian crossings, direction and relative speed of movement and the time of crossing. The data is presented in a meaningful manner at the hand of a user friendly graphical interface.

1.7 Scope Of Work

This thesis focuses on the implementation of a pedestrian-detection and -tracking system on existing surveillance infrastructure.

Only cameras facing the pedestrian bridges without infrastructure occluding pedestrians are considered. Ultimately only one camera was chosen as the best-case scenario and will act as a proof of concept.

The detection system is based the works of [10] and [2] where the optimal parameters are calculated and empirically evaluated. The motivation of the generic parameter values is beyond the scope of this thesis and is used as is unless otherwise stated. The detection method implemented is Integrated Channel Features (ICF) and the detector is trained with an AdaBoosted cascade classifier. For training, only supervised learning methods are considered and thus all forms of unsupervised learning methods such as neural networks are not included in the scope of this thesis. Only rigid-body detection methods are considered as these are the most popular methods for pedestrian detection and are implemented more easily than other methods.

The pedestrian-tracking system is based on the multi-object tracking example from MATLAB, and as such the MATLAB implementation of the Kalman filter and track-management functions are used mostly as is. Thus the derivation of the Kalman filter equations, as well as the Murkez' adaptation of the Hungarian optimization algorithm are not included in the scope of the thesis. The Kalman filter and other tuning parameters were determined empirically and not modelled mathematically . This is due to the practical nature of the system where a mathematical model does not accurately describe the setup, along with the fact that it is challenging to gather a generalized motion model of pedestrian movement from the source footage. The counting algorithm uses the tracking results and user-defined counting line to determine the number of pedestrian crossings.

The pedestrian-detection and -tracking system is integrated, which results in a system that is able to count pedestrian crossings and also give information about the time of crossing, direction, and relative speed of movement. There is little focus on interpreting the counting results, as the system is only developed as a tool to provide added information. The interpretation of the results is only given as an example as to how the data could be used meaningfully.

The system is developed to focus on accuracy, and no significant amount of time is put into making the system computationally efficient as historical source footage will be used. The system is not directly compared to high-end pedestrian-detection solutions, as it was not practically possible to obtain sufficient information on high-end pedestrian detection solutions to be able to make a meaningful comparison.

1.8 Thesis Structure

Chapter 2: presents a review of image processing in pedestrian detection and tracking. The chapter also gives a comprehensive review of the related work, including training, detection and tracking methods. The chapter also discusses the methods used to evaluate detection and tracking results as used in the rest of the thesis.

Chapter 3: discusses the setup and methodology of the detection and tracking stage. It is proposed that a detector be specifically trained with data from the respective camera. Thus the detector is scene specific and not generic. The chapter describes how footage is collected from SANRAL, pre-processed, annotated, and divided into training and verification data sets during the data-collection phase. Chapter 3 discusses the methodology followed in the training phase where the required parameters are set, and the annotated training data is used to train a cascade classifier/detector with AdaBoost. Further discussions on the setup of the detection stage focus on the parameter setting. The chapter describes the methodology followed when defining a detection region as well as the output structure of the detection phase. Chapter 3 also presents the methodology followed in evaluating the detection results. For the tracking stage, chapter 3 discusses the data flow between the stages as well as discussing the Kalman filter setup and other tuning parameters. The chapter also describes the verification methods used to evaluate the tracking results. Lastly the implemented software and physical hardware setup are presented.

Chapter 4: presents the implementation of the detection stage proposed in chapter 3. Motivations for the parameter values chosen for both the training and detection phase are also discussed. A summarized example of the output log file after the training has been completed is also given and discussed. The choice of the non-maximum suppression method (NMS) is motivated, and empirical results are given. Chapter 4 presents the graphical user interface implemented to allow for training and detection with black-box behaviour. The functionality of the graphical user interface is also discussed. Lastly this chapter discusses the challenges faced during the detection stage.

Chapter 5: discusses the implementation of the tracking stage as proposed in chapter 3. The chapter motivates the tuning- and Kalman filter-parameter values implemented as well as how variables are parsed between the stages. Chapter 5 describes how detections are assigned to tracks, and also discusses other implemented track-management functions. The proposed counting algorithm implemented is then discussed with the help of a flow diagram. An example of the track- and count-variable structures is given and briefly discussed. The chapter further discusses how the tracking algorithm displays the tracking results while the tracking stage is in progress. The tracking-verification graphical user interface used to evaluate the counting results is also presented and its functionality is discussed. Lastly, chapter 5 discusses challenges that

arose during the implementation of the tracking stage.

Chapter 6: presents the results of the implemented system. The detection results of the detection stage are presented and discussed. The system is verified by comparing the results to manually counted ground-truth data for selected days. A enhanced comparison view one of the days is also presented. Chapter 6 discusses the count differences in total pedestrians and presents the data percentage-wise. The chapter further presents a comparison of the total number of pedestrians counted automatically and manually. Chapter 6 presents the results of the system that was run on selected days of the week. The counting results are presented in the form of histograms and shows movement in both directions. Lastly, the chapter presents an alternative cumulative count histogram comparison to represent the counting results, and the relative pedestrian speeds in both directions are compared.

Chapter 7: concludes the work of this thesis by validating the hypotheses from Chapter 1 using the results from the preceding chapters. Significant findings and contributions of the work in this thesis are also provided and discussed. Lastly recommendations for future work and concluding remarks are presented.

Chapter 2

Literature Survey

2.1 Introduction

The field of computer vision and non-rigid (pedestrian) detection is rapidly expanding, with new techniques and higher benchmarks being set in quick succession. Various training methods and pedestrian-detection techniques have been proposed over the years with varying degrees of success.

This chapter discusses the building blocks of image processing in pedestrian detection in section 2.2. In particular, different non-maximum suppression methods and different training methods are discussed. Section 2.3 discusses how to interpret the graphical representations of the detection results, and the verification methods commonly used. Section 2.4 presents the related classical works in the field of pedestrian detection. Modern detection methods and state-of-the-art detectors are discussed in section 2.5. Section 2.6 discusses the different tracking methods used in pedestrian tracking. Lastly, section 2.7 presents a summary of the literature survey.

2.2 Image Processing In Pedestrian Detection

This section contains the relevant theory and building blocks of image processing used in pedestrian detection.

2.2.1 Bounding Box

A bounding box is the smallest bounding or enclosing box that can be drawn around an object. For pedestrian detection and image processing a bounding box is either drawn around an object of interest in the labelling stage, or given as an output by the detector/classifier to indicate the position of a pedestrian.

An example of bounding-box outputs from a detector run on the Caltech pedestrian-evaluation data set is shown in figure 2.1 from [2].

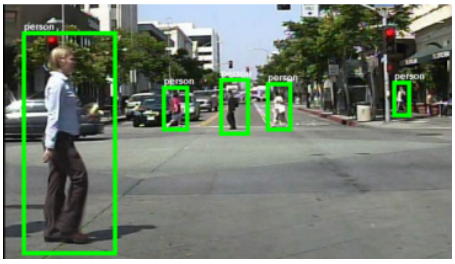


Figure 2.1: Example of bounding-box output from [2]

2.2.2 Detection Window/ Stride Length

The detection/sliding window is a square window that densely scans the input image. The sliding window moves according to the window stride length.

The window stride length is how many pixels the sliding window densely scanning the image moves to form each new detection window. This has a significant impact on performance, as it determines how much consecutive detection windows will overlap. In [9] it is shown that using overlapping windows reduces the detector miss rate by up to 5%. The original and subsequent positions of the sliding window are shown in figure 2.2 from [3].

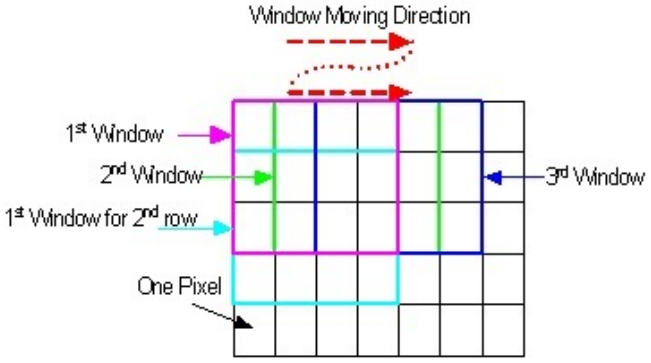


Figure 2.2: Sliding window adapted from [3]

The detection is originally at the "*1st Window*", and moves to subsequent positions ("*2nd Window, 3rd Window*") according to the window stride length. In this example the window stride length is [1 1], thus moving to the right by one pixel in each detection window until it reaches the image boundary, where the window then resets to its original position and moves one pixel down. The amount by which detection windows overlap can thus be controlled.

The sliding window stride length can have a large impact on the runtime of the algorithm, as it determines the density or sparsity at which the image will be scanned.

2.2.3 Image Pyramid

An image pyramid is a multiscale representation of an image. The pyramid is created by applying a smoothing filter (usually Gaussian) [5] and subsampling the image, usually by a factor of 2 in each direction. The subsampled image is then subjected to the same procedure and it is repeated a set number of times. For pedestrian detection, this is done for an octave, creating an image pyramid of eight scales [9] [11] [14].

An example of an image pyramid is shown in figure 2.3 from [4].

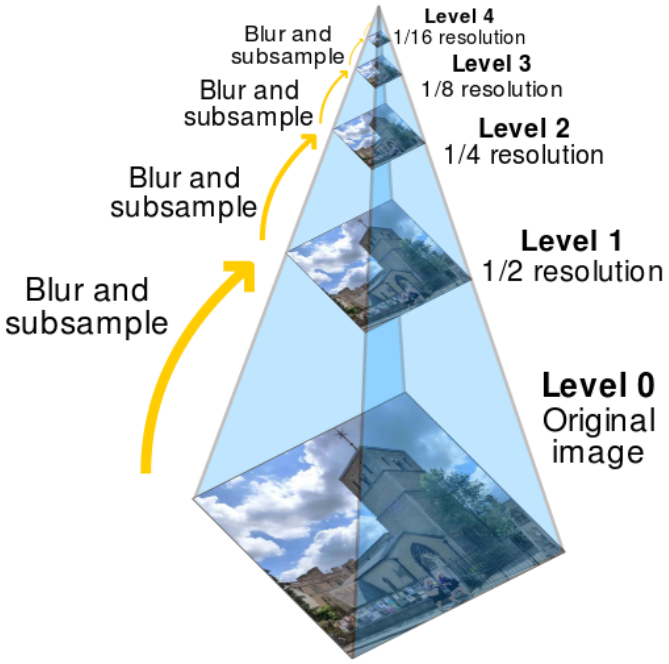


Figure 2.3: Example of an image pyramid from [4]

The use of image pyramids in image processing allows the creation of multiscale detectors, where an image pyramid is created of either the input image or the classifier, and thus contains feature information at multiple scales. A multiscale detector/classifier is one that can identify objects (pedestrians) at scales/sizes different to those used in training the detector/classifier.

2.2.4 Image Features

Feature extraction typically captures intensity patterns, texture details, and/or shape and contour information acording to [9].

There are many reasons for using image features rather than using pixels directly. The most common reason is that features can act to encode nearby domain information that is difficult to learn using a finite quantity such as pixels for the training data.

Features thus give added information about the object's surroundings. Another critical reason for using features is that a feature-based system operates much faster than pixel-based systems [5].

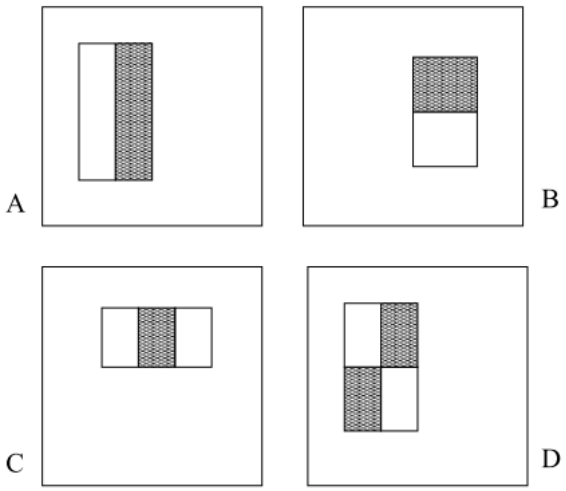


Figure 2.4: An example of rectangle features is shown relative to the enclosing detection window from [5]

.

In figure 2.4, the sum of the pixels that lie inside the white rectangles is subtracted from the sum of the pixels within the grey rectangles. Figure 2.4(A) and (B) shows two-rectangle features, figure 2.4(C) shows a three-rectangle feature, and figure 2.4(D) shows a four-rectangle feature. These rectangular features shown are similiar to Haar-like features used in early detectors as in [5].

Features can also be computed using other custom functions and transformations to add extra information/channels as in [9] [11] [14] [2].

2.2.5 Integral Image

An integral image is used as a way of representing an image to reduce the computational complexity, which allows rectangle features to be computed rapidly. By using the integral image, any rectangular sum can be computed in four array references, shown in figure 2.5 obtained from [5].

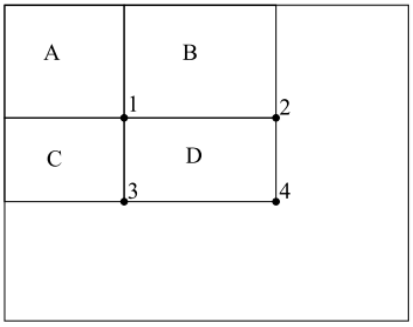


Figure 2.5: Example of an integral image from [5]

The sum of the pixels inside the rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels inside rectangle A. The value at location 2 is $A+B$, at location 3 it is $A+C$, and at location 4 it is $A+B+C+D$. The sum of the pixels inside D can be computed as $4+1-(2+3)$.

2.2.6 Non-Maximum Suppression (NMS)

Non-maximum suppression (NMS) is the act of reducing multiple bounding boxes, drawn at different scale, to one bounding box with the highest score/probability through optimization. This means that NMS suppresses values, or in this case bounding boxes/detection scores.

Multiple bounding boxes are originally created as the classifier detects the same object at multiple scales of the image pyramid. NMS is critical as it reduces mutliple detected bounding boxes to a single bounding box corresponding to a single detection. An example of NMS is shown in figure 2.6 from [6].

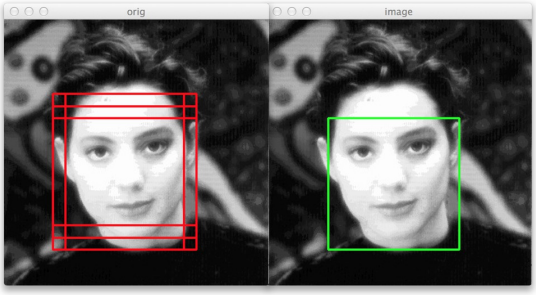


Figure 2.6: Example of NMS adapted from [6]

Four of these NMS methods will be discussed, although numerous other methods exist.

2.2.6.1 Mean Shift (MS)

Mean shift (MS) non-maximum suppression is a procedure for locating the maxima of a density function given discrete data sampled from that function [15]. A kernel with a radius is set, which controls the amount of suppression along each axis [16]. The radial width of the kernel should be set according to the spatial and scale stride of the bounding box. [13]. A example of mean shift is shown in figure 2.7 from [7].

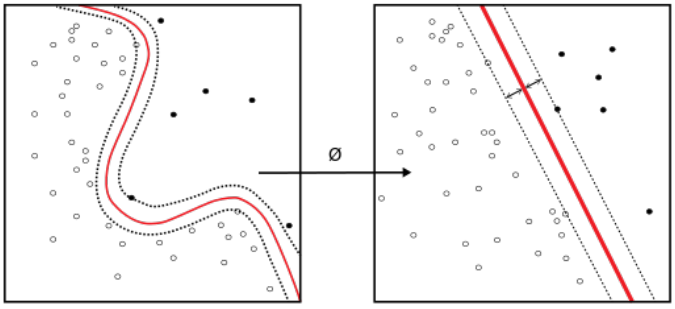


Figure 2.7: Example of Mean Shift NMS adapted from [7]

For the application of mean shift in image processing, the left side of figure 2.7 can be seen as the true maxima, and the right as the bounding boxes that need to be suppressed. The mean is shifted in such a way that an optimal rejection of overlapping bounding boxes is obtained.

2.2.6.2 Maximum (Max)

Maximum (max) non-maximum suppression uses the area of overlapping bounding boxes. For each pair of bounding boxes the overlap is defined by:

$$overlap(bb1, bb2) = area(intersection(bb1, bb2))/area(union(bb1, bb2) \quad (2.1)$$

Figure 2.8 shows the intersection and union of two overlapping bounding boxes. The overlap is the intersection divided by the total area, thus it is a measure of the percentage to which the two bounding boxes overlap.

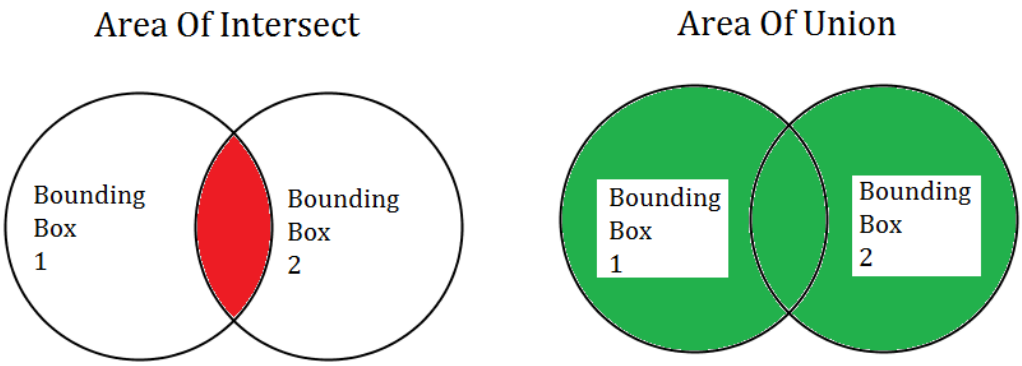


Figure 2.8: Overlap is computed using the area of the intersection divided by the area of the union of bounding box 1 and bounding box 2.

The overlap threshold can be set. If the overlap of the bounding box pair is greater than the threshold set, the bounding box with the lower detection score is suppressed.

2.2.6.3 Optimized Maximum (Maxg)

Maximum optimized (maxg) non-maximum suppression uses the same algorithm as "max", but performs optimization in a greedy fashion. Bounding boxes are processed in order of decreasing score, and, unlike "max", once a bounding box is suppressed it can no longer suppress other bounding boxes [13].

2.2.6.4 Cover (cover)

Cover (cover) non-maximum suppression attempts to choose the smallest subset of bounding boxes so that each remaining bounding box is within overlap of one of the chosen bounding boxes.

The above reduces to a weighted set cover optimization problem, which will not be discussed in detail. The score of each bounding box is set to the sum of the scores of the bounding boxes that it covers. This is in practice similar to the maximum optimized NMS (maxg) method mentioned above[13].

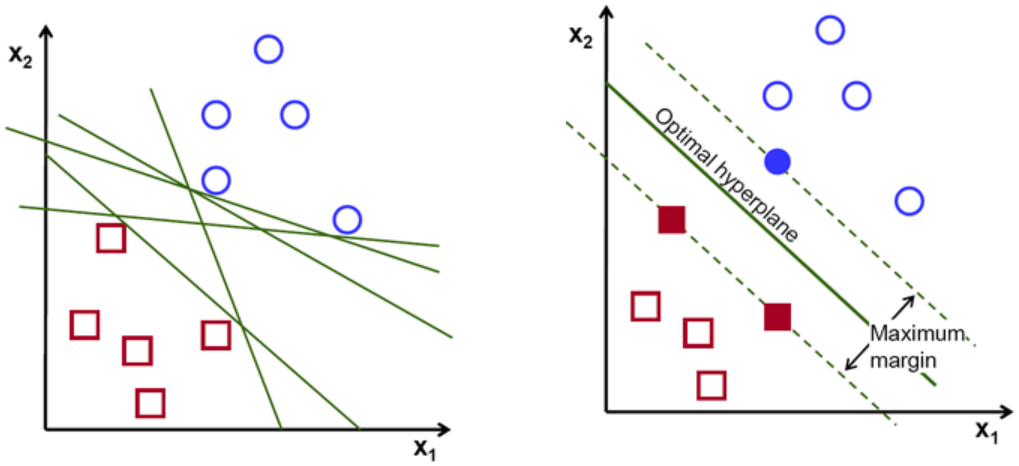
2.2.7 Training Methods

2.2.7.1 Support Vector Machine (SVM)

Support Vector Machines (SVMs) have been used widely for object recognition in the past decade. A SVM is a discriminative classifier that uses a seperating hyperplane to classify objects. In image processing and in simplified terms, given labelled training data (known as supervised learning), the SVM outputs an optimal hyperplane that can

categorize new input examples [17]. This algorithm is used by [9] in the original HOG algorithm.

Figure 2.9a shows an example of multiple hyperplanes that try to optimally separate the types of data; red squares and blue circles in this example. For pedestrian detection the data types would be pedestrian/positive or non-pedestrian/background/negative. This is a simplification of the problem and is represented in the Cartesian plane instead of a higher dimensional one.



(a) Example of hyperplanes trying to separate the two data types from [17].

(b) Example of optimal hyperplane separating the two data types with a maximum margin from [17]

Figure 2.9: Examples of hyperplanes

A hyperplane is said to be bad if it passes too close to the data points, as it will be sensitive to noise and will thus not generalize correctly [17]. The SVM algorithm is based on finding hyperplanes that give the largest minimum distance to the training data. Twice this distance is considered as the *margin*.

It can thus be said that an optimal hyperplane maximizes the margin of the training data.

Figure 2.9b shows an optimal hyperplane separating the data with a maximized margin [17].

2.2.7.2 Cascade Adaptive Boosted Learning (AdaBoost)

AdaBoost in its original form is used to boost the classification performance of a simple (sometimes called weak) learning algorithm. The output of the weak learning algorithms is combined into a weighted sum that represents the final boosted classifier [18]. Adaboost can be seen as an incremental learning from mistakes.

AdaBoost combines a collection of weak classifiers to form a stronger one. In image processing, it is used particularly to build cascades of pattern rejecters, with each level of the cascade choosing the features most relevant for its rejection task. Although AdaBoost cascades are slow to train owing to their selective feature encoding, they offer a significant improvement (compared to SVMs) in the run-time of the final detectors.

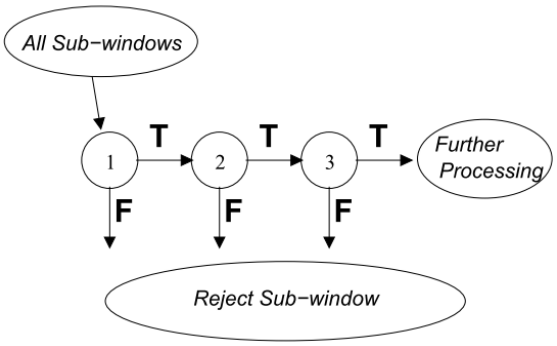


Figure 2.10: An example of a classifier cascade. The first classifier eliminates a large number of negative samples with little processing. Subsequent cascade stages eliminate additional negative samples but require additional computation. After several stages the number of sub-windows have been greatly reduced. Further processing can include adding additional stages or an alternative detection system. [5]

Figure 2.10 shows an example of the rejection at each cascade stage. The weak learning algorithm acts as a decision tree to classify the detections as objects or non-objects. At each cascade it detects all the objects it can with its set of parameters, and allows the rest through to the next stage. Each stage performs more dense detection scans, at the cost of computational time.

This can be seen as muddy water (complete data set) falling through a multi-stage filter, cascading with increasing density. The first coarse stage collects all the large and easy-to-filter material (positive samples) quickly, while the last fine stage takes a longer time to remove the dirt from the water, ultimately leaving fresh water, (negative samples). It then becomes an optimization problem to assign weights to positive and negative samples and obtain a final optimized classifier/detector.

An example of how the weighted weak classifiers are added and used to create a cascaded classifier is shown in figure 2.11 as adapted from [5].

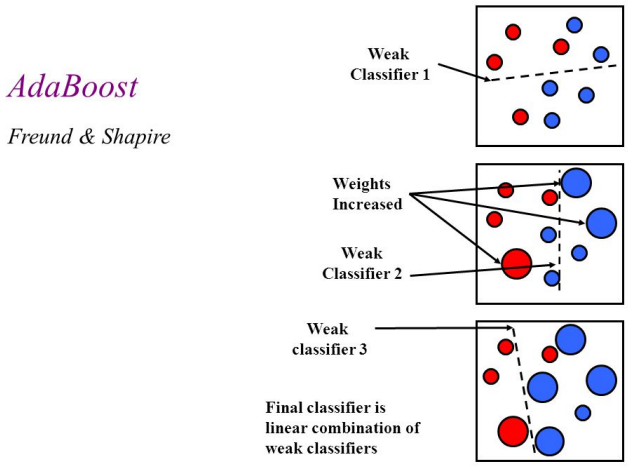


Figure 2.11: An example of the AdaBoost algorithm from [5] as originally proposed by [8]

At each stage a weak classifier is learned, represented by a dashed line, to separate the two data types. After each stage the weights of the data are adjusted to be easily classified in the next stage. Ultimately the weak classifiers are linearly combined to form a final boosted cascade classifier.

2.3 Interpreting Detection Results

A detector or classifier's performance is commonly referred to with the term hit rate. This states the percentile of objects that will be detected (hit) out of the total number of positive objects present. Some graphs use miss rate, which is one minus hit rate. Others use normalized true positives to detonate the y-axis, which is the same as hit rate.

When the detector accurately detects an object it is called a Tp, or true positive. A true positive is when the detection is the same as the annotation or data label assigned. An Fp, or false positive, would be a false detection, i.e. detecting an object that is not there. The detections are compared to the annotated or ground truth data. The same goes for Tn, true negatives, and Fn, false negatives. The detection results are compared to the annotated/labelled frames in the verification data set.

2.3.1 Receiver Operating Characteristics (ROC-Graphs)

A ROC graph plots the miss rate against the number of false positives per image, FPPI. FPPI can be set for a specific detector by adjusting the threshold in the NMS-parameters as well as the cascade weights during training.

This is useful in applications where few or no false detections occur by setting the FPPI low – alternatively setting the FPPI high for applications where false detections are acceptable. A lower FPPI results in an increase in the number of false negatives, or missed positives, but also decreases the number of false positives. A good ROC curve tends to the left-bottom corner of the graph, which indicates a low miss rate for a low number of FPPI. Equation 2.2 shows the formula to calculate the miss rate.

$$MissRate: \frac{Fn}{Fn+Tp} \tag{2.2}$$

2.3.2 Precision / Recall Graphs (PR-Graphs)

A precision-recall graph gives a different view of the data. The precision is compared against the recall rate. Nothing can be explicitly set and evaluated, but a relationship between the total detections and a ratio of true/false positives can be seen.

A good PR curve tends to the upper-right corner of the graph, which indicates a high precision at a high recall rate. Equation 2.3 and equation 2.4 show the formula to calculate the precision and recall respectively.

$$Precision: \frac{Tp}{Tp+Fp} \tag{2.3}$$

$$Recall: \frac{Tp}{Tp+Fn} \tag{2.4}$$

2.3.3 Verification By Inspection

Verification is also done by inspection, as the verification set is not always sufficiently complete over all scenarios to give a definitive result with the help of PR or ROC graphs. The PR and ROC graphs occasionally give misleading results, although they can be used to evaluate the impact of parameters.

The final assessment on whether the classifier or detector is "good" and delivers satisfactory performance is left to a physical inspection of the resulting output frames in the detecting stage. The problems that then occur, e.g. misclassification, can be handled on an individual basis by either expanding the training data set or adjusting parameters.

2.4 Classic Pedestrian Detection Methods

2.4.1 Basis Of Pedestrian Detection: Viola&Jones

The methods and techniques proposed by Viola & Jones [5] are commonly accepted as the basis of modern face and pedestrian detection. The detection algorithm proposed by [5] is distinguished by three key contributions.

The first is the introduction of a new image representation called the "*Integral Image*". The "*Integral Image*" allows the features used by the detector/classifier to be computed faster than without it. Rectangular features can be computed very rapidly using this intermediate representation of the image.

The second contribution is a learning algorithm based on AdaBoost, which is a form of boosted learning. The key difference and adaptation made is that Viola & Jones' algorithm only selects a small number of critical visual features from a larger set, and this yielded extremely efficient classifiers at the time it was published, according to[19]. The algorithm uses simple Haar-like features similar to the basis functions used in the first detectors.

The third application of the learning algorithm of [5] demanded a very aggressive approach that would discard the vast majority of features.

Lastly, the method of combining increasingly more complex classifiers in a *"cascade"* from *"weak"* classifiers is proposed. This adaptation allows the background regions of an image to be quickly discarded while spending more computation time on promising object-like regions [5]. Cascade classifiers are discussed in more detail in subsection 2.2.7.2.

2.4.2 Histogram Of Orientated Gradients (HOG)

In a paper by [9], the authors suggested HOG (Histogram of Oriented Gradients) for the application of pedestrian detection. This together with the work of [5] form the basis of most rigid-body pedestrian-detection algorithms, and is widely regarded as the foundation of practical multi-scale pedestrian detection.

For static images, the author proposed locally normalised Histograms of Oriented Gradients (HOG) as descriptors/clasifiers. A descriptor is the way the data set is encoded, or how the data set is represented. The HOG descriptors are computed from image gradients (edge information) and are sufficiently robust to deal with (a) small changes in image contour locations and directions, (b) significant change in image illumination and colour, while (c) remaining as discriminative and separable as possible [9].

An example of how a HOG descriptor and its comprising components are computed is shown in figure 2.12.

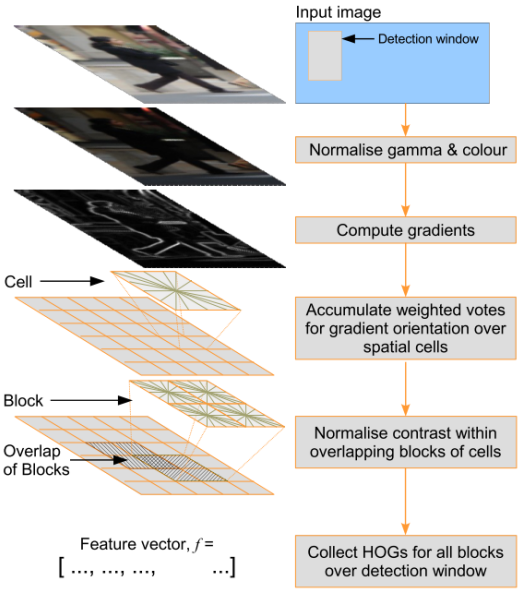


Figure 2.12: Example of HOG Descriptor from [9]

A detection window densely scans the input image at multiple scales using an image pyramid. The algorithm creates a dense image pyramid by down-sampling the image to create a multi-scale representation or descriptor of the image.

The HOG algorithm extracts features on a per-detection window basis. The gradient information contained within the detection window is extracted and normalized, and the image gradients are computed. The weighted votes of the image gradients (the magnitude of the edges, and in which direction) are then accumulated over the spatial cells and normalized over blocks of cells.

The HOGs of all the blocks of cells are then collected over the detection window to create the feature vector. The choice of cell or block format and size, as well as the impacts of different gamma and colour normalisations, are discussed in greater detail in [9].

To "train" the descriptor/classifier, HOG uses a *linear* SVM (Support Vector Machine) as it offers three properties that are important to the authors. The *linear* SVM converges reliably and repeatably during training, it handles large data sets gracefully, and it has good robustness towards different choices of feature sets and parameters [9].

Examples of the different representations of trained HOG classifier/detector as well as the trained SVM weights from [9] are shown in figure 2.13.

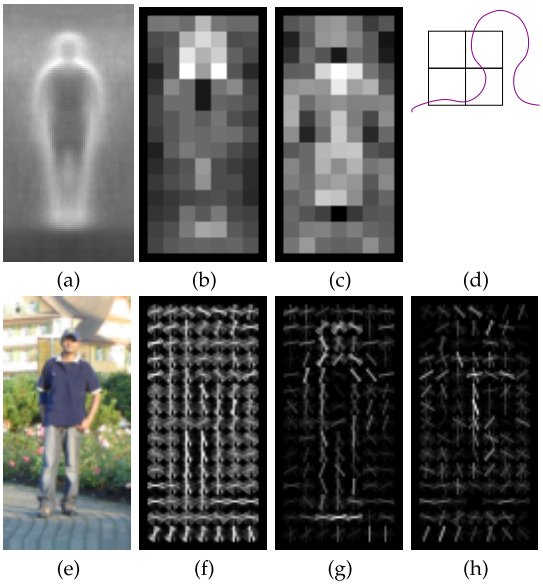


Figure 2.13: For pedestrians, the HOG classifiers cue mainly on silhouette contours, especially the head, shoulders and feet. (a) The average gradient image over the training examples. (b) Each *"pixel"* shows the maximum positive SVM weight in the block centered on the pixel. (c) Likewise for the negative SVM weights. (d) A sketch portraying the most relevant blocks, those lying just outside the contour. (e) A test input image containing a pedestrian. (f) The computed HOG descriptor. (g,h) The descriptor weighted respectively by the positive and negative SVM weights. Note that only the dominant orientations are shown. The figure is adapted from [9].

2.5 Modern Pedestrian Detection Methods

A paper published by [10] shows popular detectors from the last decade that are compared experimentally.

Figure 2.14 adapted from [10] below shows nine of the most used pedestrian detectors.

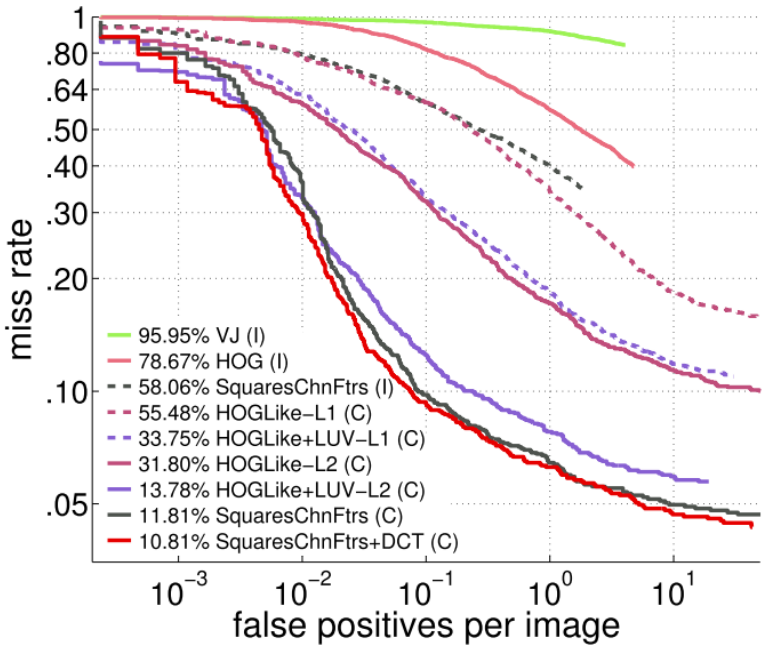


Figure 2.14: ROC curve of nine modern pedestrian detectors as adapted from [10]

The (I) and (C) next to the detector's name indicates that the detector is trained with the *INRIA* or *Caltech* standard pedestrian data sets respectively.

The article compares the log-average miss rates of various detectors. VJ [5], scores lowest with a miss rate of 94%, while HOG, [9], has a miss rate of 78%.

The basis of the algorithm used in this thesis, SquareChnFtrs [11]), scored 58% when trained on the *INRIA* data set, but scored a miss rate of 11% when trained with the *Caltech* data set.

Products on the market today typically have a miss rate of 10%-30% depending on the environment and camera.

It is clear that the data set and detecting algorithm used play vital roles in the accuracy of the detector.

2.5.1 Integral Channel Features / Square Channel Features (ICF/SCF)

Integral channel features (ICF) in its simplest form uses the same basics as the HOG detector/classifier, but instead of only using gradient, or edge, orientation and magnitude information, it combines colour space (RGB/LUV/CMYK) channel features with HOG data to add more fields/channels of information, and thus increases the robustness of the algorithm.

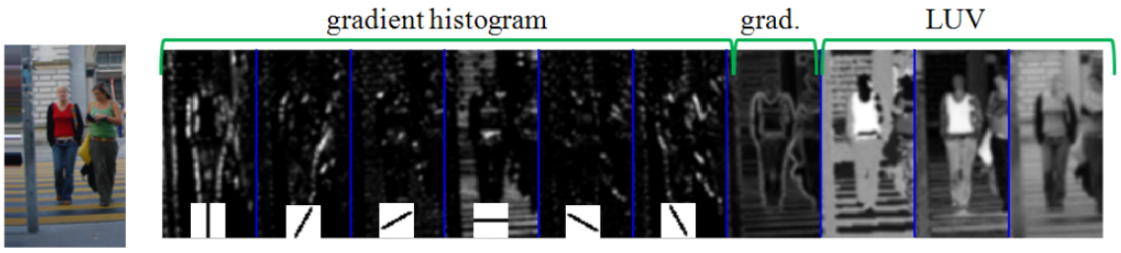


Figure 2.15: Example Of The ICF Descriptor Channels From [11]

.

The channels of the descriptor are shown in figure 2.15 as adapted from [11]. Custom channels can easily be added and can include an integral histogram, point-wise transformations, or any custom function.

Other than HOG[9], ICF uses boosted learning for training similar to AdaBoost and the early training methods of [5] with the variation known as soft cascade from [20].

Instead of having multiple distinct cascade layers, a threshold is set after the evaluation of every *"weak"* classifier/detector trained. A single boosted classifier is trained on the entire training data set. In post-training a simple heuristic can be used to set the thresholds, resulting in a greatly simplified training procedure [20] [11]. The gradient histograms (HOG) are seen as the integral channels (ICF) as they use integral images to rapidly compute the image gradients.

In computer science, a heuristic is a method designed to solve a problem more quickly when classical methods are too slow. It can also be used in finding an approximate solution when classical methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed [21].

There will always be a set of "positive" samples (containing object of interest) that are extremely difficult to detect, or worse are mistakenly labelled positive (false positive). In practice the final threshold of the complete cascade classifier will be set so that these *particular* negative samples are rejected. In [20], these *particular* negative examples can be rejected early in the computation of the cascade.

The proposed training technique runs much faster than classic AdaBoosted cascade techniques. In a test done by [11] it takes 5-10 minutes on a quad-core machine compared with 2 weeks for the original classifier/detector from [5].

2.5.2 Fastest Pedestrian Detector In The West (FPDW)

The authors of ICF [11] made some improvements on their algorithm in terms of speed to achieve near-realtime performance with state-of-the-art detection performance according to [14].

The authors propose a technique to avoid the construction of finely sampled image pyramids without sacrificing performance. Their key insight is that for a broad family of features, including gradient histograms, the feature responses computed at a single

scale can be used to approximate the features' responses at nearby scales. This yields an overall speed up of 10-100 times according to [14].

An example of the different image pyramid approaches from [14] is shown in figure 2.16.

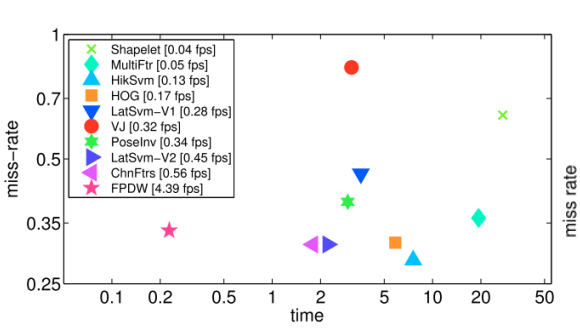
N models, 1 image scale
(a) Naive approach

1 model, N image scales
(b) Traditional approach

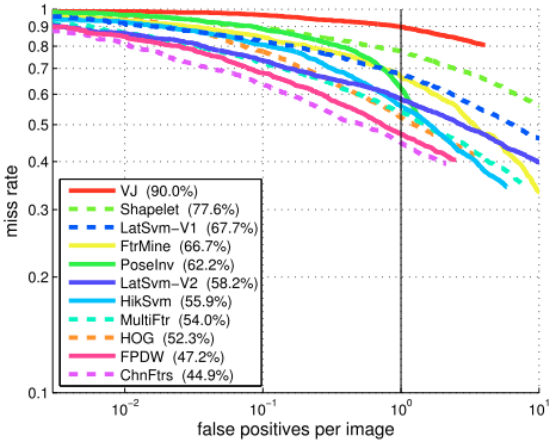
1 model, N/K image scales
(c) FPDW approach

Figure 2.16: An example of different multi-scale image pyramid techniques as adapted from [14]. (a) Shows a naive approach to construct a model pyramid for training by using models (descriptors) at N different scales.(b) The traditional approach used in [9] and [11] where an image pyramid is created from the input image at N scales. (c) The approach proposed by [14], where an image pyramid of N/K is created from the input image.

The FPDW approach computes N/K real scales, and approximates $(N-K)/K$ scales. The paper provides proof and experimental results that minimal information is lost when approximating certain scales, but greatly improves the computational speed [14]. Experimental proof includes figure 2.17a and figure 2.17b from [14].



(a) Speed of detectors running on the Caltech pedestrian data set. All detectors are tested on the same machine, and thus the speeds are comparable [14].



(b) ROC-curve of different detectors' results/miss rate when tested on the Caltech "Reasonable" Data Set [14].

The results show that the FPDW algorithm runs 10-100 times faster than the rest of the detectors/classifiers in figure 2.17a. The results in figure 2.17b show that the effect of approximating image scales has a negligible impact on performance when compared to the previous version of the algorithm, ChnFtrs (ICF), discussed in section 2.5.1. Additional results, evaluation scripts and detector descriptions, including the pedestrian data set used, are available online according to the authors at [22].

2.6 Tracking Methods

2.6.1 Appearance-Based Tracking

Appearance-based tracking is a simple, non-parametric tracking solution and is thus easy to implement for simple applications. It relies on consistency of the object's appearance for template matching, and thus it is difficult to distinguish between – and track – multiple moving objects simultaneously and correctly.

2.6.1.1 Mean-shift/ Camshift

Mean shift is a simple iterative procedure that shifts data points to the average of the data points in its neighbourhood [15]. In image processing it is a basic computational module used in pattern recognition and to track the motion of a cluster of "interesting features".

Interesting features are commonly defined by a set of attributes such as connecting three lines (a corner), which is more distinct than edge information [23].

It is said that mean-shift is an efficient approach to tracking objects whose appearance is defined by histograms. Firstly a cluster of interesting features is chosen, e.g colour + texture. The mean-shift algorithm scans the subsequent frame using a sliding window that starts at the peak of the feature distribution (presumably the center of the object) in the previous frame.

The mean-shift algorithm then seeks to find the new peak (mode) of the feature distribution in the current frame, which is also (presumably) centred around the object that produced the feature distribution from e.g. the color and texture.

In this way the mean-shift algorithm tracks the object frame by frame based on its histogram and appearance by template-matching the feature distribution.

Figure 2.18 shows an example of an initial window placed over a 2D array of data points. The sliding window is successively recentred over the local peak of the data until it converges.

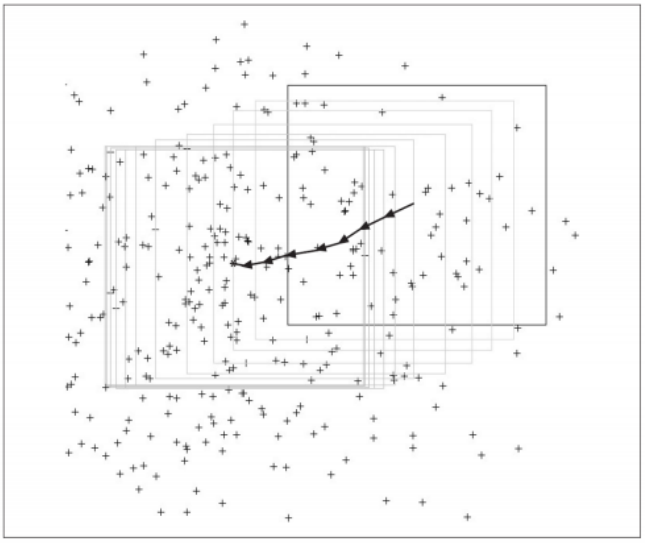


Figure 2.18: Example of successively recentred window by means of Mean-Shift [12].

In order to work accurately, this algorithm relies on feature distribution to remain the same in subsequent frames, and can thus be error-prone in practical applications. This is because some objects move/rotate discontinuously due to occlusion or changes in illumination, which could possibly change the histogram feature distribution.

2.6.1.2 Motion-Template Matching

Motion templates are useful in tracking general movement and are especially apt to be used in gesture recognition.

Motion-template matching is a gradient method (it uses edges) and thus requires silhouettes (or part of a silhouette) [12].

According to [12], object silhouettes can be obtained in several ways:

- Object silhouettes can be obtained by using a stationary camera and employing frame-by-frame differencing (what has changed from the one frame to the next). Differencing results in the acquisition of the moving edges of the object, which yields sufficient features to make motion templates work.
- Chroma keying can also be used. If, for example, one has a known background, the foreground can then be taken as anything that is not the background. This is a method of background subtraction / foreground detection. The background model can be obtained with various methods, such as using a pixel-wise moving average, or using thresholds to calculate the persistence of a pixel (background pixels tend to be unchanging and persistent for long periods of time).

Once a object silhouette is obtained, a sliding window performs a local scan and tries to minimize the distance between silhouettes in successive frames by matching the silhouette template to its closest obtained silhouette. This is similar to the mean-shift algorithm, but instead of using a feature distribution as the template, it uses object silhouettes.

This technique is used mostly in rigid-body detection (cars, bicycles) as the silhouettes remain mostly constant over the duration of motion. In pedestrian detection (non-rigid), pedestrian silhouettes constantly change due to the motion of walking resulting in swinging arms and moving legs, and thus results in discontinuous tracking.

2.6.2 Parametric Tracking

Parametric tracking is an object-tracking technique that employs a motion model and a predictive filter. This technique is useful to track objects with changing appearance histograms as it uses previous information about the object's position, the detection score, and optimization to predict the object's location in future frames. The motion model of moving objects should be obtained from either calculations or physical observations.

2.6.2.1 Kalman Filter

A Kalman filter is a method of tracking objects mathematically. The Kalman filter is a set of mathematical equations that provides an effective and computationally efficient way to estimate the state of a process (position of an object). It is done in such a way that it minimizes the mean of the square error (least square error optimization) [12].

The Kalman filter has two groups of equations:

- *Time Update* equations. These equations are used to project the current state and error covariance estimates forward in time to obtain the appropriate estimates for the next time step. These equations can be seen as the predictor equations.
- *Measurement Update* equations. These equations are used for the feedback loop. This incorporates the new measurements into the appropriate estimates to obtain an improved predictive estimate. These equations can be seen as the corrector equations.

The Kalman filter is useful in multi-object tracking, where objects tend to change their feature distribution due to occlusion, rotation, or the illumination effect. This is because the Kalman filter only uses the object's motion, the motion model, and a corrective feedback loop in order to make accurate predictions. This also means that pedestrians briefly occluded will still be tracked, as the predicted bounding box would be where the detected object should have been had it not been occluded. This makes the Kalman filter ideal for pedestrian tracking.

One obstacle that arises with parametric tracking, and the Kalman filter in particular, is that the motion model used should be as accurate as possible. This also means that the filter is not generic if objects have different movement in different footage, e.g. pedestrians commonly walk from left to right at a certain speed, versus pedestrians walking diagonally at a different speed, which would require a different or adjusted motion model.

Various parts of the practical Kalman filter and its usage as a tracking algorithm together with the track- assignment algorithm are discussed in section 3.8.2.1 and section 5.2.

2.7 Summary

This chapter reviewed the relevant building blocks of image processing in pedestrian detection, and discussed various NMS and classifier training methods. The chapter also presented methods used to interpret the detection results through graphical methods or through inspection. Classical pedestrian-detection methods, including the works of [5] and [9] were discussed as these are regarded as the basis of modern pedestrian detection. Modern pedestrian detection methods are also discussed, including the method used in this thesis (ICF by [2]). Lastly, various methods for tracking pedestrians by appearance or by parametric methods were presented and discussed.

Chapter 3

Setup And Methodology (Detection & Tracking)

The main objective of this thesis is the automatic processing of low-resolution surveillance footage from existing infrastructure to obtain information about the movement and speeds of pedestrians using pedestrian bridges.

The pedestrian-monitoring system includes a detection stage, where a cascade classifier is trained with AdaBoost and the ICF detection algorithm is used. The tracking stage uses parametric tracking in the form of a Kalman filter and helper functions. The tracking results are then used in the counting algorithm to determine the number of pedestrians crossing pedestrian bridges, as well as the direction and relative speed of movement.

3.1 Introduction

This chapter describes the setup and methodology used to allow for pedestrian detection and tracking. Referring to figure 3.1 below, section 3.2 (Detection Dataflow (1)) discusses the data-collection phase including the choice of data as well as how the data is collected. Section 3.2(1) also describes how the collected datasets are pre-processed. Section 3.3 (Training Phase (2)) discusses the training phase and the general parameter settings and setup for training. Section 3.4 (Detection Phase (3)) discusses the detection phase. This discussion includes the parameter settings and setup of the detection phase, how the detection regions should be defined, and the detection output format. Section 3.5 (Verification Phase (4)) discusses the verification phase used to evaluate the detection results.

For the tracking stage, section 3.7 (Tracking Dataflow (5)) discusses how the two stages are linked. Section 3.8 (Tracking Phase (6)) discusses the parameter settings and general setup of the tracking phase, including tuning parameters as well as the Kalman filter model. Section 3.9 (Tracking Verification (7)) discusses the verification phase of

the tracking stage.

The physical hardware and software setup is discussed in section 3.11. A conclusion in respect of both stages as an integrated unit is discussed in section **??**, and a general conclusion of this chapter is discussed in section 3.13.

Figure 3.1 shows a brief overview of the setup for the integrated system.

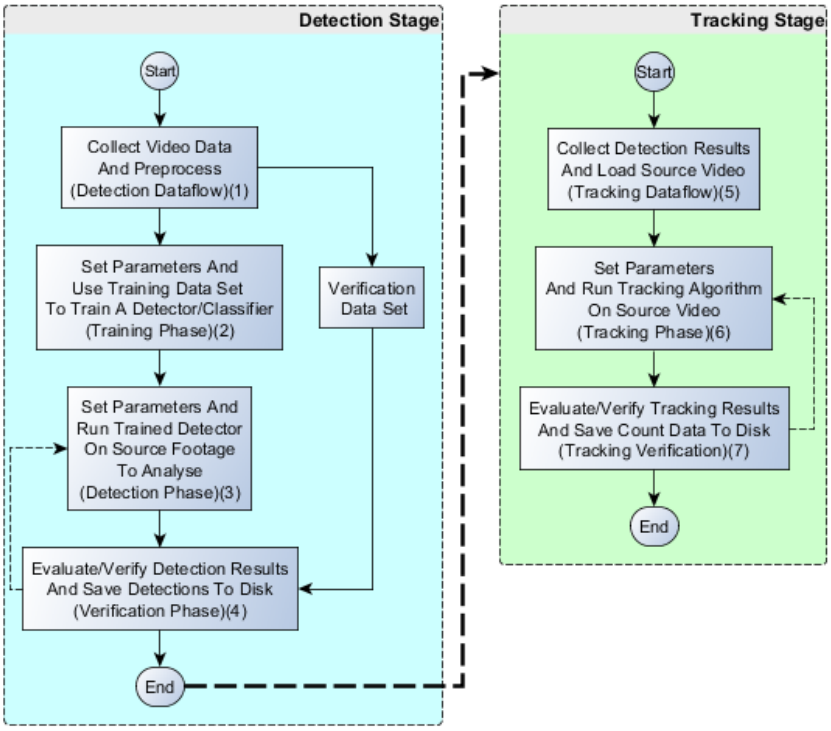


Figure 3.1: General overview of the setup of the integrated system containing the Detection and Tracking Stages

Figure 3.1 shows the setup for both the detection and tracking stages, which will be discussed in the following sections.

3.2 Detection: Data

3.2.1 Introduction

The pedestrian bridge video footage from SANRAL is converted from a live feed to historical footage. The historical footage is then stored in SANRAL archives for up to 5 years if an incident occurs. Normally the footage is kept for 6 months. This means that an abundance of historical data is available.

Figure 3.2 below shows a methodology used in the dataflow and processing of the source video data from the SANRAL archive.

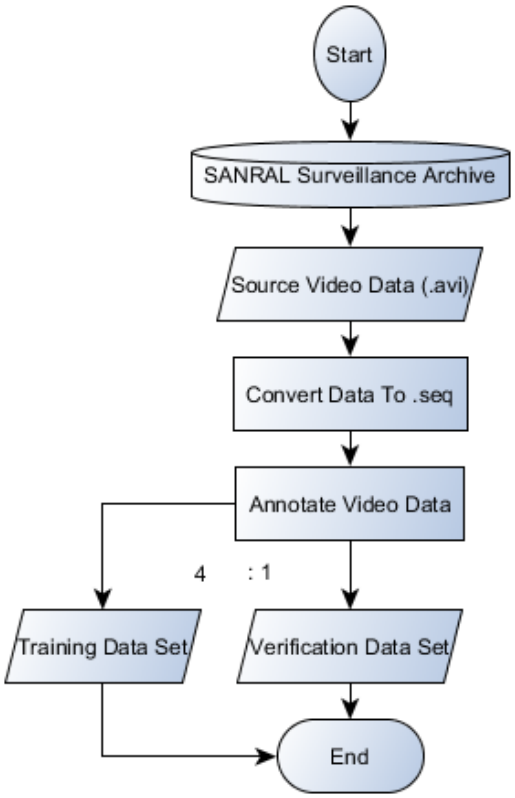


Figure 3.2: Data flow overview for the source data and data collection

Figure 3.2 shows the methodology followed in collecting and processing the data needed to train and verify a classifier. The source video data is chosen and collected from the SANRAL pedestrian surveillance archive. The source videos are converted into a format usable by the annotation/labelling MATLAB program by [13]. The footage is pre-processed by annotating/labelling the data and dividing the data into training and verification datasets. The training dataset is used in the training phase (section 3.3) and the verification dataset is used in the verification phase (section 3.5).

3.2.2 Data Collection

Two pedestrian-training datasets from *INRIA* [24] and *Caltech* [25] used in [16] were initially downloaded to test whether the dataset used to train the detector could be generic or needed to be scene specific. A sample frame from the *INRIA* pedestrian

training dataset is shown in 3.3, and a sample frame from the *Caltech* training dataset is shown in 3.4.

(a) Sample from INRIA dataset(1)

(b) Sample from INRIA pedestrian dataset(2)

Figure 3.3: Samples of positive samples from the INRIA dataset

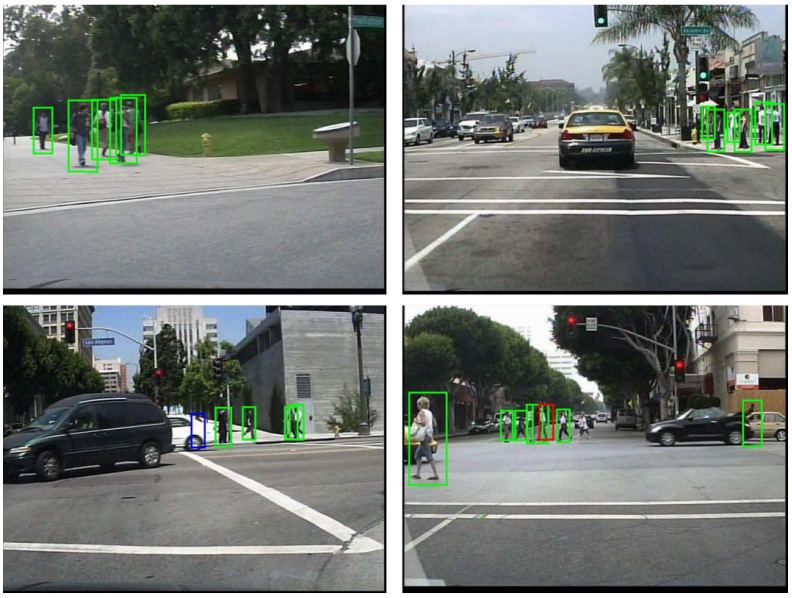


Figure 3.4: Samples from Caltech pedestrian dataset with annotations

The training datasets from INRIA and Caltech are used to train their respective detectors. These detectors are tested on surveillance footage acquired from the SANRAL video archive.

Some cameras at the pedestrian bridges taken into consideration are shown below; note that some pedestrian bridges have multiple cameras per bridge to ensure pedestrian safety.

Figures 3.5a and 3.5b show samples from camera 303A at the pedestrian bridge on the R300. Figure 3.6c, 3.6d, and figure 3.7e show samples from camera 307 at the pedestrian bridge on the R300. Figure 3.7f and figure 3.8g show samples from *camera 309* at the pedestrian bridge on the R300. Lastly, figure 3.8h shows samples from camera 220 at the pedestrian bridge on the N2.

(a) Pedestrian bridge at camera 303A on the R300 with the camera facing East

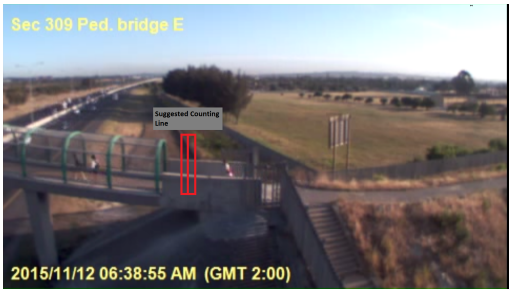
(b) Pedestrian bridge at camera 303A on the R300 with the camera facing West

(c) Pedestrian bridge at camera 307A on the R300 with the camera facing East

(d) Pedestrian bridge at camera 307A on the R300 with the camera facing West

(e) Pedestrian bridge at camera 307A on the R300 with the camera facing West

(f) Pedestrian bridge at *camera 309* on the R300 with the camera facing West



(g) Pedestrian bridge at *camera 309* on the R300 with the camera facing East

(h) Pedestrian bridge at camera 220A on the N2 with the camera facing West

Figure 3.8: Sample frames from SANRAL pedestrian bridge surveillance cameras under consideration

Figure 3.9 corresponds to figure 3.8 shown above and indicates the locations of the pedestrian bridges and surveillance cameras. All the cameras are static and transmit a live feed to the Cape Town TMC (Tactical Management Center), where it is monitored by traffic authorities to ensure pedestrian safety.

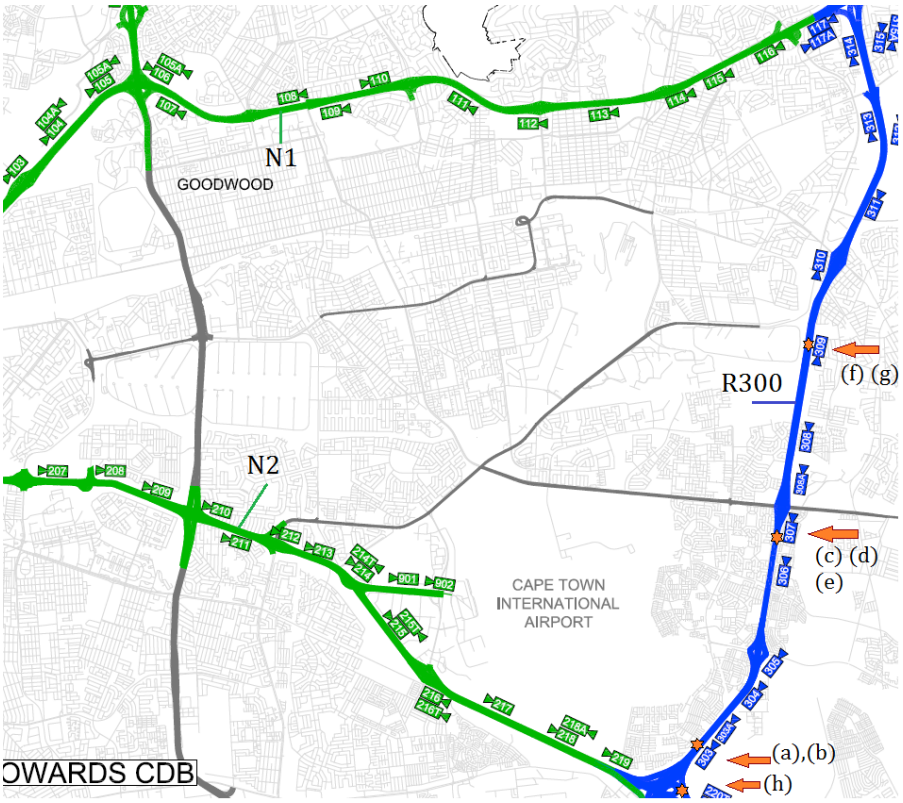


Figure 3.9: Map indicating the locations of cameras under consideration for data collection

After some consideration, two camera sources were chosen as possible candidates. Camera 307A at the pedestrian bridge on the R300 facing East, figure 3.6c, and *camera 309* at the pedestrian bridge close to Van Riebeeck on the R300 facing East, figure 3.8g, were chosen as best-case-scenario samples for data collection.

These cameras where chosen as they both seem to have a partially unoccluded view of pedestrians walking along the bridge, with *camera 309* in figure 3.8g being the best candidate, as pedestrians can walk from the left of the image to the right mostly unoccluded by fencing. The camera also faces the pedestrians from the side instead of the front, resulting in minimal occlusion of pedestrians in large groups as the bridge width only allows a maximum of two or three pedestrians to walk side-by-side. During periods of high pedestrian traffic, pedestrians mostly walk single file in each repective direction.

Camera orientation is important as one pedestrian walking in front of another would partially block/occlude the pedestrian behind him/her and would thus make it harder for the classifier to distinguish between individual pedestrian silhouettes used by HOG and subsequent classifiers.

On initial inspection of the pedestrian-training datasets collected from SANRAL, it is evident that the data contained in the pedestrian-training dataset from *INRIA* and *Caltech* differ from that of surveillance footage collected from SANRAL.

Clear differences would be:

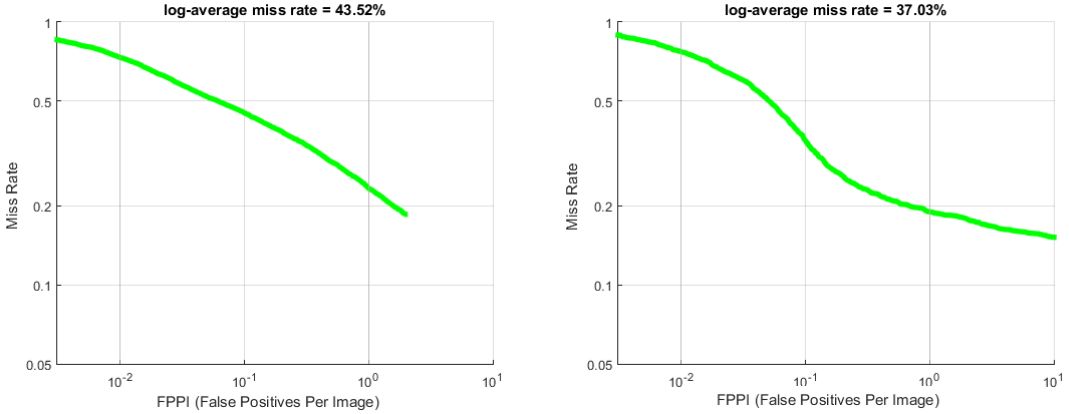
- The angle of the footage. Both *INRIA* and *Caltech* training datasets contain images of pedestrians taken from eye level, where the SANRAL surveillance footage collected is taken from an elevated surveillance position. This means that pedestrian appearance or feature channels will differ due to the difference in perspective.

- The resolution of the (*INRIA, Caltech*) training dataset samples is also significantly higher than that of the pedestrian-bridge footage collected from SANRAL. The (*INRIA, Caltech*) training datasets use a positive sample size (pedestrian size) of 96x160 pixels, while the SANRAL pedestrian bridge footage has a sample size, at data collection, of 36x28 pixels (now 34x16 pixels). This means that a mismatch in the scale pyramid of the trained multi-scale would result in diminished detection results. The pedestrian-bridge footage has a significantly lower resolution than that of the (*INRIA, Caltech*) training datasets. There are thus fewer total features in the low-resolution footage, resulting in the features that exist becoming of critical importance.

- Although down-sampling can be done as a pre-process to lower the positive sample sizes, this proved troublesome in practice. Down-sampling could be achieved with the use of an image pyramid and Gaussian smoothing, but clear differences between pedestrians in the (*INRIA, Caltech*) training datasets still exist. Differences would include the angle between the camera and pedestrians as mentioned above, as well as the negative background samples, which in the case of surveillance footage remain mostly static (ignoring illumination effects). The negative background samples thus contain useful information absent in the (*INRIA, Caltech*) training datasets.

- The aspect ratio differs between the (*INRIA, Caltech*) training datasets and those collected from SANRAL, requiring the remapping/resizing of the samples and possibly resulting in information losses during conversion.

The results of testing/evaluating the (*INRIA, Caltech*) trained detectors on the limited annotated footage collected from SANRAL are shown below. Figure 3.10a shows the ROC graph of the *Caltech* classifier and figure 3.10b shows the ROC graph of the *INRIA* classifier applied to *camera 309* from figure 3.8g.



(a) ROC graph of the *Caltech* pedestrian training dataset applied to *camera 309* from figure 3.8g

(b) ROC-graph of the *INRIA* pedestrian training dataset applied to *camera 309* from figure 3.8g

Figure 3.10: ROC graphs of trained (*INRIA, Caltech*) detectors on limited annotated SANRAL surveillance footage

Figure 3.10a shows that the *Caltech* detector obtained a log average miss rate of 43.52%, and figure 3.10b shows that the *INRIA* detector obtained a log average miss rate of 37% when tested on the SANRAL surveillance footage. These respective miss rates are far above those of the state-of-the-art detectors shown in section 2.5, with the lowest miss rate being 11%.

It should be noted that the ROC graphs can be misleading as they were only tested/evaluated on a small subset of frames from the SANRAL footage, which was during peak visibility and medium pedestrian activity (from 6:30am to 07:45am).

The resulting miss rates and evaluation based on visual inspection suggested that the detectors/classifiers should in fact be scene specific in order to obtain state-of-the-art detection performance.

It is thus clear that the detectors should be trained individually using training data from the camera that it is to be used with.

3.2.3 Dataset Selection

The training and testing data, in the form of .AVI (Audio Video Interleave) videos, was collected from the SANRAL archival database. Exported videos are said to be of reasonably ideal quality and do not contain downtime or signs of packet loss, which introduces unwanted noise or losses in the dataset.

Losses and downtime observed include:

- Signs of packet loss in the footage manifested as corrupted frames.

- Losses manifested as general glitching, where frames would skip back five frames every second or third frame.
- Downtime caused by cameras being out of focus (the main reason for downtime).
- Camera downtime caused by cameras having connectivity issues.

To remove influential environmental impacts from the results, only sunny days or days with clear visibility between the camera and pedestrians are chosen. This is referred to as reasonable conditions. Reasonable conditions would give a best-case scenario for camera performance during average operating conditions.

At the time of the dataset collection, the specifications for the archived videos were to have a resolution of 320x240 at a frame-rate of 12.5fps (now 480x272 at 5fps), with positive sample sizes (pedestrian sizes) averaging a size of 36x28 pixels (now 30x15 pixels).

3.2.4 Preparing The Dataset

The training datasets from *camera 307* at the pedestrian bridge on the R300 facing East, figure 3.6c, and *camera 309* at the pedestrian bridge close to Van Riebeeck on the R300 facing East, figure 3.8g, both need to be prepared individually. Ultimately *camera 309* is chosen as the best-case-scenario for pedestrian monitoring using existing infrastructure.

Preparations include:

- Annotating/Labelling the dataset.
- Dividing the dataset into training and verification datasets respectively.

3.2.4.1 Dataset Annotation/Labelling

Annotations are meta-data indicating that the region inside a drawn bounding box is an object of interest, or a positive sample in a given image. Here the given image is a frame from the camera footage collected. Regions outside of the bounding box indicate that there are no positive samples present (pedestrians), and thus are called negative samples or negative background samples.

Negative samples contain important information due to the nature of the static camera setup, where the background stays mostly constant. Negative samples are similar to positive samples, but instead show the algorithm what is not of interest or does not contain objects of interest. In a static background, negative background samples are useful, as distinct hyper-planes for the cascade classifier can thus be computed (see section 2.2.7.2 on page 18) between positive and negative samples. The negative samples (background) reoccur in all the frames of the specific camera and thus receive significant weights during training.

The detection algorithm is said to use supervised learning, and thus can only classify/detect based on the information/descriptors given during training.

Positive samples should include as much of the expected detection scenarios as possible, e.g. if pedestrians climbing over fences should be detected reliably, positive samples of pedestrians climbing over fences should be included in the training set. This includes positive samples at different times of the day, as shadows, color variance, and illumination change during the course of the day and would result in a disparity between the trained classifier and the computed descriptor.

Figure 3.11a, 3.11b and 3.11c show samples from *camera 309* at three distinct times of the day.

(a) Sample from *camera 309* at 06:00am

(b) Sample from *camera 309* at 11:00am

(c) Sample from *camera 309* at 04:30pm

Figure 3.11: Example of illumination effects on *camera 309* throughout an average day

Note the shadows cast from the fencing around the pedestrian bridge. The majority of the data collected focus on peak times, as these coincide with peak pedestrian activity and are thus the most important times to detect pedestrians. Collecting data around peak times also has the benefit that it minimizes the total number of annotated frames required as they contain a high pedestrian density.

An annotation toolbox for MATLAB by [13]) is used for labelling. A sample from the annotation toolbox, with bounding boxes, is shown in figure 3.12.

The toolbox allows the labelling of key frames to annotate a continuous set of frames. To accomplish this, the toolbox uses a basic speed model to estimate the linear movement of objects using the start and end frames. The annotation toolbox is used as is. Both the training and validation datasets are annotated.

Figure 3.12: Sample from the annotation toolbox by [13]

The dataset video files are converted to .seq format before the annotation can commence. Changing the format converts the video file into a large sequence of images that can easily be skipped through without the need to decode each frame. The reasonable .seq files are annotated by hand.

Labelling by hand is very time consuming, as only about 70 frames can be annotated at a time, with 12 hours of video accumulating to 600 000 frames. It must be noted, however, that only a fraction of the frames contain objects of interest, i.e. pedestrians.

The footage from *camera 309* is partially annotated over different times of the day as shown in table 3.1.

Table 3.1: Number of frames collected at different times of day for *camera 309*

Number of frames	From (Time)	To (Time)
14700	06:00am	06:50am
9500	11:00am	11:30am
8000	04:00pm	04:25pm

The annotation information is of critical importance, as it has a considerable impact on the performance of the detector and should thus be done meticulously. If frames are mislabelled, there is no ground-truth data. This would introduce false negatives in the training stage and result in classification as a false positive in the verification stage.

The misclassification would be a result of the mismatch between the detector output and the ground-truth data. The detector/classifier would still detect the object, but would be classified as a false positive compared to the ground truth.

After labelling has been completed, the annotated .seq file and corresponding ground-truth data are extracted to a directory in the form of text-image pairs for each frame of the annotated .seq file.

The bridge of *camera 309* partially occludes pedestrians at the suggested counting line with a concrete wall. To overcome this obstacle, two individual annotation objects are created, resulting in two distinct detectors: A detector for pedestrians in full view, and a detector for pedestrians with their lower halves occluded by the concrete barrier.

The positive sample size of the occluded pedestrians differs from that of the *full* size pedestrians. The occluded *half* size pedestrians have a model size of 17x17, and the *full* size pedestrians have a sample size of 34x16. Figure 3.13 shows the two detection areas with sample bounding boxes around both *full* and *half* pedestrian objects.

Figure 3.13: Sample frame showing two different regions to be annotated

During counting, the only pedestrians of interest are located in the occluded (*half*) section where the proposed counting line would be. Thus some may argue that it would be unnecessary to annotate the *full* pedestrian object. This would be true if only detection location and thus the detection stage are of importance, but movement information and continuous detection on a wider range of frames are critical for the tracking algorithm. The added information assists in making both stages more robust, and improves accuracy.

The number of frames taken by an average pedestrian to cross the occluded/half section is only 14, which does not provide enough information for the Kalman filter (discussed in section 2.6.2.1 on page 30, section 3.8.2.1 on page 55, and section 5.2.1.2 on page 76) in the tracking algorithm. More movement information is required to accurately predict the object's (pedestrian's) location.

3.2.4.2 Dividing The Dataset (Train/Validation)

The annotated datasets are divided into training and validation datasets. The training dataset is used to train the cascade classifier, and the validation dataset is used as validation to evaluate detector performance.

A good starting ratio recommended by [9] is to divide the dataset with a ratio of 4:1 training to testing set. The optimal size of training versus testing dataset size can be determined empirically by comparing validation performance.

In practice, only 25% of the dataset around 06:30am was taken as the validation set, with the evaluation results only serving as an indicator of detector performance or used for comparison when adjusting parameters.

Annotating overall scenarios for both chosen cameras is very time consuming and thus the annotated frames are rather used in the training dataset to increase

the number of descriptors/features available, which in turn could increase detector performance.

3.3 Detection: Training

3.3.1 Introduction

Training is to 'train' a classifier/detector to recognize pedestrians by 'showing' the algorithm a number of positive samples – images containing pedestrians in this case. The training algorithm used is a form of optimized AdaBoost from [13] as discussed in section 2.2.7.2 on page 18.

Before training can commence, the parameters should be set up appropriately, with most parameters giving a trade-off between accuracy and speed. Only significant training parameters will be discussed here, as there are too many for the scope of this thesis.

A detailed discussion and empirical results of generic default parameter values are provided in [9] for the HOG parameters and in [11], where different parameters and their impact on different kinds of detectors and training methods are compared. Most of the generic parameters used are set to the values as recommended by the authors in their respective papers used as the basis of the MATLAB toolbox [13] [14] [2] [26]. Important parameters are discussed in Chapter 4, where the implementation of the parameters is presented. [14] presented the recommended number of approximated and real scales in the image pyramid. [2] presented the number of weak classifiers and training stages, and [26] presented the jitter/filter parameter values and recommendations.

Figure 3.14 shows the methodology and setup of the training phase.

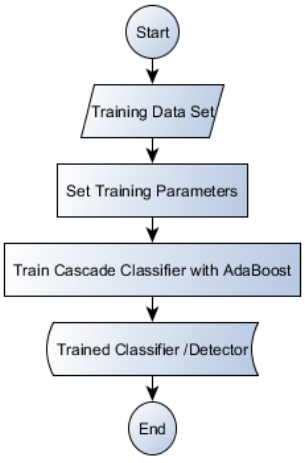


Figure 3.14: Training overview

Figure 3.14 shows the methodology followed in the training phase of the detection stage. Firstly the training dataset is imported from data collection phase (section 3.2). The static and tuning parameters of the training algorithm are set, and a cascade classifier is trained with AdaBoost. The trained classifier is then parsed to the detection phase (section 3.4).

3.3.2 Discussion of Parameter Settings And Setup

A complete list of important generic parameters with empirical results is given in [11].

Some important parameters and recommendations include:

- An accurate minimum model size of expected pedestrians is important, as bounding boxes annotated smaller than this minimum model size will be ignored.
- An optimal sliding window stride length and sliding window size will give a trade-off between speed and accuracy as discussed in section 2.2.2 on page 12.
- The amount of padding also plays an important role. Padding adds a set number of blank pixels around the positive samples in order to avoid boundary conditions when densely scanning the image. The amount of padding also has an impact on the detection window overlap.
- By using the ICF (section 2.5.1 on page 25) algorithm, the feature channels are convolved by a set of basic filters such as 1D Gaussian, DoG (Difference of Gaussian), and 1D Gabor filters as discussed in detail in [2] and [26]. These filters remove local correlations, and thus remove data that could skew the detector. This slows performance, as convolution of 9 large image channels with even a simple filter is computationally complex [27].

The implementation of the tuning parameters is discussed in chapter 4.

3.4 Detection: Discussion

3.4.1 Introduction

The detection phase will be discussed in this section. Focus is placed on some important parameters used in the setup of this phase, the setup of the detection regions, and the detector output.

Figure 3.15 shows the methodology followed in the detection phase of the detection stage.

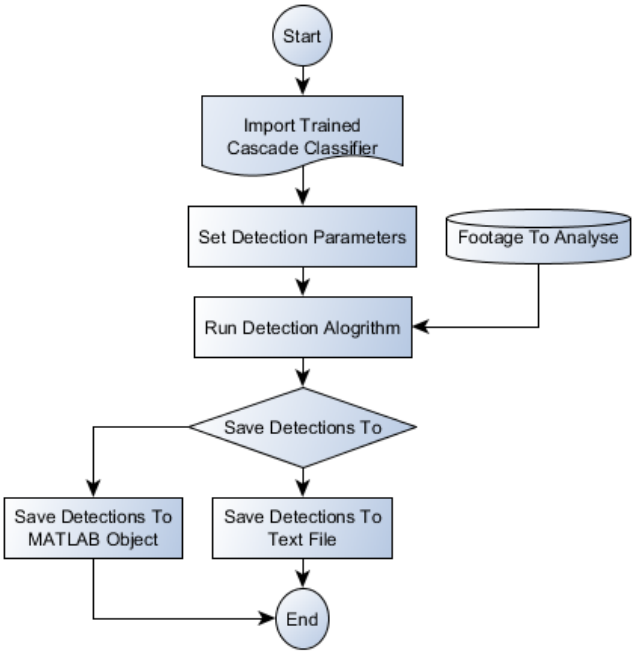


Figure 3.15: Detection-phase methodology

Figure 3.15 shows a broad overview of the detection phase. The trained classifier is imported from the training phase to the detection phase. Initially, static parameters and tuning parameters of the classifier are set. The detection algorithm analyzes the source footage collected from SANRAL's existing surveillance infrastructure (section 3.2.2 on page 3.2.2). After the analysis is completed, the detection output from the classifier is saved to a hard disk to be reviewed in the verification phase (section 3.5 on page 51).

3.4.2 Discussion of Parameter Settings And Setup

A detailed list of generic detection parameters and their empirical impacts are discussed in [26] [10]. The MATLAB toolbox from [13] is used as it incorporates speed optimizations made by the author.

Some important parameters include:

- The expected average model/positive sample size for pedestrian detection is used as the first scale of the image pyramid, where scale octave down is computed. This should not be confused with the minimum model size set during training.
- The toolbox offers four NMS methods as discussed in section 2.2.6 on page 15, being mean-shift, absolute maximum, an optimized absolute maximum

algorithm, and cover suppression. Non-maximum suppression is n^2 efficient, but can become a time consuming process if the threshold is not set correctly and more than (for example) 50 bounding boxes need to be processed over 200 000 frames.

3.4.3 Defining The Detection Region/ Region Of Interest (ROI)

To reduce the computational time spent densely scanning image regions where no pedestrians are expected to be, a detection region can be defined.

The detection region crops each frame to the indicated region of interest, and thus reduces the image resolution and run time. This is especially useful when working with *camera 309*, as the two distinct classifiers (*full* and *half*) need not scan/search the full image for pedestrians. The proposed regions of interest can be seen in figure 3.13.

Cropping the frames to the region of interest reduces the frame size that need to be scanned, e.g. an original frame of 480x272 pixels is broken into a 250x100-pixel frame for the *full* classifier and a 120x75-pixel frame for the *half* classifier.

3.4.4 Detection Output

The classifier outputs the detected bounding boxes in either a specified text file or to a MATLAB object containing the bounding boxes.

The outputs contain six fields for each detection:

- The frame it was detected on, Id.
- The upper-left x coordinate of the bounding box, with the axis starting at the left.
- The upper-left y coordinate of the bounding box, with the axis starting at the top.
- The width w of the detected bounding box.
- The height h of the detected bounding box.
- The detection/confidence score s of the bounding box.

The output has the format $[Idxywhs]$ for each detection.

The classifier/detector can either take a single image as an input or an array of images (frames of a video), and return either a single detection in a text file/single bounding box variable, or an array of detections in a single text file/array of bounding box variables.

3.5 Detection Verification Methodology

Figure 3.16 shows a broad overview of the setup and procedure followed in the verification phase of the detection stage.

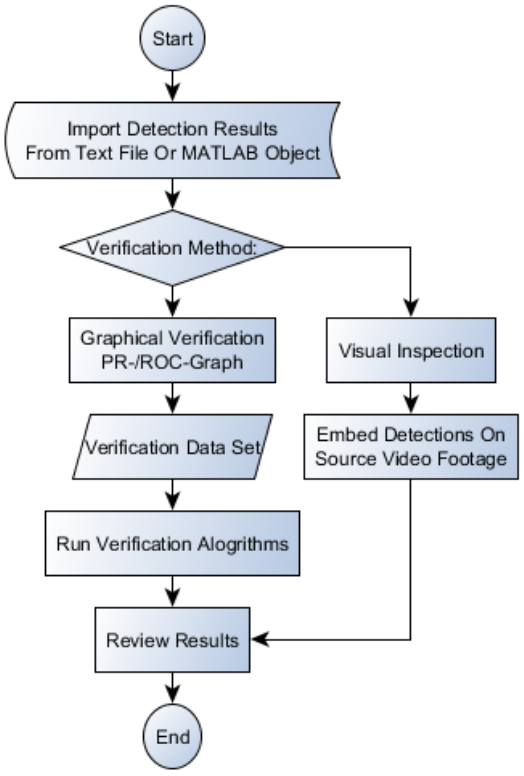


Figure 3.16: Detection-verification methodology

Figure 3.16 shows the methodology of reviewing the detection results. The imported detection results from the output of the detection stage are verified and evaluated by means of visual inspection or by means of the graphical verification methods as discussed in section 2.3. The visual or graphical results give insight into how to improve detection accuracy, and are used in the detection and training phase to make the necessary adjustments to the training dataset or parameters.

3.6 Detection: Overview

Figure 3.17 shows an overview of the complete detection stage as discussed in sections 3.2 to 3.5.

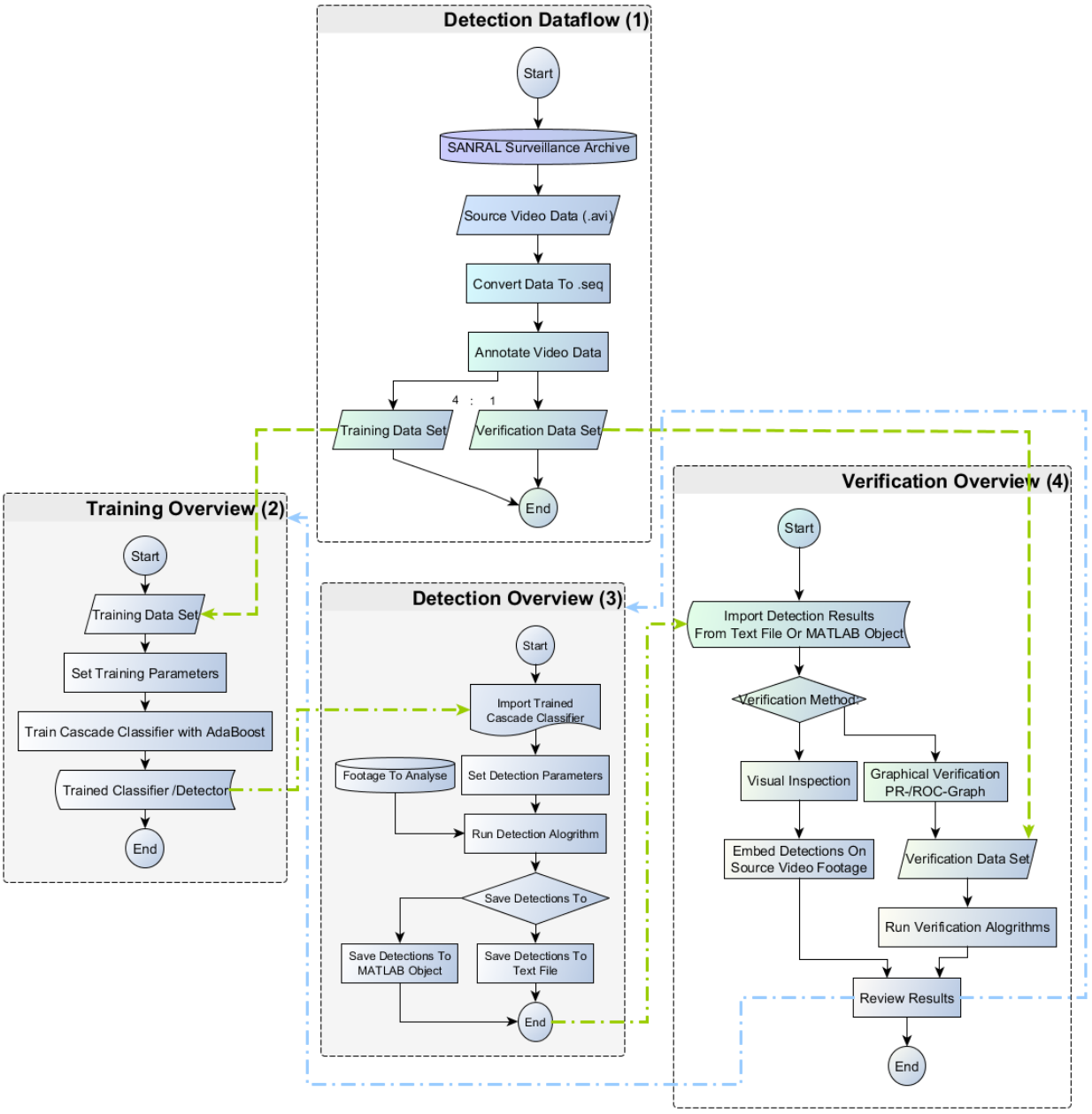


Figure 3.17: Detection Stage Overview

Figure 3.17 shows a complete algorithm for the detection stage. The algorithm starts at the detection dataflow (section 3.2), then moves to the training overview second (section 3.3), then to the detection overview (section 3.4), and lastly to the verification overview (section 3.5).

After the verification phase is completed, the results, or impact of parameters on the training/detecting phase, can be used to adjust the parameters in the training/de-

tecting phase to improve performance. This creates an iterative feedback loop to obtain the optimal parameters.

3.7 Tracking: Data

3.7.1 Introduction

Figure 3.18 shows the methodology used in the data-flow phase of the tracking stage.

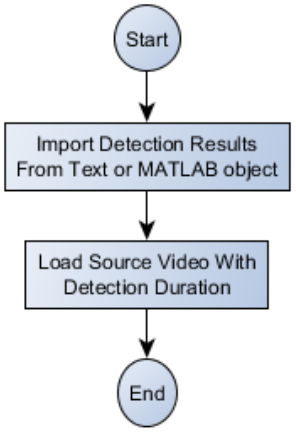


Figure 3.18: Tracking stage initial dataflow

The dataflow to input the data in the tracking stage is shown in figure 3.18. The detection results from the detection stage (section 3.4), and the video source to embed the tracking results, are imported and parsed to the tracking stage (section 3.8).

3.7.2 Interfacing With Detection Stage

In the tracking stage, data is parsed from the detection stage in the form of text files or a MATLAB object. These files/MATLAB object contain an array of bounding boxes and detection scores that correspond to each frame. The camera used, the name of the video footage, and the time interval used in the detection stage are also parsed to the tracking stage.

Parsing information about the camera, video footage, and time interval to the tracking stage allows the tracking stage to overlay the tracking/detection results on the source footage used. The complete tracking results can then be saved as an array of image frames, or as a video to review at a later stage.

3.8 Tracking: Discussion

3.8.1 Introduction

Figure 3.19 shows the setup and methodology followed in the tracking phase of the tracking stage.

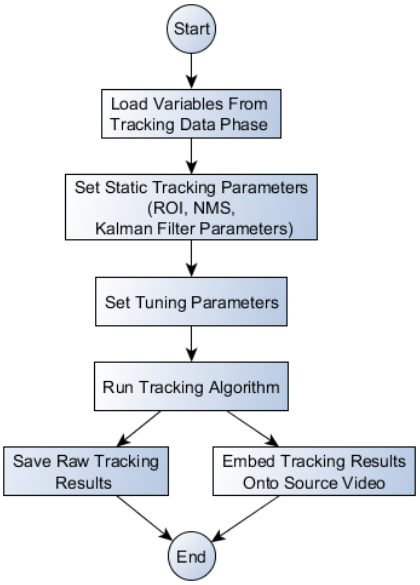


Figure 3.19: Tracking algorithm broad overview

Figure 3.19 shows the tracking phase. The variables are loaded/imported from the data phase (section 3.7). The static and tuning parameters of the tracking algorithm are set and the tracking algorithm is run. The raw detections and embedded tracking results are saved to the hard-drive and parsed to the verification phase (section 3.9).

3.8.2 Discussion of Parameter Settings And Setup

The tracking algorithm is based on a tracking-by-optimization example provided by MATLAB. The algorithm is adapted to support the counting of multiple tracked bounding boxes or centroids. Most of the parameters to facilitate the tracking are tuning parameters and have to be set through trial and error, but estimated suggested parameters are given.

The only important static parameter that is initially set is the confidence threshold. The confidence threshold is used to determine if the track is a true detection or a false alarm. The confidence threshold corresponds to the detection scores, and should be

set after observing the average detection scores of detected pedestrians in the detection stage.

Once an object with a detection score higher than that of the confidence threshold is detected, the track and detection is marked as true positive, and is displayed.

3.8.2.1 Kalman Filter Model

The tracking algorithm uses a Kalman filter to predict the position of a bounding box in the current frame, given the position of the bounding box in previous frames. The MATLAB implementation of the Kalman filter is used, as this includes optimized and time-efficient processes.

The Kalman filter is critical in the tracking stage as it ultimately links the detections between frames, which allows the collection of direction and duration data. It is important that the predicted location be close to the actual location to obtain accurate counting results as a result of less fragmented tracks.

The parameters used to configure the Kalman filter include the following:

- The pedestrian-movement model is chosen as a constant velocity model, as pedestrians are expected to move at a constant pace. This is the motion model the filter will be using when predicting locations.
- The initial location is used as the position of the detected bounding boxes' centroids. This is used as the starting point from which the Kalman filter should predict subsequent centroid locations.
- The estimated initial error is the expected error variance of the initial value. It is assumed that the objects of interest, pedestrian centroids in this case, have zero initial velocity. The value of the estimated initial error can be set to an approximated value as shown in equation 3.1.

$$InitialEstimateError = (assumedvalues - actualvalues)^2 + thevarianceofthevalues \tag{3.1}$$

- The motion noise is the deviation of the selected and actual motion model. The motion noise tolerance specifies the tolerance of the Kalman filter for the deviation from the chosen model. This tolerance compensates for the difference between the object's actual motion and that of the model chosen.
- The measurement noise is the variance inaccuracy of the detected location. This is set to a high value, as the bounding boxes are noisy due to the non-maximum suppression and do not tend to move in a uniform way. The measurement noise also assists in rejecting false detections that were not suppressed in the detection stage.

3.8.2.2 Tuning Parameters

The optimal tuning parameters aid in the assignment of true tracks to detections, and thus maximize the number of accurate tracks while also minimizing the amount of track fragmentation. The tracking algorithm also applies a filtering layer to the detections, which rejects false tracks according to the parameter thresholds set.

Important tuning parameters include the following:

- The gating threshold is an important tuning parameter, as it determines the threshold to reject a candidate match between the detection and the track. It determines how close the predicted location must be to the detection location for it to be considered a true track. When the cost of matching the detected bounding box exceeds the threshold, the algorithm removes the association of the two bounding boxes from tracking consideration.

- The gating cost is a large value and is used for the assignment cost matrix. The assignment cost matrix enforces the rejection of candidate matches. This forces the weight-optimization step used to assign new tracks (discussed in section 5.2.4 on page 79) and to ignore some candidate matches. This is useful when removing unrealistic matches and helps with the filtering of false detections. This gating cost adds a weight to the assignment of tracks to bounding boxes that do not match the predicted location closely.

- The cost of non-assignment is a tuning parameter to control the likelihood of the creation of a new track. Setting this too low increases the likelihood of creating a new track, and may result in track fragmentation. Setting it too high may result in a single track corresponding to a series of separate moving objects. The algorithm uses the Munkres version of the Hungarian algorithm to compute an assignment that minimizes the total cost and becomes an optimization problem, as briefly discussed in section 2.6.2.1 on page 30.

Other general tuning parameters are as follows:

- The time-window size is the number of frames required to estimate the confidence of the track based on the detection scores in the previous frames.

- The age threshold is used to determine the minimum length for the track to become a true positive.

- The visibility threshold is used to determine the minimum percentage that the object is visible for the track to become a true positive.

These tuning parameters are set iteratively by evaluating the tracking results through empirical methods such as visual verification (section 3.9).

3.9 Tracking: Verification

3.9.1 Introduction

Figure 3.20 shows a methodology followed in the verification phase of the tracking stage.

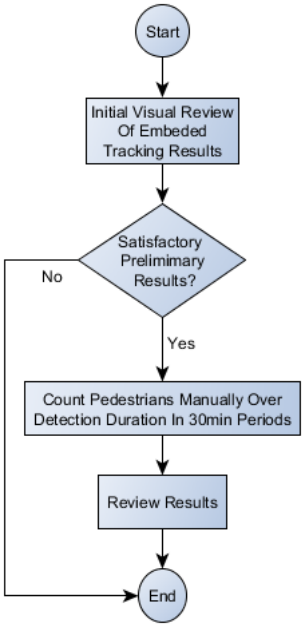


Figure 3.20: Tracking verification methodology

Figure 3.20 shows a methodology of the tracking-verification process. The embedded tracking results are initially briefly inspected. If the preliminary performance is satisfactory, a manual count of the pedestrian crossings is done in 30-minute periods in order to compare the results on a histogram over the entire day. If the preliminary tracking results are not satisfactory, or adjustments to the tracking performance must be made, tuning parameters in the tracking stage are adjusted (section 3.8)

3.9.2 Tracking Verification By Inspection

The tracking results are verified empirically through inspection by manually counting the pedestrians and comparing the number of crossings per time interval.

Although a number of persistent false detections remain due to shadows and environmental geometry, this does not greatly influence the counting results as the false detections created do not cross the counting line, or are not marked as a true track.

Initially the tracking evaluation is performed on smaller subsets of the source videos, as the impacts of most of the tuning parameters set in tracking stage are almost immediately apparent. This reduces the evaluation time, as the complete video set does not need to be parsed to the tracking stage until performance close to the desired level is achieved. The automated pedestrian count for each time period is collected and compared to the manual pedestrian count for each of the video subsets.

The direction information cannot be directly verified without tediously going through the source video and independently counting only pedestrians moving in one direction, thus only the absolute counted values are considered. The duration information will also be next to impossible to verify manually through inspection methods, but it is not critical as it only aids in giving additional information about pedestrian movement.

3.10 Tracking: Overview

Figure 3.21 shows a broad overview of the tracking stage as discussed in sections 3.7 to 3.9.

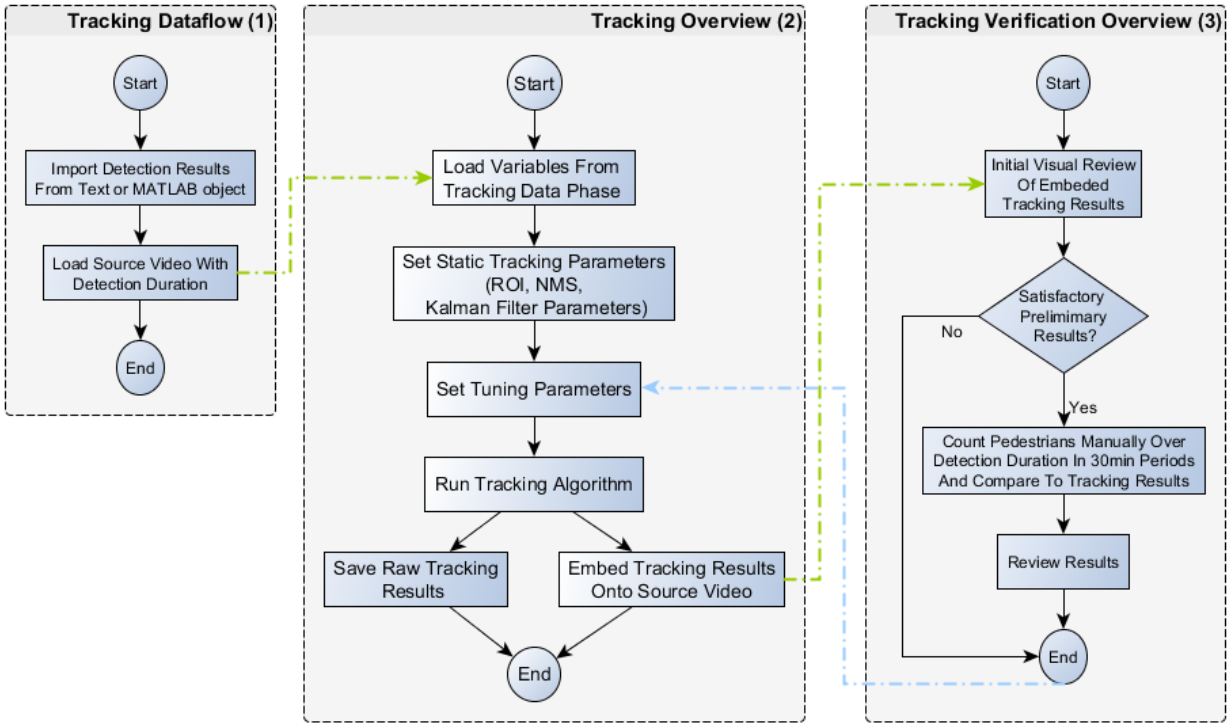


Figure 3.21: Tracking-stage overview

Figure 3.21 shows an overview of the tracking stage. The data phase interfaces with the detection stage (section 3.4) to load the detection results and source video. The variables are then parsed to the tracking phase. The embedded and raw tracking results are parsed to the verification phase, where the results are reviewed. If tracking performance is not satisfactory, adjustments to tuning parameters in the tracking phase (section 3.8) can be made following the evaluation of the tracking results.

3.11 Hardware And Software Setup

3.11.1 Software

The MATLAB toolbox for pedestrian detection by [13] is used, and includes the integral channel feature (ICF) algorithm [11], together with the adaptations made as discussed in 2.5 [27], [14], [26].

The toolbox includes an optimized variation of the AdaBoost algorithm for training, as well as a basic annotation tool. The toolbox was used as it provides all the tools needed for pedestrian detection, together with speed optimizations made over years of continuous improvement. The toolbox also includes added functionality to image manipulation, generating filters, as well as the complete work and detectors from [11] [14], [26], and [2].

The MATLAB parallel-computing toolbox is also used, since it dramatically speeds up training and detection times as multiple workers, or cores, can be used to perform simultaneous computations. This is mostly done in the form of a *parfor* (parallel for) loop, which breaks the iterator into multiple threads for simultaneous calculation of the *parfor* loop.

3.11.2 Hardware

The hardware used is an 8-core, 8-thread AMD CPU @ 4.3Ghz, 32GB DDR3 RAM @ 1866Mhz, a 500GB Samsung Evo Pro SSD, and an AMD R9 280x GPU. The detection and tracking programs are resource hungry, and care had to be taken to limit the lengths of the videos being processed to avoid running out of memory. The multi-core CPU enables MATLAB to distribute the parallel processes to up to 8 workers.

The addition of an SSD (solid-state drive) improved the run time of the detection and tracking programs by 15% as the programs require a constant stream of small data being written to the hard drive, e.g. detection results and tracking frames.

The programs are run on a powerful machine as there was little focus on programming efficiency.

3.12 Integrated Unit Overview

Figure 3.22 shows the overview of the complete integrated unit including the detection stage, section 3.2 to section 3.5, and the tracking stage, section 3.7 to section 3.9.

The detection stage starts at the data-collection phase, shown in the detection dataflow group, and ends at the verification phase, shown in the verification overview group. After the detection verification phase, all the detection results are parsed to the tracking stage.

The tracking stage starts at the tracking dataflow phase shown in the tracking dataflow group, and ends in the verification phase shown in the tracking verification overview.

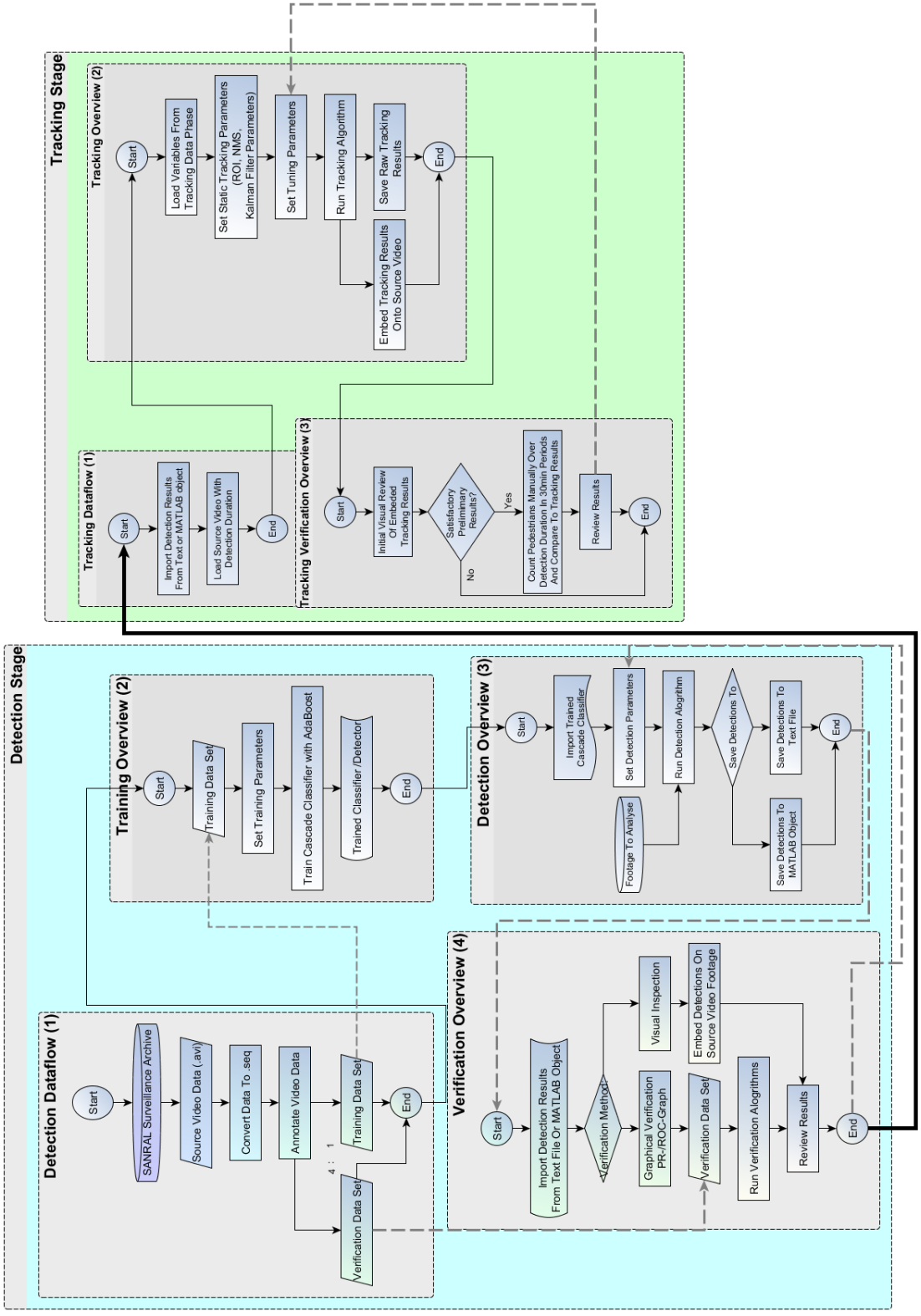


Figure 3.22: Integrated Unit Overview

3.13 Summary

This chapter presented the setup and methodology followed in developing a pedestrian-detection system comprising two distinct stages, namely the detection and tracking stages. Firstly the data phase of the detection stage was discussed. The methodology followed for collecting surveillance footage and the choice of pedestrian bridge as best-case scenario was presented. In section 3.2.2 it was demonstrated that classifiers need to be specifically trained for each camera, and thus it can be said that the classifiers are scene specific. The detection-phase section discussed the important parameters that must be set and presented an overview of the phase. The verification methodology used in the verification phase of the detection stage was presented and an overview of the detection stage was given.

For the tracking stage in the data phase, the data flow from the detection stage was presented. The tracking parameters of the tracking phase were discussed, as well as the Kalman filter parameters and the important tuning parameters associated with this phase. The section on the methodology of the verification phase of the tracking stage presented the methods by which the tracking results were evaluated and reviewed. Finally, a setup overview of the complete tracking stage was presented.

Lastly, the physical hardware and software setup was discussed and an overview of both stages as an integrated unit was presented and discussed.

Chapter 4

Implementation: Detection Stage

4.1 Introduction

This chapter discusses the implementation of the detection stage as described in section 3.2 - 3.5.

Section 4.2.1 discusses which static variables were implemented in the training stage and gives motivations for the choices. Section 4.2.2 discusses the effects on performance of the tuning parameters in the training phase. A summarized example of a log output after training is completed appears in section 4.2.3.

Section 4.3.1 provides an overview of the implementation of the detection phase with the help of a flow chart. The implementation of the static parameters in the detection phase is discussed in section 4.3.2. Section 4.3.3 discusses the choice of the non-maximum suppression method as well as the implemented parameters. Section 4.3.4 discusses the graphical user interface used in the detection stage as well as its functionality.

Section 4.4 discusses the verification methods used to evaluate classifier performance.

Section 4.5 discusses the obstacles faced in the detection stage.

Finally, section 4.6 presents the conclusion of this chapter and the implementation of the detection stage.

4.2 Training

4.2.1 Static Setup Parameters Used

The importance of good parameter choices is explored in [10], where parameter settings and their influences on detector performance are shown empirically.

Important static setup parameters (section 3.3.2) implemented in the training stage are:

- The minimum positive sample size. Objects smaller than this size will be ignored. Pedestrians are observed down to a minimum of of 24x18 pixels while annotating respective data sets.
- The positive sample size padding. The minimum model size with padding is suggested to have a ratio of 1:1.25 with the minimum model size from [11]. This equates to total padded sample sizes of 30x22 pixels each.
- The HOG and LUV channels are enabled, giving the classifier nine channels, six HOG channels, and three LUV channels. LUV is chosen as it is considered to be the most invariant to illumination effects. However, in practice this is still a problem at different times of day as previously discussed.
- The HOG descriptor is set to have six bins and apply soft-binning to edge cases. Soft-binning uses a Gaussian distribution to divide an edge case into the appropriate bin.
- The channel image pyramid is chosen to compute eight actual images per octave, and compute seven approximated images per octave as suggested in [2]. This parameter greatly impacts training speed as it takes longer to compute the actual pyramid scales than it does to approximate them.
- The use of a [4 5] filter as discussed in [27] is seen to have significant improvements on accuracy, with an increase in hit rate of $5\% - 36\%$. The default value of a [121]/4 (weak Guassian) filter is used as discussed in detail in [9].
- The number of positive samples is set to infinite, and the number of negative samples per image is set to 15. The number of negative samples per stage is set to 150 000, and the number of total negatives is set to 600 000. The total number of negative samples gathered is limited by the amount of memory/RAM in the machine. In this implementation a total of 600 000 negative samples was the upper limit for 32GB of RAM.

Other general static parameters include that the slight Gaussian smoothing algorithm from the MATLAB toolbox [13] be applied as an image pre-process.

The AdaBoost training algorithm is set to have 4096 weak classifiers and a tree depth of 3 (three cascade stages) as recommended by [2].

The number of negative samples collected from background images (not containing positive samples) can be set in detection. This has an impact on computer resources, as more negative samples increase memory usage.

The effects of including more negatives in training and of an increase in the minimum model size can be seen in figure 4.1.

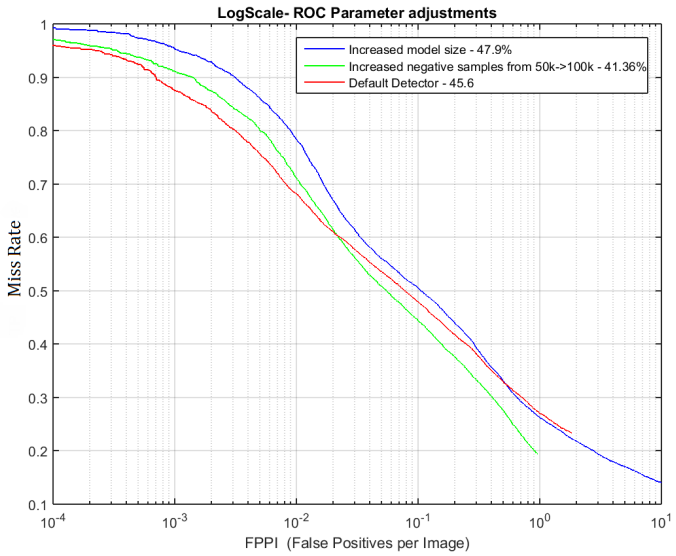


Figure 4.1: Log-scale ROC curve showing the impact of increasing the number of negative samples used in training as well as adjusting the model size

Figure 4.1 shows that a slight adjustment in model size or static parameters can have an impact on the performance of the detector, where a slight increase in expected model size increases the miss rate by 2%.

This implies that optimal parameters must be set to obtain optimal detector performance. Increasing the minimum model size means that some of the annotated positive samples in the training dataset will be ignored if their model size is smaller than the minimum model size. Ultimately this means that the training algorithm has fewer total positive samples to train a classifier, as some positive samples are ignored.

The figure also shows that the detector performance increases when the number of negative samples increases. The number of negatives reach a limit at which diminishing returns will be seen.

4.2.2 Training Tuning Parameters

The cascade calculation offset can be adjusted to increase the output scores of the classifier. This adds a constant offset to the weights of positive samples in the training stage. This makes the true positives even more positive (positive detections are given a higher score) and can have an effect on the detection performance where detection scores decrease below the threshold due to environmental factors such as shadows cast by infrastructure.

This was useful to adjust when working with the "half" detector, as the detection scores fell below the display threshold at certain times of day. Increasing the thresh-

old introduces more false positives, as the boundary between positive and negative samples was offset; thus false detections also receive a higher detection score.

The relative impact of the cascade calculation threshold on ROC and PR curves can be seen in figure 4.2 and 4.3.

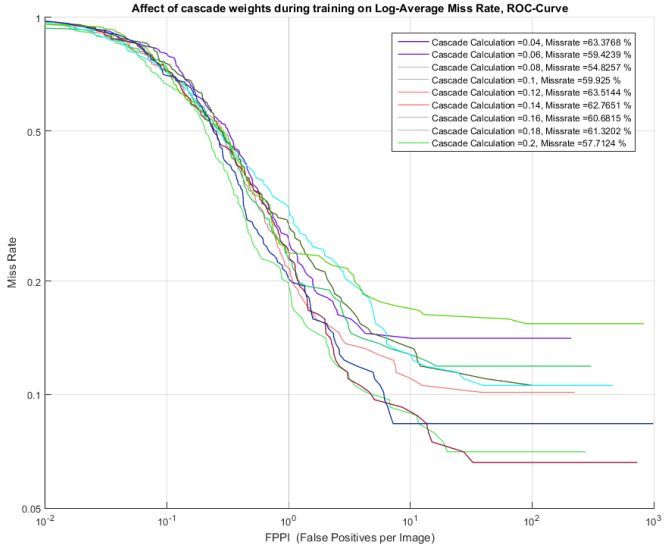


Figure 4.2: Log-scale ROC curve of different cascade calculation values

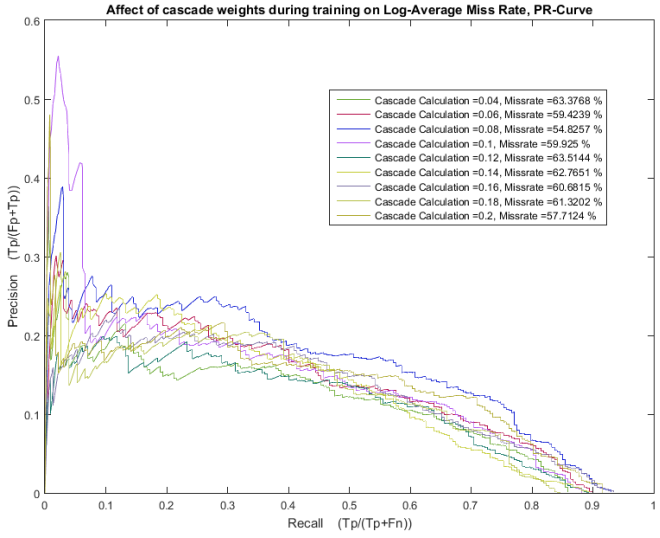


Figure 4.3: Log-scale PR curve of different cascade calculation values

From the figures it can be seen that the miss rate varies between a miss rate of 63% and 54%, which gives the correct choice of the cascade calculation offset an improvement in detection accuracy of 9% on the log-average miss rate.

Cascade calculation offset is set to 0.015 for the full pedestrians, and 0.05 for the half pedestrians. This is done as the detections in the half ROI generally had a lower detection score and had to be artificially boosted.

The detection sliding window stride length is chosen as two pixels, as this was seen to give the best performance in preliminary tests and by inspection. This has an impact on the run-time of the training algorithm, as it determines how densely the input image will be scanned by the detection sliding window. Due to the low resolution nature of the source footage, a low value for the sliding window is chosen.

4.2.3 Training Results

A summarized example of the log file output after the cascade classifier training has been completed is shown in table 4.1.

Table 4.1: Summarized example of the output log file after training has been completed

Stage:	nWindows	nImages	nWeak	nPositive	nNegative	Run-time
Stage 0 (Pos)	1086	17663	64	19548	150000	633s
Stage 0 (Neg)	150000	10048	64	19548	150000	633s
Stage 1 (Neg)	150000	10368	256	19548	300000	10457s
Stage 2 (Neg)	150000	13504	1024	19548	450000	2865s
Stage 3 (Neg)	150000	10368	4096	19548	600000	4038s

Referring to table 4.1, *nWindows* is the number of windows that are sampled from *nImages*. *nWeak* is the number of weak classifiers used to cascade in the stage. *nPositive* is the number of positive samples obtained from the training dataset. *nNegatives* is the number of negative samples collected from the training dataset, and *Run-time* is the total time in seconds it took to complete each cascade stage.

From the table it can be seen that the total run-time of the training phase is 17993 seconds, or about 5 hours. The training algorithm ultimately extracts 19548 positive samples and 600000 negative samples, or 150000 per stage from, on average, 11000 images in the training dataset.

The negative *nWindows* and *nNegative* are the same, as the algorithm only extracts a single feature from each negative sampling window, whereas positive windows have multiple features to extract.

4.3 Detection

4.3.1 Detection phase implementation overview

Figure 4.4 shows the implementation and general program flow of the detection phase.

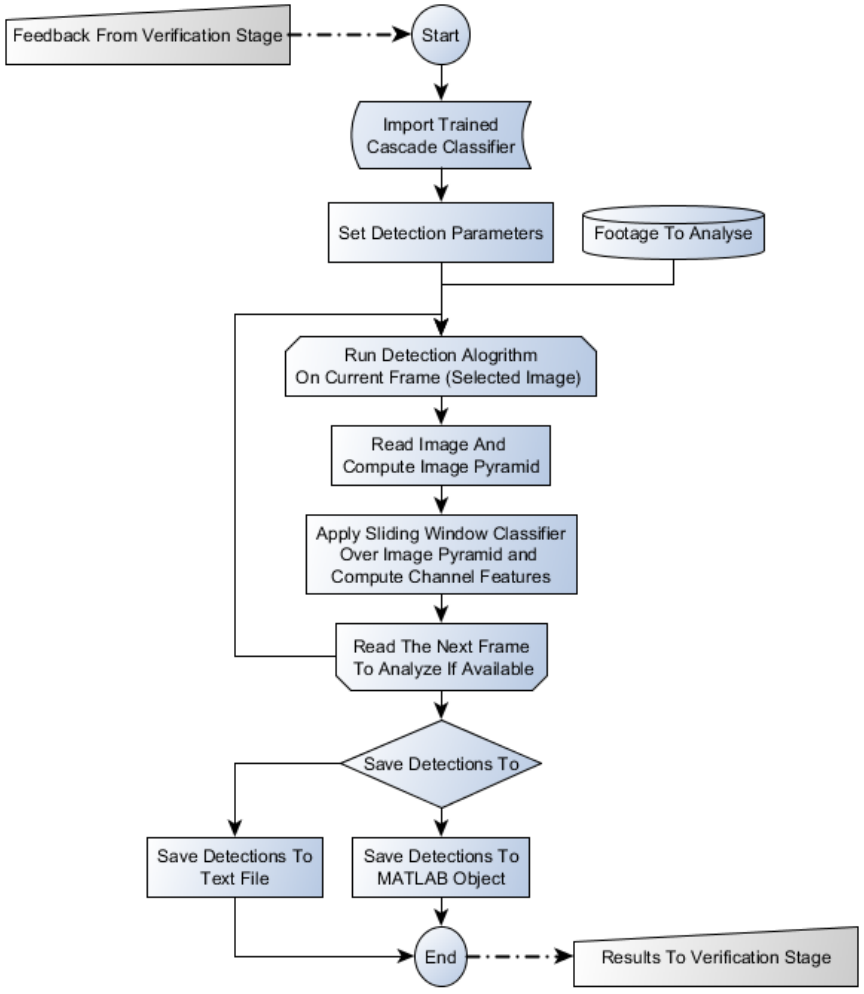


Figure 4.4: Training overview for the footage from camera 309

Referring to figure 4.4, the trained classifier is parsed from the training stage and detection parameters are initially set. The detection algorithm is then run on a frame-by-frame basis until no frames are left to read, signalling the end of the source footage being analyzed. The detection results are then saved to hard disk and parsed to the verification phase.

The detection phase interlocks with the training and verification phase of the detection stage. Feedback from the verification stage influences the detection parameters, e.g. NMS threshold and model size. The bounding box output results from the detection stage are also parsed to the verification stage for review and evaluation.

4.3.2 Static Detection Parameter Implementation

Important static detection parameters are:

- The average approximate positive sample size of pedestrians is observed and is set to 32x26 pixels.
- The same recommended padding ratio of 1:1.25 (as in 4.2.1, suggested by [2]) is used, which equates to total padded average sample sizes of 40x32 pixels.
- The detection window stride length in the detection phase is also set to 2 pixels – the same as the window stride length set in the training phase.

4.3.3 Choice Of Non-Maximum Suppression(NMS)

The initial evaluation of the different non-maximum suppression methods (section 2.2.6) on the verification data set is shown in figure 4.5.

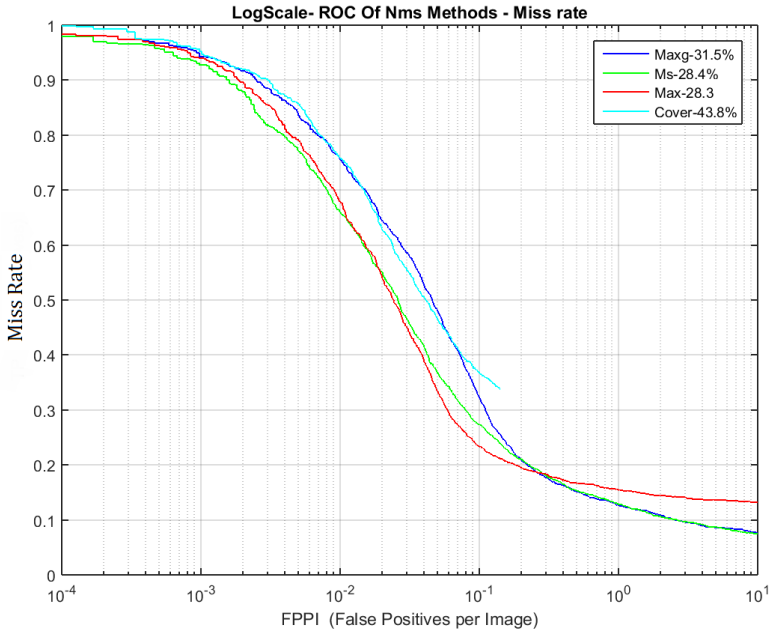


Figure 4.5: Log-scale ROC curve of the different non-maximum suppression methods

The figure shows that *max* has the highest true positive rate (lowest miss rate), with mean shift *ms* a close second. *Max* was chosen as it appeared to run the fastest when compared to mean shift *ms*, while also delivering the lowest miss rate observed.

From the figure it can also be seen that it is important to use the correct NMS method, as the miss rates differ by as much as 15%, from 43.8% for *cover* to 28.3% for *max*.

The minimum detection threshold value should be set according to the true positive detection scores observed, as any bounding boxes below the threshold will be ignored. This threshold is thus an important parameter to adjust, as it adjusts the number of false detections, thereby adjusting the FPPI (False Positive Per Image). Referring to a ROC graph, this means that if the NMS detection threshold is lowered, the classifier's working point will move more to the right (a higher FPPI means a lower miss rate).

This parameter must therefore be tuned to obtain satisfactory performance and a good balance between false and positive detections.

4.3.4 Functionality

4.3.4.1 Graphical User Interface

A sample from the graphical user interface with November 17 automatically loaded is shown in 4.6

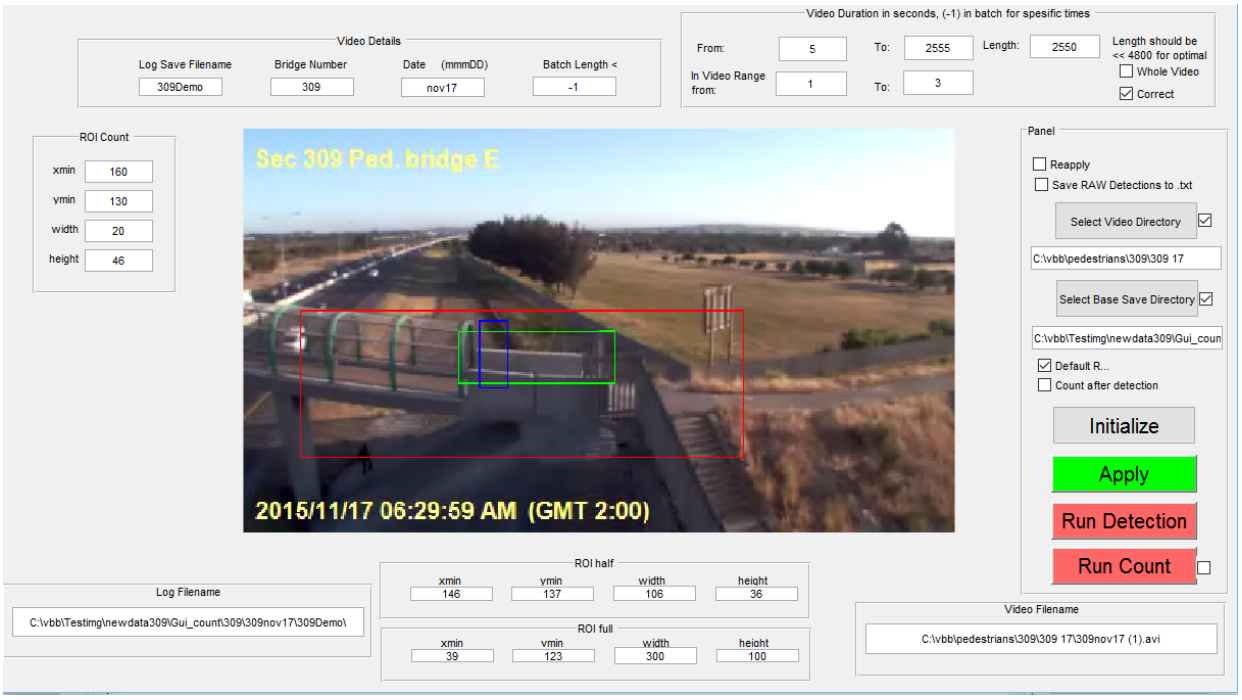


Figure 4.6: Sample from graphical user interface

4.3.4.2 Graphical User Interface Functionality

Making a Graphical User Interface (GUI) simplifies the workings of the detection process by removing the need to work through a command terminal, and thus removing the user from the complicated back-end.

Using the GUI entails the following:

- The bridge number and date can be entered; the interface will automatically load the first frame of the selected source video, or first frame of the first video if multiple videos are selected.
- Each video in the desired *Video Range* set can either be analyzed for a specific duration, or set to analyze the whole video if the *Whole Video* box is ticked. If the latter option is selected, the batch length must be set. This selects the duration of the chunks in which the videos will be analyzed.
- The interface will automatically tick the *correct* box if all information up to this point is correct.
- The interface can be set to reapply the current analysis, thus overwriting any information currently saved with the same *Log Save Filename* entered.
- The interface can also be set to save the detections to text file only, and to be loaded to MATLAB at a later time.
- The location of the source video as well as the desired save location of the results can be selected.
- The option to select the default ROI (*full*-pedestrians ROI, red, *half*-pedestrians ROI, green, and counting ROI, blue for *camera 309*) can be selected, or custom ROIs can be drawn directly onto the sample in the center of the GUI. The ROIs can also be drawn by entering the coordinates in the designated fields. It should be noted smaller *full* and *half* pedestrian ROIs decrease the detection area, and thus the total run-time decreases.
- Finally, an option can be set to automatically run the counting program after detection is completed. This allows the GUI to run completely automated from detection stage to tracking stage, producing results for an entire day's footage without user interaction.
- The push buttons(*Initialize, Apply, Run Detection, Run Count*) will automatically change from red to green if everything in the preceding steps was correct.
- After initialization, all the information entered is assumed to be correct when pressing *Apply*. This parses all the variables entered to MATLAB.

- If all the information entered up to *Apply* is correct, the detection phase will begin and the detector will run.
- If footage had previously been analyzed, the *Run Count* push button can be pressed to initialize and run the tracking stage instead.

4.4 Verification

Verification must be done in a number of ways to ensure that it is accurate and representative of the complete dataset.

4.4.1 Plotting PR/ROC-Graphs

One way to perform an initial verification and to help set tuning parameters is to plot PR/ROC graphs as discussed in section 2.3. These graphical methods help to quantify the effects of the parameters on the detector output.

The graphs also assist in setting the tuning parameters in a brute-force manner by running the detection process multiple times and incrementally adjusting the parameters. The different results are then plotted on the same axis, as shown in figure 4.2 from section 4.2.2 on page 65.

The challenges to using the graphical verification methods (some of which have been previously discussed) are as follows:

- The results shown are not always a good representation of the actual detector outputs. This is because the verification data set only covers a fraction of all the detection scenarios (only a few hours out of the whole day are annotated).
- The quality of the annotations or the ground truths can also influence the detection results, as misannotated objects would result in a false detection and thus increase the detector's miss rate.
- The log-average miss rates in these graphs are also plotted over a spectrum, e.g. false positives per image in a ROC graph, where practical implementation sets this value by means of a non-maximum suppression (NMS) threshold, among others.

4.4.2 Visual Verification

Visual verification is required for the final analysis of the detection output. For this the detection bounding box output and confidence scores are embedded in the source frames. These frames are then converted to an .AVI video file to be reviewed. This is useful as it shows whether the NMS threshold is set too high or too low (either too

many false positives or pedestrians not being detected). The visual verification also indicates if adjustments need to be made in the training stage.

This is a laborious task and is only performed after the initial evaluation is completed and parameters have been set with the help of PR/ROC graphs.

4.5 Challenges

Creating a pedestrian-detection system using existing low-resolution surveillance camera footage produced a number of challenges.

The challenges that occurred during the detection stage are listed below.

- Adjustments to the tuning parameters in the training phase are not mutually exclusive. This means that adjusting parameters in this phase may have an impact on performance in the detection phase, and vice versa. For example, adjusting the cascade threshold offsets the detection/confidence scores, and therefore the NMS threshold also needs to be offset.

- Adjustments to the camera resolution, aspect ratio, frame-rate, and camera orientation can be undertaken by the governing authorities and would result in re-tuning some of the parameters or, in the worst case, retraining the detector/-classifier.

- The training set requires data from throughout the day, and this can be a time consuming process to collect. This is due to the variation in illumination caused by the sun over the period of a day, which means that the footage will have shadows cast on different locations resulting in false positives. The change in the orientation of the sun also means that the footage contains different saturation and warmth values, from cool in the morning to warm at midday. This was partly overcome by means of the LUV-colour space, which is partly invariant in respect of illumination effects such as saturation and brightness.

- Parameters for each phase need to be set for each specific camera. This means that each camera's detector need to be trained and tuned for each specific setup. This is one of the limiting factors to using the low-resolution surveillance footage from the existing infrastructure.

- The optimal operating point for the tuning parameters in the detection stage can only be a obtained once the tracking stage is realized, as there are some interactions between the stages, and the tracking stage gives quantitative results in the form of pedestrian count numbers that can be compared to manually verified results.

4.6 Summary

This chapter presented and discussed the implementation of the detection stage. The discussion of the training phase presented the important parameters used, and a summary of the terminal output after the training of a classifier was completed. The discussion of the detection phase included a detailed overview of the implementation of this phase, as well as the static parameters used. The choice of NMS methods was discussed, and *Max-NMS* was chosen as the most suitable method for the detection phase. The graphical user interface for the detection stage was presented and its functionality was discussed. The verification methods implemented in the verification phase were discussed, and lastly the challenges faced during the implementation of the detection phase were presented.

Chapter 5

Implementation: Tracking Stage

5.1 Introduction

This chapter discusses the implementation of the tracking stage as previously discussed in section 3.7 - 3.9.

Section 5.2.1.1 discusses the implementation of tuning parameters used in the tracking phase. Section 5.2.1.2 motivates the parameter values used in the implementation of the Kalman filter. Section 5.2.2 discusses the importance of correctly placing the counting region of interest (ROI). Section 5.2.3 describes how variables are parsed to the tracking stage from the detection stage, or hard disk. Section 5.2.4 describes how the implemented function assigns tracks to detections. The implementation of various other track-management functions is discussed in section 5.2.5. Section 5.2.6 discusses the counting algorithm used to count tracked pedestrians. Section 5.2.7 presents an example of the structure when working with the track and counted variables. The tracking results that are displayed and saved to hard disk are discussed in section 5.2.8.

Section 5.3 presents and discusses the tracking verification graphical user interface and section 5.3.1 discusses the functionality.

Section 5.4 discusses the challenges faced in the implementation of the tracking stage. Lastly, a summary of the work in this chapter is given in section 5.5.

5.2 Tracking

5.2.1 Setup Parameter Implementation

5.2.1.1 Tuning Parameters

The tuning parameters are dependent on the camera footage used, as pedestrian motion and environmental conditions differ from camera to camera – e.g. the number of frames in which a pedestrian can be seen unoccluded, or the angle at which the camera views the pedestrians.

Important tuning parameters (discussed in section 3.8.2.2 on page 56) implemented in the tracking stage are:

- The bounding box overlap parameter, or gating threshold, was chosen as a large value, i.e. 0.99 or 99%. This allows bounding boxes to easily be assigned to a track/predicted location. A large value is chosen, as pedestrians crossing the boundary between the *full* and *half* regions cause a jump in the centroid and hence a mismatch in the predicted centroid. This in turn can cause the track to be lost. The jump in centroid is a result of the *full* pedestrian model (34x16 pixels) having a centroid at 17x8 pixels, and the *half* pedestrian model (17x17 pixels) having a centroid at 8x8 pixels. Thus a horizontal jump of 11 pixels occurs when the pedestrian centroid moves over the boundary between the two regions.

- The gating cost used in conjunction with the track-assignment optimization was tuned to 780, a value that is observed to deliver the best tracking results. A large value helps reduce the number of false positives in the detection stage.

- The cost of the non-assignment parameter is set to 12. This low cost helps reduce the creation of new tracks but strikes a balance so that multiple detections are not assigned to the same track.

- The time window size is set to 4 frames, as this allows the tracks sufficient time to stabilize, but not so much that the detection is only assigned to a track after the pedestrian has crossed the counting line.

- The confidence threshold is tuned to 60. This is set as most of the accurate detection scores are observed to be higher than 60, thus rejecting detections below the threshold.

- The age threshold is set to 3 frames, and thus the required minimum age of a track to be marked as a true positive is 3 frames.

- The visibility threshold is set to 70%. This means that the object should at least be visible on 70% of the 4 frames allowed in the time window size.

The optimal tuning parameters in the tracking stage are obtained by iteratively running the tracking algorithm, visually evaluating the results, and adjusting the parameters accordingly.

5.2.1.2 Kalman Filter Parameters

The Kalman filter parameters (as discussed in section 2.6.2.1 on page 30 and section 3.8.2.1 on page 55) are tuned for optimal performance, as they are important in the prediction of detection locations in the frames that follow.

The Kalman filter parameters implemented are as follows:

- The error variance is tuned to [3,3]. This means that the Kalman filter can expect the bounding box to be located within three pixels in both the x- and y-axis of the predicted location.
- The motion noise is chosen to be [8,8]. This is observed to give the best results, indicating that the chosen model differs from the actual model. The difference could be due to pedestrians moving non-linearly. For *camera 309*, the pedestrians move in a vertical manner when walking up the stairs, and then move horizontally as they cross the bridge.
- Lastly the measurement noise is chosen as a large value, 45, as the bounding box output from the detection stage seems to contain discontinuous jumps even though averaging effects are applied. Choosing a large value for the measurement noise will help to predict the centroid position within a large tolerance. This ultimately helps to improve the pedestrian track life, and less track fragmentation occurs.

These Kalman filter parameters are observed to give the best tracking results when compared to the manually counted ground truth as empirically determined.

5.2.2 Defining The Count Line or ROI

Defining the counting line or ROI is an important step in improving and maintaining tracking accuracy. The position of the ROI is important, as the tracking results show that a total accuracy difference of as much as 15%-20% can be seen when pedestrians move in certain directions. An accuracy difference is seen as some ROI positions do not allow for the minimum age and visibility requirements (section 5.2.1.1) to be met due to infrastructure occlusions or other environmental factors.

Figure 5.1 shows a sample from 11 November on *camera 309*. The red ROI indicates the detection area of the *full* pedestrian detector as previously discussed. The green ROI indicates the *half* pedestrian detector, and the blue ROI indicates the counting line, or counting ROI.

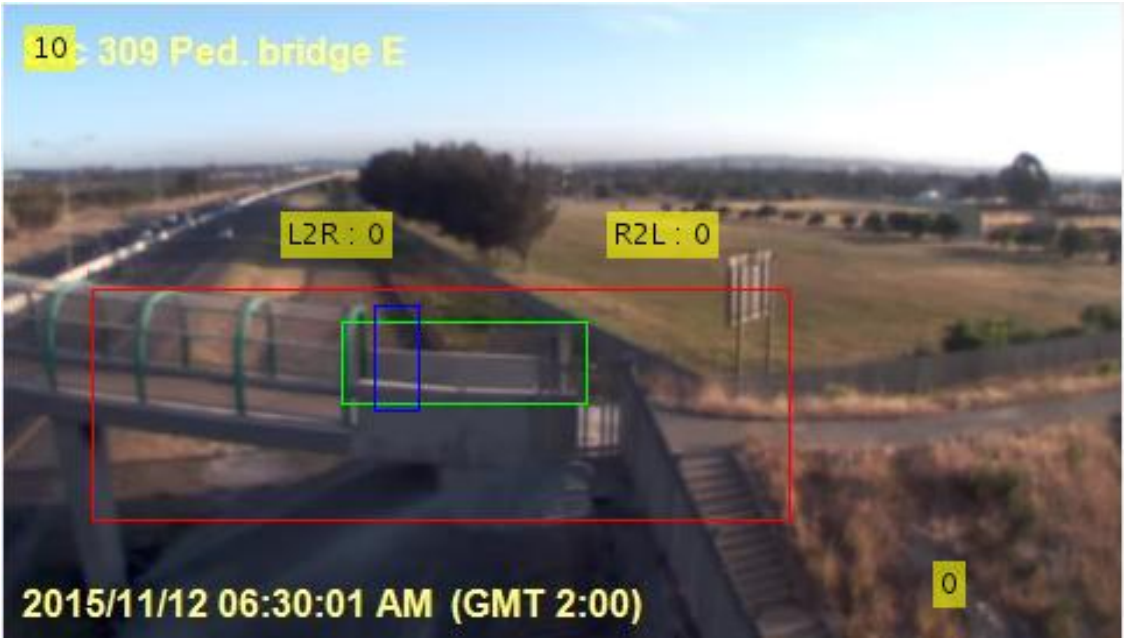


Figure 5.1: Sample showing different ROIs on *camera 309*

As an example, and referring to figure 5.1:

It is observed that moving the counting ROI closer towards the boundary of the green ROI results in the window size and age parameters of the track being below the threshold. Being below the threshold means that the track is unable to be classified as a true positive or true track, and thus the track is missed. The lack of visible frames is due to pedestrians only appearing in the detection region briefly before they pass the counting ROI. The sparsity of available track information means that the Kalman filter does not have time to stabilize the track predictions.

Moving the ROI far to the left shows a loss in the "left to right" count , and moving it far to the right shows a loss in the "right to left" count.

It is thus recommended that the counting line be placed at different logical locations and the accuracy over a subset be compared in order to achieve optimal results.

5.2.3 Parsing Variables From The Detection Stage

Parsing the stored variables from the detection stage or hard disk is challenging, as it is necessary to allow for the parsing of fragmented detections. This is a result of the detection user interface allowing for the detection of specific times as well as for a complete day, or a set of videos. Being able to use fragmented detections allows the tracking stage to be run independently after the detection stage has been completed.

This is accomplished by traversing the selected folders indicated by the user and searching for the needed MATLAB variables containing the bounding-box outputs. The

bounding-box outputs are then sorted and concatenated to form a single variable that corresponds to the source footage that is being analyzed.

The tracking algorithm also allows the loading of fragmented pedestrian-count variables so that the displayed pedestrian counts in each embedded frame are continuous even if the tracking of the same footage was previously interrupted. It also allows for the fragmented writing of analyzed frames, so as not to start from the beginning each time the same video footage is interrupted or stopped. This was mostly done for debugging purposes as it was sometimes useful to do the analysis in small batches and verify the results incrementally.

5.2.4 Assigning Tracks To Detections

Assigning tracks to detected pedestrian bounding boxes is done by optimization, which seeks to minimize a cost function. The cost is computed using the bounding-box overlap parameter, as discussed in 5.2.1.1, together with the bounding-box overlap-ratio function, and the physical overlap between predicted and actual bounding boxes.

A flow diagram presenting the assignment of tracks to detections is shown in figure 5.2 below.

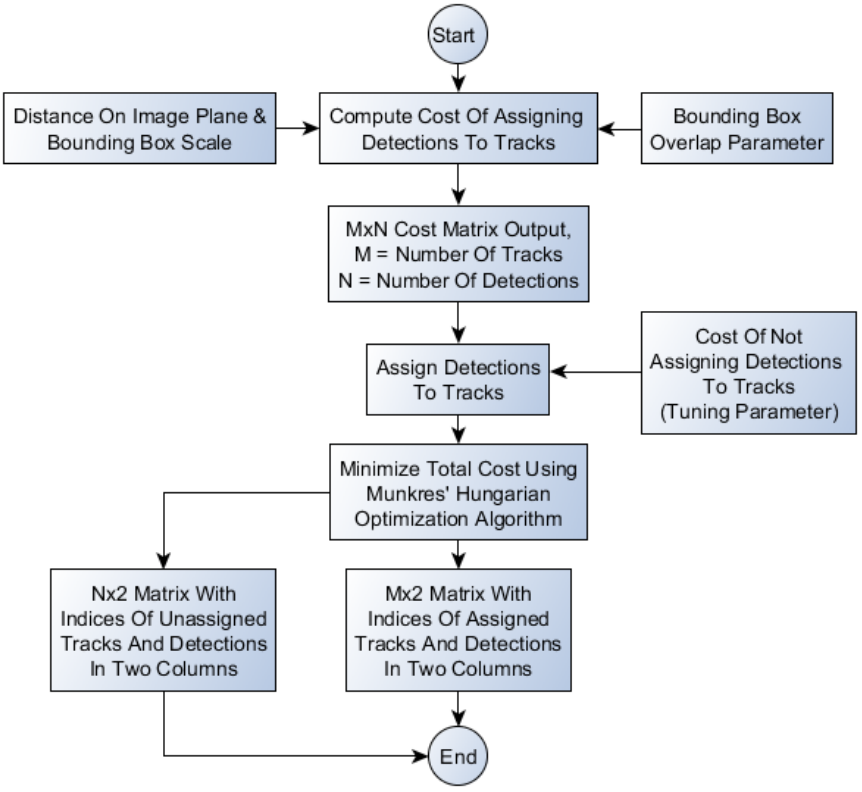


Figure 5.2: Flow diagram of assigning detections to tracks

Figure 5.2 shows how a track is assigned to a pedestrian bounding box obtained from the detection stage. The track assignment algorithm involves two steps as can be seen from the MATLAB multi-object tracking example, referring to figure 5.2:

The **first step** is to compute the cost of assigning each pedestrian-detected bounding box to each track using the *bounding box overlap* parameter as a measure. As people move across the pedestrian bridges, their motion will not be accurately described by the centroid point alone.

The cost takes into account the distance on the image plane as well as the scale of the bounding box. This helps to prevent the assignment of tracks far away from the camera to detections close to the camera, even if their centroids coincide.

The choice of this cost function will ease the computation without resorting to a more sophisticated dynamic model. The results are stored in a MxN matrix, where M is the number of tracks and N is the number of detections.

The **second step** is to solve the assignment problem represented by the cost matrix using the *assign-detections-to-tracks* function. This function takes into account the cost matrix and the *cost-of-not-assigning-any-detections-to-a-track*.

The value of the *cost-of-not-assigning-a-detection-to-a-track* parameter depends on the range of values returned by the cost function. The parameter value must be tuned experimentally as discussed in 3.8.2.2 on page 56 and 5.2.1.1 on page 75.

The *assign-detections-to-tracks* function uses Munkres' version of the Hungarian optimization algorithm to compute an assignment that minimizes the total cost. The function returns a Kx2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of the tracks and detections that remained unassigned in a Jx2 matrix.

A discussion on the Hungarian optimization algorithm and alterations made by Munkres is beyond the scope of this thesis as discussed in section 1.7.

5.2.5 Managing Tracks

Managing the tracks is an important step in the tracking phase, as assigned and unassigned tracks need to be updated, newly assigned, or removed.

Important track management functions include:

- *Updating assigned tracks* and using the update-assigned-tracks function updates each assigned track with the corresponding detection. It calls the "correct" method inside the Kalman filter toolbox to correct the location prediction in the current frame after a match between detection and track has been made. Next, the function computes the new averaged bounding-box location by taking the average sizes of the recent (up to four) bounding boxes. The function then increases the age and total visible count of the assigned track by one. Finally the function adjusts the confidence score for the track based on the previous detection scores.

- *Updating the unassigned tracks* and using the update-unassigned-tracks function marks each unassigned track as invisible, increases its age by one, and matches the predicted bounding box to the track. The confidence is set to zero since it is not known why the detection was not assigned to a track.

- *Deleting lost tracks* and using the delete-lost-tracks function deletes tracks that have been invisible for too many consecutive frames. It also deletes recently created tracks that exceeded the total invisibility threshold as set by the invisible/age threshold as discussed in section 3.8.2.2 on page 56 and 5.2.1.1 on page 75. Noisy detections result in the creation of false tracks as false positives are assigned tracks. To combat this, tracks are removed if:
  - The object was tracked for a short time. This typically happens when a false detection shows up for a few frames and a track is initiated for it.
  - The track was marked invisible for most of the frames.
  - The track failed to receive a strong detection score within the last few frames, which is expressed as the maximum confidence score.

- *Creating new tracks* and using the create-new-tracks function creates new tracks from unassigned detections, assuming that any detection is the start of a new track.

These functions use the tuning parameters implemented in section 5.2.1.1 on page 75 and are run each time a new frame from the source video feed is analyzed.

5.2.6 Counting Algorithm

The counting algorithm is implemented to quantify the pedestrian-tracking data into pedestrian-crossing count data. In *camera 309* shown in figure 5.3 for example, pedestrians move horizontally across the pedestrian bridge, thus their centroid's x-axis value is considered an indication of movement and direction.

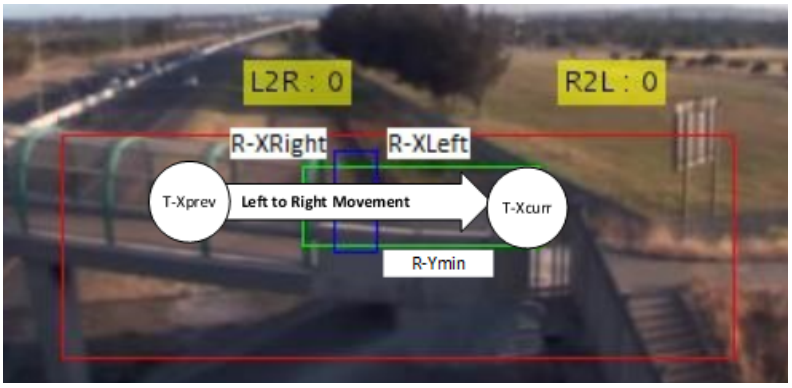


Figure 5.3: Example showing centroid moving from left to right in *camera 309*

Figure 5.4 shows the implementation of the counting algorithm based on the detection and tracking results.

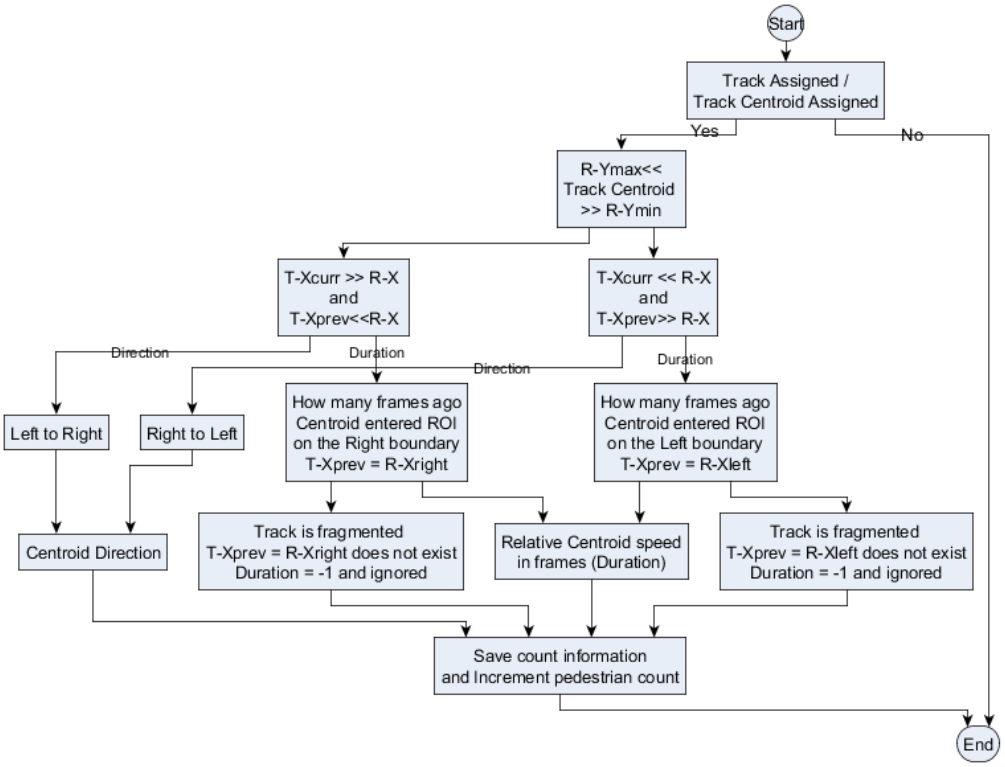


Figure 5.4: Flow chart of the implementation of the counting algorithm

The counting algorithm first checks whether there are tracks assigned and whether there are track centroids assigned. The algorithm then scans each assigned track, and evaluates each respective centroid's current and past positions.

Tracks are only considered if their centroid's y-value falls within the ROI y-range (*R-Ymin* to *R-Ymax*). If it is found that the centroid's current x-axis position (*T-Xcurr*) is larger than the x-value of the counting ROI (*R-X*), and that some historical value of the track's centroid (T-Xprev) is smaller than *R-X*, it can be said that the pedestrian is moving from left to right over the frame. The same process is applied when working with pedestrians from right to left, but instead *T-Xcurr* should be smaller than *R-X*, and *T-Xprev* larger.

The ROI is rectangular in form as it is used to obtain the relative speeds of pedestrians. Pedestrians moving from left to right, for example, first cross the ROI at the left boundary of the rectangle. Entering the ROI increments a counter to show that there are currently centroids inside the ROI. Only when a track leaves the rectangle, e.g. the right boundary of the ROI, and the counter is decremented, are the count variable and direction recorded. It is thus useful to use the centroid's direction of movement to determine if it is entering or exiting the ROI.

If a track leaves the counting ROI, the algorithm scans through the centroid history to determine how many frames ago the track crossed the entering ROI. This is used

to obtain the relative speed of pedestrian movement in frames. If the track leaves the ROI and it is found that the track did not enter the ROI, this could be due to track fragmentation, and the duration is ignored and set to a (-1) flag.

The pedestrian count from respective directions is incremented separately to give an indication of the pedestrian traffic in either direction. Once a pedestrian enters the ROI, the involved side of the ROI rectangle will flash bright green for a few frames to indicate a track has entered. When a pedestrian exits the ROI, the corresponding ROI side will flash bright red to indicate a track has left the ROI and was counted.

5.2.7 Track and Count Variable Structure

The tracking variable structure holds all the information about the current and predicted location, and the position and confidence (detection) score of the detected bounding. The tracking variable also holds information about the track, such as the total visible count and age of the track.

This can be seen in table 5.1, which shows the structure of the tracking variable.

Table 5.1: The tracking variable structure

Track Variable Field	Description
Id	The track's id identifier.
Color	The color used to draw the bounding box.
Scores	The detection scores from the detection stage.
Kalman Filter	The Kalman filter model and parameters.
Age	The current age of the track.
Total Visible Count	The total number of frames the track was visible.
Confidence	The average of the past detection scores.
Predicted Position	The location predicted by the Kalman filter.

The pedestrian-count information is saved to a dense MATLAB variable with the structure shown in table 5.2.

Table 5.2: The counting variable structure

Count Variable Field	Description
Direction	The direction the pedestrian centroid is moving.
Duration	The number of frames it takes to cross the count ROI.
FrameTime	The frame number the count occurred.
Countnr	The total number of pedestrian crossings

The track variable is updated each time a track management function changes or updates any tracks. This is done in almost every frame that is being analyzed.

The count variable is updated each time a new pedestrian crossing is detected and the pedestrian count variable is incremented.

Creating complex structures to hold the tracking and counting information assists in setting a standard so that fragmented variables can easily be loaded, or exported without compatibility problems.

5.2.8 Tracking Stage Output

The tracking results are displayed and embedded in the source footage being analyzed and displayed to the user, together with indications of the current pedestrian count in respective directions, as well as the current frame number being analyzed.

It was necessary to redraw/recreate/clear the figure displaying the tracking results, as in practice it seems that each frame is drawn on top of the previous one, thus making the display of the current frame significantly longer the longer the tracking algorithm runs.

For example, it would initially take 60ms to display a frame, but after 250 000 frames it took almost 1.2s to display each frame. This represents significantly more computational time for the tracking and detection stage per frame.

The tracking results are saved to the hard drive after the tracking phase is completed. The results are saved in the form of multiple images corresponding to the analyzed and tracked frames. This can be converted to an .AVI video to be reviewed quickly at a later stage. The frames are saved at the end of the tracking phase and not in real time, as this allows for the use of parallel processing and accelerates the writing of a large quantity of frames to the hard drive.

5.3 Tracking Verification Graphical User Interface (GUI)

A graphical user interface was created to easily review the MATLAB variables containing the tracking output. A sample from the tracking user interface with the results from 16 November loaded is shown below in figure 5.5.

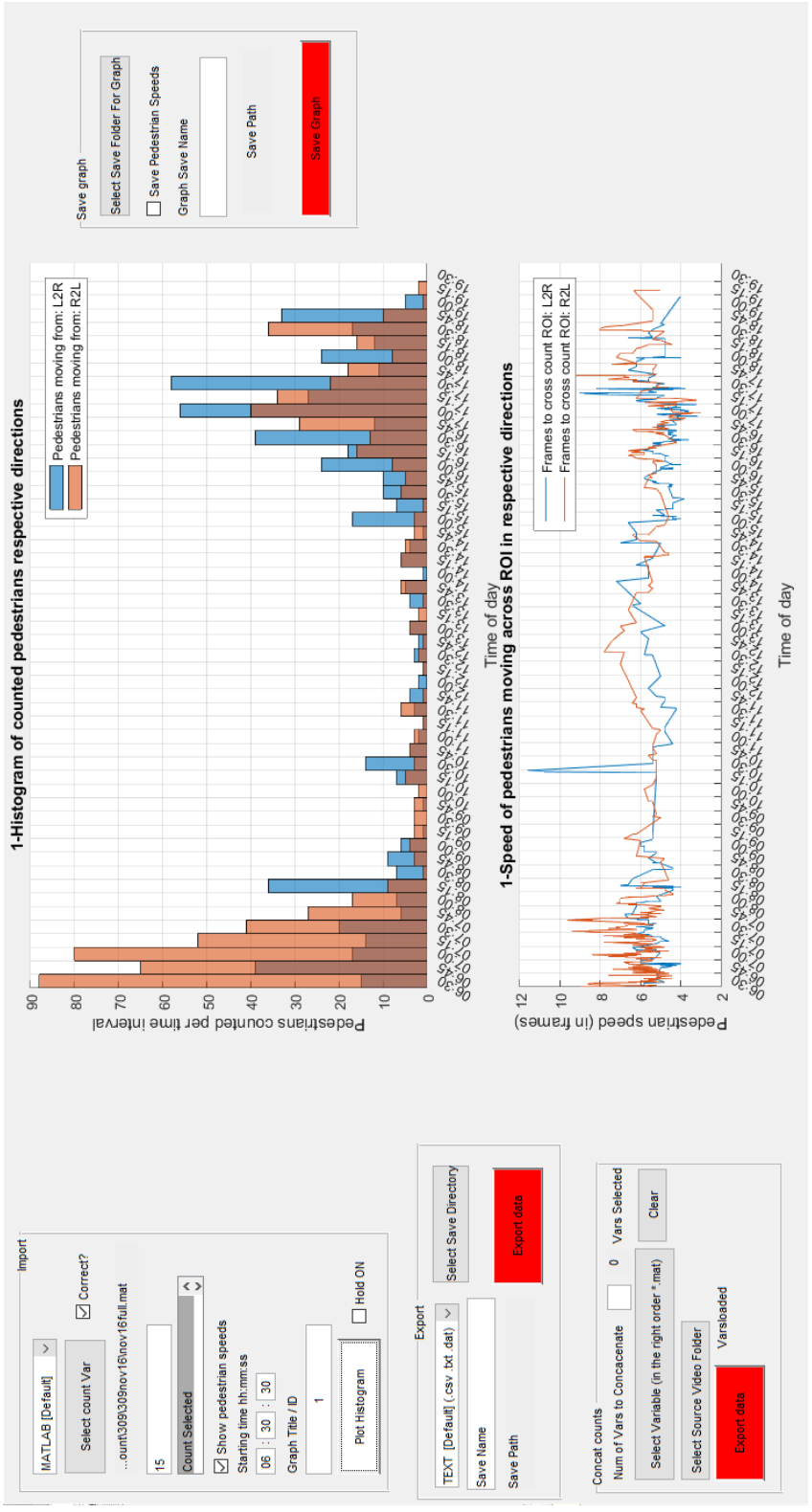


Figure 5.5: Sample showing the tracking verification user interface

The figure shows a histogram of counted pedestrians in both directions in the middle of the figure, with blue indicating pedestrians moving from *L2R*, and orange indicating pedestrians moving from *R2L* overlaid on the above mentioned plot. Thus the brown colour displayed shows the blue histogram behind the orange histogram (the overlap of R2L on L2R). The relative pedestrian speeds, in frames, is shown below the counting histogram.

The time period is set from 06:30am in 15-minute intervals. The import panel is at the top-left, below that is the panel to export the MATLAB variable, and at the bottom-left is the panel to concatenate fragmented tracking results. Lastly, the panel to export the figures to a desired image format appears at the top-right.

5.3.1 Verification GUI Functionality

The verification GUI is used to evaluate the tracking results and has a number of functions. It is used to simplify the interaction with the pedestrian-count data, and to remove the user from the complicated back-end.

5.3.1.1 Histogram Plot Settings

The count histogram in the verification user interface can be tuned and adjusted inside the interface. The type of plot can be selected, from a normal count distribution to cumulative count, probability, and probability density functions.

The x-axis intervals can be set, as well as the starting time. The graph can also be independently named to distinguish the exported graphs.

Lastly, the interface can be set to hold the current plot on the axis. This is similar to the "hold on" command in MATLAB, where the current graph is held as is, with a new graph being drawn on top of the previous one in order to compare results.

5.3.1.2 Conversion Of Output Data To Multiple Formats (.csv; .xls)

The MATLAB variable containing the tracking results can be exported to multiple formats to be reviewed by an external program, such as Microsoft Excel.

This is done by converting the MATLAB variable to a table format, and then writing the table to the desired format, be it to plain text (*.txt), comma separated values (*.csv), or specialized worksheet formats (*.xls), among others.

Exporting the variable to multiple formats enables multi-disciplinary entities to analyze and research the results without knowledge about MATLAB.

5.3.1.3 Save Histogram To Multiple Formats (.png; .fig)

The verification user interface allows the user to save the graphs plotted inside the user interface to a variety of image formats including .PNG, or .JPG, or as a MATLAB figure, .FIG . The plots are saved to the file path entered by the user in the export panel.

The relative pedestrian speeds can also be saved to the hard disk together with the histogram if the option is selected. The histogram- and speed-plots share the same axis intervals and labels to allow for easy cross-referencing between both plots.

5.4 Challenges

Working with existing low-resolution surveillance cameras not optimized for pedestrian tracking resulted in challenging situations. Challenges that occurred during the tracking stage were as follows:

- It was critical to get the counting line/ROI in the right position to maximize the counting accuracy.
- It was challenging to adjust the tuning parameters to their optimal value, as parameters in the detection stage had an influence on the tracking tuning parameters. This meant that if parameters in the detection stage were adjusted, it was likely that parameters in the tracking stage also needed to be adjusted.
- For *camera 309*, it was difficult at first to get the tracking algorithm to keep the detections correctly tracked when pedestrians were briefly occluded. One major challenge was to prevent the tracks getting lost/fragmented as pedestrians moved from the *full*-pedestrian region into the *half*-pedestrian region, as discussed in previous sections.
- Some consideration had to be given to run-time optimization as many frames had to be tracked and displayed, and a few fractions of a second saved on each frame would result in significant decreases in total run-time.
- Lastly it was a challenge to integrate the two stages, to allow for the parsing of fragmented detection variables, and to allow the continuation of tracking from an interrupted tracking variable. This forced the incorporation of many fail-safes and other trial-and-error methods.

The challenge of getting the ROI/counting line in the optimal position was overcome by iteratively adjusting the position of the counting line and noting the counting results of a subset of frames at different times of day. The obtimal position of the ROI was determined by the maximum pedestrian count in both respective directions.

The challenge of adjusting the tuning parameters was overcome by bruteforce, and required multiple iterations of parameter values to be run and the results evalutated over all the phases and stages.

The challenge of tracks being lost over a occluded region (transition between *half* and *full* region) was mostly overcome by adjusting the overlapping threshold so that even a slight overlap in bounding boxes would result in a candidate match.

Lastly the challenge of integrating the two distinct stages was overcome by incorporating the necessary protocols and variable formats, as well as including fail-safes and trail-and-error parsing methods.

5.5 Summary

This chapter discussed the implementation of the tracking stage. The important parameters used, including the Kalman filter and tuning parameters, were discussed. The importance of a well-placed counting line together with the method implemented to handle the parsing of variables between phases and stages were also discussed. A detailed discussion was provided on the implementation of a function that assigns detections to tracks. Other track-management functions were also discussed. Some of these functions included updating tracks, or deleting lost tracks.

The implementation of the counting algorithm was discussed, together with an example of the track and count variables. The output of the tracking stage, and the implementation of a verification graphical user interface to interpret the counting results, were presented. The functionality added by the verification graphical user interface was also presented. Lastly, challenges faced in the implementation of the tracking stage were presented and discussed.

Chapter 6

Results

6.1 Introduction

This chapter presents the results of the detection, tracking, and counting system developed in the preceding chapters.

Section 6.2 discusses the detection results obtained from the detection stage. Section 6.3 presents the tracking validation results when compared to manually counted ground-truth data. The results from the counting system are then presented and discussed in section 6.4. The peak hours of pedestrian activity in the morning and afternoon are discussed in section 6.5. Lastly, a summary of the results is discussed in section 6.7.

6.2 Detection Results

The detection results can be represented by an ROC-/PR-curve to quantify the accuracy (hit rate) of the detector before any tracking is applied.

Figure 6.1 shows an ROC-curve of the detector applied to camera 309. The miss rate of the detector at various intervals used to compute the log-average miss rate is shown on the graph. As a reminder, a good ROC-curve tends to the lower-left corner where the miss rate and FPPI are at a minimum.

The figure shows that the detector obtained a good miss rate of only 28%, meaning only 28% of all the detections on every frame were missed. This is competitive when compared to other modern detectors as discussed in section 2.5. With the addition of the tracking algorithm, it is unnecessary for each pedestrian to be detected on every single frame. The miss rate improved by as much as 15% by adjusting the tuning parameters to optimal values in the detection stage.

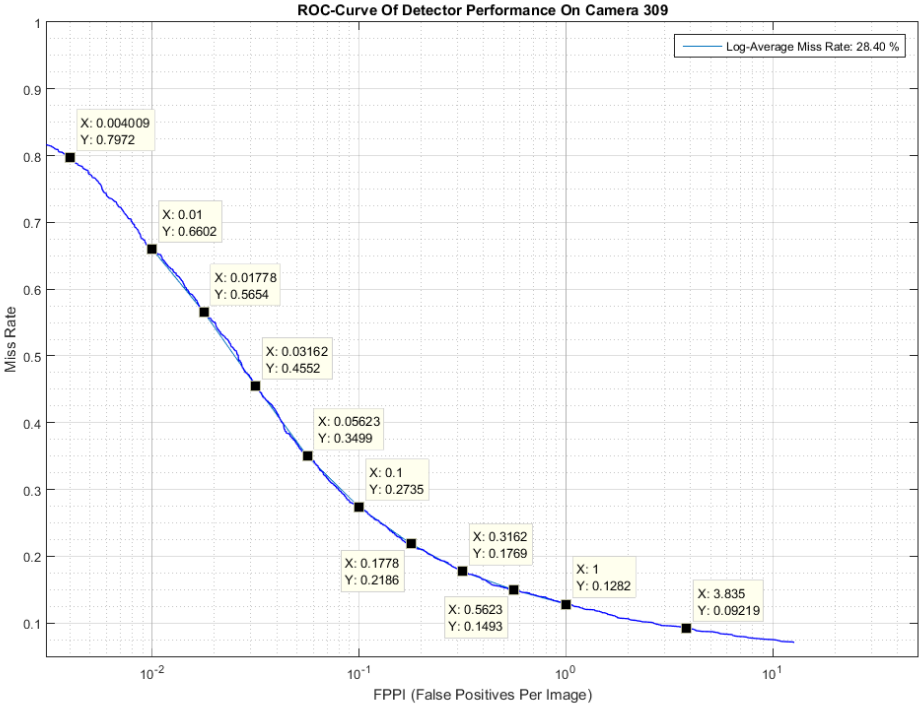


Figure 6.1: Camera 309 ROC With Log-average Miss Rate

For this application, and because the tracking stage adds a filtering layer, the working point is on the right-hand side of the ROC curve, due to the high number of FPPI. Due to this added layer, it is acceptable to have one or more false positives per image, and thus have a miss rate of less than 9%. As discussed in previous chapters, the tracking stage measures each detection input against a set of criteria such as total visibility, which helps with the rejection of assigning tracks to false detections.

It should be noted that the ROC-curve is a function of the validation data set, and does not necessarily represent the detector performance over the whole day. It does however help to quantify the relative miss rate that can be obtained under ideal validation conditions.

These conditions include that all detection scenarios in the validation set be included in the training set, and that the validation set be meticulously annotated.

6.3 Tracking Validation (*Manual* vs *Autonomous*)

The tracking results are verified and validated by manually counting pedestrians in footage by hand for each day that was automatically analyzed. The manual count was done in 30-minute intervals from 06:00am to 07:00pm for 16 to 18 November 2015.

This verification excludes the comparison of pedestrians moving in opposite directions, and only compares the total number of pedestrian crossings per 30-minute interval.

Figure 6.2 shows a histogram of the results of the manual validation, where the light colours denote the *manually* counted results, and the dark colours denote the *automatically* counted pedestrians.

The figure shows that the *manual* and *automatic* counts both have peaks around 06:30am and 05:00pm that coincide, and that the various *manual* counts are close to their *automatic* counterparts. The pedestrian-crossing distribution also displays bell-curve-like behaviour around the peaks as would be expected from a natural distribution. The difference in count values can be attributed to environmental factors, such as shadows or illumination effects. Pedestrians moving in large groups also influence the performance as this results in pedestrians being occluded, making it difficult for the algorithm to detect and track.

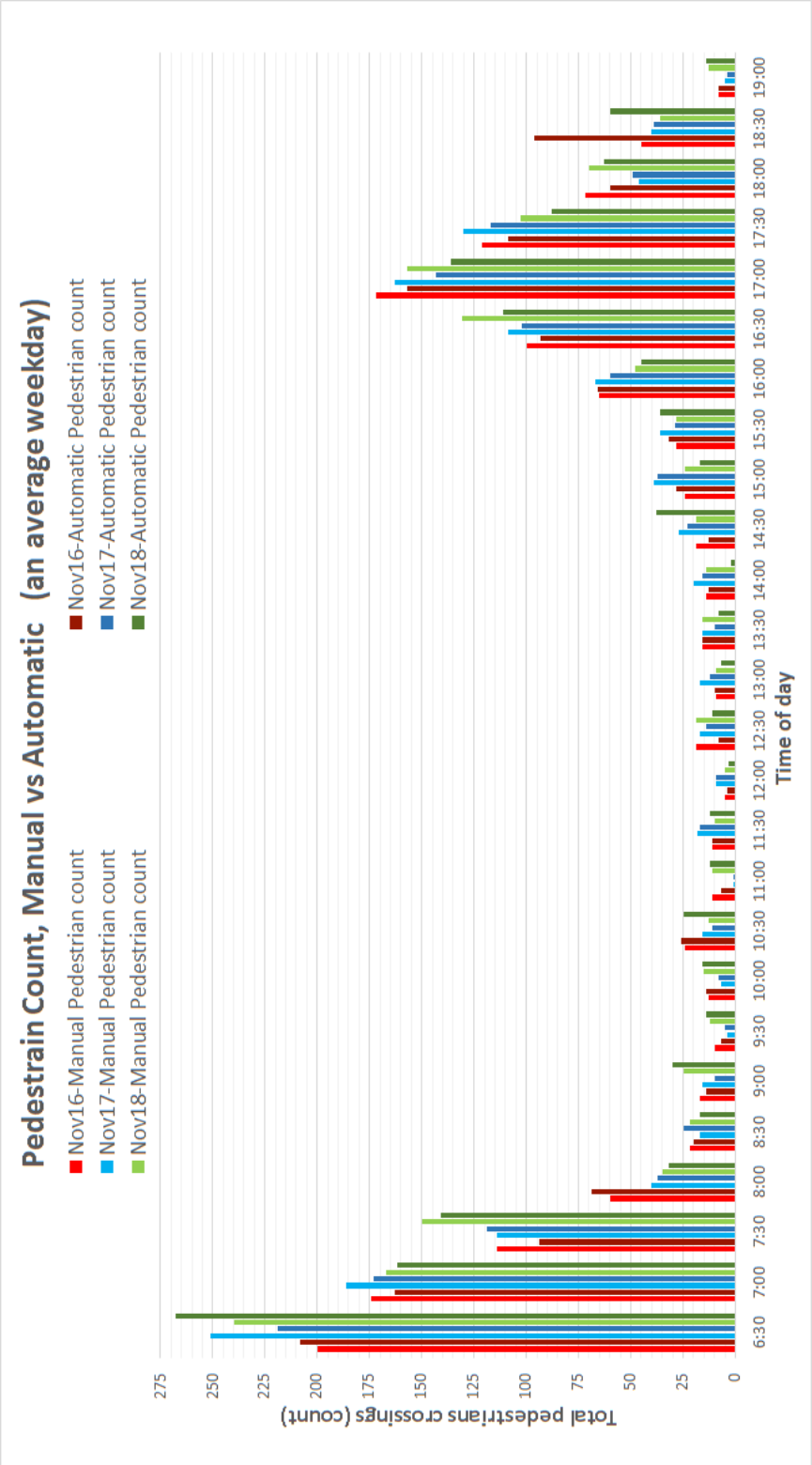


Figure 6.2: A comparison between *Manual* and *Automatic* counted pedestrians for selected days

An enhanced view of the *manual* and *automatic* counts for 16 November 2015 is shown in figure 6.3. Once again the light colours denote the *manually* and dark colours denote the *automatically* counted pedestrians.

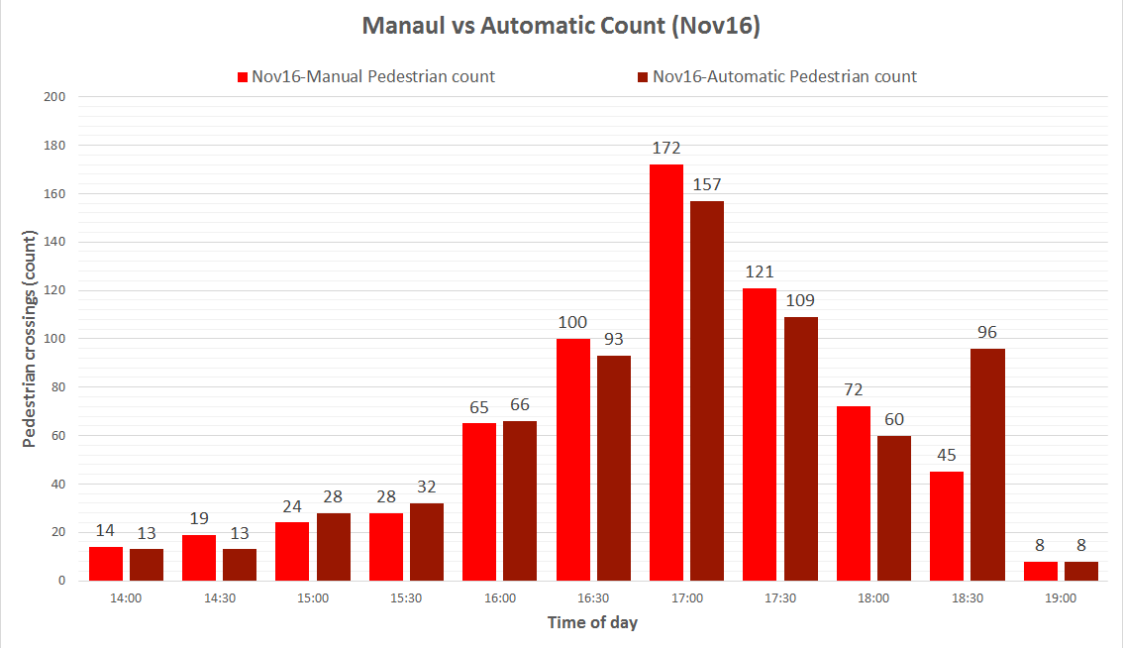


Figure 6.3: Afternoon comparison of the *Manual* vs *Automatic* counts For 16 November

It can be seen from figure 6.3 that *manually* and *automatically* counted pedestrians only differ slightly and coincide with a tolerance. Some outliers do still exist, which is mostly due to environmental effects as previously discussed.

A number of the values seem to be far off, e.g. around 05:00pm the automatic count missed 15 pedestrians. This would be significant were it not for the fact that the 15 pedestrians missed are out of a total of 172 total pedestrians counted *manually* in the time period. It is thus evident that the automatic count missed only a small fraction of the total number of pedestrians.

Figure 6.4 shows the total *manual* minus *automatic* count differences for 16 to 18 November. It can be seen that both under-counting and over-counting occur at various stages of the day, with the most significant differences being around the peak times, i.e. 06:30am and 05:00pm. The higher differences are due to the total number of pedestrian crossings during those time periods. Higher pedestrian density means that more pedestrians are moving in groups, which increases the probability that pedestrians occlude one another.

The higher pedestrian density makes it harder for the classifier to distinguish between groups of partially occluded pedestrians, and results in pedestrians being misclassified as false detections.

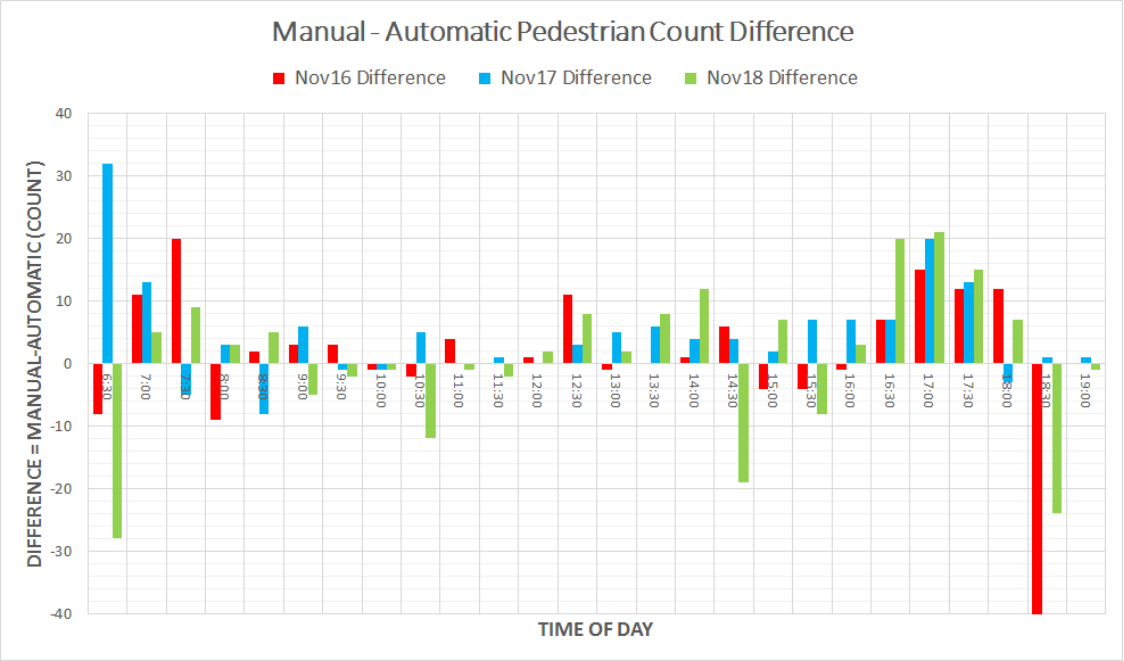


Figure 6.4: Absolute count difference of the *Manual* minus *Automatic* counts over three days

Figure 6.5 shows the percentage overcount done by the *automatic* count. This is the percentage by which the *automatic* count is higher than the *manual* count. For example, a -5% overcount means that the *automatic* count was actually 5% less than the *manual* count.

Figure 6.5 presents another view (percentage-wise) of the data compared to figure 6.4 where the difference is represented by the total number of pedestrians.

Figure 6.5 shows that the highest percentile of missed pedestrians is actually from 09:00am to 03:00pm, where the total count is the lowest. The high missed percentile is due to the low total number of pedestrians counted during that time period. If, for example, one pedestrian is missed out of a total of two pedestrians during that time period, this would have an overcount percentage of -50%. It can however also be seen that the percentile of pedestrians is lowest during peak times where it is most important.

Figure 6.5, together with figure 6.4, aim to give better insight into the raw performance of autonomous counting and provide both the total difference in pedestrians as well as a percentage-based representation.

Lastly, figure 6.6 shows the total differences between the *manual* and *autonomous* counting over three days. The light colours denote the *manually* and the dark colours denote the *automatically* counted pedestrians.

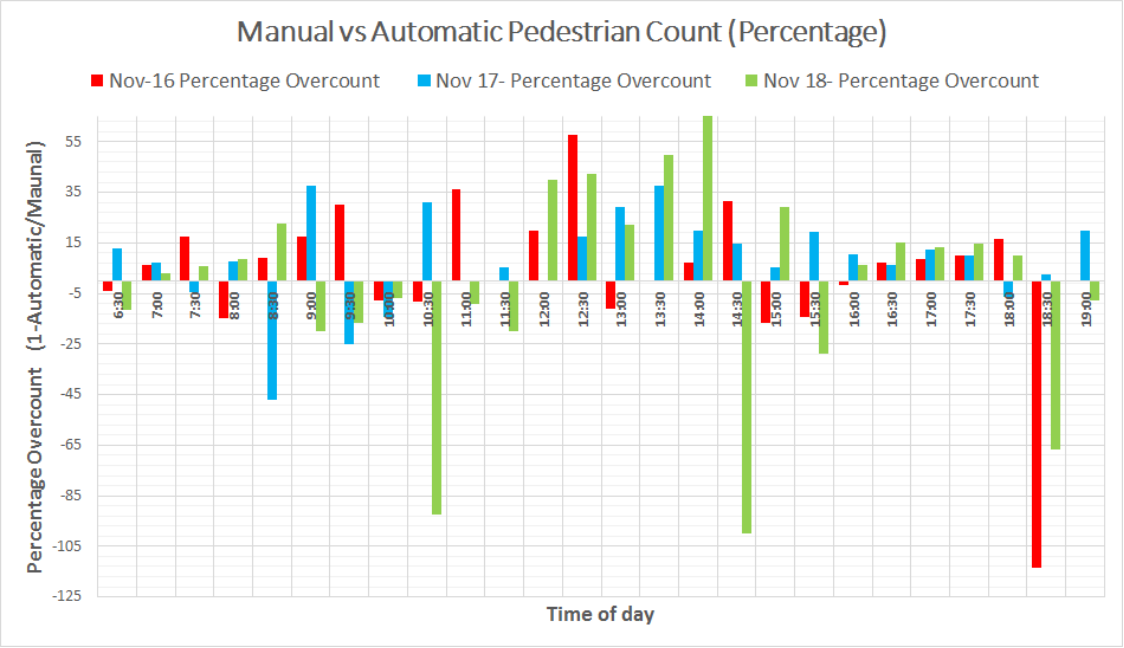


Figure 6.5: Percentage *Automatic* overcount for three days

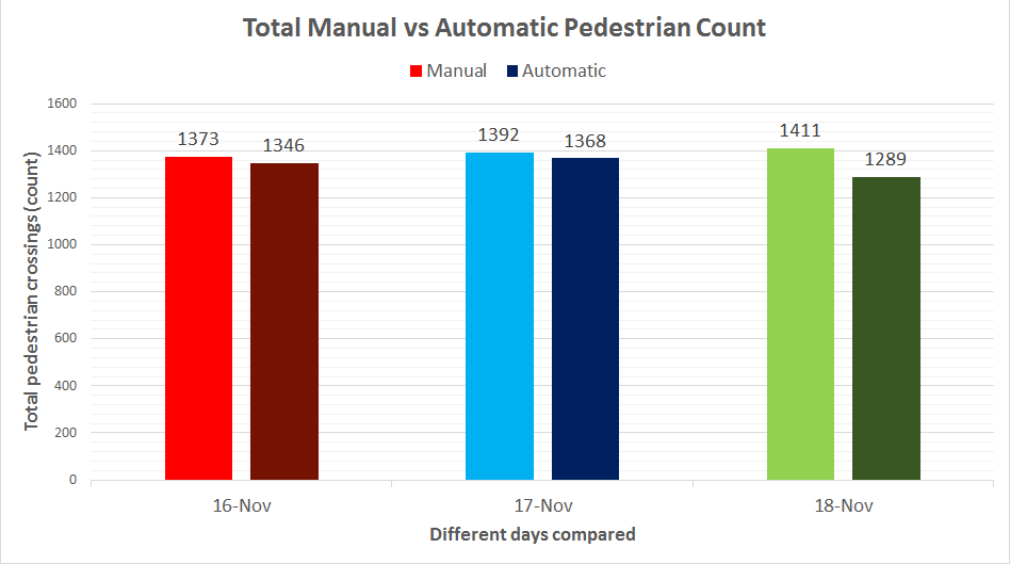


Figure 6.6: A comparison between the totals of *Manual* and *Autonomous* counting of pedestrians over a whole day

Figure 6.6 shows that the total number of *automatically* counted pedestrians closely matches the totals of the *manually* counted pedestrians to within a small tolerance.

Ultimately the undercounting and overcounting done by the *automatic* count even out, and the end result is an *automatic* count with an accuracy of more than 90%. The variations in count results are insignificant as the end total number of pedestrian crossings closely match the ground truth.

6.4 Autonomous Counting Results

The output from the counting results for *camera 309* can be used to analyze the pedestrian activity and behaviour.

The pedestrian crossings from right to left, and left to right, as well as the relative pedestrian speed (in frames), are the tracking stage output as previously discussed in section 5.2.8. The output count histograms together with a relative pedestrian speed example are shown. As per the legend blue indicates pedestrians moving from left to right, and orange indicates pedestrians moving from right to left. The dark brown indicates the overlap of R2L (orange) on L2R (blue) in subsequent figures.

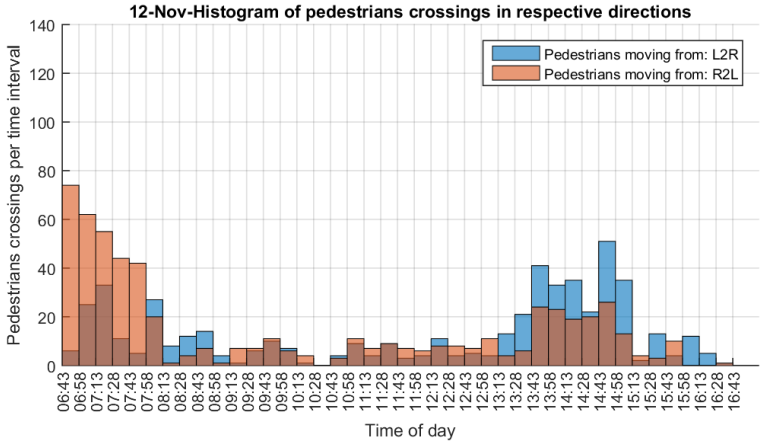


Figure 6.7: Count Histogram -12 November 2015 (Thursday)

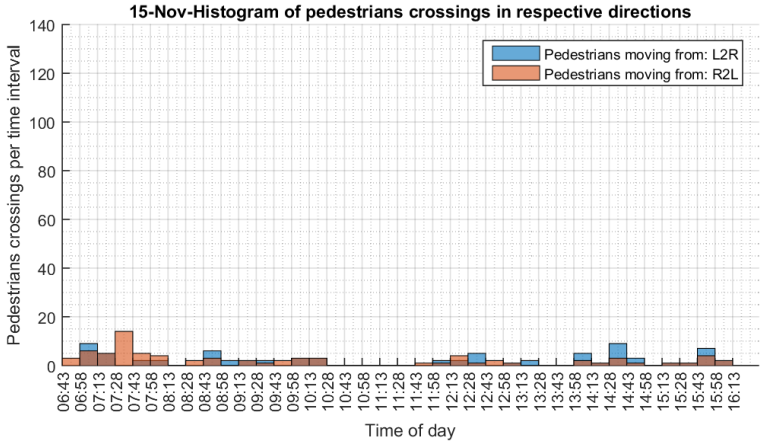


Figure 6.8: Count Histogram -15 November 2015 (Sunday)

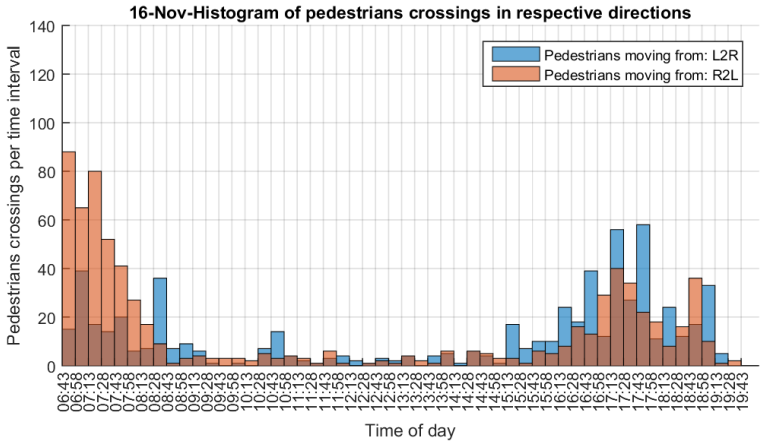


Figure 6.9: Count Histogram -16 November 2015 (Monday)

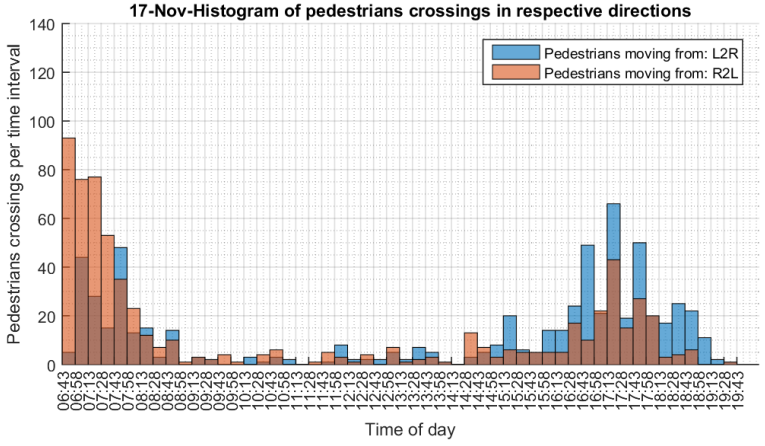


Figure 6.10: Count Histogram -17 November 2015 (Tuesday)

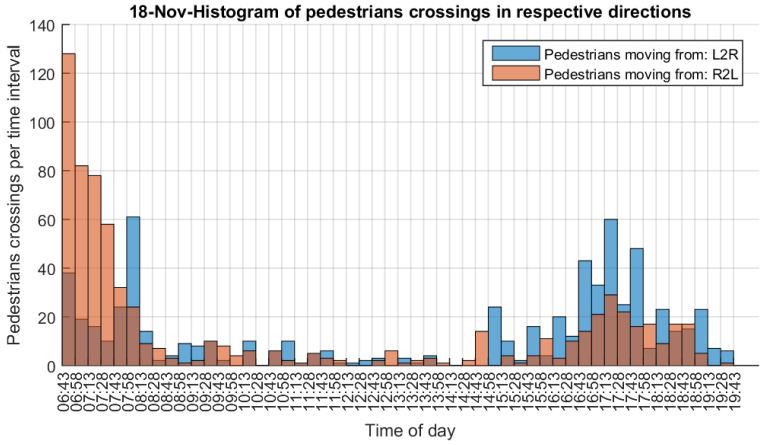


Figure 6.11: Count Histogram -18 November 2015 (Wednesday)

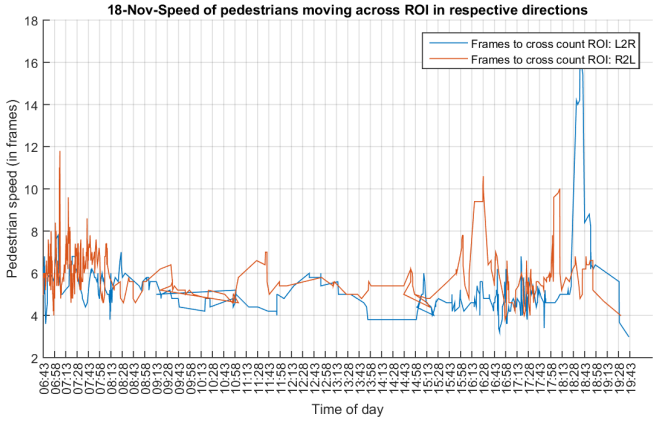


Figure 6.12: Relative Unsmoothed Pedestrian Speeds -18 November 2015 (Wednesday)

It can be seen that figure 6.8 has the lowest pedestrian activity, as would be expected from a Sunday. Figure 6.7, Thursday, and figure 6.9, Monday, show the highest number of pedestrian crossings.

It should be noted that no definitive conclusion about activity during the days of the week can be made as the sample size is too small, and some obstacles such as camera downtime prevented the analysis of multiple complete weeks.

Figure 6.12 shows the average relative speeds of 18 November. The measurements are noisy and the results can be improved with an optimized fitting curve or other smoothing techniques.

As an example of the possible representations of the pedestrian counting data with the Verification user interface, figure 6.13 compares the cumulative count in both directions for two arbitrary days, namely 16 and 18 November.

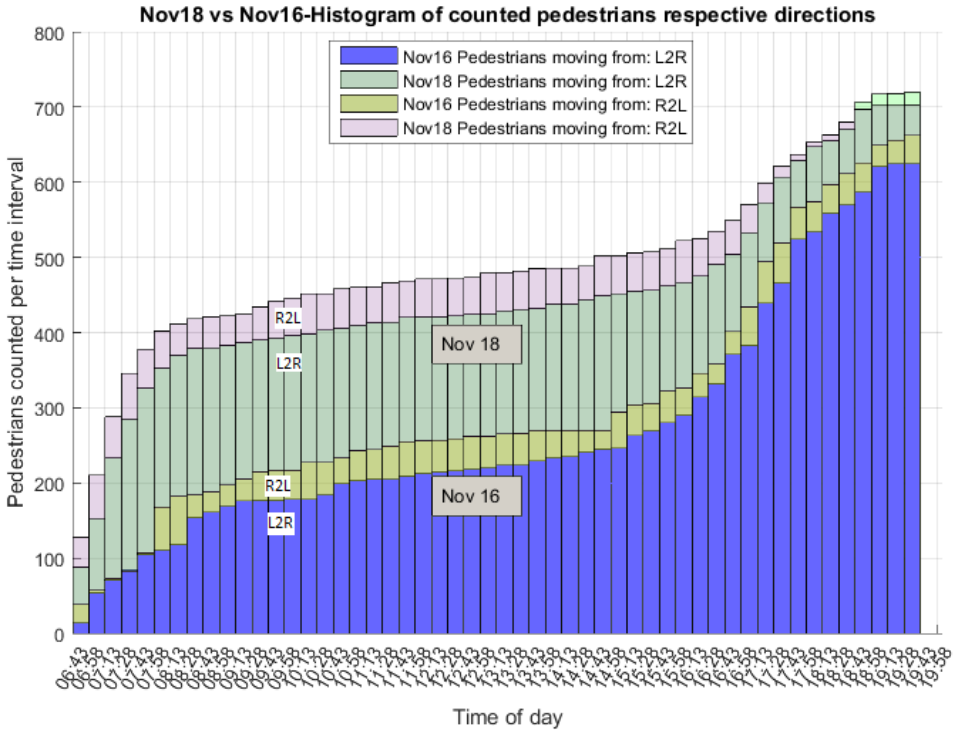


Figure 6.13: Cumulative histogram comparing 16 and 18 November in both directions

Figure 6.13 shows that 18 November has the highest total pedestrian count. The figure also shows that pedestrians move more from right to left (towards the CBD) in the morning, and more from left to right (away from the CBD) in the afternoon as seen from the count histograms in figures 6.7 to 6.11.

6.5 Peak Hour Pedestrian Activity

From the counting results, the peak hour factor (PHF) and other interesting information about the peak hour of pedestrian crossings can be obtained. The peak hour factor is a measure of traffic demand fluctuations within the peak hour. It is the hourly volume during the highest activity hour of the day, divided by the peak 15-minute flow rate within the hour.

$$PeakHourFactor(PHF) = \frac{Max_{hour}}{4Max_{15-min}} \tag{6.1}$$

The PHF is thus an indication of the narrowness or broadness of the lobs around the 15-minute peak within the hour, with values closer to 1 being a broad peak, and closer to 0 being a narrow or sharp peak.

Table 6.1 shows the morning peak of the pedestrian activity. The table presents the direction of movement, the time of the peak hour, the time of the 15-minute period

with the highest pedestrian activity, the value of the maximum 15-minute period, the total number of crossings in the peak hour, the average crossings per 15-minute interval, and the peak hour factor.

Table 6.1: Morning Peak Hour Results

Date	Direction	Peak Hour	Peak 15min	MAX	SUM	AVG	PHF
12-Nov	R2L	06:30-07:30	06:45	74	236	59	0.797
16-Nov	R2L	06:30-07:30	06:45	88	285	71	0.81
17-Nov	R2L	06:30-07:30	06:45	93	229	75	0.804
18-Nov	R2L	06:30-07:30	06:45	128	346	87	0.676
Average	**R2L**	**06:30-07:30**	**06:45**	**96**	**292**	**73**	**0.771**
12-Nov	L2R	06:30-07:30	07:15	33	74	19	0.561
16-Nov	L2R	07:00-08:00	07:00	39	85	21	0.545
17-Nov	L2R	07:00-08:00	07:30	48	104	26	0.542
18-Nov	L2R	07:30-08:30	08:00	61	109	27	0.447
Average	**L2R**	**07:00-08:00**	**07:26**	**45**	**93**	**24**	**0.523**

Table 6.1 clearly shows that the highest number of pedestrians crossing pedestrian bridges in the morning is from right to left (R2L towards CBD), with a total average sum of 292 pedestrian crossings in the hour, and an average of 73 pedestrians per 15-minute period.

The average PHF of 0.77 in the R2L direction shows that the peak hour features a constant flow of pedestrians, with a broad peak in the peak hour. It can also be seen that the PHF in the L2R direction is 0.5, which indicates a narrower peak in the peak hour.

It can also be seen that the highest peak 15 minutes on average contains a value of 96 pedestrian crossings, which is higher than the average sum in the L2R direction of 93. It can also be seen that the highest peak has a time of 06:43am, which is where analysis of the footage started (cameras only turned off night vision mode after about 06:30am). It could thus be that the highest peak may lie earlier where footage is not analyzed.

The table also shows that the average morning 15-minute peak in the L2R direction is on average 30 minutes later than that observed in the R2L direction.

Table 6.2 shows the afternoon peak hour pedestrian activity in both directions. Data from 12 November was omitted as the dataset was incomplete.

Table 6.2 shows the afternoon peak pedestrian activity. It is clear that the highest number of pedestrians move from left to right (L2R away from the CBD). The peak pedestrian traffic period perfectly coincides at 17:13 in the R2L direction, and only deviates slightly in the L2R direction. It can be seen that the PHF of the max peak hour

Table 6.2: Afternoon Peak Hour Results

Date	Direction	Peak Hour	Peak 15min	MAX	SUM	AVG	PHF
16-Nov	R2L	16:45-17:45	17:15	40	125	31	0.781
17-Nov	R2L	16:45-17:45	17:15	43	107	27	0.622
18-Nov	R2L	16:45-17:45	17:15	28	88	22	0.759
Average	**R2L**	**16:45-17:45**	**17:15**	**37**	**107**	**27**	**0.721**
16-Nov	L2R	17:00-16:00	17:45	58	152	38	0.656
17-Nov	L2R	16:45-17:45	17:15	66	156	39	0.59
18-Nov	L2R	16:45-17:45	17:15	60	166	42	0.691
Average	**L2R**	**16:45-17:45**	**17:28**	**61**	**158**	**40**	**0.646**

of both directions, and both morning and afternoon peak, is in the morning peak in the R2L direction (Towards the CBD). It can also be seen that the PHF in the L2R direction increased from 0.52 in the morning to 0.65 in the afternoon.

Thus the PHF indicates that the peak traffic is towards the CBD in the morning, and away from the CBD in the afternoon. Although the L2R direction has the highest peak, with the largest sum and averages in the afternoon, it is still far less than those observed in the R2L direction in the morning peak.

The pedestrian traffic from right to left is significantly higher in the morning peak, representing people going to work. Pedestrian traffic from left to right appear to be only marginally higher in the afternoon peak when compared to movement in the opposite direction. From this it could possibly be inferred that some pedestrians use a different route when returning from work.

6.6 Relative Pedestrian Speeds

Figure 6.14 and figure 6.15 below show the curves fitted onto the raw pedestrian durations, together with the average speeds for both directions. The durations are calculated as the number of frames it takes pedestrians to cross the rectangle counting ROI, as previously discussed in section 5.2.6 on page 82.

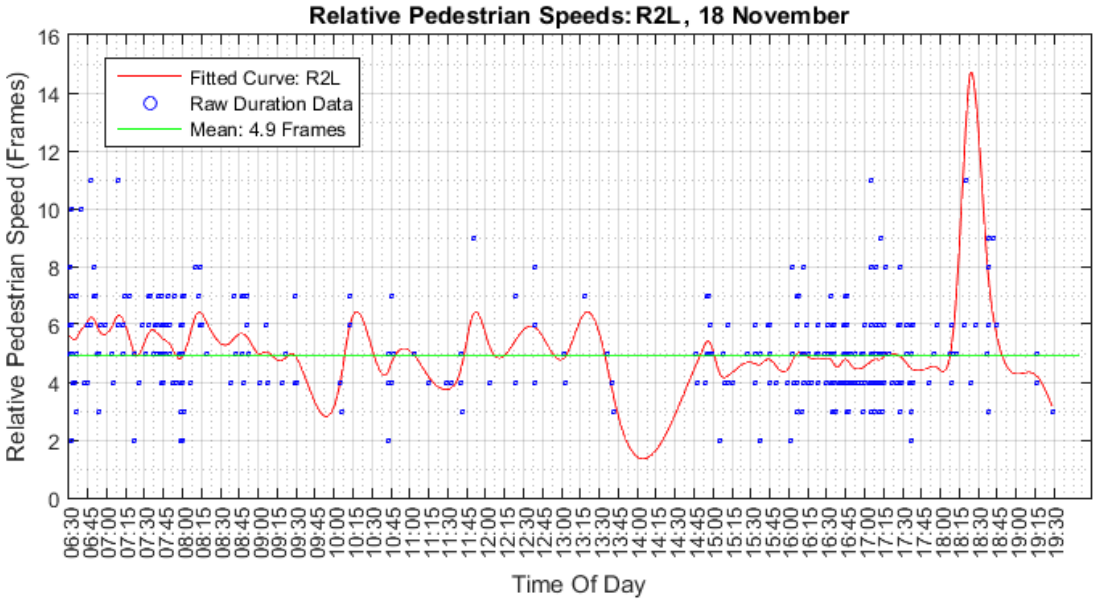


Figure 6.14: Relative Pedestrian Speed With Fitting Curve (R2L) - 18 November 2015

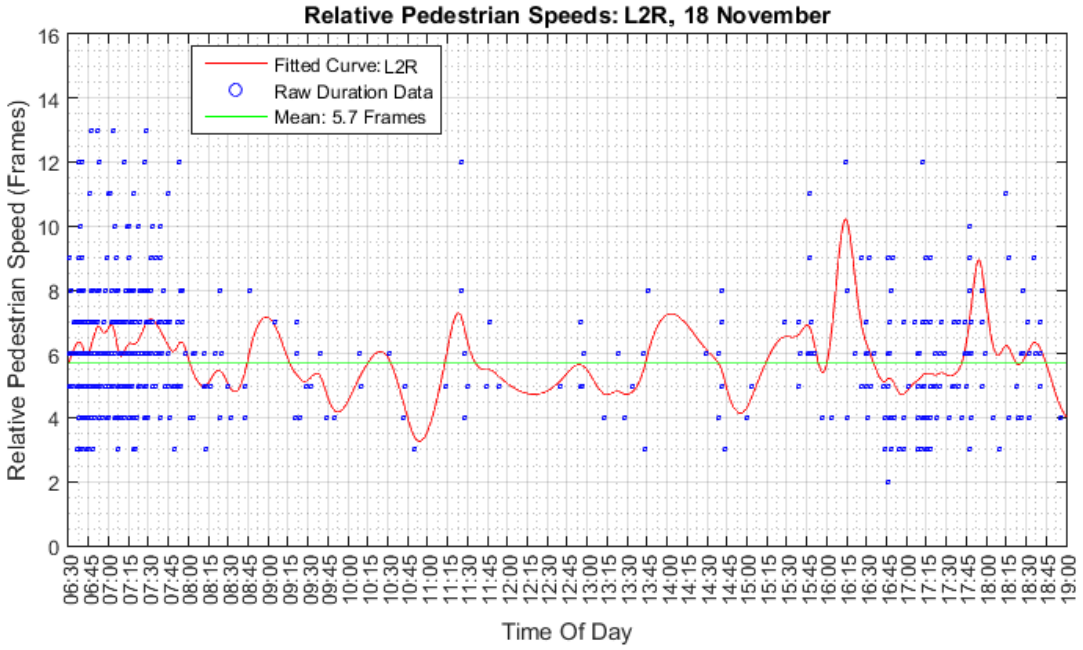


Figure 6.15: Relative Pedestrian Speed With Fitting Curve (L2R) - 18 November 2015

The duration data is smoothed with a smoothing spline, and with fitting curves having the same p value. The pedestrian speed activity around the peaks at 06:30am

and 05:00pm looks similar for both directions. There is not enough data around the off-peak times to be able to define a correlation between time and pedestrian speeds.

The fitted curves of the pedestrian speeds in both directions are compared in figure 6.16.

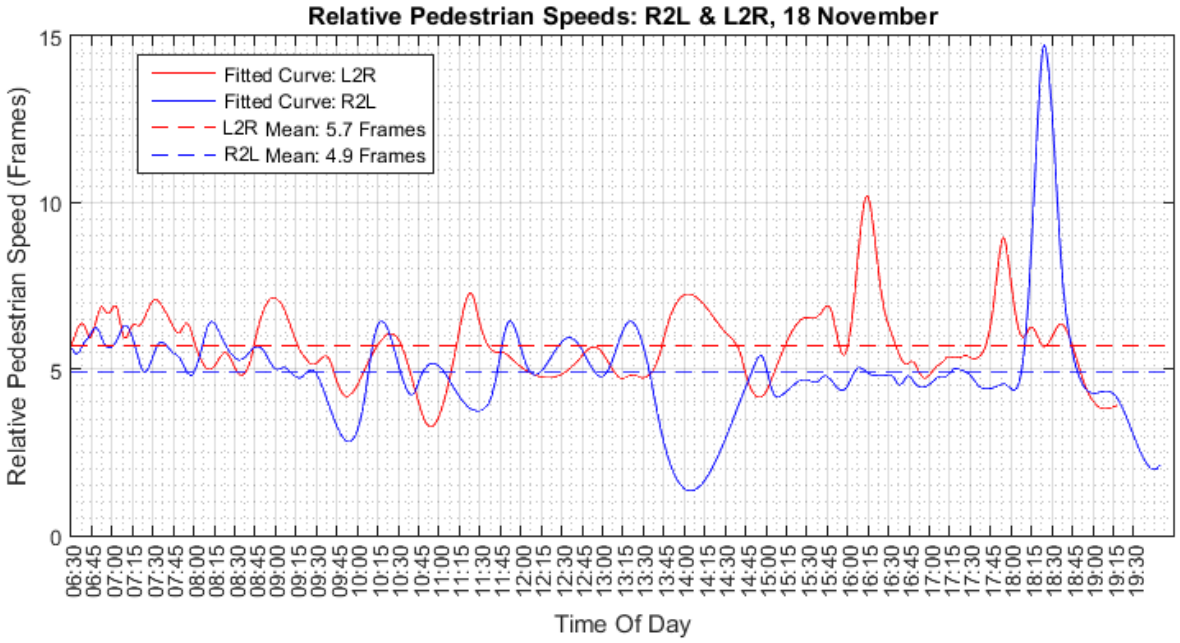


Figure 6.16: Relative Pedestrian Speed With Fitting Curve - 18 November 2015

From the figure it can be seen that the relative pedestrian speeds differ for either direction. Pedestrians moving from left to right who cross the counting ROI move on average one frame faster than in the opposite direction.

The deduction can be made from the data that pedestrians walk faster towards work (towards CBD: R2L) than they walk from work (away from CBD: L2R).

These figures merely show that the speed data can be obtained and the capabilities exist to represent and analyze the data in a meaningful way.

6.7 Summary

This chapter presented the results of the pedestrian detection and tracking system. For the detection stage, it could be seen that the detector achieved state-of-the-art performance, with a miss rate of less than 10%.

The tracking and counting results are presented, where the *automatically* counted pedestrian crossings were compared to *manually* counted pedestrian crossings. From the comparison it could be seen that the counting system achieved a hit rate of more

than 90%, and that the *automatically* counted pedestrian crossings are directly comparable to *manually* counted pedestrian crossings done by hand.

From the analyses of selected days, it could be seen that the morning peak towards the CBD (R2L) contained the highest number of pedestrian crossings of the day, with an average of 73 pedestrians crossing in the peak hour. The highest afternoon peak hour had an average of 40 pedestrian crossings in the peak hour with pedestrians moving away from the CBD (L2R). It could also be seen from the results that pedestrians move on average up to 10% faster when walking in the R2L direction when compared to the L2R direction. This chapter thus confirmed that the tuning parameters in the implementation chapters were correctly set through iterative methods, and that the pedestrian-monitoring system can be seen as proof that existing surveillance infrastructure can be used to monitor and count pedestrians under ideal conditions.

Chapter 7

Conclusion

This chapter concludes the work of this thesis by presenting a summary and discussing the motivation behind the work done. The conclusions and validations of the hypotheses made in Chapter 1 in respect of the work done in this thesis are also discussed. Next, recommendations for future work on pedestrian monitoring using image processing techniques are presented and discussed. Finally, important concluding remarks are presented.

7.1 Summary Of Work

This study focuses on the monitoring and safety of pedestrians who use freeways and pedestrian bridges. A large number of pedestrian incidents occurs annually on the freeways as shown in Chapter 1. Data collection and pedestrian-movement surveys are also done by hand. In Chapter 1 the need to automatically collect the pedestrian data in surveillance footage was discussed. Environmental and physical challenges to the implementation of a pedestrian-monitoring system number among the limitations currently faced by the authorities, and these are also presented.

In an effort to address this problem, an application that processes images from existing surveillance infrastructure to enable the collection of pedestrian mobility data on pedestrian bridges is developed as a proof of concept. This study has the future goal of expanding the pedestrian-monitoring system to the freeways.

Chapter 2 discussed the basic building blocks of image processing in pedestrian detection, as well as modern pedestrian detection and classifier training methods. The chapter also discussed alternative pedestrian-tracking methods used in related work.

In Chapter 3 the setup and methodology of the pedestrian-monitoring system, which consists of a detection and tracking stage, are discussed. It was shown that detectors needed to be trained specifically for a camera, and that detectors are not generic by applying pre-trained *INRIA* and *Caltech* classifiers to test footage from a

surveillance camera under ideal conditions. Significant parameters of both stages are discussed along with the methodology followed during each phase of each stage.

Chapter 4 discussed the implementation of the detection stage. The chapter discussed the important parameters used together with empirical results of the impact of suboptimally tuned parameters on detector accuracy. The choice of NMS (non-maximum suppression) was presented by way of graphical verification methods and iterative testing. The graphical user interface that enables the user to run the detection and tracking stage was presented and its functionality was discussed. The chapter was concluded by discussing the verification methods used in the detection stage, along with the challenges faced.

Chapter 5 discussed the implementation of the tracking stage. The chapter also discussed the implementation of important tuning parameters in the tracking algorithm and Kalman filter. The placement of the counting line and its impact on detector performance were then presented. The implementation of the counting algorithm and the implementation of track-management functions were also presented. The verification graphical user interface implemented to allow the user to review the counting results easily was presented and its functionality was discussed. Lastly, the challenges faced during the implementation of the tracking stage were discussed.

Chapter 6 presented the results from the detecting and tracking stage, together with example results from the counting algorithm. The chapter discussed the accuracy results obtained from the classifier in the detection stage. The chapter also presented a comparison between *manually* and *automatically* counted pedestrian-bridge crossings. Examples of five days' worth of analyzed footage were presented in the form of multiple count histograms showing pedestrian crossings in both directions. The relative pedestrian speeds were also discussed.

This work thus presents the setup and methodology, implementation, and results of a proof-of-concept pedestrian-monitoring system that uses existing surveillance infrastructure to function as a tool to automatically collect pedestrian-mobility information.

7.2 Conclusions

The conclusions in this section are divided into those related to *thesis statement 1* (Detection stage), *thesis statement 2* (Tracking stage), and *thesis statement 3* (Mobility Information from tracking results) as defined in Chapter 1.

7.2.1 Detection Stage Conclusions

Thesis statement 1:
Pedestrians on the pedestrian bridges can be detected with high accuracy using ex-

isting surveillance cameras.

The methodology and setup of the detection stage presented in section 3.2 to 3.5 is validated by the detection results from section 6.2.

A camera could be chosen with the optimal orientation and adequate resolution as can be seen from section 6.2 on page 90, showing that the final detector obtained a state-of-the-art hit rate of more than 90%. These results verify *hypothesis 1.1*, *1.2*, and *1.4*.

Hypothesis 1.1:
The existing surveillance camera has a resolution adequate to perform pedestrian detection.

Hypothesis 1.2:
There is an existing surveillance camera with the correct orientation to allow for pedestrian detection that can be used as a best-case scenario.

Hypothesis 1.4:
The detector is able to detect multiple distinct pedestrians reliably throughout a day with a hit rate of more than 80%.

Visual evaluation of the detection results verifies that the detector can distinguish easily between partially occluded pedestrians, as an abundance of partially occluded pedestrians were included in the training set as discussed in section 3.2 on page 33. This verifies *hypothesis 1.3*.

Hypothesis 1.3:
The detector is able to distinguish between partially occluded pedestrians moving in a group.

A pre-trained detector is tested on subsets of footage from the existing surveillance infrastructure, and it is shown in section 3.2 on page 33 that detectors need to be trained specifically for each camera. Pre-trained detectors obtained at best a hit rate of 64%. This verifies *hypothesis 1.5*.

Hypothesis 1.5:
The detector needs to be trained specifically for each scene in order to obtain a hit rate of more than 70%.

False detections or false negatives can be suppressed to a rate that does not interfere with the visual evaluation of the detection results by way of methods discussed in section 3.5 on page 51 and section 4.4 on page 72. The false detections can be suppressed in such a way as not to also suppress positive detections. This is accomplished by selecting the correct NMS method and threshold as discussed in section 4.3.3 on page 69. These results verify *hypothesis 1.6*.

Hypothesis 1.6:
False detections can be suppressed to a visually acceptable rate.

Lastly, the parameters implemented as discussed in section 3.3.2 on page 48, section 4.2.1 on page 63, and section 4.2.2 on page 65 did not deviate significantly from those normally employed for pedestrian detection, and it was possible to apply only slight adjustments to the parameter values to obtain a state-of-the-art detector hit rate performance of more than 90% as presented in section 6.2 on page 90. The results verify *hypothesis 1.7*.

Hypothesis 1.7:
The training and detection parameter values do not deviate significantly from those normally employed for pedestrian detection.

7.2.2 Tracking Stage Conclusions

Thesis statement 2:
Pedestrians on the pedestrian bridges can be tracked with a high degree of certainty from their detections.

The setup and methodology presented in section 3.7 to 3.9 and implementation of the tracking stage discussed in Chapter 3 are verified with the results in section 6.3.

A tracking system is designed and implemented to track pedestrians from the detection results of the detection stage as discussed in Chapter 5. From the results in section 6.3 where the automatic tracking results are compared to manually counted ground-truth data, it can be seen that the tracking stage and pedestrian-monitoring system achieved a hit rate of more than 90%. Section 6.3 on page 91 also shows that pedestrians can be tracked within a 15% tolerance during peak pedestrian activity. This proves that the Kalman filter can be optimally tuned through iterative methods and incorporated into

a tracking algorithm to achieve a hit rate exceeding 90%. These results verify *hypothesis 2.1* and *2.2*.

Hypothesis 2.1:
Pedestrians can be identified and tracked during peak pedestrian activity and throughout the day with a hit rate of more than 80%.

Hypothesis 2.2:
Kalman filter parameters can be optimally tuned to achieve a tracking hit rate of more than 80%.

It can be seen from evaluation via the verification methods discussed in section 3.9 and 5.3 that the track-management functions implemented in the tracking algorithm aid in the rejection of false/noisy detections allowed through from the detection stage to boost detector performance. The noisy detections do not have a significant impact on the tracking performance, as section 6.3 shows that the pedestrians overcounted by the *automatic* system due to false detections are kept within reasonable limits, i.e. less than 10% overcount on average, if overcount occurs. These results verify *hypothesis 2.3*.

Hypothesis 2.3:
Further rejection of false/noisy detections from the detection stage can be achieved with the tracking algorithm.

Visual verification was used in the iterative process to tune the parameters in the tracking stage. The system was tested on small subsets of footage, and the parameters were adjusted accordingly to increase perceived tracking performance as discussed in section 3.9 on page 57. The final detector achieved state-of-the-art tracking performance of more than 90% when tested on the full dataset. It can thus be deduced that the visual evaluation of subsets of the full dataset is an adequate method for the rapid tuning of parameters in the tracking stage to obtain a tracking hit rate of more than 90%. These results and the performance results of the final tracking algorithm verify *hypothesis 2.4*.

Hypothesis 2.4:
Visual verification, and thus quantitative evaluation, is an adequate method to evaluate the tracking results of subsets of footage to tune the required tracking parameters.

7.2.3 Pedestrian Mobility Conclusion

Thesis statement 3:
The tracking results can be used to count pedestrians crossing pedestrian bridges, and to provide useful pedestrian-mobility information.

The implementation of the counting algorithm presented in section 5.2.6 to collect pedestrian-mobility information is validated in section 6.4.

In this section the results from analyzing footage from selected days and from inspection of the raw counting results show that the direction and relative speed of pedestrian movement can be obtained from the tracking results. This can also be confirmed visually by the verification user interface presented in section 5.3 on page 85. The total number of pedestrian crossings and the time of crossing can reliably be obtained as proposed in section 5.2.7 on page 84, and validated in section 6.3 on page 91, where the times and total number of pedestrian crossings from both the *manually* counted and *automatically* generated data coincide to within a small tolerance. The counting results can be represented in a user-friendly graphical user interface with various options of histogram plots and by presenting the count data alongside the relative pedestrian speeds in both directions. This verifies *hypothesis 3.1, 3.2, 3.3, and 3.4.*

Hypothesis 3.1:
Information about the direction of movement can be reliably obtained from the tracking results.

Hypothesis 3.2:
Information about the relative pedestrian speeds can be obtained from the tracking results.

Hypothesis 3.3:
The time of crossing and total number of pedestrian crossings can be obtained from the tracking stage.

Hypothesis 3.4:
The counting results can be represented in a meaningful way using a graphical user interface.

7.2.4 Interesting Findings

Section 6.4 on page 97 presented the pedestrian-mobility information obtained by analyzing the data from selected days. The results show that more pedestrians on

average cross the pedestrian bridges from left to right (towards the CBD), with the highest peak in the morning. This could mean that pedestrians possibly use other routes when returning from work. The section also shows the relative pedestrian speeds in both directions, *Left2Right* (Towards CBD) and *Right2Left* (Away from CBD). The speed results show that pedestrians move on average 15% faster from *Left2Right*, with pedestrians moving at relative speeds of 5.7 frames *R2L* and 4.9 frames *L2R*.

7.3 Future Work

Recommendations and applications for future work are discussed in this section. Although the pedestrian-detection system developed and implemented proves that existing surveillance infrastructure can be used to obtain-state-of-the-art performance, it is far from practically viable on a wider scale. This is due to technical and operational challenges and issues faced as discussed in Chapter 1. In short, only a select few static cameras are currently viable for the implementation of the pedestrian-detection system due to occlusion by infrastructure on pedestrian bridges, or cameras that face pedestrians at suboptimal angles.

Recommendations for future work would be to reduce the computational time so as to enable the real-time detection and tracking of pedestrians. The system can also be integrated with behaviour-recognition algorithms to enable authorities to detect possible suspicious pedestrian behaviour. Ultimately, once the operational and technical limitations on the existing infrastructure monitoring the freeways have been overcome, the system can be expanded to monitor pedestrians walking along the freeways. This would aid authorities in the gathering of pedestrian-mobility information, and conceivably also alert authorities of high-risk pedestrian behaviour along the freeway. It is also recommended that attention be given to the placement and orientation of surveillance cameras with a view to the future implementation of a pedestrian-monitoring system. To this end, pedestrians should be minimally occluded by the environment and infrastructure, and the camera should provide a side view/silhouette view of pedestrians to increase the chance of obtaining state-of-the-art detection and tracking performance.

Future work can easily incorporate real-world measurements to indicate pedestrian speeds in m/s. This could be done by using known lengths for camera calibration, e.g. the size of infrastructure.

Other recommendations include automating the training-dataset annotation process, adding support to collect information about height (and thus possibly distinguishing between adults and children), and to allow for the application of cable- or crime-hotspot monitoring.

7.4 Concluding Remarks

A pedestrian-monitoring system is developed and implemented to collect pedestrian-mobility information and present the results in a convenient graphical user interface. The system was successfully developed and tested, and it proved that existing surveillance infrastructure can be used. The inspiration behind this study was to improve the available pedestrian information in order to assist the authorities in their pedestrian-safety efforts.

The results show that the system obtained state-of-the-art performance. Accommodating image-processing techniques via existing infrastructure to enable pedestrian monitoring should thus be borne in mind when performing future infrastructure upgrades or installations.

Bibliography

[1] SANRAL, Western Cape Goverment, T.F.C.T.: Pedestrian activity and accidents on and along western cape freeways and arterial roads. 2014. x, 1, 2

[2] Dollár, P., Wojek, C., Schiele, B. and Perona, P.: Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012. x, 8, 11, 12, 14, 31, 47, 48, 59, 64, 69

[3] Leeser, M. and Yu, H.: Available at: https://sites.google.com/site/hsi2013logan99/computer-vision/implementation, [2016, June 08], 2013. x, 12

[4] Lee, C.: Available at: https://commons.wikimedia.org/wiki/File:Image_pyramid.svg https://upload.wikimedia.org/wikipedia/commons/4/43/Image_pyramid.svg, [2016, August 08], 2016. x, 13

[5] Viola, P. and Jones, M.: Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. I—-511—-I—-518, 2001. ISSN 1063-6919. x, 13, 14, 15, 19, 20, 21, 22, 25, 26, 31

[6] Rosebrock, A.: Available at: http://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python, [2016, June 09], 2015. x, 15

[7] Zirguezi: Available at: https://en.wikipedia.org/wiki/Mean_shift/media/File/Kernel_Machine, [2016, June 09], 2011. x, 16

[8] Freund, Y., Iyer, R., Schapire, R.E. and Singer, Y.: An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, vol. 4, pp. 933–969, December 2003. ISSN 1532-4435.
Available at: http://dl.acm.org/citation.cfm?id=945365.964285 x, 20

[9] Dalal, N.: Finding People in Images and Videos. *I Can*, p. 149, 2006.
Available at: http://eprints.pascal-network.org/archive/00002373/ x, 12, 13, 14, 18, 22, 23, 24, 25, 26, 27, 31, 46, 47, 64

[10] Benenson, R., Mohamed, O., Hosang, J. and Schiele, B.: Ten Years of Pedestrian Detection , What Have We Learned ? *European Conference on Computer Vision - ECCV*, 2014. 1411.4304. x, 8, 24, 25, 49, 63

[11] Dollár, P., Tu, Z., Perona, P. and Belongie, S.: Integral Channel Features. *BMVC 2009 London England*, pp. 1–11, 2009. x, 13, 14, 25, 26, 27, 47, 48, 59, 64

[12] Hlavac, V.: Digital image proccessing. Available at: `http://sccg.sk/~cernekova/Benesova_Digital20Image20Processing20Lecture20Objects20tracking2020motion20detection.pdf`, 2001. [Online]. x, 29, 30

[13] Dollár, P.: Piotr's Computer Vision Matlab Toolbox (PMT). Available at: `https://github.com/pdollar/toolbox` Copyright 2014 Piotr Dollar, 2016. xi, 16, 17, 34, 43, 44, 47, 49, 59, 64

[14] Dollar, P., Belongie, S. and Perona, P.: The Fastest Pedestrian Detector in the West. *Proceedings of the British Machine Vision Conference 2010*, pp. 68.1–68.11, 2010. 13, 14, 26, 27, 47, 59

[15] Cheng, Y.: Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 790–799, August 1995. ISSN 0162-8828.
Available at: `http://dx.doi.org/10.1109/34.400568` 16, 28

[16] Rodríguez Fernández, J.M.: *Computer vision for Pedestrian detection using Histograms of Oriented Gradients*. Ph.D. thesis, Universitat Polite ecnica de Cataluna, 2014.
Available at: `http://hdl.handle.net/2099.1/21343` 16, 34

[17] Hastie, T., Tibshirani, R. and Friedman, J.: *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 18

[18] Kegl, B.: The return of adaboost.mh: multi-class hamming trees. *CoRR*, vol. abs/1312.6086, 2013.
Available at: `http://arxiv.org/abs/1312.6086` 18

[19] Freund, Y. and Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: *European conference on computational learning theory*, pp. 23–37. Springer, 1995. 22

[20] Zhang, C. and Viola, P.A.: Multiple-instance pruning for learning efficient cascade detectors. In: *Advances in neural information processing systems*, pp. 1681–1688. 2008. 26

[21] Wikipedia: Heuristic (computer science) — wikipedia, the free encyclopedia. 2016. [Online].
Available at: `https://en.wikipedia.org/w/index.php?title=Heuristic_(computer_science)&oldid=732792198` 26

[22] Dollár, P.: Caltech pedestrian detection benchmark. Available at: `http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/`, 2016. [Online]. 28

[23] Shi, J. and Tomasi, C.: Good features to track. In: *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pp. 593 – 600. 1994. 28

[24] Dalal, N.: Inria pedestrian data set. Available at: `http://pascal.inrialpes.fr/data/human/`, 2006. [Online Pedestrian Data Set (French Institute for Research in Computer Science and Automation)]. 34

[25] Dollar, P.: Caltech pedestrian data set. Available at: `http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/datasets/USA/`, 2009. [Online Pedestrian Data Set(California Institute of Technology)]. 34

[26] Appel, R., Belongie, S., Perona, P. and Doll, P.: Fast Feature Pyramids for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532 – 1545, 2014. 47, 48, 49, 59

[27] Nam, W., Dollár, P. and Han, J.H.: Local Decorrelation For Improved Detection. *Nips*, pp. 1–9, 2014. 48, 59, 64