University of Cyprus

DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING

A Vehicle Detection, Counting and Tracking system for Traffic Assessment Using Machine Vision and Aerial Video

ANDRI A. THEOPHANOUS

A Dissertation Submitted to the University of Cyprus in Partial Fulfilment of the Requirements for the Degree of Master of Science

NICOSIA, DECEMBER 2020

ΠΕΡΙΛΗΨΗ

Τις τελευταίες δύο δεκαετίες, η κοινωνία έχει μεταβεί στην ψηφιακή εποχή, με τη σύγκλιση των τηλεπικοινωνιών, του βίντεο και της πληροφορικής και την ενσωμάτωσή τους στην καθημερινότητα μας. Είναι όλο και πιο δύσκολο όμως να παρακολουθήσει κανείς και να έχει πρόσβαση σε αυτό το περιεχόμενο μέσω μηχανών αναζήτησης βάσεων δεδομένων, λόγω του εξαιρετικά μεγάλου και αυξανόμενου όγκου δεδομένων από ψηφιακές εικόνες και βίντεο. Η αναζήτηση με βάση το περιεχόμενο μέσω τεχνικών αυτόματης ανίχνευσης (detection) και αναγνώρισης (recognition) αντικειμένων έχει γίνει ένα από τα πιο σημαντικά και μείζοντα ζητήματα για τα επόμενα χρόνια, για να αντιμετωπιστεί και να περιοριστεί η χρήση των παραδοσιακών συστημάτων πληροφοριών. Το σύστημα ανίχνευσης αντικειμένων (object detection) χρησιμοποιείται ήδη σε μεγάλο βαθμό για την παρακολούθηση αντικειμένων (object tracking), όπως για παράδειγμα την παρακολούθηση μιας μπάλας κατά τη διάρκεια ενός ποδοσφαιρικού αγώνα, την παρακολούθηση της κίνησης ενός ροπάλου cricket, την παρακολούθηση ενός ατόμου ή ενός οχήματος σε ένα βίντεο κ.λπ.

Το “Computer Vision” ή αλλιώς «η μηχανική όραση» είναι ένα από τα πιο συναρπαστικά τμήματα της επιστήμης των υπολογιστών. Ο προγραμματισμός ενός υπολογιστή και ο σχεδιασμός αλγορίθμων για την κατανόηση του τι βρίσκεται μέσα από τις εικόνες ή τα βίντεο είναι το πεδίο αυτού που ονομάζουμε “Computer Vision”. Η γλώσσα προγραμματισμού “Python” έρχεται με πολλές ελεύθερα διαθέσιμες ισχυρές ενότητες για την ανάλυση και επεξεργασία εικόνας και βίντεο.

Ο κύριος σκοπός της διατριβής αυτής είναι να μελετηθεί η κίνηση οχημάτων σε κυκλικούς κόμβους χρησιμοποιώντας ένα σύστημα ανίχνευσης, καταμέτρησης και παρακολούθησης οχημάτων, μέσω επεξεργασίας εικόνας και βίντεο. Πέρα από την παρακολούθηση της κίνησης των οχημάτων, η διατριβή αυτή στοχεύει και στην πρόβλεψη των πιθανών σημείων ρηγματώσεων του οδοστρώματος, αφού με το σύστημα παρακολούθησης της πορείας οχημάτων εντοπίζεται η πιο συχνή πορεία κάποιου οχήματος που κινείται μέσα σε ένα κυκλικό κόμβο, άρα συνεπακόλουθα η περιοχή πιο συχνής χρήσης του συγκεκριμένου οδοστρώματος. Το σύστημα αυτό έχει δημιουργηθεί στη γλώσσα προγραμματισμού “Python” και έχει χρησιμοποιηθεί κυρίως η βιβλιοθήκη “OpenCV”. Σημειώνεται ότι, ο κώδικας της συγκεκριμένης διατριβής επικεντρώνεται σε βίντεο τα οποία έχουν μαγνητοσκοπηθεί από μη επανδρωμένα αεροσκάφη, γνωστά και ως “Drones”.

ABSTRACT

In the last two decades, we have entered the digital era, with the convergence of telecommunication, video, and informatics. It is more and more difficult to track and access this content via database search engines due to the enormous and increasing amounts of digital images and videos. Content-based indexing via automatic object detection and recognition techniques has become one of the most important and challenging issues for the years to come, to face the limitation of traditional information systems (Hirano, et al., 2006). Object detection system is also used in tracking objects, for example tracking a ball during a football match, tracking the movement of a cricket bat, tracking a person or a vehicle in a video, etc. (Prasanna, et al., 2019).

Computer vision is one of the most exciting divisions of computer science. Computer vision uses image processing algorithms to analyse and understand visuals from a single image or a sequence of images. The Python programming language (Van Rossum, G. & Drake, F.L., 2009) comes with many freely available powerful modules for image/video analysing and processing.

The main purpose of this thesis is to study the movement (traffic) of vehicles in roundabouts using a built-up detection, counting, and tracking system, through image and video processing. In addition, this dissertation aims to predict the possible areas of cracks in the road surface, since the vehicle tracking system detects the most common path of a vehicle moving in a roundabout, so the most commonly used area of this surface. This system generated in the Python programming language and the "OpenCV" library (Bradski, G., 2000) was mainly used. It is noted that the code of this thesis focuses on videos that have been filmed by unmanned aerial vehicles, known as "Drones".

ACKNOWLEDGEMENT

Foremost, I would like to express my sincere gratitude to my advisor Prof. Symeon Christodoulou, for the support, patience, motivation, enthusiasm, as well as continuous encouragement for undertaking this research. His guidance helped me in all the time of research and writing of this thesis. His contribution was of enormous importance to everything I learned throughout the study and the write-up of this thesis.

Besides my advisor, I would also like to acknowledge Prof. Loukas Dimitriou, to whom I am gratefully indebted for his precious aspect on this thesis. He gave me the opportunity to develop my knowledge in transportation research and inspired me to write this dissertation based on that field.

To my friends, thank you for your thoughts, well-wishes/prayers, phone calls, e-mails, texts, visits, editing advice, and being there whenever I needed a friend.

Finally, my deep and sincere gratitude to my parents and family, for their continuous and unparalleled love, help and support. I am grateful to my sisters and brothers for always being there for me as friends. I am forever indebted to my parents for giving me the opportunities and experiences that have made me who I am. They selflessly encouraged me to explore new directions in life and seek my destiny.

This accomplishment would not have been possible without any of them, and I dedicate this achievement to them.

Thank you

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Motivation

Currently, there is a strong drive within the construction and transportation industry to automate and integrate the multitude of data sources and to modernize the mechanisms for calculating and managing information. A well-designed road mobility data management system is a significant improvement over the current management methods. It allows quick and efficient changes to face many anomalies on the road network.

Several problems with the current road network have been identified. The proposed system will be able to record, at a first level, the mobility over a roundabout in the form of a video from a drone. At a next level, will process the video to extract the road distress, the total number of vehicles, as well as the total number of vehicles per entry/exit for a specific time period.

The continuous increase use of cars, as the primary means of transport, induces to the rapid increase of traffic congestion on the road network. This situation affects our lives, particularly in the urban areas, since people need, more and more, to move rapidly between different places. The results are traffic congestion, accidents, transportation delays and higher vehicle pollution emissions.

The road network of Cyprus should be improved. This cannot be done by constantly building new roads, which requires a lot of cost and time, but by improving the mobility and quality of existing ones. We notice that the biggest traffic jam occurs at the roundabouts. Roundabouts have a maximum vehicle service capacity. That is why their locations should be selected after an analysis of the existing traffic.

But what about existing roundabouts that need improvement? This question is answered by this dissertation.

There are three basic strategies to relieve congestion (Xiaoping & Mengting, 2009): The first strategy is to increase the transportation infrastructure. However, this strategy is costly and can only be accomplished in the long term. The second strategy is to limit traffic demand or make travelling more expensive, which will be strongly disapproved by travelers. The third strategy is to focus on the efficient and intelligent utilization of the existing transportation infrastructures. This strategy seems the best trade-off and is gaining more and more attention (Raiyn & Toledo, 2014). Currently, an Intelligent Transportation System (ITS) is the most promising approach to the implementation of the third strategy.

1.2 Aims and Work Objectives

This project aims to develop an effective response to the challenges currently faced by roads and especially roundabouts. It is intended that the research findings will contribute to the development of a framework or toolkit designed to support municipalities in managing road congestion and road distress.

These above aims raise the following core project objectives:

- To evaluate the current condition of roads and traffic congestion.
- To propose ways in which these specific aspects may be addressed.
- To support municipalities in piloting approaches to meeting these aspects.

The long-term goal of the project is to increase the number of municipalities that use drones to control and analyse the traffic on their roads.

1.3 Outline-Thesis Organization

The dissertation is organized into five Chapters.

The introductory chapter (Chapter 1) embraces the prime motivation, objectives, related work, and a brief organization of the dissertation.

Chapter 2 describes, in brief, the findings of the literature relevant to the computer and machine vision. The review is focused on studies considering detection methods and applications.

Chapter 3 highlights the theoretical background of image processing: an introduction to OpenCV, NumPy and image basics, background subtraction, morphological transformations, image thresholding, contours, etc.

Chapter 4 discusses the proposed methodology.

Chapter 5 presents a real-life application of the system. Exported analysis charts/figures from the algorithms discussed in previous chapters are presented with a discussion on their accuracy.

Chapter 6 presents a summary and the conclusions drawn from the performed research work and suggestions on related future work.

2 LITERATURE REVIEW

A review focused on studies and techniques considering the computer vision field are presented in this section.

Computer vision is an interdisciplinary scientific field that deals with how computers can gain a high-level understanding from digital images or videos (Boscacci, 2018). From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do (Zhanyu Ma, et al., 2018).

The scientific discipline of computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, multi-dimensional data from a 3D scanner or medical scanning device. The technological discipline of computer vision seeks to apply its theories and models to the construction of computer vision systems.

In the late 1960s, computer vision began at universities which were pioneering artificial intelligence. It was meant to mimic the human visual system, as a steppingstone to endowing robots with intelligent behaviour. In 1966, it was believed that this could be achieved through a summer project, by attaching a camera to a computer and having it "describe what it saw" (Lalan, n.d.).

What distinguished computer vision from the prevalent field of digital image processing at that time was a desire to extract three-dimensional structure from images to achieve full scene understanding. Studies in the 1970s formed the early foundations for many of the computer vision algorithms that exist today, including extraction of edges from images, labelling of lines, non-polyhedral and polyhedral modelling, representation of objects as interconnections of smaller structures, optical flow, and motion estimation.

The next decade saw studies based on more rigorous mathematical analysis and quantitative aspects of computer vision. These include the concept of scale-space, the inference of shape from various cues such as shading, texture and focus, and contour models known as snakes. Researchers also realized that many of these mathematical concepts could be treated within the same optimization framework as regularization and Markov random fields. By the 1990s, some of the previous research topics became more active than the others. At the same time, variations of graph cut were used to solve image segmentation. This decade also marked the first-time statistical learning techniques were used in practice to recognize faces in images. Toward the end of the 1990s, a significant change came about with the increased interaction between the fields of computer graphics and computer vision. This included image-based rendering, image morphing, view interpolation, panoramic image stitching and early light-field rendering.

Recent work has seen the resurgence of feature-based methods, used in conjunction with machine learning techniques and complex optimization frameworks. The advancement of Deep Learning techniques has brought further life to the field of computer vision. The accuracy of deep learning algorithms on several benchmark computer vision data sets for tasks ranging from classification, segmentation and optical flow has surpassed prior methods.

Computer vision techniques, in conjunction with acquisition through unmanned aerial vehicles (UAVs), offer promising non-contact solutions to civil infrastructure condition assessment. The ultimate goal of such a system is to automatically and robustly convert the image or video data into actionable information.

There are relevant researches in the fields of computer vision, machine learning, and structural engineering. These researches are classified into two types: inspection applications and monitoring applications. The inspection applications include identifying context such as structural components, characterizing local and global visible damage, and detecting changes from a reference image. The monitoring applications include static measurement of strain and displacement, as well as the dynamic measurement of displacement for modal analysis.

For inspection applications, researchers frequently envision an automated inspection framework that consists of two main steps: a) utilizing UAVs for automated data acquisition; b) data processing using computer vision techniques (F. Spencer, et al., 2019).

Intelligent UAVs are no longer a thing of the future, and the rapid growth in the drone industry over the last few years has made UAVs a viable option for data acquisition. These efforts have primarily focused on taking photographs and videos that are used for onsite evaluation or subsequent virtual inspections by engineers. The ability to automatically and robustly convert images or video data into actionable information is still challenging (F. Spencer, et al., 2019).

3 THEORETICAL BACKGROUND OF IMAGE PROCESSING

3.1 An introduction to OpenCV

Initially developed by Intel, OpenCV (Open Source Computer Vision) is a free cross-platform library for real-time image processing that has become "a de facto" standard tool for all things related to Computer Vision. The first version was released in 2000 under BSD licence, and since then, its functionality has been very much enriched by the scientific community. In 2012, the non-profit foundation OpenCV.org took on the task of maintaining a super site for developers and users.

OpenCV is available for the most popular operating systems, such as GNU/Linux, OS X, Windows, Android, iOS, and some more. The first implementation was in the C programming language; however, its popularity grew with its C++ implementation as of Version 2.0. New functions are programmed with C++. Yet, nowadays, the library has a full interface for other programming languages, such as Java, Python, and MATLAB/Octave (García, et al., 2015).

Python (Van Rossum, G. & Drake, F.L., 2009) is a general-purpose programming language started by Guido van Rossum, which became very popular in a short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability. Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++.

The support of Numpy (Oliphant, T.E., 2006) makes the task easier. Numpy is a highly optimized library for numerical operations that gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases the number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

3.2 NumPy and Image Basics

What is an image, what is a pixel and how does a computer represent image data?

The smallest element of an image is called a pixel, or a picture element. All images consist of pixels which are the raw building blocks of images. An image contains multiple pixels arranged in rows and columns. A 640 x 480 image has 640 columns (the width) and 480 rows (the height). There are 640 * 480 = 307200 pixels in an image with those dimensions.

Each pixel in a grayscale image has a value representing the shade of gray. In OpenCV, there are 256 shades of gray, from 0 to 255 (figure 1). So, a grayscale image would have a grayscale value associated with each pixel.

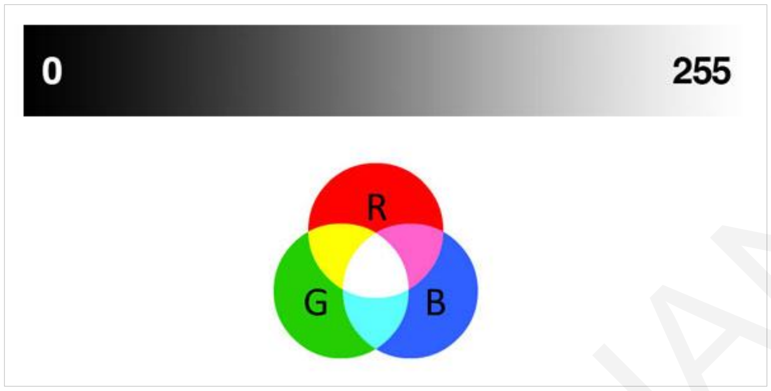


Figure 1: RGB colour model

Pixels in a colour image have additional information. In OpenCV colour images in the RGB (Red, Green, Blue) colour space, have a 3-tuple associated with each pixel: (B, G, R). Additive colour mixing allows us to represent a wide variety of colours by simply combining different amounts or R, G, B.

Each value in the BGR 3-tuple has a range of [0, 255] and the shape of the colour array then has three dimensions (figure 2): a) Height, b) Width, and c) Colour Channels. Therefore, when you read in an image and check its shape, it will look something like: *(1280,720,3)*, 1280 for pixel width, 720 for pixel height and 3 for colour channels.

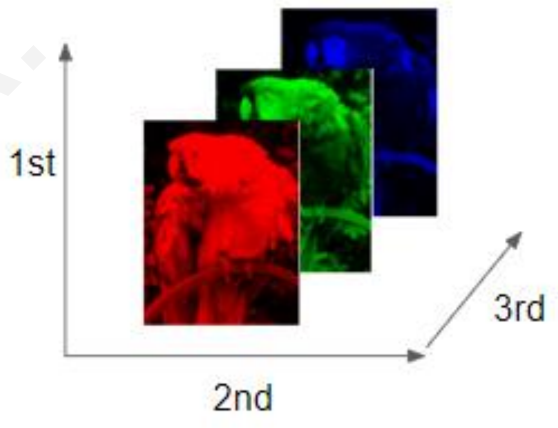


Figure 2: Shape of the colour array
(Source: Udemy course - Python for Computer Vision with OpenCV and Deep Learning)

Notice the ordering is BGR rather than RGB. This is because when OpenCV was first being developed many years ago, the standard was BGR ordering. Over the years, the standard has now become, RGB but OpenCV still maintains this "legacy" BGR ordering to ensure no existing code breaks.

Since a computer only understands numbers, every image is represented as an array. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with an extensive collection of high-level mathematical functions to operate on these arrays.

How a computer represents image data, it is easy to be explained through a simple example.

Let us suppose that we have a simple image of a handwritten number. Each single digit image can be represented as an array. For the following image (figure 3), a number is 28 by 28 pixels.

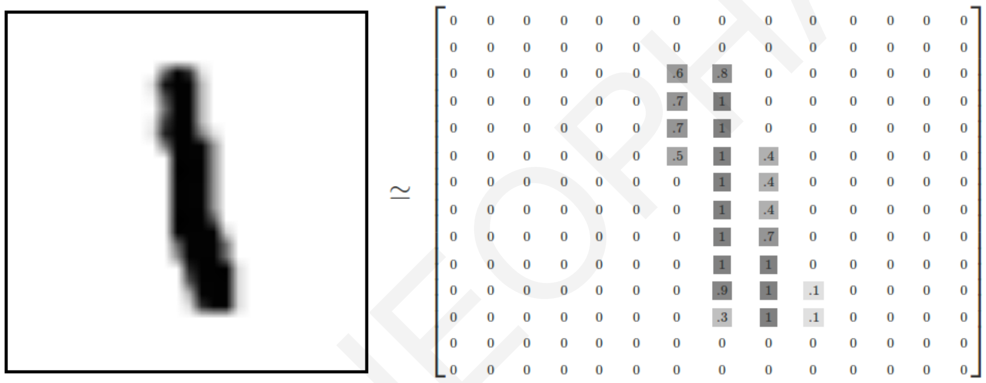


Figure 3: Handwritten number in pixels

How dark a pixel should be can be represented as a value between 0 and 1. It is common that the default images have values between 0 and 255. The range of 0 to 255 has to do with how computers store 8-bit numbers. But all the values can be divided by 255 to normalize to between 0 and 1.

3.3 Image Processing

Image processing is a method to perform some operations on an image, to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image (Mary, 2011). Nowadays, image processing is among rapidly growing technologies. It forms the core research area within engineering and computer science disciplines too.

3.3.1 Background subtraction

Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model (figure 4), containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.

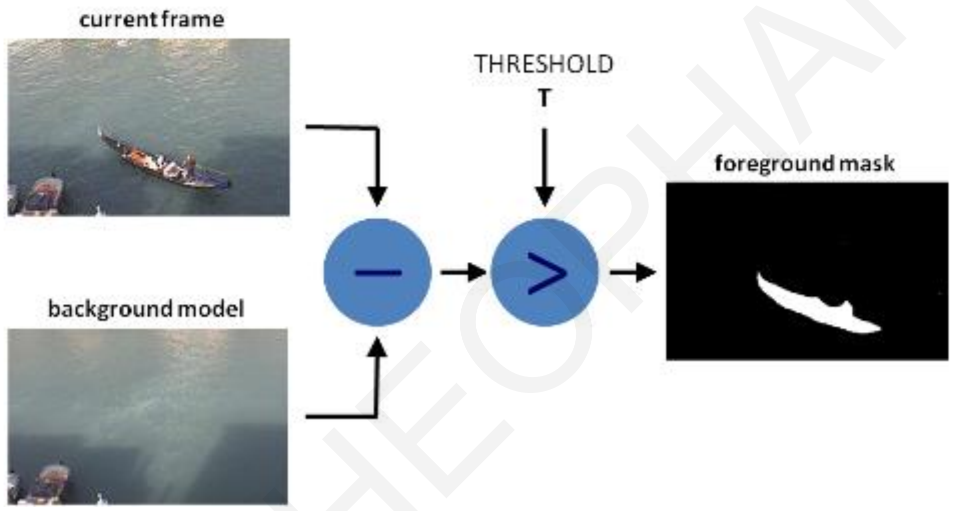


Figure 4: Background subtraction (Anon., 2020)

Background modelling consists of two main steps:

a. Background Initialization
b. Background Update

In the first step, an initial model of the background is computed, while in the second step that model is updated to adapt to possible changes in the scene.

<u>Syntax:</u> *cv.createBackgroundSubtractorMOG2(history, varThreshold, detectShadows)*

Parameters:

- history: Length of the history.
- varThreshold: Threshold on the squared Mahalanobis distance between the pixel and the model to decide whether a pixel is well described by the background model. This parameter does not affect the background update.
- detectShadows: If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false.

3.3.2 Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is usually performed on binary images. It needs two inputs; one is the original image, the second one is called a structuring element or kernel which decides the nature of the operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc. also comes into play.

3.3.2.1 Erosion

The basic idea of erosion is just like soil erosion only, it erodes the boundaries of the foreground object. So, what does it do? The kernel slides through the image. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise, it is eroded (made to zero).

So, what happens is that all the pixels near the boundary will be discarded depending upon the size of the kernel. The thickness or size of the foreground object decreases or white region decreases in the image. It is useful for removing small white noises, detach two connected objects etc.(figure 5-6).

<u>Syntax:</u> *cv2.erode(src, kernel, dst, anchor, iterations, borderType, borderValue)*

Parameters:

- src: It is the image which is to be eroded
- kernel: A structuring element used for erosion. If element = Mat(), a 3 x 3 rectangular structuring element is used. The kernel can be created using getStructuringElement
- dst: It is the output image of the same size and type as src
- anchor: It is a variable of type integer representing anchor point and its default value Point is (-1, -1) which means that the anchor is at the kernel center
- borderType: It depicts what kind of border to be added. It is defined by flags like cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc
- iterations: It is the number of times erosion is applied
- borderValue: It is border value in case of a constant border

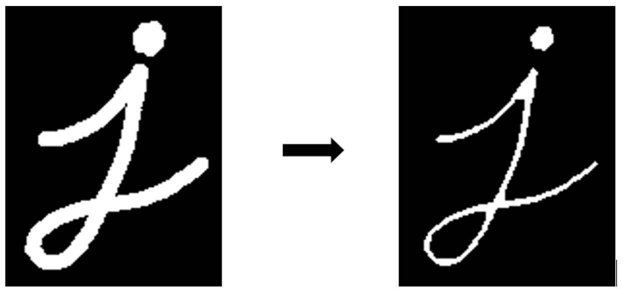


Figure 5: Morphological transformations: Erosion (a)

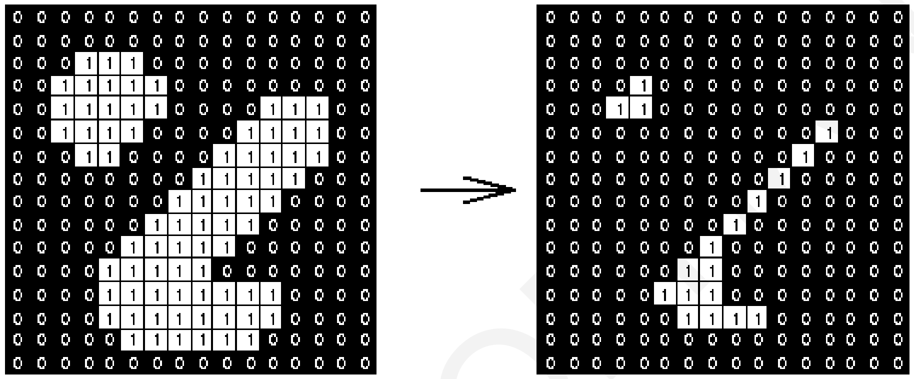


Figure 6: Morphological transformations: Erosion (b)

3.3.2.2 Dilation

It is just opposite of erosion. In this case, a pixel element is '1' if at least one pixel under the kernel is '1'. So, it increases the white region in the image or size of foreground object increases (figure 7-8). Usually, in cases like noise removal, erosion is followed by dilation. Because erosion removes white noises, but it also shrinks the object. Since noise is gone, they won't come back, but the object's area increases. It is also useful in joining broken parts of an object.

<u>Syntax:</u> *cv2.dilate(src, kernel, anchor, iterations, borderType, borderValue)*

The parameters are the same as mentioned for erosion.

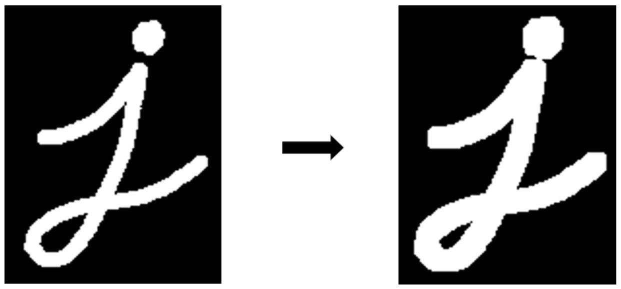


Figure 7: Morphological transformations: Dilation (a)

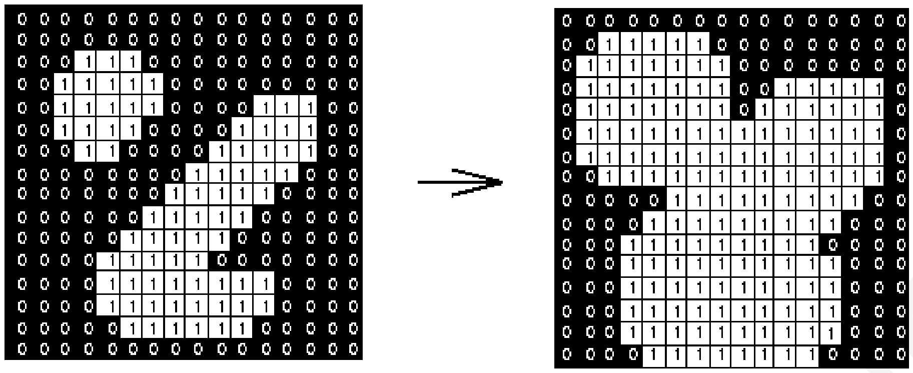


Figure 8: Morphological transformations: Dilation (b)

3.3.2.3 Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as explained above. Opening operation can open a gap between objects connected by a thin bridge of pixels (figure 9). Any regions that have survived the erosion are restored to their original size by the dilation. Opening is an idempotent operation: once an image has been opened, subsequent openings with the same structuring element have no further effect on that image.

<u>Syntax:</u> *cv2.morphologyEx(src, cv2.MORPH_OPEN, kernel)*

Parameters:

- src: Input Image array
- cv2.MORPH_OPEN: Applying the Morphological Opening operation
- kernel: Structuring element

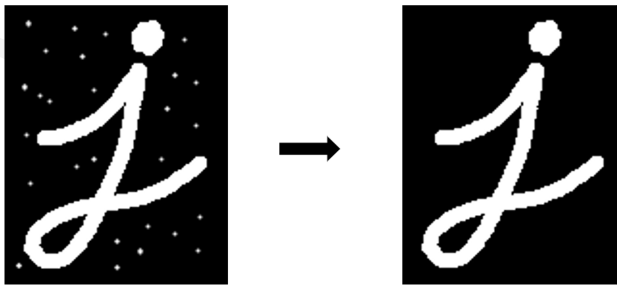


Figure 9: Morphological transformations: Opening

3.3.2.4 Closing

Closing is reverse of opening, dilation followed by erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object. Closing operation can fill gaps in the regions while keeping the initial region sizes (figure 10-11). Similar to opening, closing is idempotent, and it is the dual operation of opening (just as opening is the dual operation of closing). In other words, closing (opening) of a binary image can be performed by taking the complement of that image, opening (closing) with the structuring element, and taking the complement of the result.

Syntax: *cv2.morphologyEx(src, cv2.MORPH_CLOSE, kernel)*

Parameters:

- src: Input Image array
- cv2.MORPH_CLOSE: Applying the Morphological Closing operation
- kernel: Structuring element

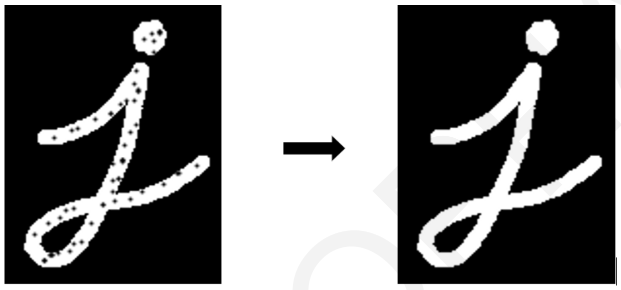


Figure 10: Morphological transformations: Closing (a)

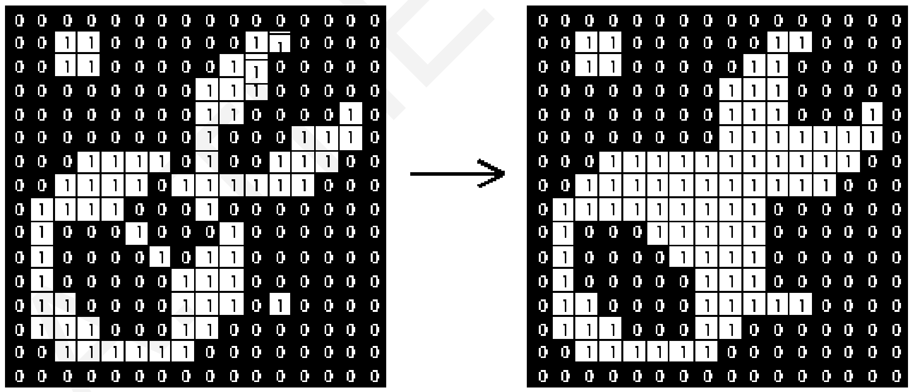


Figure 11: Morphological transformations: Closing (b)

3.3.3 Image Thresholding

Thresholding is a technique in OpenCV, which is the assignment of pixel values with the threshold value provided. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0. Otherwise, it is set to a maximum value (generally 255).

Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background. A threshold is a value which has two regions on its either side, i.e. below the threshold or above the threshold.

In Computer Vision, this technique of thresholding is done on grayscale images. So initially, the image must be converted in grayscale colour space.

Syntax: *cv2.threshold(src, thresholdValue, maxVal, thresholdingTechnique)*

Parameters:

- src: Input Image array (must be in Grayscale)
- thresholdValue: Value of Threshold below and above which pixel values will change accordingly
- maxVal: Maximum value that can be assigned to a pixel
- thresholdingTechnique: The type of thresholding to be applied

There are five types of thresholding (figure 12):

1. THRESH_BINARY: This merely turns any value above and below your threshold into the minimum possible and maximum potential value, respectively.
2. THRESH_BINARY_INV: This does the opposite of a THRESH_BINARY where any value above and below your threshold into the maximum possible and minimum potential value respectively.
3. THRESH_TRUNC: Values lower than the threshold value are unchanged, but any value higher is set to the threshold value of 127.
4. THRESH_TOZERO: Pixel values lower than the threshold value are set to the 0 and value above are unchanged.
5. THRESH_TOZERO_INV: Pixel values less than the threshold value are fixed, and the value above is set to 0.

The basic Thresholding technique is Binary Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0. Otherwise, it is set to a maximum value.

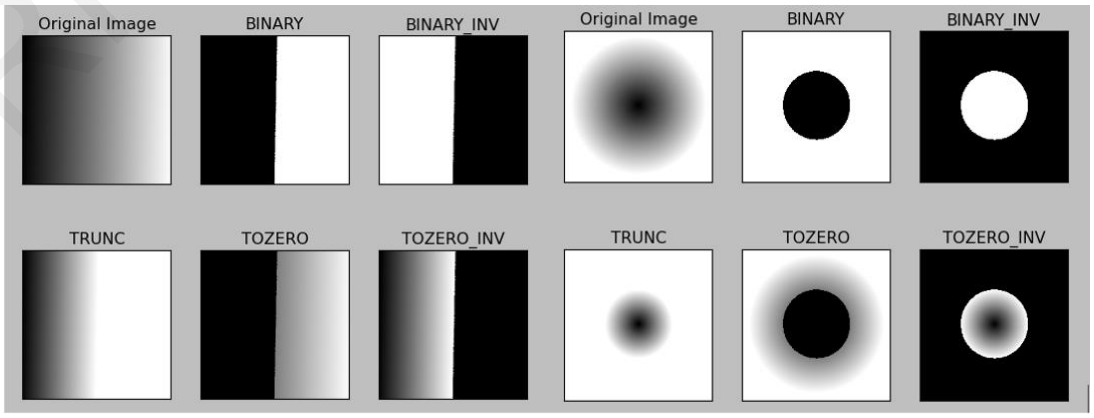


Figure 12: Types of thresholding (Ranjit, 2019)

3.4 Contours in OpenCV

You may be already familiar with the word 'contour'. A contour line indicates a curved line representing the boundary of the same values or the same intensities. A contour map is the most straightforward example.

Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. Contours come handy in shape analysis, finding the size of the object of interest, and object detection.

What's the difference between edges and contours? The two terms are often used interchangeably so it could be a bit confusing. To put it simply, the concept of edges lies in a local range while the concept of contours is at the overall boundary of a figure. Edges are points whose values change significantly compared to their neighbouring points. Contours, on the other hand, are closed curves which are obtained from edges and depicting a boundary of figures.

OpenCV has *cv2.findContour()* function that helps in extracting the contours from the image. It works best on binary images, so we should first apply thresholding techniques, Sobel edges, etc.

There are three essential arguments in cv2.findContours() function. First one is the source image, second is contour retrieval mode, third is contour approximation method, and it outputs the image, contours, and hierarchy. 'Contours' is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object.

It returns three values: a) Input image array, b) Contours, and c) Hierarchy. Hierarchy is the parent-child relationship in contours. It is represented as an array of four values: [Next contour, previous contour, First child contour, Parent contour].

Syntax: *cv2.findContours(src, contour_retrieval, contours_approximation)*

Parameters:

- src: Input Image of n – dimensional array (n = 2,3) but preferred 2-dim binary images for better result
- contour_retrieval: This is contour retrieval mode. Possible values are: cv2.RETR_TREE, cv2.RETR_EXTERNAL, cv2.RETR_LIST, cv2.RETR_CCOMP etc.
- contours_approximation: This is Contour approximation method. Possible values are: cv2.CHAIN_APPROX_NONE and cv2.CHAIN_APPROX_SIMPLE

OpenCV provides *the cv2.drawContours()* function, which is used to draw the contours. It is also used to draw any shape by providing its boundary points.

Syntax: *cv.DrawContours(src, contours, contourIdx, colour, thickness)*

Parameters:

- src: the name of the image
- contours: All the input contours.
- contourIdx: Parameter indicating a contour to draw. If it is negative, all the contours are drawn
- colour: Colour of the contours
- thickness is how thick are the lines drawing the contour

3.4.1 Contour Features (Moments, Contour Area)

The word 'moment' is a short time period in common usage. But in physics terminology, a moment is the product of the distance and another physical quantity meaning how a physical quantity is distributed or located. In computer vision, image moment is how image pixel intensities are distributed according to their location. It's a weighted average of image pixel intensities and we can get the centroid or spatial information from the image moment.

In image processing, computer vision and related fields, an image moment is a certain particular weighted average (moment) of the image pixels' intensities or a function of such moments, usually chosen to have some attractive property or interpretation.

The function *cv2.moments()* gives a dictionary of all moment values calculated.

There are three types of moments- spatial moments, central moments, and central normalized moments. We can get the image moment with the function *cv2.moments()* in OpenCV, and it returns 24 different moments. If you print the output M as shown below, it'll return the 24 moments in a dictionary format.

For the moments, you can extract useful data like area, centroid, etc. A centroid is given by the relations, $C_x = \frac{M_{10}}{M_{00}}$ and $C_y = \frac{M_{01}}{M_{00}}$. This can be done as follows:

```
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

Contour area is given by the function *cv.contourArea()* or from moments, M['m00'].

3.4.2 Hierarchy

What is the Hierarchy? With the *cv2.findContours()* function we detect objects in an image. Sometimes objects are in different locations. But in some cases, some shapes are inside other shapes, just like nested figures. In this case, we call the outer one as a parent and the inner one as a child. This way, contours in an image has some relationship to each other. And we can specify how one contour is connected, like, is it the child of some other contour, or is it a parent etc. Representation of this relationship is called the Hierarchy.

How is the hierarchy Representation in OpenCV? Each contour has its own information regarding what hierarchy it is, who is its child, who is its parent etc. OpenCV represents it as an array of four values: [Next, Previous, First_Child, Parent].

- Next denotes next contour at the same hierarchical level.
- Previous denotes the previous contour at the same hierarchical level.
- First_Child denotes its first child contour.
- Parent denotes the index of its parent contour.

Consider an example image below:

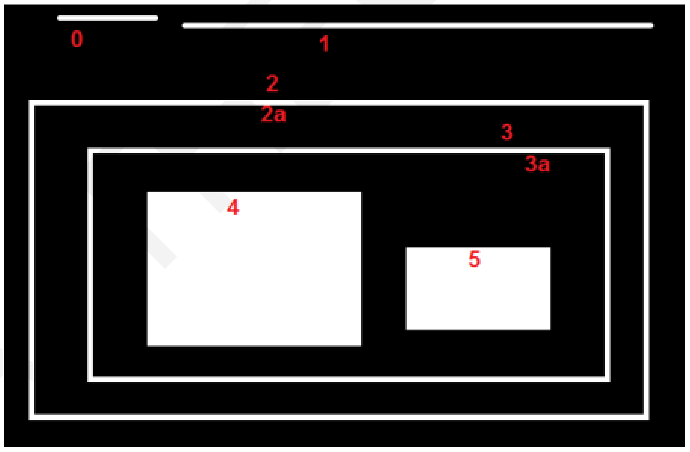


Figure 13: Contour hierarchy example

In figure 13, there are a few shapes which have numbered from 0-5. 2 and 2a denotes the external and internal contours of the outermost box.

Here, contours 0,1,2 are external or outermost. We can say, they are in hierarchy-0 or they are in the same hierarchy level. Next comes contour-2a. It can be considered as a child of contour-2 (or oppositely, contour-2 is the parent of contour-2a). So, let it be in hierarchy-1. Similarly, contour-3 is the child of contour-2, and it comes in the next hierarchy. Finally, contours 4,5 are the children of contour-3a, and they come in the last hierarchy level. From the way I numbered the boxes, I would say contour-4 is the first child of contour-3a (It can be contour-5 also).

4 METHODOLOGY

As previously mentioned, this work consists of the development of a vehicle recognition system through the use of machine vision. The system will be able to detect, track and count the vehicles which are in the video and frame them with the type of vehicle (motorcycle, car, or bus/truck). The system must be able to:

- Identify vehicle's type.
- Store the number of vehicles per entrance or exit of the roundabout.
- Store the coordinates of vehicles' route (centroids).

4.1 Difference between detection and tracking

Object detection and tracking models have been a source of inspiration for many tracking-by-detection algorithms over the past decade. However, detection and tracking are commonly confused words. Object detection is simply about identifying and locating all known objects in a scene, and object tracking is about locking onto a particular moving object(s) in real-time.

The two are similar, however:

- Object detection can occur on still photos while object tracking needs video feed.
- Object detection can be used as object tracking if we run object detection every frame per second.
- Object tracking does not need to identify the objects. Objects can be tracked based solely on motion features without knowing the actual objects being tracked. Thus, pure object tracking can be extremely efficient if it leverages the temporal relationship between video frames.
- Running object detection as tracking can be computationally expensive, and it can only track known objects.

Thus, object detection requires a form of classification and localization, while for object tracking, classification can be unnecessary depending on the problem. In practice, real-time object detection can be sped up by employing object tracking and running classification once a few frames. Object detection can run on a slow thread looking for objects to lock onto and once those objects are locked on then object tracking, running in a faster thread, can takeover.

The main aim of this thesis, is to design a new tracking-by-detection algorithm, that is based on detection by contours and tracking by the location of every contour in each frame. This method is the simplest way to detect objects and track them (store the path) without being computationally expensive.

This project was developed with the use of a drone's camera. Animated videos are great for explaining a complex idea. Thus, at the first level, the whole concept of this thesis will be explained and tested in an animated aerial video (figure 14) and later in a real aerial video.

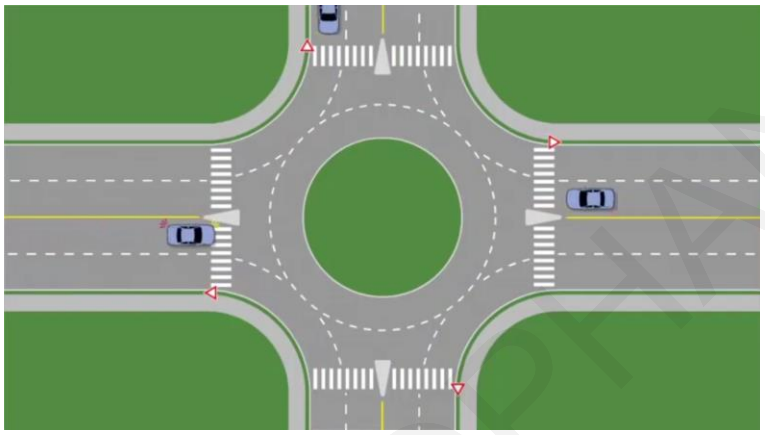


Figure 14: Animated aerial video

One of the main objectives of this chapter is to detect obstacles and see the performance of a simple animated video and a real-life video. This chapter will introduce the use cases in practice. All the use-cases were implemented in Python. The complete codes with comments are available in the appendixes of this thesis.

4.2 Step 1: Import Video, Draw Entrance/Exit Lines, Create background subtractor

The goal in this first step had four initiatives. The first was to import and read the video of interest, the second was to find video's properties (frames per second, width, height, etc.), the third part of this step was drawing the entrance and exit lines of the roundabout, and the last one was to generating the foreground mask.

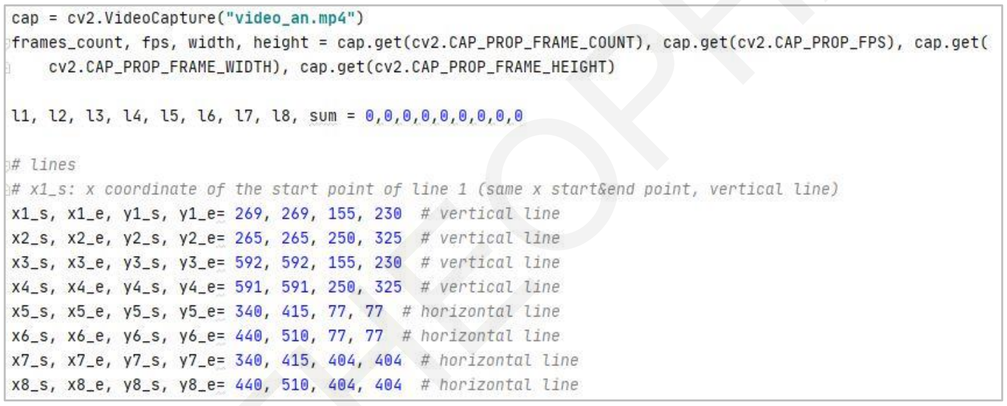
In OpenCV, a video can be read either by using the feed from a camera connected to a computer or by reading a video file. The first step towards reading a video file is to create a VideoCapture object. Its argument can be either the device index or the name of the video file to be read.

```
# Create a VideoCapture object and read from an input file
cap = cv2.VideoCapture("video_an.mp4")
```

There are many properties which we can read using the method cap.get(). In this thesis work, we read the following properties (number of frames, frames per second, width, height):

- frames_count = cap.get (cv2.CAP_PROP_FRAME_COUNT)
- fps = cap.get (cv2.CAP_PROP_FPS)
- width = cap.get (cv2.CAP_PROP_FRAME_WIDTH)
- height = cap.get (cv2.CAP_PROP_FRAME_HEIGHT)

The third part of this step was to define the region of interest; creating a line by the coordinates of each entrance and exit of the roundabout (figure 15-16). Based on those lines, we will be able at a later stage to check and count how many vehicles passed through each line. For this reason, these lines had to be either horizontal or vertical. If there were diagonals, the span of control would be increased, and there would be lots of errors.



```
cap = cv2.VideoCapture("video_an.mp4")
frames_count, fps, width, height = cap.get(cv2.CAP_PROP_FRAME_COUNT), cap.get(cv2.CAP_PROP_FPS), cap.get(
    cv2.CAP_PROP_FRAME_WIDTH), cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

l1, l2, l3, l4, l5, l6, l7, l8, sum = 0,0,0,0,0,0,0,0,0

# lines
# x1_s: x coordinate of the start point of line 1 (same x start&end point, vertical line)
x1_s, x1_e, y1_s, y1_e= 269, 269, 155, 230  # vertical line
x2_s, x2_e, y2_s, y2_e= 265, 265, 250, 325  # vertical line
x3_s, x3_e, y3_s, y3_e= 592, 592, 155, 230  # vertical line
x4_s, x4_e, y4_s, y4_e= 591, 591, 250, 325  # vertical line
x5_s, x5_e, y5_s, y5_e= 340, 415, 77, 77  # horizontal line
x6_s, x6_e, y6_s, y6_e= 440, 510, 77, 77  # horizontal line
x7_s, x7_e, y7_s, y7_e= 340, 415, 404, 404  # horizontal line
x8_s, x8_e, y8_s, y8_e= 440, 510, 404, 404  # horizontal line
```

Figure 15: Import Video, Extract video properties, Set Entrance/Exit Lines (code)

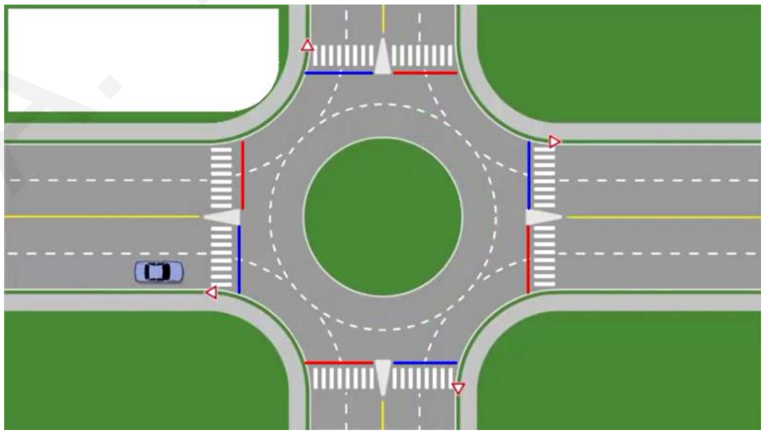


Figure 16: Define Entrance/Exit Lines

In the last part of the first step, the foreground mask is generated by performing a subtraction between the current frame and a background model, containing the static part of the scene.

sub = cv2.createBackgroundSubtractorMOG2()   #create background subtractor

"BackgroundSubtractorMOG" is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction" in 2004 (Zivkovic, 2004) and "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" in 2006 (Zivkovic & der Heijden, 2006). A critical feature of this algorithm is that it selects the appropriate number of gaussian distribution for each pixel. It provides better adaptability to varying scenes due to illumination changes etc.

After the VideoCapture object is created, the lines are defined, and the foreground mask is generated, we can capture the video frame by frame using a while loop (figure 17). A frame of a video is simply an image, and we display each frame with the function imshow().

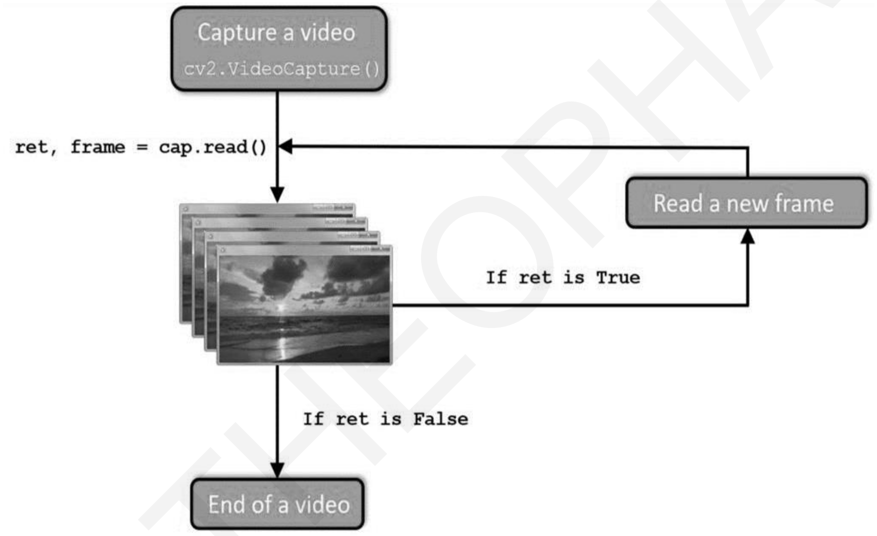
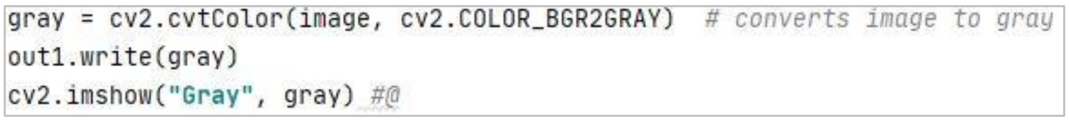


Figure 17: Example of while loop (frames)

4.3 Step2: Image/Frame processing

The goal in this step was to perform some image processing to video's frames in the loop.

The first image processing goal was to change the colour-spaces of the image. There are more than 150 colour-space conversion methods available in OpenCV. For colour conversion, we use the function *cv2.cvtColor(input_image, flag),* where flag determines the type of conversion. In this work, we only had to change the colour to grayscale (figure 18-19). For the gray conversion, we use the flags *cv2.COLOR_BGR2GRAY*.



```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converts image to gray
out1.write(gray)
cv2.imshow("Gray", gray) #@
```

Figure 18: Changing Colourspaces (code)

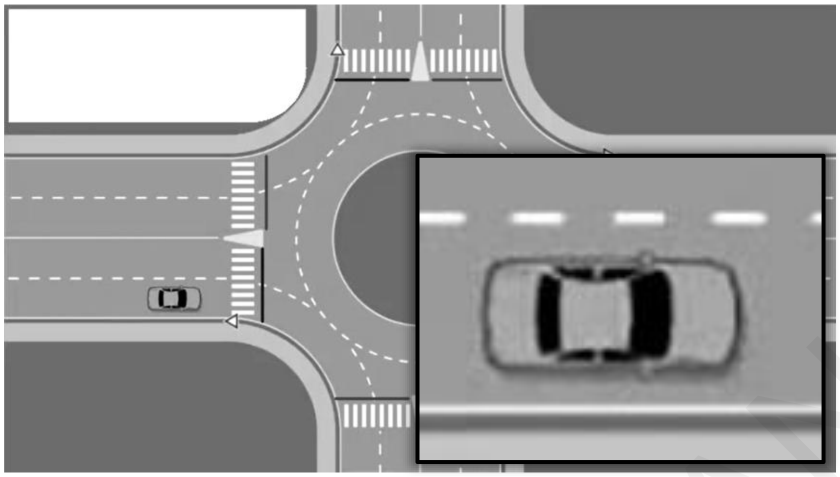
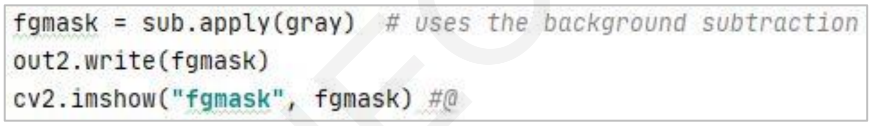


Figure 19: Changing Colourspaces (Grayscale)

In the previous step, we created the background subtractor using the *cv2.createBackgroundSubtractorMOG2()* function. Then inside the video loop, we use the backgroundsubtractor.apply() method in grayscale image to get the foreground mask (figure 20).



```
fgmask = sub.apply(gray)  # uses the background subtraction
out2.write(fgmask)
cv2.imshow("fgmask", fgmask) #@
```

Figure 20: Appling background subtraction (code)

Figure 21 shows that technically, we had extracted the moving foreground from static background. So, the only moving foreground on that frame was the car.

It would be better at this stage to apply some morphological operations to the result to remove the noises.

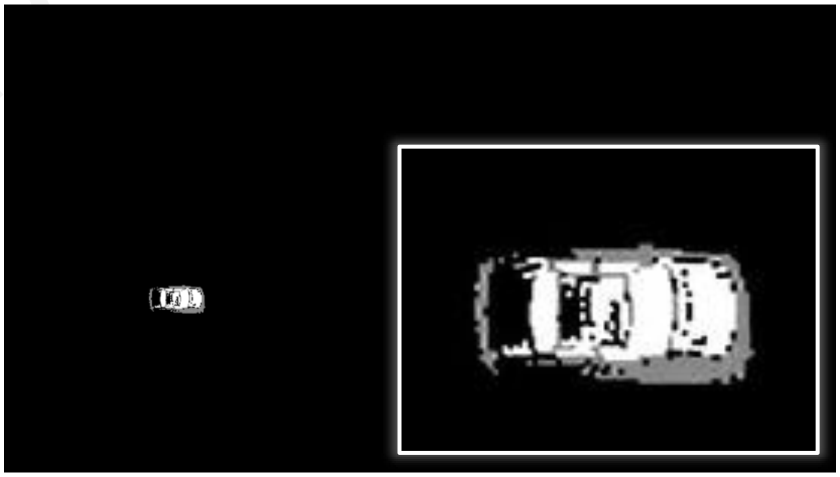


Figure 21: Result of BackgroundSubtractorMOG2

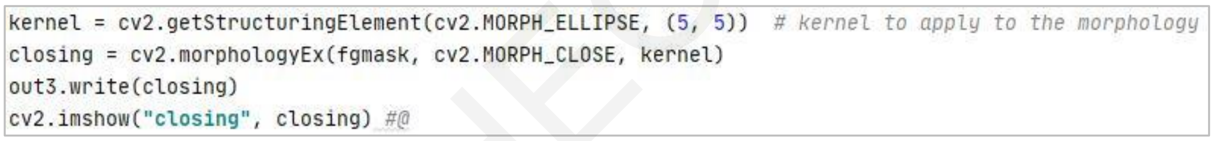
4.4 Step 3: Morphological Transformations to frames

The goal in this step was to apply some morphological operations to each frame. Morphological operators take an input image, and a structuring component as input and these elements are then combines using the set operators. It is typically performed on binary images, so we applied it on the grayscale image.

Closing:

Closing can be useful in closing small holes (i.e. "pepper") inside the foreground objects, or small black points on the object. In this work, we applied closing and then dilation operation. The dilation operation that follows ensures that dark regions that are larger than the structuring element retain their original size (figure 23).

Before applying the closing operation, we have defined the kernel (figure 22). Kernel is a structuring element which decides the nature of the operation. The kernel can be of any shape, but usually, it is a square and is defined with respect to the anchor point which is usually placed at the central pixel.



```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))  # kernel to apply to the morphology
closing = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
out3.write(closing)
cv2.imshow("closing", closing) #@
```

Figure 22: Closing operation (code)

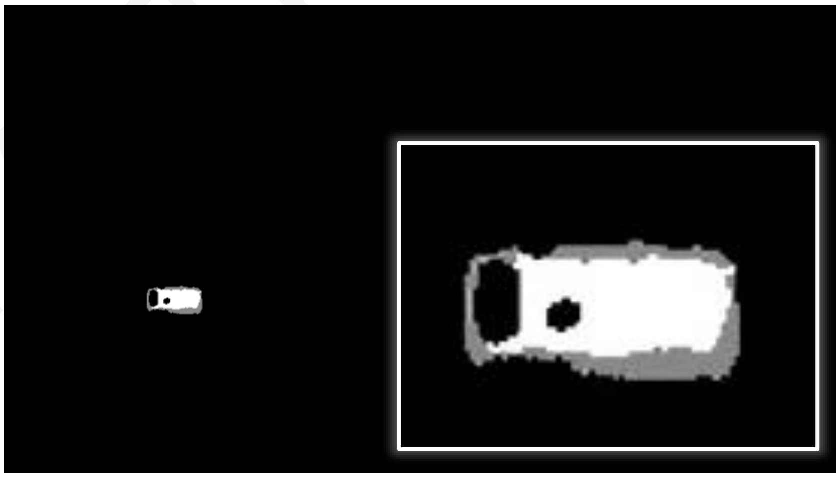
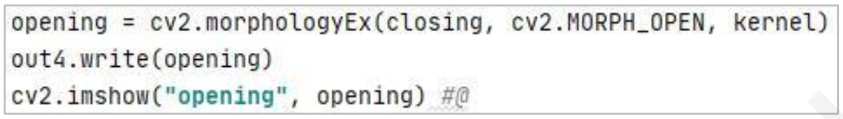


Figure 23: Result of Closing operation

Notice how the white dots get connected because of closing, but other dark region retains their original sizes.

<u>Opening:</u>

Opening can remove small bright spots (i.e. “salt”) and connect small dark cracks. The dilation operation that follows ensures that light regions that are larger than the structuring element retain their original size (figure 25).



```
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
out4.write(opening)
cv2.imshow("opening", opening) #@
```

Figure 24: Opening operation (code)

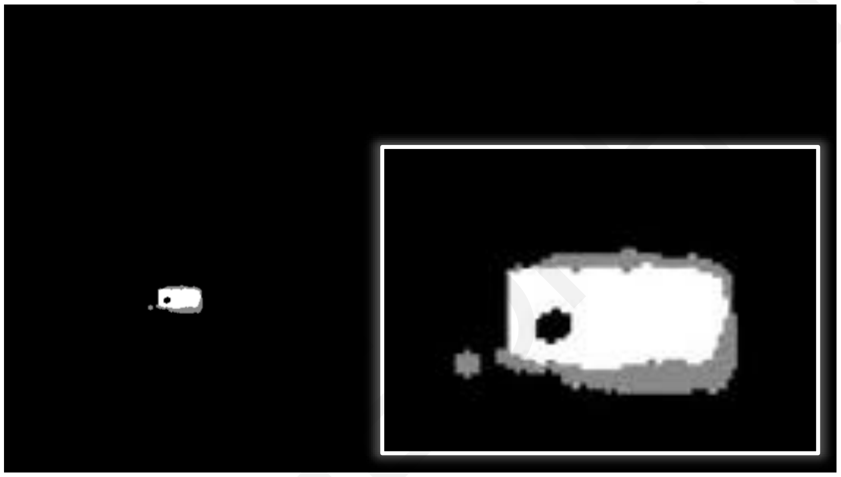
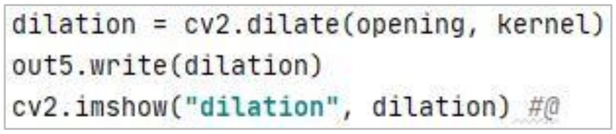


Figure 25: Result of Opening operation

<u>Dilation:</u>

Dilation is exactly what it sounds like. It is an addition (expansion) of bright pixels of the object in each image. It enlarges bright regions and shrinks dark regions. It is illustrated in the image below (figure 27).



```
dilation = cv2.dilate(opening, kernel)
out5.write(dilation)
cv2.imshow("dilation", dilation) #@
```

Figure 26: Dilation (code)

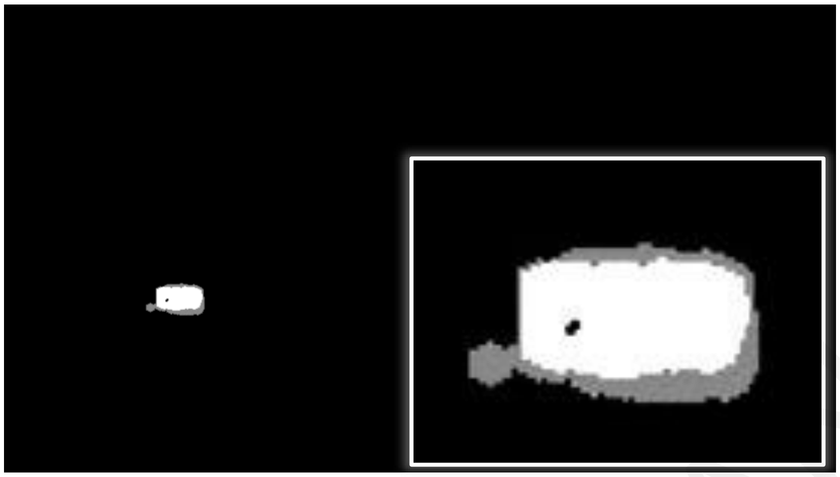


Figure 27: Dilated image

Notice how the white boundary of the image thickens or gets dilated (figure 27). Also notice the decrease in size of the black spot in the center, and the thickening of the light grey circle in the left part of the image.

<u>Image Thresholding:</u>

In the last image processing, we use functions to apply thresholding to an image. Most frequently, we use thresholding as a way to select areas of interest of an image, while ignoring the parts we are not concerned with. In this case, we use thresholding to remove shadows.

As it pertains to images, a histogram is a graphical representation showing how frequently various colour values occur in the image. Looking at the histogram of the grayscale image (figure 28), we can see clearly that there is a large number of gray pixels, as indicated in the chart by the spike around the grayscale value 160. That is not so surprising, since the original image is mostly gray background. Looking at the histogram of the dilated image (figure 29), we notice that there are lots of and black pixels and few gray pixels (shadows). On the last histogram (figure 30), those gray pixels disappear, because of the thresholding operation we have applied (binary image; only black and white pixels).

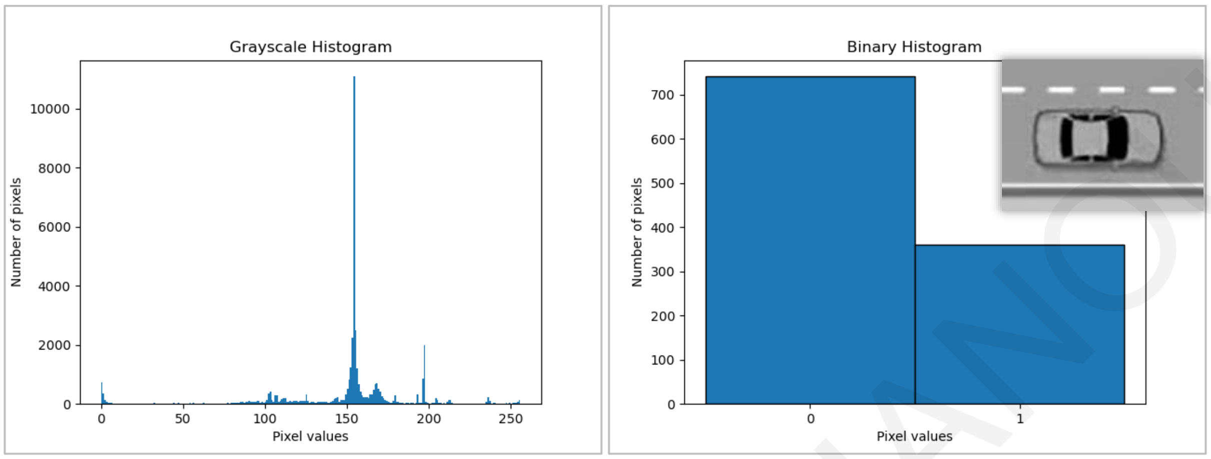


Figure 28: Histograms of a grayscale image

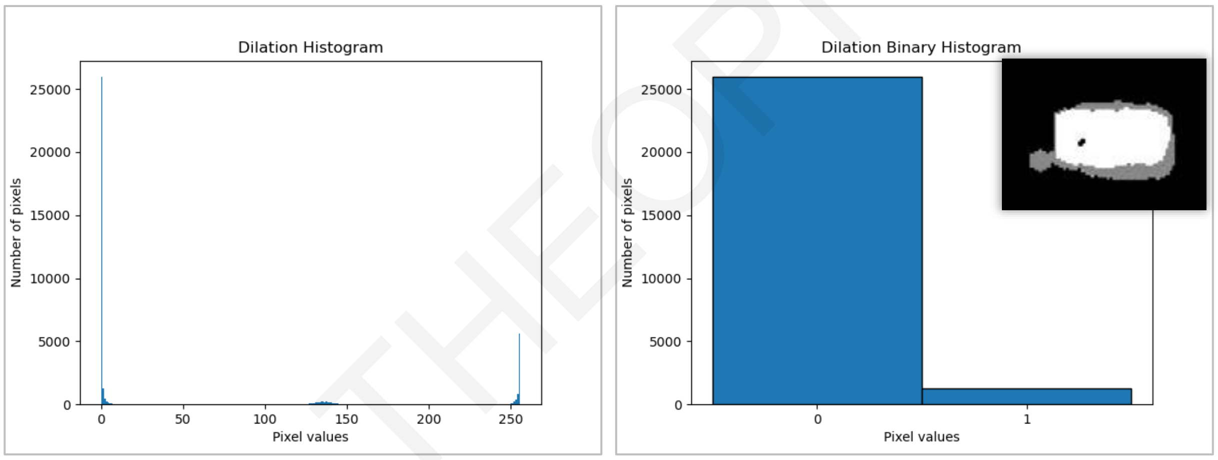


Figure 29: Histograms of a dilated image

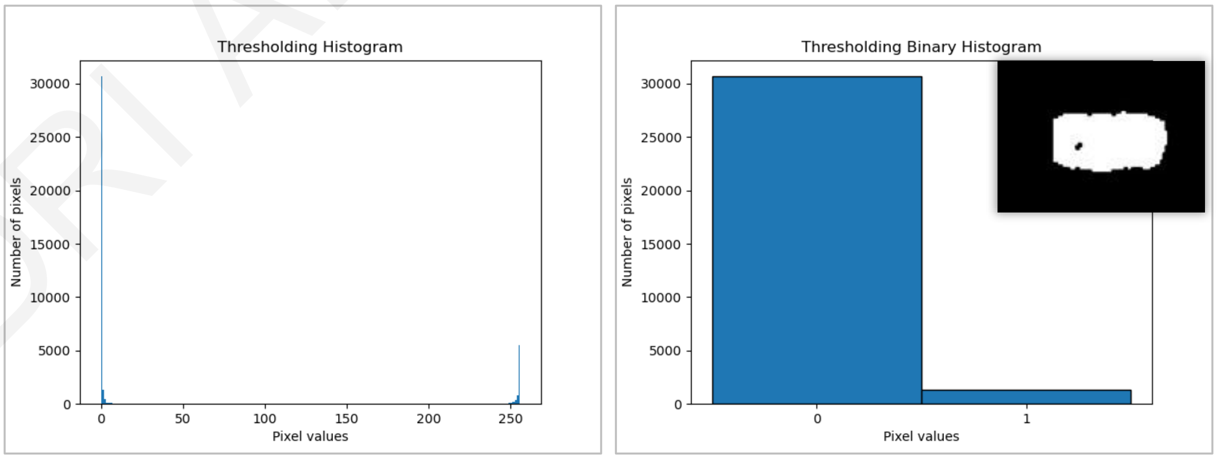


Figure 30: Histograms of a thresholding image

4.5 Step 4: Finding and drawing Contours in frames

This step focuses on finding and drawing contours on each frame of the video. Contours are continuous lines or curves that bound or cover the full boundary of an object in an image. Contours are critical in object detection. OpenCV has findContour() function that helps in extracting the contours from the image. It works best on binary images, so we have used the thresholding image for the previous step.

contours,hierarchy=cv2.findContours(bins,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

After finding all the contours of the frame, the next step was to find area and the co-ordinates of the centroid of the area. The Co-ordinates of each vertex of a contour is hidden in the contour itself. In this approach, we will be using Numpy library to convert all the co-ordinates of a contour into a linear array. This linear array would contain the x and y co-ordinates of each contour. The key point here is that the first co-ordinate in the array would always be the co-ordinate of the topmost vertex and hence could help in detection of orientation of the image.

It is important to note that before of finding the co-ordinates of the center, we use a while loop for hierarchy (figure 31) to only count parent contours (contours not within others). Furthermore, inside of this loop, we check only contours that have area between the minimum and maximum area. The minimum and maximum area have been defined manually at an early stage in the code.

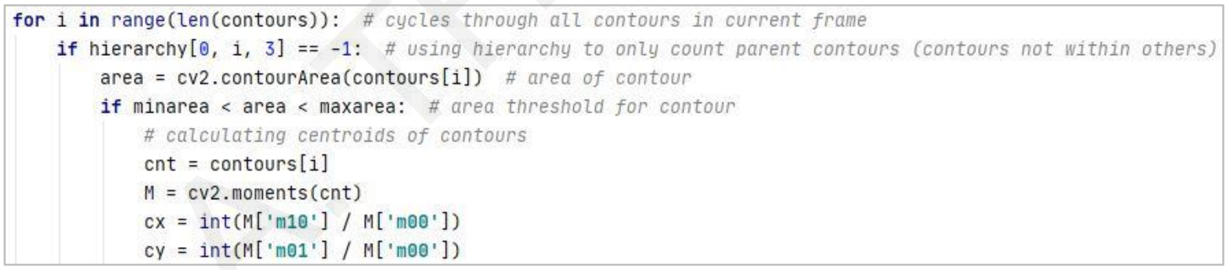

```
for i in range(len(contours)):  # cycles through all contours in current frame
    if hierarchy[0, i, 3] == -1:  # using hierarchy to only count parent contours (contours not within others)
        area = cv2.contourArea(contours[i])  # area of contour
        if minarea < area < maxarea:  # area threshold for contour
            # calculating centroids of contours
            cnt = contours[i]
            M = cv2.moments(cnt)
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])
```

Figure 31: Hierarchy and contour area (code)

We then use the cv2.drawContours() function to draw the contours of the image by finding the corners (x,y,w,h) as shown in the figure 32.

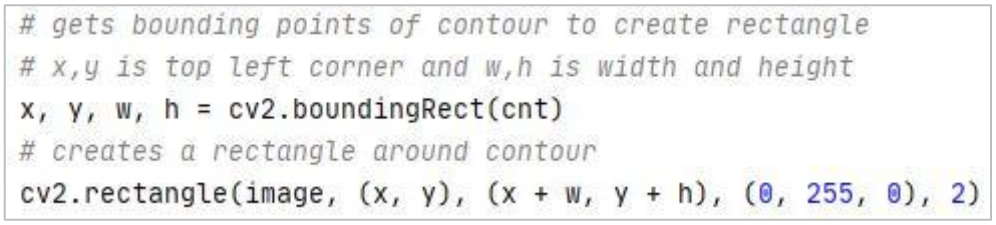

```
# gets bounding points of contour to create rectangle
# x,y is top left corner and w,h is width and height
x, y, w, h = cv2.boundingRect(cnt)
# creates a rectangle around contour
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Figure 32: Drawing contours (code)

4.6 Step 5: Identify vehicle's type

The goal in this step was to identify the vehicle's type by using the contour area. For this thesis work, there are three possible vehicle types (figure 33):

a. Motorcycle
b. Car
c. Bus/Truck

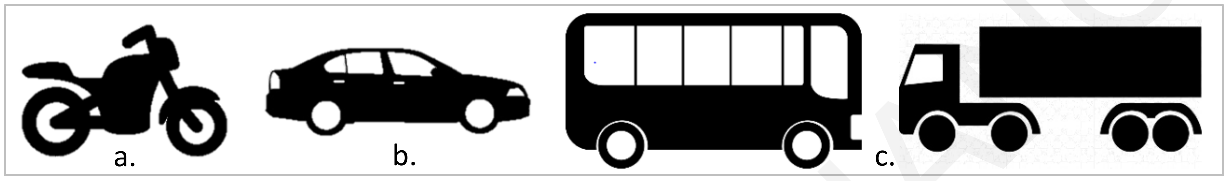
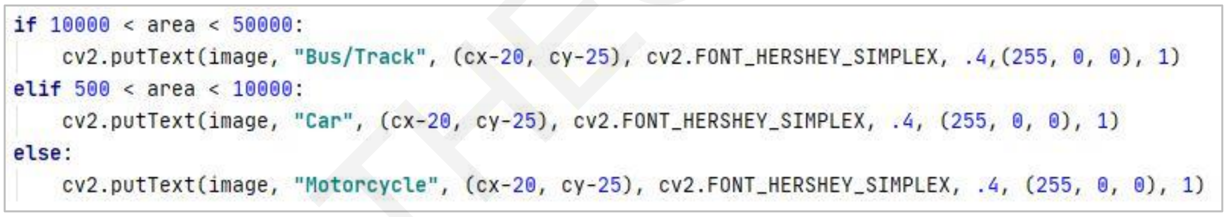


Figure 33: Types of vehicles

Contours that had an area between 1000 and 50000 are considered as buses or trucks. Areas between 500 and 10000 are considered as cars, and the rest of the contour areas are considered as motorcycles (figure 34). This range was manually defined by a trial and error method.



```
if 10000 < area < 50000:
    cv2.putText(image, "Bus/Track", (cx-20, cy-25), cv2.FONT_HERSHEY_SIMPLEX, .4,(255, 0, 0), 1)
elif 500 < area < 10000:
    cv2.putText(image, "Car", (cx-20, cy-25), cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)
else:
    cv2.putText(image, "Motorcycle", (cx-20, cy-25), cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)
```

Figure 34: Identify vehicle's type (code)

4.7 Step 6: Counting system

In videos, we will always have some small moving objects that we do not want to count. So, we have to define regions of interest. We will count a moving object only if it is in that region. In this case, the regions of interest are the entrances and exits of the roundabout.

In a previous step (step 1), we defined the lines of entrances and exits. In this step, we want to check and count how many vehicles passed through each line.

These lines, as described before, had to be either horizontal or vertical. If there were diagonals, the range of control would be increased, and there would be lots of errors. For example, if a line is horizontal, we only have to check if the center of contour passes through one line with a constant y-coordinate. In the same way, for vertical lines, we have to check if the center of contour passes through one line with a constant x-coordinate (figure 35).

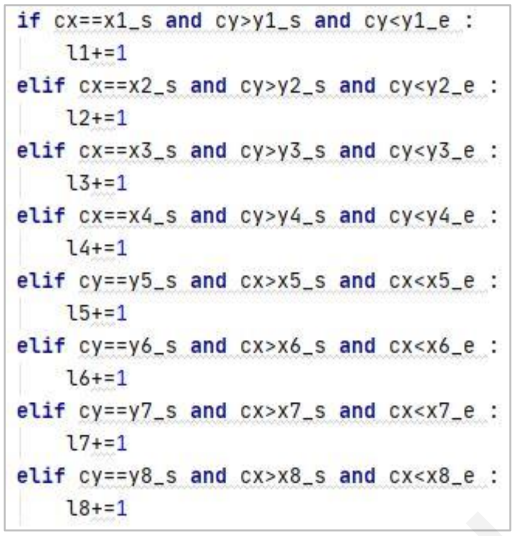

```
if cx==x1_s and cy>y1_s and cy<y1_e :
    l1+=1
elif cx==x2_s and cy>y2_s and cy<y2_e :
    l2+=1
elif cx==x3_s and cy>y3_s and cy<y3_e :
    l3+=1
elif cx==x4_s and cy>y4_s and cy<y4_e :
    l4+=1
elif cy==y5_s and cx>x5_s and cx<x5_e :
    l5+=1
elif cy==y6_s and cx>x6_s and cx<x6_e :
    l6+=1
elif cy==y7_s and cx>x7_s and cx<x7_e :
    l7+=1
elif cy==y8_s and cx>x8_s and cx<x8_e :
    l8+=1
```

Figure 35: Counting system (code)

By summing up all the vehicles passes by all entrance lines, we will have the total numbers of vehicles that enter the roundabout for a specific period (video duration). For example, in the following figure, the entrances are lines 2, 3, 5 and 8. So, the total number of vehicles that have entered the roundabout is four (figure 36).

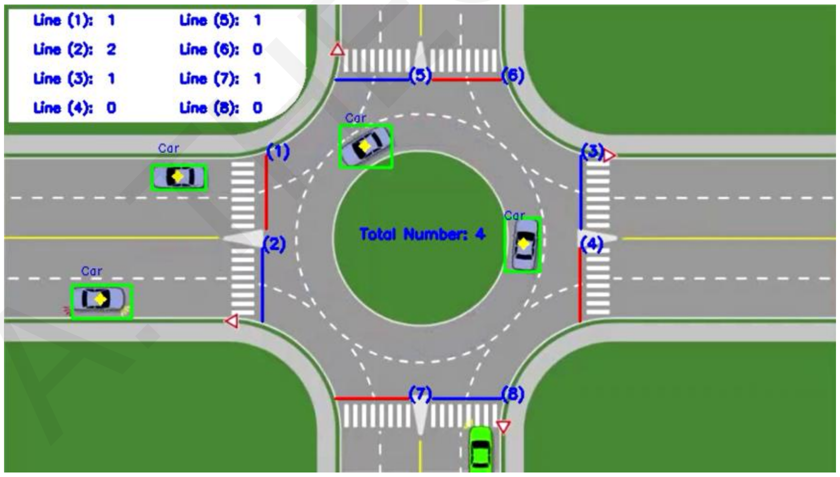
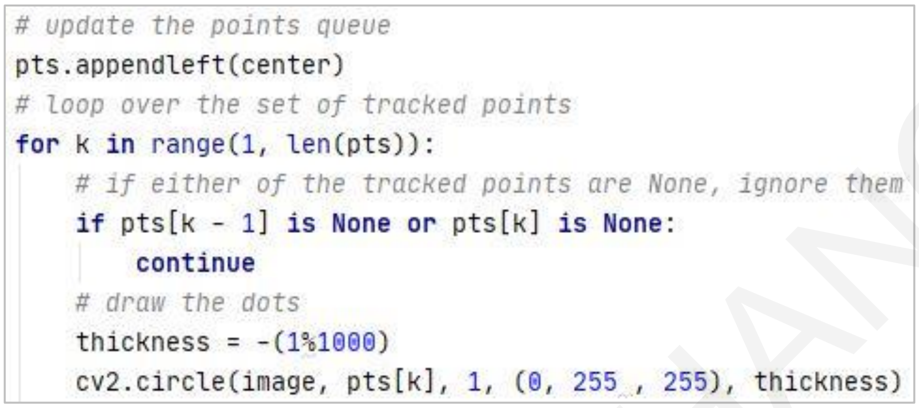


Figure 36: Total numbers of vehicles

4.8 Step 7: Contour-based tracking system

Contour based vehicle tracking is an efficient method for tracking vehicles. Using contours, we can detect the centroid of the vehicles, and this centroid will represent the vehicle. So, we can follow the path of every vehicle easily. And this contour-based vehicle tracking can be used in several computer vision applications.

In this case, we store the centroid of each contour in the list “pts” and display it using a yellow circle. Every time a contour is detected, this list updated with the new centroid and the new circle is displayed creating a path (figure 37-38).



```
# update the points queue
pts.appendleft(center)
# loop over the set of tracked points
for k in range(1, len(pts)):
    # if either of the tracked points are None, ignore them
    if pts[k - 1] is None or pts[k] is None:
        continue
    # draw the dots
    thickness = -(1%1000)
    cv2.circle(image, pts[k], 1, (0, 255 , 255), thickness)
```

Figure 37: Tracking system (code)

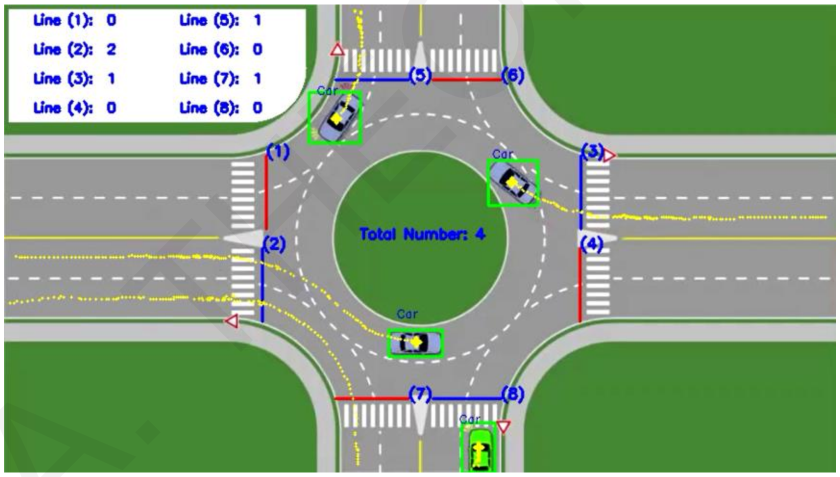


Figure 38: Tracking system (result)

4.9 Methodology Flow-chart

The flowchart in figure 39, is a simplification of the complete code and shows all the steps described before.

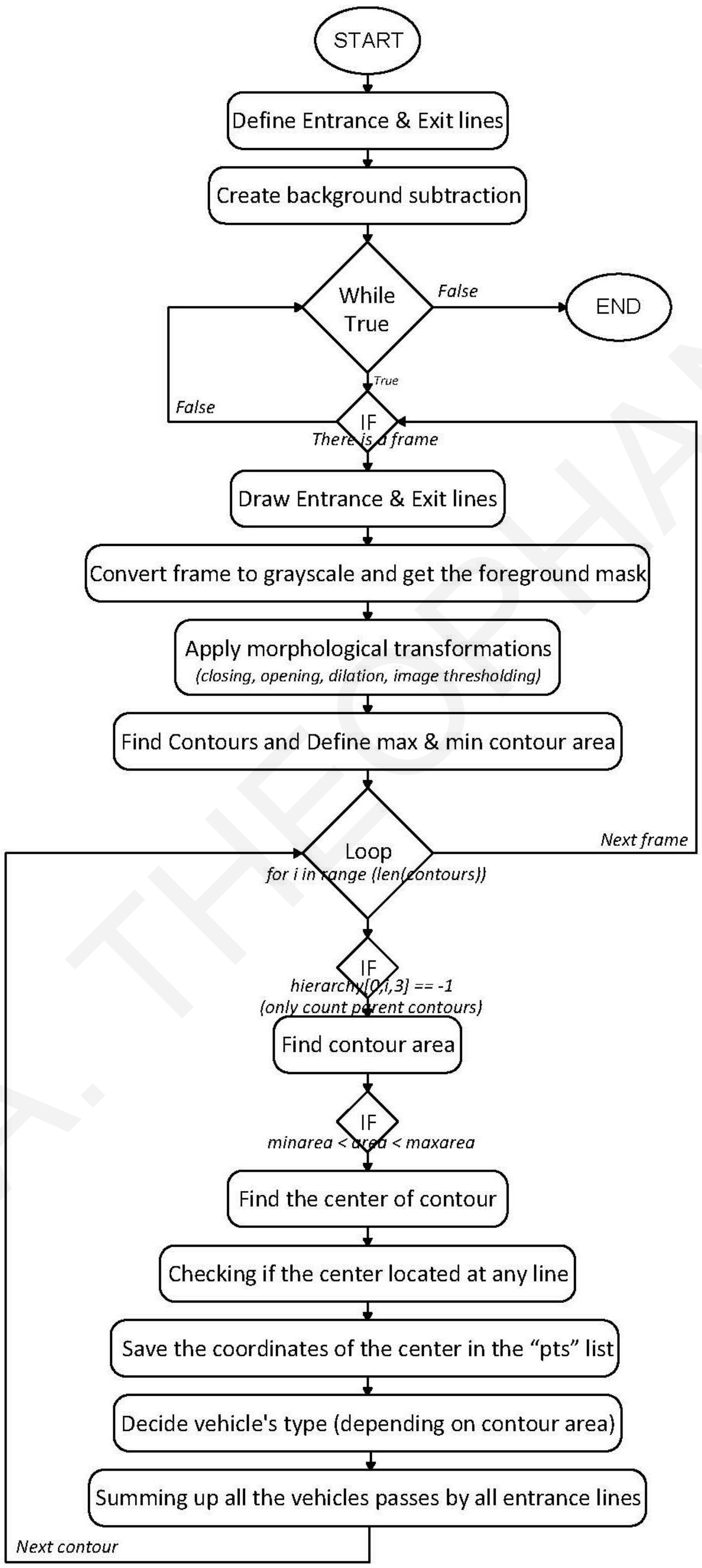

START
Define Entrance & Exit lines
Create background subtraction
While True
False
END
True
False
IF
There is a frame
Draw Entrance & Exit lines
Convert frame to grayscale and get the foreground mask
Apply morphological transformations
(closing, opening, dilation, image thresholding)
Find Contours and Define max & min contour area
Loop
for i in range (len(contours))
Next frame
IF
hierarchy[0,i,3] == -1
(only count parent contours)
Find contour area
IF
minarea < area < maxarea
Find the center of contour
Checking if the center located at any line
Save the coordinates of the center in the "pts" list
Decide vehicle's type (depending on contour area)
Summing up all the vehicles passes by all entrance lines
Next contour

Figure 39: Methodology Flow-chart

5 REAL-LIFE APPLICATION

5.1 Aerial video

Humans can recognize any object in the real world easily without any efforts, on contrary machines by itself cannot recognize objects. It is a challenging or difficult task in the image processing to detect, track, and count the objects into consecutive frames. Various challenges can arise due to complex object motion, the irregular shape of the object, occlusion of object to object and object to scene and real-time processing requirements.

In the previous chapter, an animated aerial video has been tested. In this chapter, a real-life aerial video is tested. This video was taken by another student of University of Cyprus at Polis Chrysochous, from a height of 90-120 meters above the roundabout (figure 40). This height is ideal for recognizing vehicles without particular errors. Polis is a small town at the north-west end of the island of Cyprus, at the centre of Chrysochous Bay, and on the edge of the Akamas peninsula nature reserve.

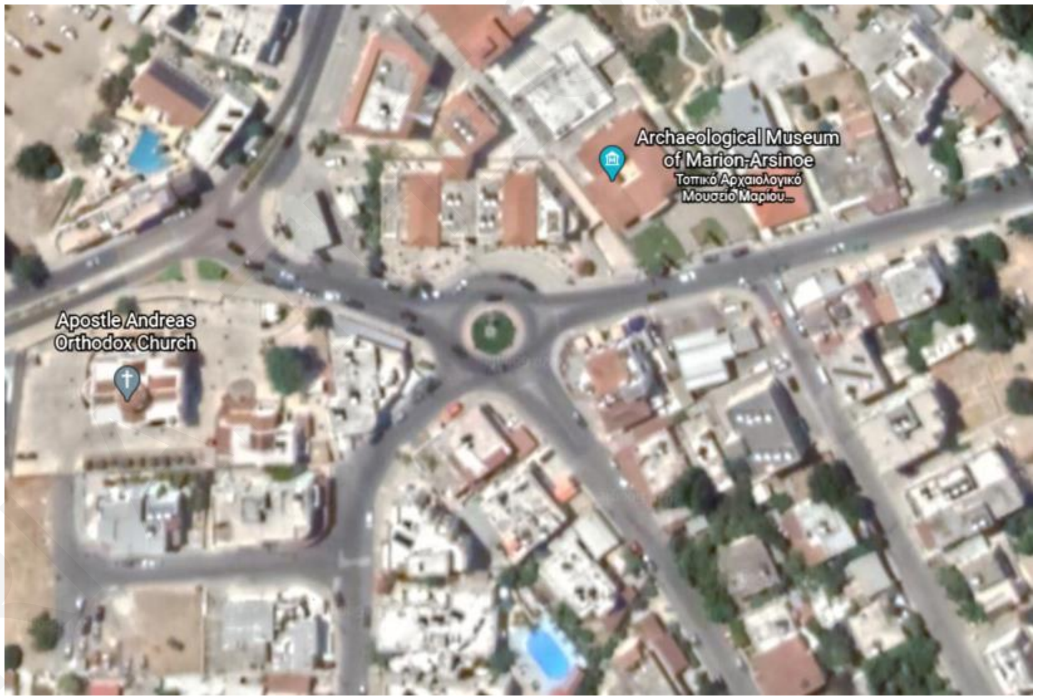

Archaeological Museum of Marion-Arsinoe
Τοπικό Αρχαιολογικό Μουσείο Μαρίου...
Apostle Andreas Orthodox Church

Figure 40: Location of roundabout tested in this thesis (Google Earth)

As anyone can easily understand, the real-life aerial video is more complex and challenging. The detection of vehicles is difficult as the background is not clear and contains many objects.

Figure 41 shows the actual video in a specific frame.

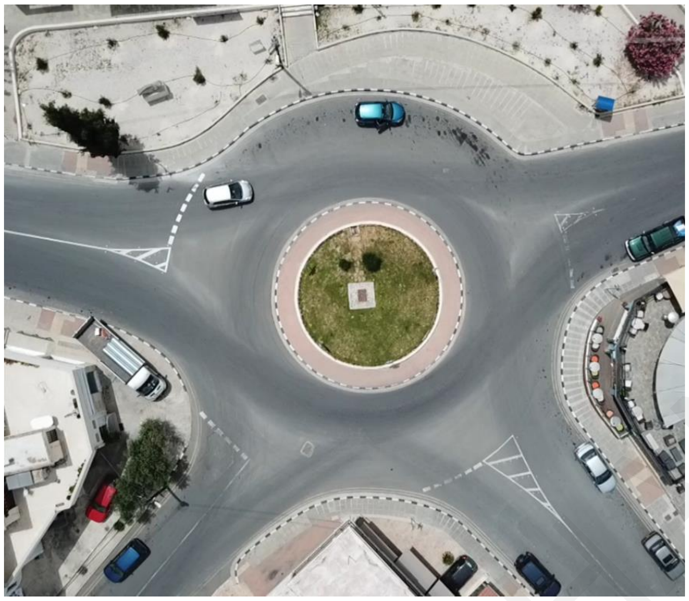
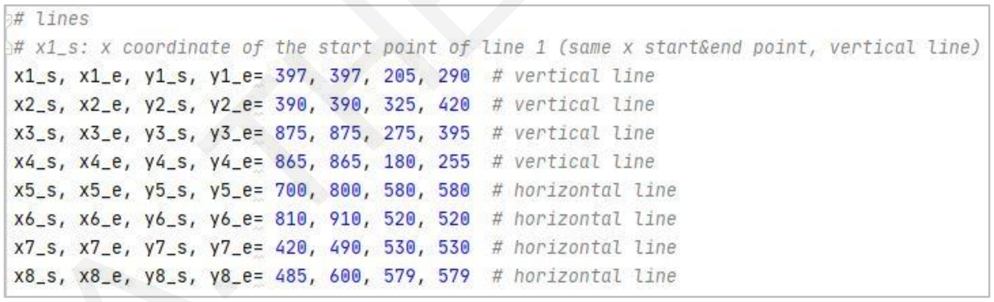


Figure 41: Real-life aerial video

The steps remain the same, as explained in chapter 4.

Firstly, we had to import the video, draw the entrance and exit lines (figure 42) and create the background subtractor.



```
# lines
# x1_s: x coordinate of the start point of line 1 (same x start&end point, vertical line)
x1_s, x1_e, y1_s, y1_e= 397, 397, 205, 290  # vertical line
x2_s, x2_e, y2_s, y2_e= 390, 390, 325, 420  # vertical line
x3_s, x3_e, y3_s, y3_e= 875, 875, 275, 395  # vertical line
x4_s, x4_e, y4_s, y4_e= 865, 865, 180, 255  # vertical line
x5_s, x5_e, y5_s, y5_e= 700, 800, 580, 580  # horizontal line
x6_s, x6_e, y6_s, y6_e= 810, 910, 520, 520  # horizontal line
x7_s, x7_e, y7_s, y7_e= 420, 490, 530, 530  # horizontal line
x8_s, x8_e, y8_s, y8_e= 485, 600, 579, 579  # horizontal line
```

Figure 42: Entrance/Exit Lines (real-life video)

Notice that the coordinates are not the same as the ones from the animated aerial video. The coordinates have to be defined manually by the user for each video separately. They are related to the position and height of the drone, and consequently, the camera. It is considered that the user of the specific code and drone will make a test flight and will note the coordinates of these lines so that he can use them in all other flights and analyses of traffic for the specific roundabout he will make.

The roundabout that tested, is not a typical roundabout with four entrances the one opposite the other forming a cross (with horizontal and vertical lines). This roundabout also has four exits but forming a strange shape, as shown in figure 43 and figure 44.

Under standard conditions, the entrance/exit lines should be as shown in figure 43.

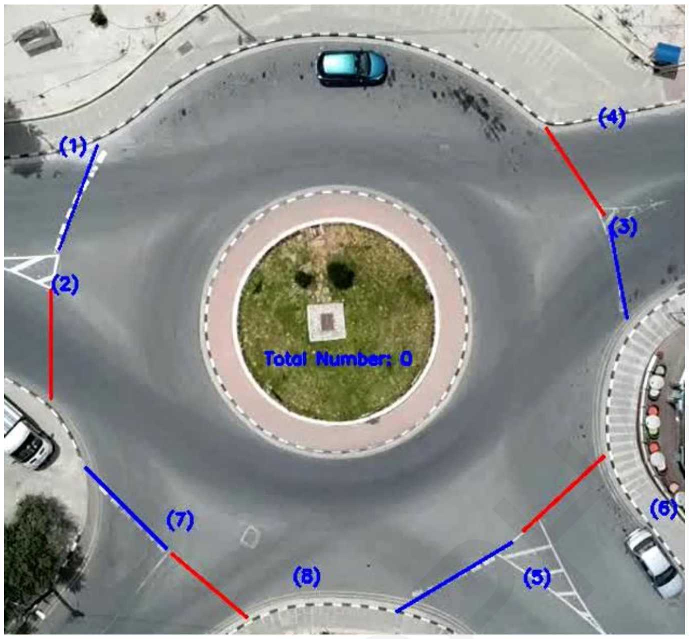


Figure 43: Diagonal Entrance/Exit Lines

Despite the situation of each roundabout that is tested, the lines must be vertical and horizontal in a location that a vehicle counted only if it is going to pass through the specific line of interest. So, the image and lines change as follows (figure 44):

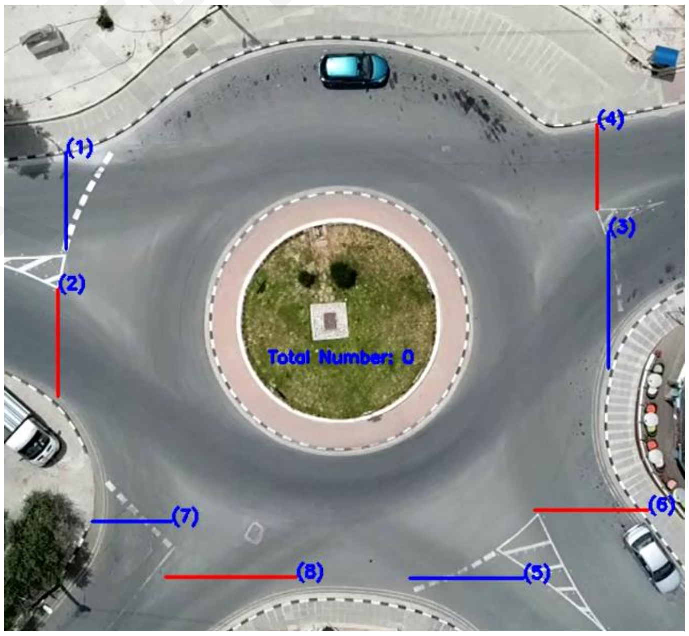


Figure 44: Horizontal and vertical Entrance/Exit Lines

As long as these lines are defined, the foreground mask is generated, and the next step is the most significant part of this code; the image/frame processing. The colour space of each frame changes to grayscale, as shown in figure 45.

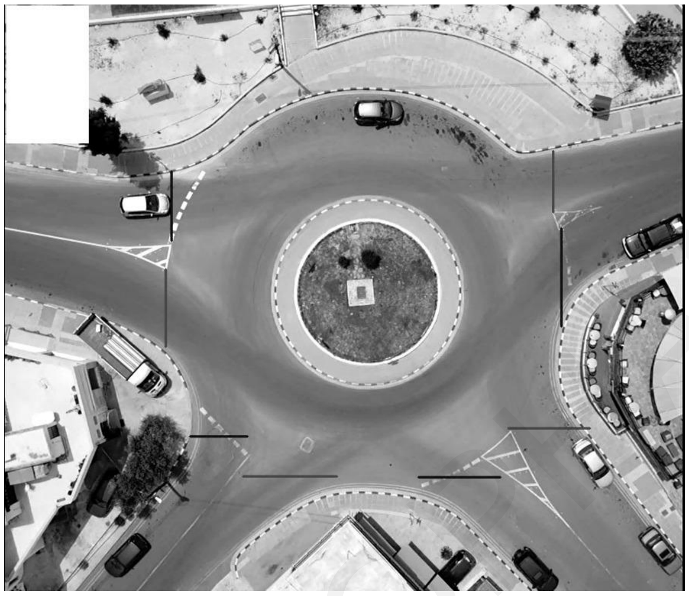


Figure 45: Grayscale frame (real-life video)

Using the grayscale frames, we apply the background subtraction to extract the moving foreground. It is worth mentioning that the stationary vehicle was recognized as part of the background (figure 46).

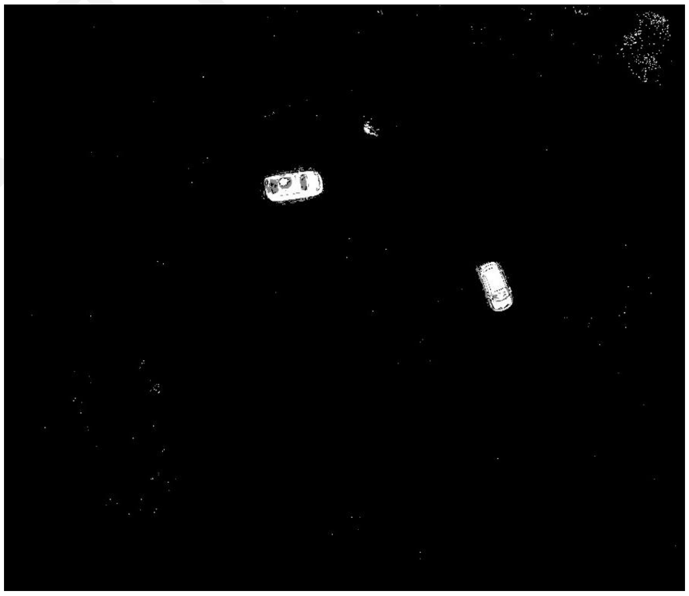


Figure 46: Background Subtractor

On the contrary, as shown in the figure above (figure 46), in the right corner of the frame, the tree is recognized as a moving object. Thus, we realize that the tree moves from the wind and, therefore, cannot be recognized as background.

For this reason, in a next step, we determine the minimum and maximum contour area, since we do not want small moving objects (such as the dots shown in the figure 46) to be recognized and measured in traffic. It was evident that the frames so far contain a lot of noise. By applying some morphological operations (closing, opening, dilation, and image thresholding) we will be able to remove this noise.

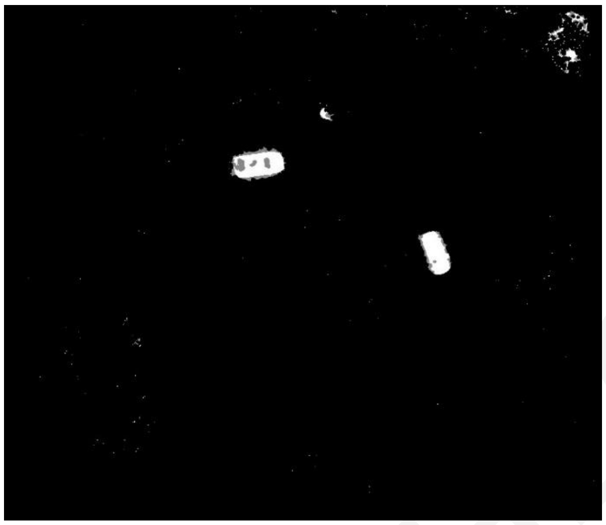


Figure 47: Closing operation

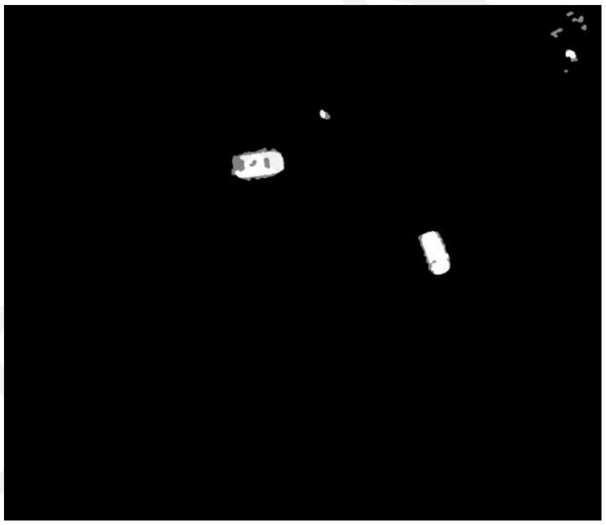


Figure 48: Opening operation

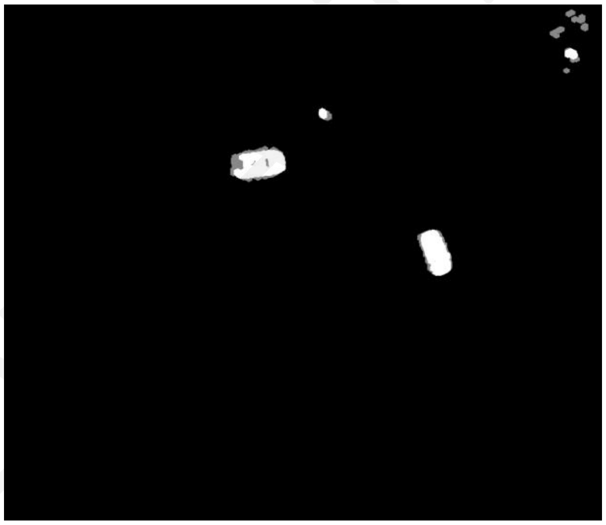


Figure 49: Dilation operation

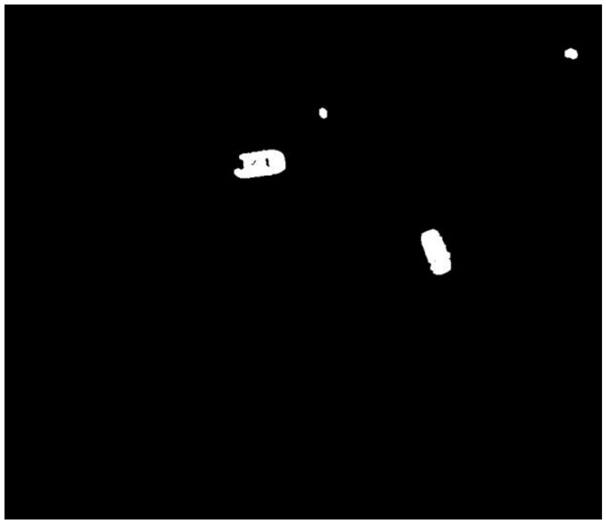


Figure 50: Thresholding operation

It becomes clear that all the noise and shadows from frames disappeared (figure 47-50). Apart from the two vehicles, we notice that two additional contours detected. The first one, as mentioned before, belongs to the tree and the other one is a pedestrian who gets away from his vehicle (a vehicle that is not recognized as a moving object).

In this work, it was decided that the traffic will be evaluated only for motorcycles, cars and buses or trucks. Thus, by determining the minimum and maximum contour area, we not only remove from our check the small moving objects, but also the pedestrians that we will very often meet moving around the roundabouts.

The next step comes with finding and drawing the contour in each frame. Firstly, we use a while loop for hierarchy to only count parent contours (contours not within others). Then, we find the four edges of the contour and draw it using the function v2.drawContours(). The coordinates of the center of each contour stored in the 'pts' list.

According to the size of each contour, is decided the type of vehicle. In this code for the real-life aerial video, the area range was not changed from the previous code (animated aerial video). This is because the two videos have similar features, and the height of the drone is quite close. Thus, if we manage to keep the height close to 90-120 meters from the height of the road, then this range of comparison does not need to change for each different roundabout that be examined.

Proceeding to the penultimate step, the counting system, all we need to do is count each vehicle pass by each line (entrance or exit line). If a line is horizontal, we only have to check if the center of contour passes through one line with a constant y-coordinate. In the same way, for vertical lines, we have to check if the center of contour passes through one line with a constant x-coordinate.

The total number of vehicles is calculated by summing up all the vehicles passed through all the entrances. As shown in figure 51, in the case of real-life aerial video, the entrances of this roundabout are considered to be 1,3, 5 and 7 (blue lines).

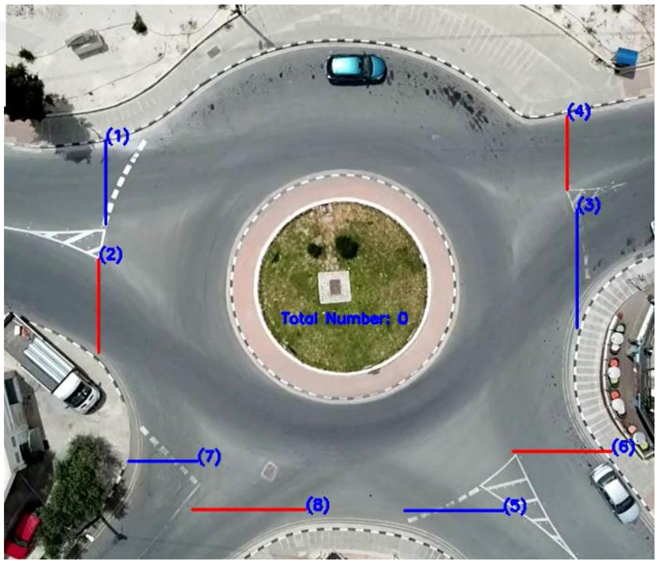


Figure 51: Inputs (entrances-blue) & Outputs (exits-red)

The final step comes with tracking the path of vehicles. We create dots for each contour that been detected using the coordinates of the center, which the function provides to us. Those dots stored in a list with maximum length one million dots. This length was manually defined, and it could be changed. Most of the time, we do not need to store the entire route of a vehicle, especially when the period we are considering is very long. So, this length is good to change depending on the time of the video.

The following figure (figure 52) shows the path of three vehicles.

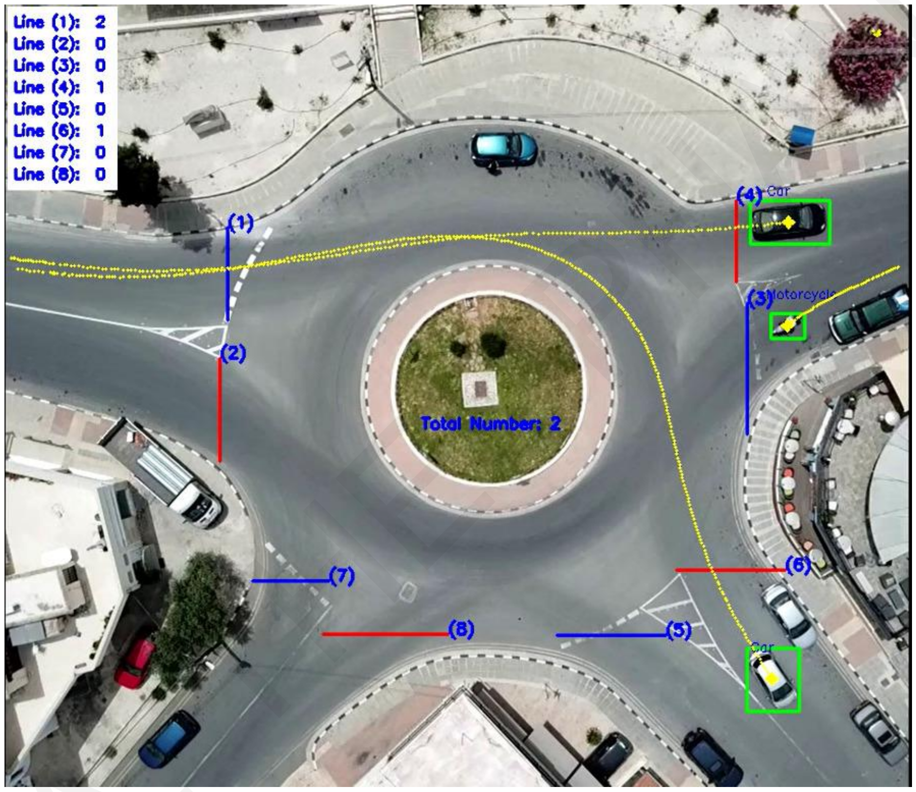


Figure 52: Result of full code

5.2 Results and discussion

This part of chapter 5 will present the research results of this project, looking primarily at the data collected from the aerial video taken in Polis. The system that was built in this thesis work focuses on the traffic assessment of roundabouts in Cyprus. The code can be used for other purposes, for example in traffic on major avenues, but will require a few basic modifications on the code.

<u>Road Surface Inspections:</u>

Furthermore, from the results and the data stored in the list 'pts' (centroids of contours, path), we can predict the road distress and damages. This could be helpful on road surface inspection applications. The path that vehicles create in this video, as it shown in the figure bellow with a more intense gray colour (figure 53), is the path than any vehicle usually takes.

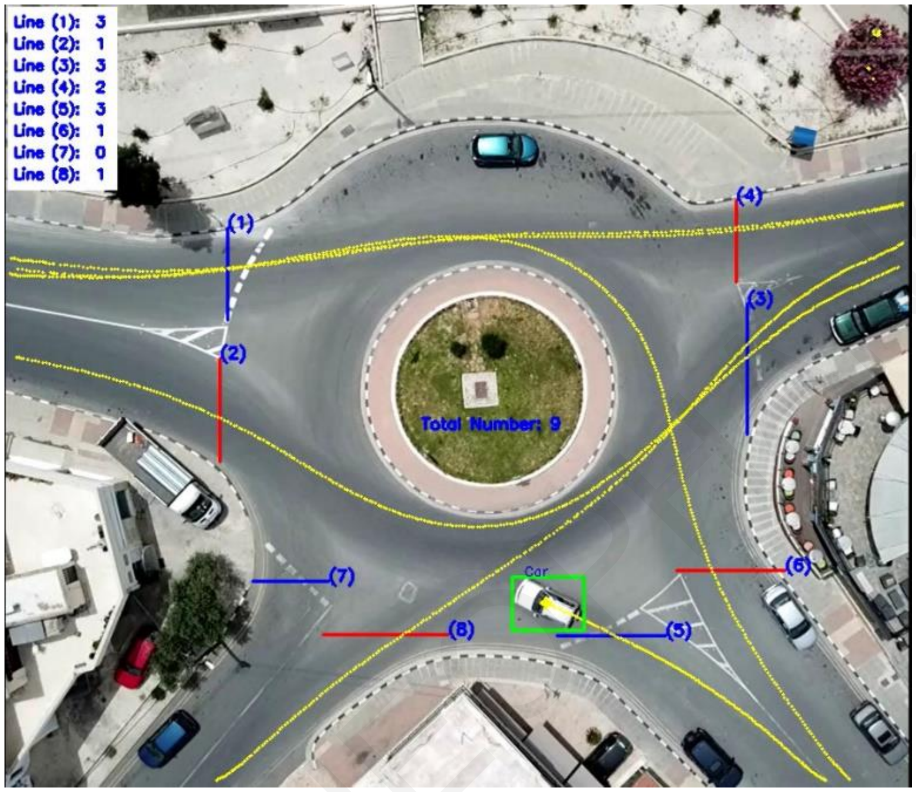


Figure 53: Road surface inspection applications

5.2.1 Accuracy

A code that is applied to real data and is based on the human decision could not be 100% perfect. So, similarly this code, while it works accurately in an ideal model, when applied to imperfect compositions, it presents some weaknesses.

For instance, we notice that when a vehicle is stationary, the system does not recognize it and does not measure it. Small moving objects are also not counted. But what happens when a vehicle is almost stopped at the entrance line of the roundabout and is moving with a minimum speed, but not zero?

Then as we see, the system, which counts vehicles depending on the frames, counts the same vehicle two or three times, because the same vehicle detected next or at the line successively for two or three frames (figure 54). A good tactic to face this system’s weakness is to place the control line just ahead of the entry or exit line of the roundabout. Applying this tactic, the vehicle will be able to increase speed before passing the control line and not be measured double or triple.

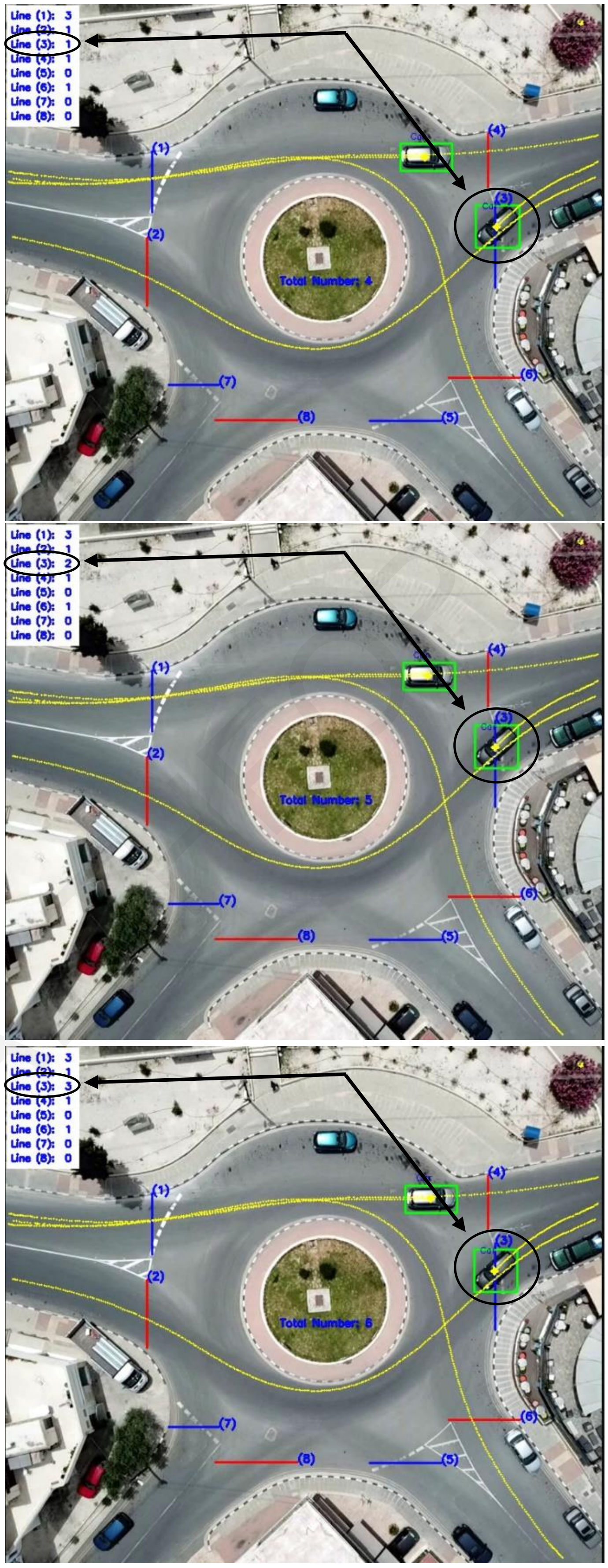


Figure 54: Weakness of algorithm

5.2.2 Precision and Recall

Precision and recall are two numbers which together are used to evaluate the performance of classification or information retrieval systems. Precision is defined as the fraction of relevant instances among all retrieved instances. Recall, sometimes referred to as 'sensitivity', is the fraction of retrieved instances among all relevant instances. A perfect classifier has precision and recall both equal to 1.

In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labelled as belonging to the positive class) divided by the total number of elements labelled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labelled as belonging to the class).

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Recall in this context is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labelled as belonging to the positive class but should have been).

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

The true positive rate is the number of instances which are relevant and which the model correctly identified as relevant. The false positive rate is the number of instances which are not relevant but which the model incorrectly identified as relevant. The false negative rate is the number of instances which are relevant and which the model incorrectly identified as not relevant.

Usually, precision and recall scores are given together and are not quoted individually. This is because it is easy to vary the sensitivity of a model to improve precision at the expense of recall, or vice versa. Precision can be seen as a measure of quality and recall as a measure of quantity. Higher precision means that an algorithm returns more relevant results than irrelevant ones, and high recall means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

In this project, the algorithm in the real-life application identifies nine vehicles and of the nine it identifies as vehicles, six actually are vehicles (true positives), while the other three have been identified incorrectly (false positives). All the other contours were correctly excluded, and zero vehicles were missed (false negatives). The program's precision is then 6/9 (true positives/all positives), while its recall is 6/6=1 (true positives/relevant elements).

The model has a precision of 0.67, so in other words, when it identifies a vehicle, it is correct 67% of the time. Respectively, the model has a recall of 1.00, so it correctly identifies 100% of all vehicles. The percentage of precision may not seem large enough, but it should be taken in account that the sample (in this case the number of vehicles in the video) is small.

Similarly, in the testing video (animated video) the precision and the recall are equal to 1, as all the vehicles were identified correctly and no one vehicle was missed.

If a single number is required to describe the performance of a model, the most convenient figure is the F1-score, which is the harmonic mean of the precision and recall:

$$F1 - Score = 2 \times \frac{precision \ \times recall}{precision + recall}$$

This allows us to combine the precision and recall into a single number.

Putting the figures for the precision (=6/9) and recall (=1) into the formula for the F-score, we obtain:

$$F1 - Score = 2 \times \frac{precision \ \times recall}{precision + recall} = 2 \times \frac{\frac{6}{9} \times 1}{\frac{6}{9} + 1} = 0.80$$

If the precision or the recall were zero, the F1 score will be zero. So, you will know that the algorithm is not working well. When the precision and recall both are perfect, that means precision is 1 and recall is also 1, the F1 score will be 1. So, the perfect F1 score is 1 and the higher the F1 score is, the more accurate the model is in doing predictions. As we see, the 80% is quite satisfactory in terms of accuracy and the model is working well enough.

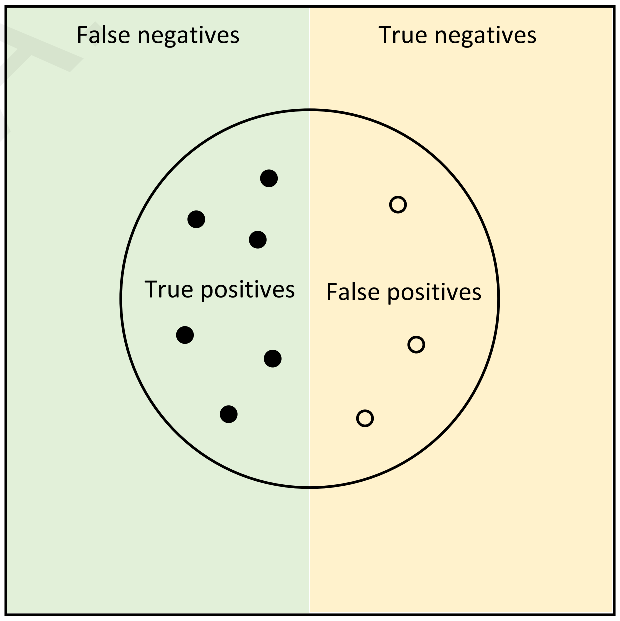


Figure 55: Precision and Recall

5.2.3 Method's Limitations/Shortcomings

As with the majority of studies, the design of the current study is subject to limitations. Pointing out the limitations and shortcomings of the study is a very important part of the research. It helps future studies and researches to focus on more innovative ways to conduct research and ignore the issues faced. No study is complete without its set of limitations.

The findings of this study have to be seen in the light of the following limitations. First, the code has been developed to detect, track and count vehicles only for aerial videos (top view). With any other form of perspective or view on video, the code will not work correctly and will have many vulnerabilities. In this case, the code will have to undergo some radical configurations to meet the project objectives.

The second limitation concerns the altitude that the drones are when the video is captured. As described in the previous chapter, the altitude and position of the drone play a crucial role and can significantly affect the results of the code. A 90-120 meters height restriction above the road surface should be set. Then the code will have its maximum performance. An alternative way to get rid of this restriction is to have a function in the code that make changes to the whole check for the contour area depending on the altitude at which the camera was.

This research is also subject to some shortcomings. Post-research shortcomings are the ones that are identified after the study was conducted. The main drawback of this study is that the code is not entirely automated. For instance, the user has to define the coordinates of the entrances and exits manually for each roundabout separately. As described in the real-life application chapter, a good tactic to face this problem is to take a test flight with constant altitude, latitude, and longitude for every roundabout and note the coordinates of the entrance and exit lines so that can be used in all other flights and analyses of the traffic of each roundabout.

Another shortcoming that should be taken into consideration is the accuracy of the method regarding the counting of vehicles. This issue is discussed in subsection 5.2.1. So according to this, a good strategy to take into account in the results is to consider a percentage of inaccuracy/error every time the code is run. Of course, this percentage could be higher during peak hours (vehicles move at lower speeds near the control lines of the method – method's weakness), while for the rest of the day this percentage could be lower. This percentage could be calculated by running the code for different hours of the day and recording the number of vehicles that were counted more times than they should have been, or not counted at all. Thus, this percentage may be representative of other analyses.

6 CONCLUSION

Object detection using machine and computer vision has taken some massive leaps in the past couple of years, and the field is very popular at the moment. Recently, many studies have been published on the field ((Reddy, 2020), (Skakun, et al., 2020), (Han, et al., 2020), (Reksten & Salberg, 2020), (Xiao & Kang, 2021), etc.).

The purpose of this thesis was to examine the traffic assessment using machine vision and develop a new vehicle detection, counting and tracking system by using contours for detection.

The main goal was successfully implemented, a working system which utilizes contours for object detection. However, the system does not run perfectly on the current test, so some changes need to be implemented in the nearby future to acquire more accuracy.

This whole project took about six and a half months to complete, which includes researching, documentation, development, and testing. Researching and documentation probably took the most prolonged period.

Finding the most suitable framework and tools was fulfilled mostly with trial and error. At first, benchmarking different models and codes was executed with simple demos found publicly on GitHub. After Python and OpenCV were chosen as the primary programming language and library, respectively, coding started, and after the system was mostly finished, benchmarking was performed running the main application and testing different models. It is impressive that such a relatively simple program is so functional, meets its goals efficiently and is very fast.

Before this project, my knowledge in machine and computer vision was minimal and almost nonexistent. Completing this project has given knowledge, ideas, and skills to work in future projects involving deep learning, machine, and computer vision.

The development of the system will continue as future work on better test hardware.

7 APPENDIX

APPENDIX A. Python algorithm for animated video

Filename: vehicle_tracking_video_an.py

```
import numpy as np
import cv2
import pandas as pd
from collections import deque

cap = cv2.VideoCapture("video_an.mp4")
frames_count, fps, width, height = cap.get(cv2.CAP_PROP_FRAME_COUNT),
cap.get(cv2.CAP_PROP_FPS), cap.get(
    cv2.CAP_PROP_FRAME_WIDTH), cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

l1, l2, l3, l4, l5, l6, l7, l8, sum = 0,0,0,0,0,0,0,0,0

# lines
# x1_s: x coordinate of the start point of line 1 (same x start&end point,
vertical line)
x1_s, x1_e, y1_s, y1_e= 269, 269, 155, 230  # vertical line
x2_s, x2_e, y2_s, y2_e= 265, 265, 250, 325  # vertical line
x3_s, x3_e, y3_s, y3_e= 592, 592, 155, 230  # vertical line
x4_s, x4_e, y4_s, y4_e= 591, 591, 250, 325  # vertical line
x5_s, x5_e, y5_s, y5_e= 340, 415, 77, 77  # horizontal line
x6_s, x6_e, y6_s, y6_e= 440, 510, 77, 77  # horizontal line
x7_s, x7_e, y7_s, y7_e= 340, 415, 404, 404  # horizontal line
x8_s, x8_e, y8_s, y8_e= 440, 510, 404, 404  # horizontal line

pts = deque(maxlen=1000000) # line length
width = int(width)
height = int(height)
print(frames_count, fps, width, height)

sub = cv2.createBackgroundSubtractorMOG2()  # create background subtractor
# information to start saving a video file
ret, frame = cap.read()  # import image
ratio = 1.0  # resize ratio
image = cv2.resize(frame, (0, 0), None, ratio, ratio)  # resize image
width2, height2, channels = image.shape
out1 = cv2.VideoWriter('out_an1.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
out2 = cv2.VideoWriter('out_an2.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
out3 = cv2.VideoWriter('out_an3.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
out4 = cv2.VideoWriter('out_an4.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
out5 = cv2.VideoWriter('out_an5.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
out6 = cv2.VideoWriter('out_an6.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 1)
out7 = cv2.VideoWriter('out_an7.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 1)
out8 = cv2.VideoWriter('out_an8.avi', cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'), fps, (height2, width2), 0)
```

```
while True:
    ret, frame = cap.read()  # import image
    if not ret: #if vid finish repeat
        frame = cv2.VideoCapture("video_an.mp4")
        continue
    if ret:  # if there is a frame continue with code
        image = cv2.resize(frame, (0, 0), None, ratio, ratio)  # resize
image

        cv2.rectangle(image, (5, 5), (260, 120), (255, 255, 255), -2)
        cv2.rectangle(image, (260, 5), (309, 60), (255, 255, 255), -2)
        cv2.circle(image, (255,65), 55, (255, 255, 255), -2)

        cv2.line(image, (x1_s, y1_s), (x1_e, y1_e), (0, 0, 255), 2)  #
line1
        cv2.line(image, (x2_s, y2_s), (x2_e, y2_e), (255, 0, 0), 2)  #
line2
        cv2.line(image, (x3_s, y3_s), (x3_e, y3_e), (255, 0, 0), 2)  #
line3
        cv2.line(image, (x4_s, y4_s), (x4_e, y4_e), (0, 0, 255), 2)  #
line4
        cv2.line(image, (x5_s, y5_s), (x5_e, y5_e), (255, 0, 0), 2)  #
line5
        cv2.line(image, (x6_s, y6_s), (x6_e, y6_e), (0, 0, 255), 2)  #
line6
        cv2.line(image, (x7_s, y7_s), (x7_e, y7_e), (0, 0, 255), 2)  #
line7
        cv2.line(image, (x8_s, y8_s), (x8_e, y8_e), (255, 0, 0), 2)  #
line8

        cv2.imshow("image", image) #@
        out7.write(image)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converts image to
gray
        out1.write(gray)
        cv2.imshow("Gray", gray) #@
        fgmask = sub.apply(gray)  # uses the background subtraction
        out2.write(fgmask)
        cv2.imshow("fgmask", fgmask) #@
        # applies different thresholds to fgmask to try and isolate cars
        # just have to keep playing around with settings until cars are
easily identifiable
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))  #
kernel to apply to the morphology
        closing = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
        out3.write(closing)
        cv2.imshow("closing", closing) #@
        opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
        out4.write(opening)
        cv2.imshow("opening", opening) #@
        dilation = cv2.dilate(opening, kernel)
        out5.write(dilation)
        cv2.imshow("dilation", dilation) #@
        retvalbin, bins = cv2.threshold(dilation, 220, 255,
cv2.THRESH_BINARY)  # removes the shadows
        cv2.imshow("bins", bins) #@
        out8.write(bins)
        # creates contours
        # cv2.imshow('bins',bins)
        contours, hierarchy = cv2.findContours(bins, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        center = None

        minarea = 300
        # max area for contours, can be quite large for buses
        maxarea = 50000
        # vectors for the x and y locations of contour centroids in current
frame
        cxx = np.zeros(len(contours))
        cyy = np.zeros(len(contours))

        for i in range(len(contours)):  # cycles through all contours in
current frame
            if hierarchy[0, i, 3] == -1:  # using hierarchy to only count
parent contours (contours not within others)
                area = cv2.contourArea(contours[i])  # area of contour
                if minarea < area < maxarea:  # area threshold for contour
                    # calculating centroids of contours
                    cnt = contours[i]
                    M = cv2.moments(cnt)
                    cx = int(M['m10'] / M['m00'])
                    cy = int(M['m01'] / M['m00'])

                    if cx==x1_s and cy>y1_s and cy<y1_e :
                        l1+=1
                    elif cx==x2_s and cy>y2_s and cy<y2_e :
                        l2+=1
                    elif cx==x3_s and cy>y3_s and cy<y3_e :
                        l3+=1
                    elif cx==x4_s and cy>y4_s and cy<y4_e :
                        l4+=1
                    elif cy==y5_s and cx>x5_s and cx<x5_e :
                        l5+=1
                    elif cy==y6_s and cx>x6_s and cx<x6_e :
                        l6+=1
                    elif cy==y7_s and cx>x7_s and cx<x7_e :
                        l7+=1
                    elif cy==y8_s and cx>x8_s and cx<x8_e :
                        l8+=1

                    center = (int(M["m10"] / M["m00"]), int(M["m01"] /
M["m00"]))

                    # update the points queue
                    pts.appendleft(center)
                    # loop over the set of tracked points
                    for k in range(1, len(pts)):
                        # if either of the tracked points are None, ignore
them

                        if pts[k - 1] is None or pts[k] is None:
                            continue
                        # draw the dots
                        thickness = -(1%1000)
                        cv2.circle(image, pts[k], 1, (0, 255 , 255),
thickness)
                    # gets bounding points of contour to create rectangle
                    # x,y is top left corner and w,h is width and height
                    x, y, w, h = cv2.boundingRect(cnt)
                    # creates a rectangle around contour
                    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255,
0), 2)
                    # Prints centroid text in order to double check later
on
                    # cv2.putText(image, str(cx) + "," + str(cy), (cx + 10,
cy + 10), cv2.FONT_HERSHEY_SIMPLEX,.3, (0, 0, 255), 1)
                    cv2.drawMarker(image, (cx, cy), (0, 255, 255),
cv2.MARKER_CROSS, markerSize=8, thickness=3,line_type=cv2.LINE_8)

                    if 10000 < area < 50000:
                        cv2.putText(image, "Bus/Track", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4,(255, 0, 0), 1)
                    elif 500 < area < 10000:
                        cv2.putText(image, "Car", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)
                    else:
                        cv2.putText(image, "Motorcycle", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)

                    cv2.putText(image, "(1)", (x1_e, y1_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(2)", (x2_e, y2_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(3)", (x3_e, y3_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(4)", (x4_e, y4_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(5)", (x5_e, y5_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(6)", (x6_e, y6_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(7)", (x7_e, y7_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    cv2.putText(image, "(8)", (x8_e, y8_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)

                    cv2.putText(image, "Line (1):  " + str(l1), (30, 20),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (2):  " + str(l2), (30, 50),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (3):  " + str(l3), (30, 80),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (4):  " + str(l4), (30, 110),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (5):  " + str(l5), (180, 20),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (6):  " + str(l6), (180, 50),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (7):  " + str(l7), (180, 80),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                    cv2.putText(image, "Line (8):  " + str(l8), (180, 110),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)

                    sum = l2 + l3 + l5 + l8
                    cv2.putText(image, "Total Number: " + str(sum), (365,
240), cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                    out6.write(image)
                    cv2.imshow("countours", image)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out1.release()
out2.release()
out3.release()
out4.release()
out5.release()
out6.release()
out7.release()
out8.release()
cv2.destroyAllWindows()
print("Videos successfully saved")
```

APPENDIX B. Python algorithm for Real aerial video

Filename: vehicle_tracking_video1.py

```
import numpy as np
import cv2
import pandas as pd
from collections import deque

cap = cv2.VideoCapture("video1.mp4")
frames_count, fps, width, height = cap.get(cv2.CAP_PROP_FRAME_COUNT),
cap.get(cv2.CAP_PROP_FPS), cap.get(
    cv2.CAP_PROP_FRAME_WIDTH), cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

l1, l2, l3, l4, l5, l6, l7, l8, sum = 0,0,0,0,0,0,0,0,0

# lines
# x1_s: x coordinate of the start point of line 1 (same x start&end point,
vertical line)
x1_s, x1_e, y1_s, y1_e= 397, 397, 205, 290  # vertical line
x2_s, x2_e, y2_s, y2_e= 390, 390, 325, 420  # vertical line
x3_s, x3_e, y3_s, y3_e= 875, 875, 275, 395  # vertical line
x4_s, x4_e, y4_s, y4_e= 865, 865, 180, 255  # vertical line
x5_s, x5_e, y5_s, y5_e= 700, 800, 580, 580  # horizontal line
x6_s, x6_e, y6_s, y6_e= 810, 910, 520, 520  # horizontal line
x7_s, x7_e, y7_s, y7_e= 420, 490, 530, 530  # horizontal line
x8_s, x8_e, y8_s, y8_e= 485, 600, 579, 579  # horizontal line

pts = deque(maxlen=1000000) # line length
width = int(width)
height = int(height)
print(frames_count, fps, width, height)

sub = cv2.createBackgroundSubtractorMOG2()  # create background subtractor
# information to start saving a video file
ret, frame = cap.read()  # import image
ratio = 1.0  # resize ratio
image = cv2.resize(frame, (0, 0), None, ratio, ratio)  # resize image
width2, height2, channels = image.shape
out1 = cv2.VideoWriter('out_real1.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)
out2 = cv2.VideoWriter('out_real2.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)
out3 = cv2.VideoWriter('out_real3.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)
out4 = cv2.VideoWriter('out_real4.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)
out5 = cv2.VideoWriter('out_real5.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)
out6 = cv2.VideoWriter('out_real6.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 1)
out8 = cv2.VideoWriter('out_real8.avi', cv2.VideoWriter_fourcc('M', 'J',
'P', 'G'), fps, (height2, width2), 0)

while True:
    ret, frame = cap.read()  # import image
    if not ret: #if vid finish repeat
        frame = cv2.VideoCapture("video1.mp4")
        continue
    if ret:  # if there is a frame continue with code
        image = cv2.resize(frame, (0, 0), None, ratio, ratio)  # resize
image

        cv2.rectangle(image, (195, 2), (295, 170), (255, 255, 255), -2)

        cv2.line(image, (x1_s, y1_s), (x1_e, y1_e), (255, 0, 0), 2)  #
line1
        cv2.line(image, (x2_s, y2_s), (x2_e, y2_e), (0, 0, 255), 2)  #
line2
        cv2.line(image, (x3_s, y3_s), (x3_e, y3_e), (255, 0, 0), 2)  #
line3
        cv2.line(image, (x4_s, y4_s), (x4_e, y4_e), (0, 0, 255), 2)  #
line4
        cv2.line(image, (x5_s, y5_s), (x5_e, y5_e), (255, 0, 0), 2)  #
line5
        cv2.line(image, (x6_s, y6_s), (x6_e, y6_e), (0, 0, 255), 2)  #
line6
        cv2.line(image, (x7_s, y7_s), (x7_e, y7_e), (255, 0, 0), 2)  #
line7
        cv2.line(image, (x8_s, y8_s), (x8_e, y8_e), (0, 0, 255), 2)  #
line8

        cv2.imshow("image", image) #@
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # converts image to
gray
        out1.write(gray)
        cv2.imshow("Gray", gray)  #@
        fgmask = sub.apply(gray)  # uses the background subtraction
        out2.write(fgmask)
        cv2.imshow("fgmask", fgmask)  #@
        # applies different thresholds to fgmask to try and isolate cars
        # just have to keep playing around with settings until cars are
easily identifiable
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))  #
kernel to apply to the morphology
        closing = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
        out3.write(closing)
        cv2.imshow("closing", closing)  #@
        opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
        out4.write(opening)
        cv2.imshow("opening", opening)  #@
        dilation = cv2.dilate(opening, kernel)
        out5.write(dilation)
        cv2.imshow("dilation", dilation)  #@
        retvalbin, bins = cv2.threshold(dilation, 220, 255,
cv2.THRESH_BINARY)  # removes the shadows
        cv2.imshow("bins", bins)  # @
        out8.write(bins)
        # creates contours
        # cv2.imshow('bins',bins)
        contours, hierarchy = cv2.findContours(bins, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        center = None

        minarea = 300
        # max area for contours, can be quite large for buses
        maxarea = 50000
        # vectors for the x and y locations of contour centroids in current
frame
        cxx = np.zeros(len(contours))
        cyy = np.zeros(len(contours))

        for i in range(len(contours)):  # cycles through all contours in
current frame
            if hierarchy[0, i, 3] == -1:  # using hierarchy to only count
parent contours (contours not within others)
                area = cv2.contourArea(contours[i])  # area of contour
                if minarea < area < maxarea:  # area threshold for contour
                    # calculating centroids of contours
                    cnt = contours[i]
                    M = cv2.moments(cnt)
                    cx = int(M['m10'] / M['m00'])
                    cy = int(M['m01'] / M['m00'])

                    if (cx==(x1_s-1) or cx==(x1_s) or cx==(x1_s+1)) and
cy>y1_s and cy<y1_e :
                        l1+=1
                    elif (cx==(x2_s-1) or cx==(x2_s) or cx==(x2_s+1)) and
cy>y2_s and cy<y2_e :
                        l2+=1
                    elif (cx==(x3_s-1) or cx==(x3_s) or cx==(x3_s+1)) and
cy>y3_s and cy<y3_e :
                        l3+=1
                    elif (cx==(x4_s-1) or cx==(x4_s) or cx==(x4_s+1)) and
cy>y4_s and cy<y4_e :
                        l4+=1
                    elif (cy==(y5_s-1) or cy==(y5_s) or cy==(y5_s+1)) and
cx>x5_s and cx<x5_e :
                        l5+=1
                    elif (cy==(y6_s-1) or cy==(y6_s) or cy==(y6_s+1)) and
cx>x6_s and cx<x6_e :
                        l6+=1
                    elif (cy==(y7_s-1) or cy==(y7_s) or cy==(y7_s+1)) and
cx>x7_s and cx<x7_e :
                        l7+=1
                    elif (cy==(y8_s-1) or cy==(y8_s) or cy==(y8_s+1)) and
cx>x8_s and cx<x8_e :
                        l8+=1

                    center = (int(M["m10"] / M["m00"]), int(M["m01"] /
M["m00"]))
                    # update the points queue
                    pts.appendleft(center)
                    # loop over the set of tracked points
                    for k in range(1, len(pts)):
                        # if either of the tracked points are None, ignore
them

                        if pts[k - 1] is None or pts[k] is None:
                            continue
                        # draw the dots
                        thickness = -(1%1000)
                        cv2.circle(image, pts[k], 1, (0, 255 , 255),
thickness)
                    # gets bounding points of contour to create rectangle
                    # x,y is top left corner and w,h is width and height
                    x, y, w, h = cv2.boundingRect(cnt)
```

```
                        # creates a rectangle around contour
                        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255,
0), 2)
                        # Prints centroid text in order to double check later
on
                        # cv2.putText(image, str(cx) + "," + str(cy), (cx + 10,
cy + 10), cv2.FONT_HERSHEY_SIMPLEX,.3, (0, 0, 255), 1)
                        cv2.drawMarker(image, (cx, cy), (0, 255, 255),
cv2.MARKER_CROSS, markerSize=8, thickness=3,line_type=cv2.LINE_8)

                        if 10000 < area < 50000:
                            cv2.putText(image, "Bus/Track", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4,(255, 0, 0), 1)
                        elif 500 < area < 10000:
                            cv2.putText(image, "Car", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)
                        else:
                            cv2.putText(image, "Motorcycle", (cx-20, cy-25),
cv2.FONT_HERSHEY_SIMPLEX, .4, (255, 0, 0), 1)

                        cv2.putText(image, "(1)", (x1_e, y1_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(2)", (x2_e, y2_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(3)", (x3_e, y3_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(4)", (x4_e, y4_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(5)", (x5_e, y5_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(6)", (x6_e, y6_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(7)", (x7_e, y7_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        cv2.putText(image, "(8)", (x8_e, y8_s),
cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)

                        cv2.putText(image, "Line (1):  " + str(l1), (200, 20),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (2):  " + str(l2), (200, 40),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (3):  " + str(l3), (200, 60),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (4):  " + str(l4), (200, 80),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (5):  " + str(l5), (200, 100),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (6):  " + str(l6), (200, 120),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (7):  " + str(l7), (200, 140),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)
                        cv2.putText(image, "Line (8):  " + str(l8), (200, 160),
cv2.FONT_HERSHEY_SIMPLEX, .45, (255, 0, 0), 2)

                        sum = l1 + l3 + l5 + l7
                        cv2.putText(image, "Total Number: " + str(sum), (575,
390), cv2.FONT_HERSHEY_SIMPLEX, .5, (255, 0, 0), 2)
                        out6.write(image)
                        cv2.imshow("countours", image)
```

```
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out1.release()
out2.release()
out3.release()
out4.release()
out5.release()
out6.release()
out8.release()
cv2.destroyAllWindows()
print("Videos successfully saved")
```

APPENDIX C. Python algorithm for grayscale histograms

Filename: histogram.py

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('thres_img_crop.jpg')
plt.hist(img.ravel(),256,[0,256])
plt.title('Thresholding Histogram')
plt.xlabel('Pixel values')
plt.ylabel('Number of pixels')
plt.show()
```

APPENDIX D. Python algorithm for binary histograms

Filename: histogram.py

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('grayscale_img_crop.jpg')
plt.hist(img.flatten(), bins=[-.5,.5,1.5], ec="k")
plt.xticks((0,1))
plt.title('Binary Histogram')
plt.xlabel('Pixel values')
plt.ylabel('Number of pixels')
plt.show()
plt.show()
```

8 REFERENCES

Anon., 2020. *How to Use Background Subtraction Methods.* [Online]
Available at: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

Boscacci, R., 2018. *Towards data science.* [Online]
Available at: https://towardsdatascience.com/what-even-is-computer-vision-531e4f07d7d0

Chen, S., Xu, Y., Zhou, X. & Li, F., 2019. Deep Learning for Multiple Object Tracking: A Survey. *IET Computer Vision,* 13(4), pp. 355-368.

Cohen, J., 2020. *Computer Vision and Deep Learning: From Image to Video Analysis.* [Online]
Available at: https://heartbeat.fritz.ai/computer-vision-from-image-to-video-analysis-d1339cf23961

Efford, N., 2000. *Digital Image Processing: A Practical Introduction Using JavaTM.* s.l.:Pearson Education.

F. Spencer, B., Hoskere, V. & Narazaki, Y., 2019. Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring. *Elsevier LTD,* 5(2), pp. 199-222.

García, G. B. et al., 2015. *Learning Image Processing with OpenCV.* s.l.:Packt Publishing Ltd.

Grigoriy, B., n.d. *English for IT Professionals Wiki.* [Online]
Available at: https://englishforitprofessionals.fandom.com/wiki/Computer_vision

Han, R. et al., 2020. Underwater Image Enhancement Based on a Spiral Generative Adversarial Framework. *IEEE Access,* Volume 8, pp. 218838 - 218852.

Hirano, Y., Garcia, C., Sukthankar, R. & Hoogs, A., 2006. *Industry and Object Recognition: Applications, Applied Research and Challenges.* s.l., Christophe Garcia.

Jiao, L. et al., 2019. A Survey of Deep Learning-Based Object Detection. *IEEE Access,* Volume 4.

Lalan, A., n.d. Facilitating Machine Learning Ep 1: An Introduction to Computer Vision. *BITS Goa Women In Tech*, 27 October.

Mary, R., 2011. Introduction to Image Processing. 1 April.

Memon, S. et al., 2018. A Video based Vehicle Detection, Counting and Classification System. *International Journal of Image, Graphics and Signal Processing,* 10(9), pp. 34-41.

Prasanna, R., Maruti, N. & Abdul, V., 2019. Applications of Object Detection System. *International Research Journal of Engineering and Technology (IRJET),* Volume 7.211, pp. 4186-4192.

Raiyn, J. & Toledo, T., 2014. Real-Time Road Traffic Anomaly Detection. *Journal of Transportation Technologies,* Issue 4, pp. 256-266.

Ranjit, A., 2019. *Basic Image Thresholding in OpenCV - Medium.* [Online]
Available at: https://medium.com/@anupriyam/basic-image-thresholding-in-opencv-5af9020f2472

Rawat, W. & Wang, Z., 2017. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. June, pp. 1-98.

Reddy, C. R., 2020. Autonomous Object Detection and Recognition Using a Machine Learning Based Smart System. *International Journal of Innovative Research in Computer and Communication Engineering,* 8(10), pp. 4050-4054.

Reksten, J. H. & Salberg, A.-B., 2020. Estimating Traffic in Urban Areas from Very-High Resolution Aerial Images. *International Journal of Remote Sensing*, 04 July, pp. 865-883.

Singh, R., 2019. *Recent Advances in Modern Computer Vision.* [Online] Available at: https://towardsdatascience.com/recent-advances-in-modern-computer-vision-56801edab980

Skakun, V. V., Tcherniavskaia, E. A. & Saetchnikov, I. V., 2020. Object detection for unmanned aerial vehicle camera via convolutional neural networks. *IEEE Journal on Miniaturization for Air and Space Systems*, pp. 1-1.

Walsh, J. et al., 2019. *Deep Learning vs. Traditional Computer Vision.* Las Vegas, Nevada, United States, Niall O' Mahony.

Xiao, B. & Kang, S.-C., 2021. Development of an Image Data Set of Construction Machines for Deep Learning Object Detection. *Journal of Computing in Civil Engineering*, 35(2).

Xiaoping, Z. & Mengting, L., 2009. An overview of accident forecasting methodologies. *Journal of Loss Prevention in the Process Industries*, 22(4), pp. 484-491.

Yashas111, 2019. *GitHub: Where the world builds software.* [Online] Available at: https://github.com/Yashas111/Vehicle-Detection-and-Tracking/blob/master/vehicledet.py

Zhanyu Ma, et al., 2018. Recent Advantages of Computer Vision. *IEEE Access Special Section Editorial*, pp. 31481 - 31485.

Zivkovic, Z., 2004. *Improved Adaptive Gaussian Mixture Model for Background Subtraction.* The Netherlands, IEEE Xplore.

Zivkovic, Z. & der Heijden, F., 2006. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, May, 27(7), pp. 773-780.

Oliphant, T.E., 2006. A guide to NumPy, Trelgol Publishing USA.

Bradski, G., 2000. The OpenCV Library. Dr. Dobb&#x27;s Journal of Software Tools.

Van Rossum, G. & Drake, F.L., 2009. Python 3 Reference Manual, Scotts Valley, CA: CreateSpace.