University of Cyprus
Department of Civil and Environmental Engineering

Automatic number-plate recognition (ANPR) in video streams and data anonymization for traffic monitoring, by use of machine vision

Master of Science Thesis

Arestis Pavlides

December 2022

Arestis Pavlides

University of Cyprus
Department of Civil and Environmental Engineering

Automatic number-plate recognition (ANPR) in video streams and data anonymization for traffic monitoring, by use of machine vision

Arestis Pavlides

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Cyprus

December 2022

Arestis Pavlides

APPROVAL PAGE

Master of Science Thesis

AUTOMATIC NUMBER-PLATE RECOGNITION (ANPR) IN VIDEO STREAMS AND DATA ANONYMIZATION FOR TRAFFIC MONITORING, BY USE OF MACHINE VISION

Presented by

Arestis Pavlides

Research Supervisor

Dr. Symeon Christodoulou
Professor

Committee Member

Dr. Loukas Dimitriou
Assistant Professor

University of Cyprus

December 2022

Arestis Pavlides

ACKNOWLEDGEMENTS

Researching and composing this Master’s thesis couldn’t be done without help. Foremost, I would like to express my sincere gratitude to my academic adviser Professor Dr. Symeon Christodoulou for the continuous support of my Master’s thesis study and research, for his patience, motivation, enthusiasm, and immense knowledge which inspired me greatly. His guidance helped me continuously in the research and the writing of this thesis.

Besides my adviser, I would like to thank the rest of my thesis committee members for their encouragement, insightful comments, and hard questions.

Last but not the least, I would like to thank my family: my parents Soteris and Eleni and my sister Andromachi for supporting me through every step-in life and being there when I needed them. Their sacrifices, love and advises in order to provide me a better life have molded me to the person I am today.

Arestis Pavlides

Table of Contents

Arestis Pavlides

Arestis Pavlides

Table of Abbreviations

Abbreviation	Definition
ANPR	Automatic Number Plate Recognition
CPU	Central Processing Unit
CV	Computer Vision
GDPR	General Data Protection Regulation
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
ID	Identification
MinNeigh	Minimum number of neighboring rectangles
OCR	Optical Character Recognition
PC	Personal Computer
RAM	Random Access Memory
RGB	Red Green Blue
ROI	Region Of Interest

Arestis Pavlides

List of Figures

Arestis Pavlides

List of Tables

Arestis Pavlides

Abstract

The research work described in this thesis focuses on the creation of an Automatic Number Plate Recognition (ANPR) software that is able to analyze images from video streams, to detect and to extract a vehicle's license plate for subsequent use. In order to adhere to the General Data Protection Regulation (GDPR) and thus avoid any privacy issues with video processing, the developed process involves two key stages: (1) the video streams in analysis are first 'anonymized', by blurring any detected plate numbers, and (2) the detected plated numbers are stored in a different (textual) file for later use by the users. Also listed in the thesis, are the problems that arose in the process, and the tools and methods used to solve them.

The program is written in the Python programming language. This language was preferred because of its suitability to task, its many contributions in the field of open computer vision (CV) and the abundance of tools available for public use (About OpenCV, 2022).

As aforementioned, a key consideration in this work was compliance to GDPR so that processed sensitive data (such as vehicle license plates) would be protected by an anonymization process. This was done by blurring out the parts of the image that contained such sensitive information. The detection and anonymization process used exhibited a high accuracy and could also be expanded in other applications (Janowski, 2014).

Keywords: Automatic Number Plate Recognition, optical character recognition, Open computer vision, anonymized, blur license plate

Arestis Pavlides

ΠΕΡΙΛΗΨΗ

Η ερευνητική εργασία που περιγράφεται σε αυτή τη διατριβή επικεντρώνεται στη δημιουργία κώδικα λογισμικού αυτόματης αναγνώρισης αριθμού πινακίδας οχημάτων (ANPR), το οποίο είναι σε θέση να αναλύει εικόνες από ροές βίντεο, να ανιχνεύει και να εξάγει την πινακίδα κυκλοφορίας ενός οχήματος για μεταγενέστερη χρήση. Προκειμένου να τηρηθεί ο Γενικός Κανονισμός Προστασίας Δεδομένων (GDPR) και να αποφευχθούν τυχόν προβλήματα απορρήτου με την επεξεργασία βίντεο, η αναπτυγμένη διαδικασία περιλαμβάνει δύο βασικά στάδια: (1) οι ροές βίντεο στην ανάλυση πρώτα «ανωνυμοποιούνται», θολώνοντας κάθε αριθμό πινακίδας που έχει εντοπιστεί και (2) οι ανιχνευθέντες αριθμοί που εντοπίζονται, αποθηκεύονται σε διαφορετικό αρχείο (κειμένου) για μελλοντική χρήση από τους χρήστες. Επίσης στη διατριβή αναφέρονται τα προβλήματα που προέκυψαν κατά τη διαδικασία, καθώς και τα εργαλεία και οι μέθοδοι που χρησιμοποιήθηκαν για την επίλυσή τους.

Το πρόγραμμα είναι γραμμένο στη γλώσσα προγραμματισμού Python. Αυτή η γλώσσα προτιμήθηκε λόγω της καταλληλόλητάς της για την εν λόγω εργασία, των πολλών συνεισφορών της στον τομέα της ανοιχτής όρασης υπολογιστή (openCV) και της πληθώρας εργαλείων που είναι διαθέσιμα για δημόσια χρήση (About OpenCV, 2022).

Όπως προαναφέρθηκε, ένα βασικό στοιχείο σε αυτήν την εργασία ήταν η συμμόρφωση με τον κανονισμό GDPR περί προσωπικών δεδομένων, έτσι ώστε τυχόν εντοπισμένα ευαίσθητα δεδομένα (όπως οι πινακίδες κυκλοφορίας των οχημάτων) να προστατεύονται από μια διαδικασία

Arestis Pavlides

ανωνυμοποίησης. Αυτό επιτυγχάνεται με τη θόλωση των μερών της εικόνας που περιείχαν ευαίσθητες πληροφορίες. Η διαδικασία ανίχνευσης και ανωνυμοποίησης που χρησιμοποιήθηκε επέδειξε υψηλή ακρίβεια και θα μπορούσε επίσης να επεκταθεί σε άλλες εφαρμογές (Janowski, 2014).

Arestis Pavlides

1. Introduction

The steady increase of vehicles on the streets and the pressing need to better manage and continuously optimize traffic flows have led to the need for traffic monitoring (Patel, 2013). There is a plethora of reasons why an around-the-clock traffic monitoring program can help agencies worldwide, and why it is in high demand by them (Janowski, 2014). With the necessary information about car flows, the control and prediction of traffic jams and of accidents can be improved (Cynthia Lum, 2011), and by using related camera-hosted technologies could monitor unremittingly the traffic situation in the country and provide transport authorities much needed information and unprecedented, until now, monitoring capabilities, to manage their transport networks. A typical ANPR system, as shown in the diagram of Figure 1-1, can be either fixed or mobile and is already being used by law enforcement agencies across the globe.

Arestis Pavlides

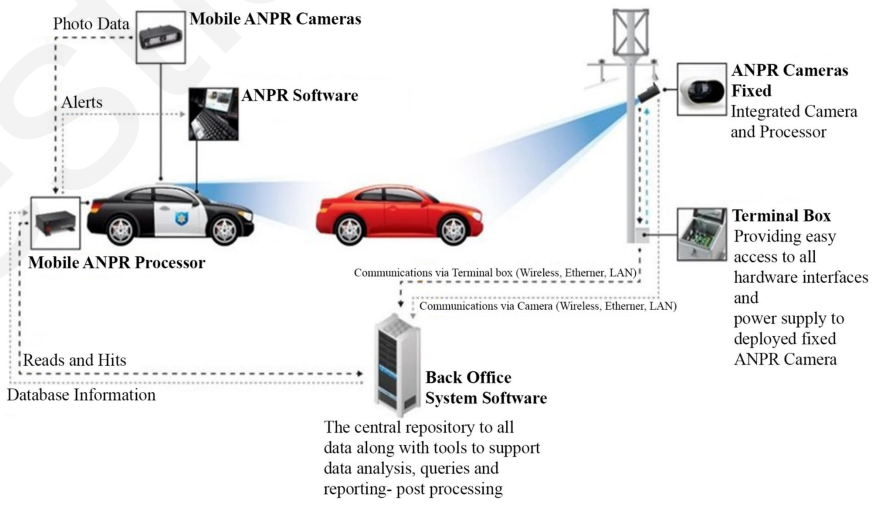


Photo Data
Mobile ANPR Cameras
Alerts
ANPR Software
ANPR Cameras Fixed
Integrated Camera and Processor
Terminal Box
Providing easy access to all hardware interfaces and power supply to deployed fixed ANPR Camera
Mobile ANPR Processor
Communications via Terminal box (Wireless, Ethernet, LAN)
Communications via Camera (Wireless, Ethernet, LAN)
Reads and Hits
Database Information
Back Office System Software
The central repository to all data along with tools to support data analysis, queries and reporting- post processing

Figure 1-1 Typical ANPR system diagram of a mobile ANPR system (left) and a fixed ANPR System (right) (Mufti, 2021)

1.1 Possible applications of machine vision in transport

Machine vision can serve the purpose of detecting and reading a license plate that can be used in solving misdemeanors such as illegal parking, driver license expiration, mechanical inspection of vehicle (roadworthiness tests certificate expiration) (Han, 2020), or other more serious cases such as car theft, and placing a car at a location and date for investigation purposes (Sharma, 2019). With regard to traffic monitoring, the technology can be utilized in traffic-counting at locations of interest and in providing information about the traffic flow. This information can subsequently be used to explore if a certain road is able to handle traffic demand and/or other key performance metrics. An example of an ANPR system used for speed limit enforcement is shown in Figure 1-2.

Aresti Pavlides

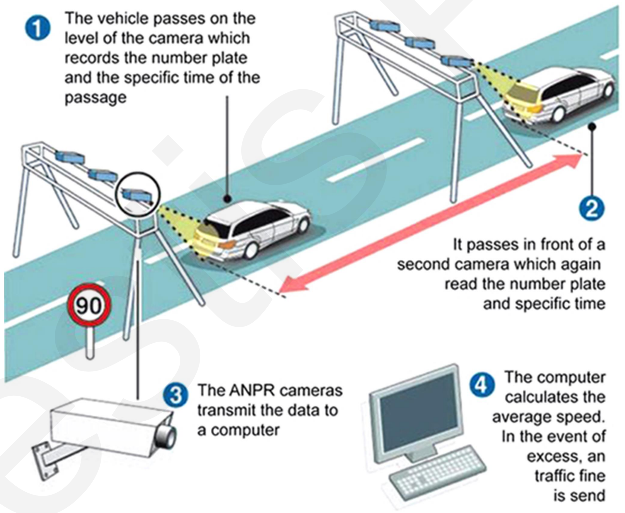

1 The vehicle passes on the level of the camera which records the number plate and the specific time of the passage
2 It passes in front of a second camera which again read the number plate and specific time
90
3 The ANPR cameras transmit the data to a computer
4 The computer calculates the average speed. In the event of excess, an traffic fine is send

Figure 1-2 ANPR system used for speed limit enforcement (Infratech, 2014)

1.2 Overview of research methodology on ANPR technology

In order to develop a software program that can detect license plates in images, a solid background in the field of computer vision had to be obtained and many related programs in optical character recognition had to be considered (Patel, 2013). The vast majority of the programs that were studied were written in the Python programming language. The choice of using this language for developing the code for the current study is further explained in Chapter 1.3.

A similar to this thesis's Python code structure, that can detect and extract a car's license plate numbers using images from a camera or video file, has been reported by other researchers in the field of computer vision. (About OpenCV, 2022; Patel, 2013; Sharifara, 2014)

Further, the specific language and generic computing technology, such as a PC and a web camera, can furnish a developer with the tools to not only develop a powerful ANPR application but to also test it in real-life situations.

1.3 The use of the Python programming language for the purpose of this research work

As explained in Dubois (2007), Python is open-source and thus available universally. It comes with a variety of standard libraries which contain support for the majority of areas of computer science and data science, and its "*extensive set of third-party tools and modules covers additional tasks, from managing a Web site to doing a fast Fourier transform to distributed or parallel programming. Python's motto, "batteries included," is meant to convey the*

Arestis Pavlides

idea that Python comes with everything you need. Python is an interpreted, high-level programming language and has become standard for exploratory, interactive, and computation-driven scientific research” (Millman, 2011).

The Python programming language is one of the most popular languages for scientific computing. Due to its top interactive nature and its continuously upgrading list of scientific libraries, it is an attractive choice for algorithmic development and investigative data analysis. Further, as a multipurpose language, it is frequently used not only in academic applications but also in industry (Pedregosa, 2011).

Furthermore, Python enables code reusability and flexibility. It has a fast edit-inspect-debug which makes debugging a straightforward task in Python programs. It has its own debugger written in Python itself. Also, Python includes a variety of third-party components present in the Python Package Index (PyPI).

In conclusion, Python is the most suitable programming language for the current study, due to its abundance of functions, readily available library of toolboxes and its vast knowledge base.

Arestis Pavlides

2. Literature review

In this chapter, necessary background knowledge is presented in order to describe ANPR-related technology. The definition and brief description of the most critical aspects of the underlying code will enable the comprehension of the code syntax subsequently presented in chapter “Code Structure”. The sequencing of the code functions, as utilized in this thesis, can be altered to serve different needs but the overall structure of the code used is similar to several machine vision systems that use Python.

2.1 ANPR technology

Automatic Number Plate Recognition (ANPR) is an application of computer vision technology and can be used in all sorts of applications such as listing all the cars that are in a parking lot, helping crime investigations etc. (Sharifara, 2014; Han, 2020).

Computer vision involves using a camera and a program that takes the information and processes it in order to find a given object or character and answer a certain question that the user asks about the object (Janowski, 2014; About OpenCV, 2022).

There are numerous machine vision tools for Python that use Optical Character Recognition (OCR) technology such as Keras OCR, PyTesseract, Calamari OCR etc. They often use image editing techniques like image binarization, edge detection, Hough transform, blob detection, neural networks (Draghici, 1997; Patel, 2013), cropping, blurring and adjusting the

Arestis Pavlides

image in order to enable the computer to identify and recognize text in images (About OpenCV, 2022).

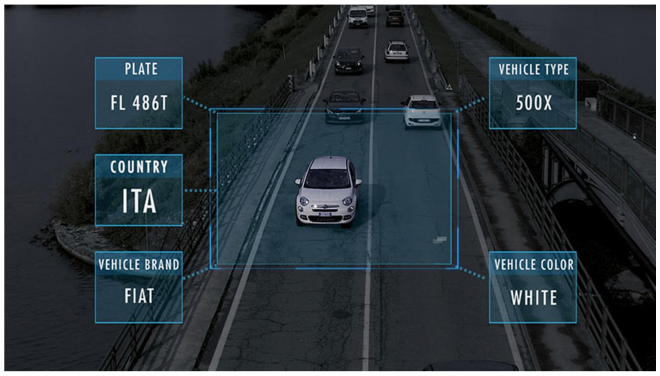


Figure 2-1 Example of Tattile ALPR camera collecting data, such as the vehicle's estimated speed, direction, and country of origin. Some cameras can even classify the vehicle and recognise the brand, colour, class and model.

Using cameras for traffic purposes must be fast and accurate, for avoiding false penalties, providing evidence for the law as well as data for statistical analysis. The aim of the current study is to fill the scientific gap. Unfortunately, in Cyprus at the time this project was executed there was a lack of using traffic cameras for law enforcement with efficiency. ANPR technology has made its appearance in the country but it is not as trusted due to the possibility of false penalties (Chatzivasilis, 2022).

2.2 Python programming language

Created by Guido van Rossum in 1991, Python is a high-level, object-oriented programming language and it is easy to comprehend and use. According to (Python® – the language of today and tomorrow), Guido van Rossum set the following goals for Python in 1999:

- An easy and instinctive language powerful as all other programming languages;

Arestis Pavlides

- Open-source, enabling contribution from everyone to its development;
- An easy and readable code;
- To have short development times.

Python is widely used by researchers in several scientific domains, it is used by companies around the world for data processing, and it supports different styles of programming including structural and object-oriented providing the user numerous programming capabilities (Srinath, 2017).

Some of the flexibilities that Python provides are its ability to use modular components that were designed in other programming languages. For example, you "*can write a program in C++ programming language and import it to python as a module and design a GUI for that program. The high-level language serves as a "glue" to tie modules and components together to rapidly create specialized applications*" (Sanner, 1999). In the community of software development this concept is known, but it can be used in a variety of other fields, for example molecular computations.

Moreover, a model exported to Python language can serve as a reference point for a final program making Python a very good choice for program writing. Though scripting in Python has some requirements on user knowledge in programming skills, it is designed to be learnt easily (Dobesova, 2011).

As a result, Python has very clear syntax and it is a suitable language for both learning and real-world programming making it a great choice for this project.

Arestis Pavlides

2.3 Python libraries

In order to reduce the writing time of a code, programming languages have a set of pre-written functions that are called libraries. They make it simpler for the user to write and compute the code for a given task.

For example, the function that computes the square root of a number, which is used frequently in scientific research, is pre written in a math library. A library eliminates the need for writing every time this function is when a new program is created.

Libraries help the user recall the function in different programs and reduce the risk of bugs in the code. These set of functions in a library were created in order to accelerate the speed that the code is written and tested. The libraries for python programming language used in this project are Opencv, Numpy and Pytesseract OCR and are discussed in the following chapters. These libraries enable the program to read and process images and extract important features from them like shapes and characters.

2.4 Integrated Development Environment (IDE)

IDE is a software that provides the capability of constructing applications by using a graphical user interface (GUI) along with frequently used developer tools.

An IDE usually contains a text editor that can assist in writing software code by using functions and text auto-completion. Also, it checks for bugs while the code is being written. It uses automations that reduce repeatable tasks. It has a compiler that turns the computer source code into binary code, packages it

Arestis Pavlides

and runs tests. Furthermore, an IDE has a debugger which is a program for testing other programs and can graphically display the location of a bug in the original code.

There were multiple tests that were made, aiming to achieve the program's scope, and an IDE made the composition of the code and the insertion of the libraries much easier. In this project, the Pycharm IDE was used in order to test the code for the ANPR program which will be discussed in the next chapter.

2.5 PyCharm IDE

PyCharm is an application developed by the firm 'JetBrains' as an IDE for Python (Islam, 2015). It is commonly used for Python application development. Some massive organizations use PyCharm as their Python IDE. PyCharm is an application that enables the user to write Python scripts/code and run them in a user-friendly environment. The code syntax is checked by the application. All the necessary dependencies for a given project are installed separately, eliminating the risk of interference between the libraries. PyCharm was used due to its convenience in writing Python scripts and easy access to multiple projects. All of this while learning some useful tips about the Python language.

2.6 Region(s) Of Interest (ROI)

ANPR technology handles a large amount of image data in order to function. Image processing is a high computational task and the PC running the code has to analyze thousands of pixels in an image (and/or video frame) and

Arestis Pavlides

determine if it contains the information of a character. In order to reduce the amount of data being processed a region of interest (ROI) has to be identified and then the original image has to be cropped. Locating a region of interest lessens the computational cost and the code entanglement. For example, a frequent size for an image has 1024x768 resolution and contains a total of 786,432 pixels. The region of interest for this project is the license plate bounding box and it may be responsible for nearly 10% of the original image area. This magnifies the simplification of the algorithm structure and overall process time (Mitra, 2016; Badr, 2011).

The method for determining the ROI is based on the Haar cascade classifiers which will be further described in the chapter 2.7.

2.7 Haar cascade classifier

The Haar feature-based cascade classifier is an object detection technique created by P. Viola and M. Jones (Viola and Jones, 2001). Haar-like features works by using machine learning based approach (involving AdaBoost (Freund, 1997)). A cascade function is trained in a large number of positive and negative images and then upgrades their classifier. It‘s been used to locate objects amidst different images by sliding a fixed size window across our image at multiple scales. At each of these phases, our window stops, computes some features, and then classifies the region as the positive images have a license plate and negative images without a plate) (Sharifara, 2014). The classifier uses the positive and negative images to train its classifier. Therefore, the classifier is able to detect objects in other images by extracting features from them. This requires a bit of machine learning. We

Arestis Pavlides

need a classifier that is trained in using positive and negative samples of a number plate. Given these positive and negative data points, we can “train” a classifier to recognize whether a given region of an image contains number plate. However, using a fixed sliding window and sliding it across every (x, y)-coordinate of an image, followed by computing these Haar-like features, and finally performing the actual classification can be computationally expensive. OpenCV can perform plate detection using a pre-trained Haar cascade like the one used in this project. Using Viola algorithm it declares numerical values for features (e.g. edges, lines) effectively with the concept of integral image (or summed-area table), which trumps the default computationally-heavy way of subtracting sums of pixels across multiple regions of an entire image.

In addition, it uses the ‘Cascade of Classifiers’. This means that instead of applying hundreds of classifiers for the many features within the image at one go (which is very inefficient), the classifiers are applied one by one.

For instance, in an image of a human face if the first classifier for the ‘eyes’ feature has failed (i.e. fail to detect human eyes in the image), the algorithm does not bother applying the subsequent classifiers (e.g., nose, mouth, etc.). Instead, it stops and declares that no face is detected.

2.8 Thresholding

Thresholding function accepts only grayscale images and provides an output where each pixel value has been mapped to either zero or 255, depending on two parameters – the current pixel value and the threshold limit. Any pixel having value less than the decided limit would be mapped to zero (black) and

Arestis Pavlides

any pixel having value greater than the limit would be mapped to 255 (white). A lot of different and advanced thresholding techniques are available in OpenCV (Sharma, 2019).

2.9 Contours

Contours can be understood as lines joining the boundary of the image based on the changes in neighboring pixel values (CV, 2022). Finding contours in an image is perhaps the most important aspect of this system as our license plate contains characters inside a rectangle therefore it has to be able to recognize the geometry arising from these lines. OpenCV provides versatile functions for finding contours. One drawback of this function is that it does not discriminate between the image and noise and therefore passing this image directly would result in an incorrect output. Thus, some filtering techniques have to be applied on the image in order to reduce noise and increase the possibility of detecting the plates (Sharma, 2019).

2.10 Optical Character Recognition (OCR)

Characters in an image like letters, numbers or symbols can be recognized through an OCR library. This library takes the original image and scans it in order to find the text. In this project we try to scan only the region of interest, in this case the number plate, in order to find extract the characters.

An industrial OCR uses algorithms studied by different researchers in the areas of image processing, pattern recognition, machine learning, language analysis, document understanding. There is no universal algorithm that is

Arestis Pavlides

suitable for any OCR problem, thus recent systems try to adapt themselves to the given features of the image or document that is processed (Marosi, 2007).

There are currently several OCR libraries that can be used in Python, some of which being the following: *PyTesseract*, *Nautilus OCR*, *Keras OCR*, *Calamari OCR*, *EasyOCR*, *Kraken OCR*. In this project, the tool that is used for text recognition is the *PyTesseract* library, which is discussed further in chapter 3.2.3.

Arestis Pavlides

3. Software code structure

The developed code is structured as seen in the below figure.

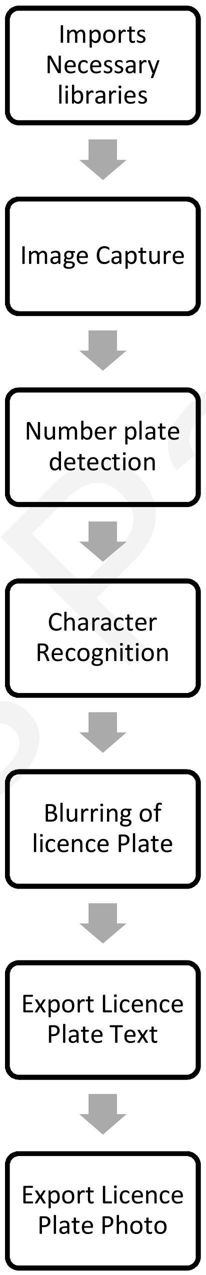

Imports Necessary libraries
Image Capture
Number plate detection
Character Recognition
Blurring of licence Plate
Export Licence Plate Text
Export Licence Plate Photo

Figure 3-1 Code Structure diagramm

Arestis Pavlides

The code that was used in this project was based on previous ANPR algorithms, as listed in Appendix A, with key implemented changes to them listed below:

- The use of a blurring function for the plates, as shown in Table 3-11 Command for blurring the plates creating rectangle around them and put text on the created rectangleTable 3-11, was added.
- The use of an OCR tool, in this case *PyTesseract*, for video streams was used for the conversion of an image to text as shown in Table 3-10.
- Fine-tuning of function values, as seen in chapter 4.1. These function values were changed from the default ones in the original codes, to fit the needs of this study.
- The insertion of a cropping command for the frame as shown in Table 3-6.
- The insertion of a rotating command for original image from video streams as shown in Table 3-7.
- The image saving command after detection of a license plate as shown in Table 3-12.

The commands shown in Table 3-1 for:

- sharpening the image,
- applying Gaussian blur, and

Arestis Pavlides

- applying canny edge detection

were also tested in order to see if the results' accuracy was significantly altered.

The final code didn't have these commands because the resulting accuracy wasn't improved by using them.

Table 3-1 Commands for : sharpening the image , applying Gaussian blur and applying canny edge detection

Python command	Code task

kernel = np.array([[0, -1, 0],
[-1, 5, -1],
[0, -1, 0]])
 	
image_sharp = cv2.filter2D(src=imgGray, ddepth=-1, kernel=kernel)
	sharpening the image

image = cv2.GaussianBlur(imgGray, (5,5), 0)	applying Gaussian blur
edge = cv2.Canny(image_sharp, 255,255, 5)	applying canny edge detection

The code segments that retained similarities with the ANPR algorithms listed in Appendix A was:

- Setting the Image size and brightness for webcam videostreams,
- The minimum area for the plate to be detected,

Arestis Pavlides

- The transformation of the image to grayscale,
- The use of the existing haar-cascade file.

3.1 Computer specifications

The PC that was used for the development and execution of the code was Dell Latitude E5470, with the following technical specifications:

Table 3-2 PC Specifications for writing and running the code

PC specifications	
Feature	Specification
Processing Unit (CPU):	Intel Core i5 - 6300U @ 2.40 Ghz
Memory (RAM)	16 GB - DDR4
Graphic Card	None
Web camera	Yes • Camera resolution 0.92 megapixels • HD Panel Resolution 1366 1377 x 768 pixels • FHD Panel Resolution 1920 x 1080 pixels • HD Panel Video Resolution (maximum) 1366 x 768 pixels • FHD Panel Video Resolution (maximum) 1920 x 1080 pixels

Arestis Pavlides

PC specifications	
	• Diagonal viewing angle 74°
Hard Disk	400 GB
Operating System	Microsoft Windows 10
Pycharm Application	Installed
PyTesseract	Installed

The overall execution speed of the program was sufficient, given the PC specifications. For faster results a PC with a graphic card and better CPU would have a reduction in the calculation speed.

3.2 Necessary libraries for running the code

In the first lines of the developed code, all the necessary dependencies are loaded in the python environment (PyCharm Application). These dependencies are:

• Opencv-python	version 4.6.0.66
• Numpy	version 1.23.3
• Pytesseract OCR	version 0.3.10

These libraries contain all fit-to-task functions in order to achieve the calculations needed to perform image processing and information extraction.

The role of each library will be discussed in subsequent chapters.

Arestis Pavlides

3.2.1 OpenCV Python library, version 4.6.0.66

In this project, the *OpenCV* library was used as a tool that can read an image and detect a license plate shape (contour) in it. *OpenCV* was used because it is an open-source library and it contains the necessary functions for reading an image or video frame. The video source can be either offline (i.e. a video file) or live (i.e. a web camera). The library has the capability of detecting contours and shapes and extracting their area in order to determine if the rectangle that it reads has the potential of being a license plate (i.e. a rectangle). A detected rectangle is the region of interest (ROI) for this project and it takes a very small portion of the image that is analyzed. Since all license plates have common width to length ratio the tool searches this kind of geometry in the picture frame in order to extract the characters. With the aim of finding this geometry, the library uses a pre-trained Haar Cascade XML file. This geometry is embedded in the weights file "*haarcascade_russian_plate_number.xml*" which contains a trained model for detecting our region of interest. After this ROI is found, a bounding box around the number plate is created and the image is cropped in order to reduce data for the optical character recognition which will be used in the next step. The functions that were imported from *OpenCV* library start with the expression "cv2." and are shown in chapter 4 (The code in).

3.2.2 NumPy version 1.23.3

NumPy is an array programming library for the Python language, and it is widely used in scientific research as it provides fundamental algorithms for scientific computing. It is used in order to enable Python to do computations between multidimensional arrays (Harris, 2020). For image processing

Arestis Pavlides

NumPy supplies numeric computing capabilities and is a dependency of *OpenCV*'s Python bindings (Howse, 2013).

An image in *OpenCV* can be seen as an array of pixels. In a grayscale image, every pixel has values for y,x coordinates and a value from 0 to 255 where 255 is white and 0 is black. We can access these values through *NumPy* and perform calculations.

NumPy also includes multiple array functions found in linear algebra, and Fast Fourier Transformations (FFT). "*Its goal is to create the corner-stone for a useful environment for scientific computing*" (Oliphant, 2006).

3.2.3 PyTesseract OCR 0.3.10

PyTesseract OCR is a library that contains the algorithms for extracting optical characters from an image and convert it into text. It is able to extract characters from an image file, in this case the characters that are been searched are latin alphabet characters A-Z and numeric values 0-9, since all license plates in Cyprus contain this sort of format.

The *PyTesseract* process for making the recognition possible involves several steps. Tesseract assumes that its input is a binary image with optional polygonal text regions defined. The first step is a connected component analysis in which the outlines of the components are stored. At this stage, outlines are gathered together, purely by nesting, into blobs. Blobs are organized into text lines, text lines are broken into words. Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance

Arestis Pavlides

to more accurately recognize text lower down the page. A final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate small-cap text. Tesseract is behind the leading commercial engines in terms of its accuracy. Its key strength is probably its unusual choice of features. Its key weakness is probably its use of a polygonal approximation as input to the classifier instead of the raw outlines (Smith, 2007).

The latest releases of Tesseract support deep-learning-based OCR that is significantly more accurate. The OCR engine itself is built on a Long Short-Term Memory (LSTM) network, a kind of Recurrent Neural Network (RNN).

3.3 Image capture

OpenCV provides image capture from video file. In the table below the commands for obtaining the image in a video file or video stream are given

Table 3-3 Commands in python for obtaining the image in a video file or video stream

Python command	Code task
cap = cv2.VideoCapture("leof strovol2.mp4")	Imports the Video File
cap = cv2.VideoCapture(0)	Imports image from camera

In order for the code to scan every frame of the video and save it, a while loop had to be written. In the table below the command for performing this task is given:

Table 3-4 While loop which scans every frame of the video and save it , in the variable “img”

Python command	Code task

Arestis Pavlides

While True: success, img = cap.read()	Scans every frame of the video stream and saves it, in the variable “img”

For every frame that the code reads is performs some modifications to the original image in order for the Haar cascade classifier to detect the license plate. In the next subchapter number, the number plates’ detection commands are explained.

3.4 Number plate detection

Most of the detection algorithms used for detecting license plates incorporate a lot of techniques, according to their application, in order to make the program to detect license plates like color changing, blurring, cropping of the image. For example, by changing the color of the image to grayscale, the background of the plates, stand out more easily due to most of the plates having a white color which is detected faster. In this project the first transformation that was chosen is the command which transforms the colored image to grayscale. In the table below the command for performing this task is given:

Table 3-5 Command that Turns image to grayscale

Python command	Code task
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)	Turns image to grayscale

Arestis Pavlides

The second transformation was the cropping of the original frame to a proportion that the remaining image only contained the license plates being studied. For example, in video file 1 by using the command

Table 3-6 Command that crops the indicated portion of the image in video file 1

Python command	Code task
img = img[200:600, 0:1000]	Crops the indicated portion of the image in video file 1

The third transformation was the rotation of the original image to any degree needed, in order to align horizontal the plate text and help the OCR detect it more easily. As shown in the table below, the degree chosen for video file 1 to be rotated was 4 degrees clockwise and was made possible by the commands :

Table 3-7 Rotates images from video stream by 4 degrees clockwise

Python command	Code task

M = cv2.getRotationMatrix2D([0, 0], 4, 1.0)
 	
img = cv2.warpAffine(img, M, (0, 0))
	Rotates images from video stream by 4 degrees clockwise

After the image color is transformed, the open source Haar cascade classifier was used with a trained model for detecting the plates by searching rectangles of a given geometry. The command for using the classifier is given in the table below:

Arestis Pavlides

Table 3-8 Command for searching the rectangle shape that number plates have using the classifier from haar cascade file

Python command	Code task
numberPlates = plateCascade.detectMultiScale(imgGray, 1.2, 4)	Looks for plate's shape using the cascade classifier

The model was pre-trained in detecting Russian plates which had similar shape and size like the plates used in Cyprus. The trained model was chosen because it is an open-source classifier that is widely used by researchers and has good accuracy for detecting the plates.

3.5 Character recognition

After the license plate is detected the minimum area of the plate is given to help program to detect the plate's rectangle size and location in the image. The portion of image that has the license plate is cropped and is saved in the variable "imgROI".

This is done by the commands in the table below:

Table 3-9 Command for setting the plate's rectangle minimum area and cropping that portion of image

Python command	Code task

for (x, y, w, h) in numberPlates:
area = w * h
 if area > minArea:
	minimum area of the plate for detecting the rectangle size and location in the image

imgRoi = img[y:y + h, x:x + w]	Crops the license plate is saves it in the variable "imgROI"

Then the characters are recognized by using the command:

Table 3-10 Command for performing character recognition with Pytesseract OCR tool and printing the output

Python command	Code task

Arestis Pavlides

pytesseract.image_to_string(imgRoi, config=f'--psm 7 --oem 3 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')	takes the cropped image named “imgROI” and performs character recognition using PyTesseract OCR.

3.6 Blurring of license plate

After the detection of the license plates, they had to be blurred out by hiding information that is on the plates for privacy issues. The command for blurring the part of the image that contained the plate, creating a rectangle around the plates and put text on the created rectangle is given in the table below:

Table 3-11 Command for blurring the plates creating rectangle around them and put text on the created rectangle

Python command	Code task
plate = cv2.blur(imgRoi, ksize=(20, 20)	Blurs the plates
cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 10)	Creates rectangle in the image around the plates
cv2.putText(img, "NumberPlate", (x, y - 5),	Puts text in above rectangle created in the image
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)	Chooses font for the text

Arestis Pavlides

3.7 Exporting license plate text

The license plate text is exported or printed in the PyCharm Terminal after Tesseract converts the image to text. The text is shown while the program runs. This is done from the command given in the **Error! Reference source not found.** below:

Table 3-12 Command for printing the output of the Tesseract OCR

Python command	Code task
print(pytesseract.image_to_string(imgRoi, config=f'--psm 7 --oem 3 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789'))	Print the OCR output in the terminal

3.8 Saving image

The license plates after are located and cropped are saved in an image file in the python project folder with the extension '.jpeg' by use of the following command:

Table 3-13 License plates' command to be saved in an image file

Python command	Code task
cv2.imwrite("images" + str(count) + ".jpg", imgRoi)	saves plate picture in an image file

Arestis Pavlides

4. The code in Python

In this chapter, further details are shown about the choices for the function values of the code that is used in this research work.

4.1 Function values

There is no universal calibration that performs optimization for any given video file. The function values used in the project were chosen because of the overall increase of accuracy of the OCR and Haar cascade file.

4.1.1 OCR function

In order for the OCR to perform with higher accuracy, adjustments in the image and OCR library had to be made through their function values. For example, through the below shown command.

```
print(pytesseract.image_to_string(imgRoi,
                    config=f'--psm 7 --oem 3 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ012345
6789'))
```

Figure 4-1 Command that performs OCR in the image and prints the output in the terminal

The command in Figure 4-1 signals the OCR to search only for the certain character list numbers from 0-9 and letters A-Z. This relates to the fact that number plates in Cyprus have only these characters in them.

The variables “psm” (Page Segmentation Modes) and “oem” (OCR Engine Mode) were calibrated. In the tables below the options that were available for “psm” and “oem” are listed.

Arestis Pavlides

Table 4-1 Options for setting Tesseract to only run a subset of layout analysis and assume a certain form of image.

--psm N
Sets Tesseract to only run a subset of layout analysis and assume a certain form of image. The options for N are:
0 = Orientation and script detection (OSD) only.
1 = Automatic page segmentation with OSD.
2 = Automatic page segmentation, but no OSD, or OCR. (not implemented)
3 = Fully automatic page segmentation, but no OSD. (Default)
4 = Assume a single column of text of variable sizes.
5 = Assume a single uniform block of vertically aligned text.
6 = Assume a single uniform block of text.
7 = Treat the image as a single text line.
8 = Treat the image as a single word.
9 = Treat the image as a single word in a circle.
10 = Treat the image as a single character.
11 = Sparse text. Find as much text as possible in no particular order.
12 = Sparse text with OSD.
13 = Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.

The options 5-13 for "psm" were chosen for calibration. The choices were tested with the same video files for objective measurement. The resulting accuracies can be seen in the tables later in the chapter Results Analysis.

For the variable "oem", which selects the OCR Engine mode, the option 3 was chosen as seen in the image below.

Table 4-2 Options for Setting Tesseract Engine mode.

--oem N
Specify OCR Engine mode. The options for N are:
0 = Original Tesseract only.
1 = Neural nets LSTM only.
2 = Tesseract + LSTM.
3 = Default, based on what is available.

Arestis Pavlides

4.1.2 Picture size and brightness

Other variables that can were calibrated were the picture frame width, the height and the brightness. The values set for these parameters depended on the video quality and lightning that were chosen to process. This was done in order for the cascade classifier to detect the plates in the images more accurately by separating the background from the plates. Subsequently, the change in these variables could help increase the detection accuracy of the image characters by Pytesseract OCR.

The commands for setting these parameters are shown below.

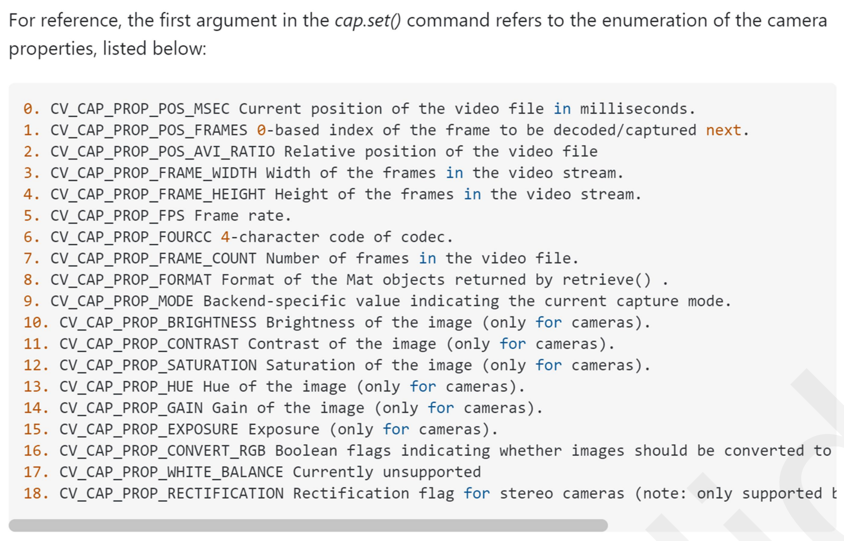
Table 4-3 Commands for picture size and brightness for webcam video stream

Python command	Code task

cap.set(3,640)
cap.set(4,480)
cap.set(10, 150)
	Sets image size and brightness in the desired value.

In Figure 4-2 one can see the various options that can be adjusted by the command *cap.set* (option 1-18, value) in order to gain more control over the image parameters.

Arestis Pavlides


For reference, the first argument in the *cap.set()* command refers to the enumeration of the camera properties, listed below:

```
0. CV_CAP_PROP_POS_MSEC Current position of the video file in milliseconds.
1. CV_CAP_PROP_POS_FRAMES 0-based index of the frame to be decoded/captured next.
2. CV_CAP_PROP_POS_AVI_RATIO Relative position of the video file
3. CV_CAP_PROP_FRAME_WIDTH Width of the frames in the video stream.
4. CV_CAP_PROP_FRAME_HEIGHT Height of the frames in the video stream.
5. CV_CAP_PROP_FPS Frame rate.
6. CV_CAP_PROP_FOURCC 4-character code of codec.
7. CV_CAP_PROP_FRAME_COUNT Number of frames in the video file.
8. CV_CAP_PROP_FORMAT Format of the Mat objects returned by retrieve() .
9. CV_CAP_PROP_MODE Backend-specific value indicating the current capture mode.
10. CV_CAP_PROP_BRIGHTNESS Brightness of the image (only for cameras).
11. CV_CAP_PROP_CONTRAST Contrast of the image (only for cameras).
12. CV_CAP_PROP_SATURATION Saturation of the image (only for cameras).
13. CV_CAP_PROP_HUE Hue of the image (only for cameras).
14. CV_CAP_PROP_GAIN Gain of the image (only for cameras).
15. CV_CAP_PROP_EXPOSURE Exposure (only for cameras).
16. CV_CAP_PROP_CONVERT_RGB Boolean flags indicating whether images should be converted to
17. CV_CAP_PROP_WHITE_BALANCE Currently unsupported
18. CV_CAP_PROP_RECTIFICATION Rectification flag for stereo cameras (note: only supported b
```

Figure 4-2 Options 1 till 18 for the command "cap.set"(option 1-18,value)

In this project the options 3,4 and 10, for setting the frame width, height and brightness respectively, as seen in Figure 4-2, were chosen to remain constant for the duration of the study.

4.1.3 Size of minimum plate area

Further, the minimum plate area value is set to 500 and is calculated by multiplying the width and height of the rectangle that is found when a plate is recognized by the Haar cascade classifier. After the area is found, the code performs cropping of the initial image. Sometimes, motorcycles and large dumpster trucks mount their plates sideways. This would have to be considered for a highly accurate license plate system. In this project we won't consider this case

Some countries and regions allow for multi-line plates with a near 1:1 aspect ratio; again, we won't consider this edge case.

Arestis Pavlides

The minimum area of the plate could be checked by the command as seen in Figure 4-3 :

```
for (x, y, w, h) in numberPlates:
  area = w * h
  if area > minArea:
    imgRoi = img[y:y + h, x:x + w]
```

Figure 4-3 Command for finding Minimum area

4.1.4 Color of the image

Monochrome (or black & white) representation of image is more appropriate for OCR analysis, because it defines clear boundaries of contained characters (Martinsky, 2007).

The initial image is turned to grayscale by the command seen in the table below:

Table 4-4 Command for turning colored image to gray

Python command	Code task
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)	Turns colored image to grayscale

4.1.5 Edge detection

The command in the table below is used for detecting the edges of the number plate:

Table 4-5 Command for the detection of the number plate using cascade classifier

Python command	Code task
plateCascade.detectMultiScale(imgGray, 1.2, 4)	Detects the edges of the number plate

Arestis Pavlides

The parameters scaleFactor (value=1.2) and Minimum Amount of Neighboring Rectangles (value=4) were calibrated and the resulting accuracy for each option is given in the 'Results' chapter.

The parameter scaleFactor is responsible for specifying how much the image size is reduced at each image scale. In this project scaleFactor is given a value of 1.2 which means that OpenCV will scale the image down by 20% for better matching the car plates.

The parameter minNeighbors specifies how many neighbors each candidate rectangle should have in order for the candidate rectangle to be retained. (OpenCV, 2022). This parameter influences the quality of the detected objects. For this project the value was checked between1-5 and the default value was set to 4.

Arestis Pavlides

5. Results

The Results for our ANPR system are shown in this chapter. The code is able to detect the license plate and extract the characters from video streams. It also blurs the license plate for privacy issues and the plate detected is exported in an image file as seen in the following images.

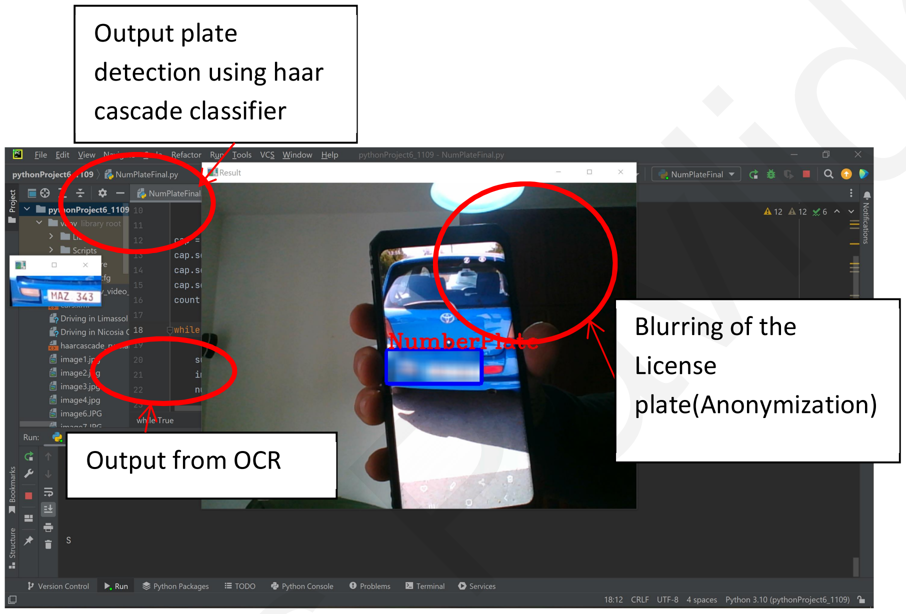
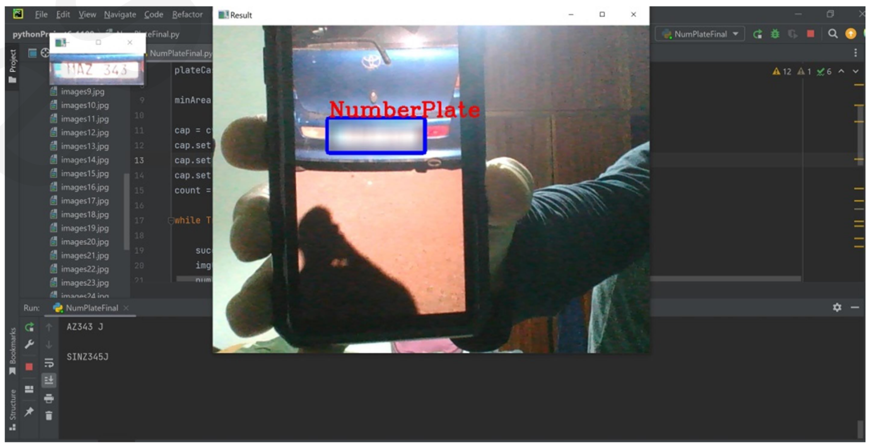

Output plate detection using haar cascade classifier
Blurring of the License plate(Anonymization)
Output from OCR
NumberPlate
MAZ 343

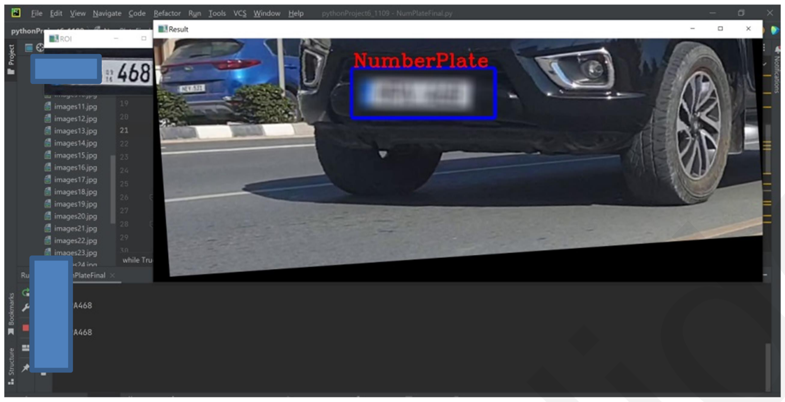
Figure 5-1 Result using PC camera to run the code in Pycharm IDE .Photo in daylight conditions



```
AZ343 J

SINZ345J
```

Figure 5-2 Result using PC camera to run the code in Pycharm IDE. Photo in nightlight conditions (poor accuracy)

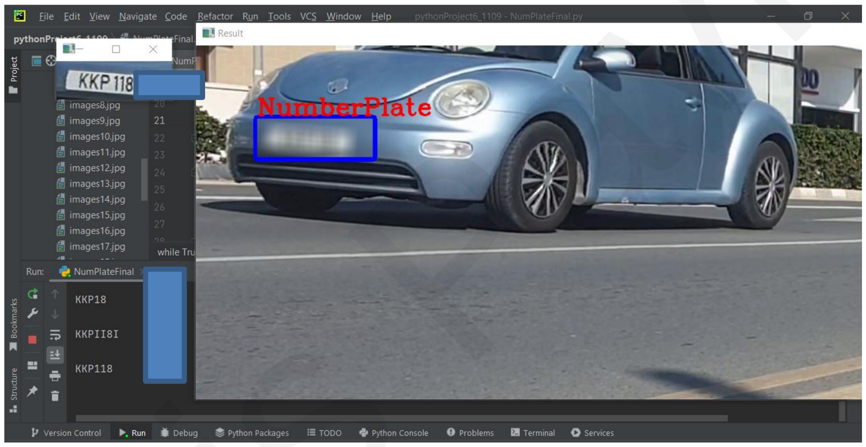


[redacted]

```
A468

A468
```

Figure 5-3 Result using video file 1 to run the code in Pycharm IDE



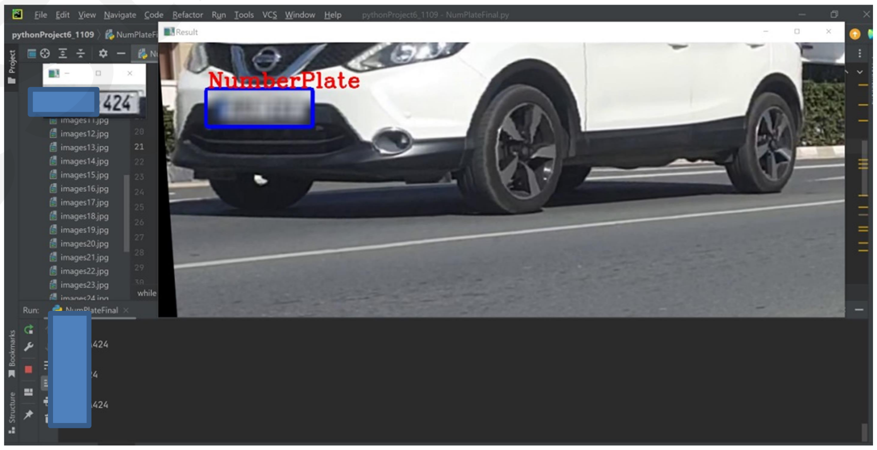
[redacted]

```
KKP18

KKPII8I

KKP118
```

Figure 5-4 Result using video file 1 to run the code in Pycharm IDE (Letter i is mistaken with number 1)



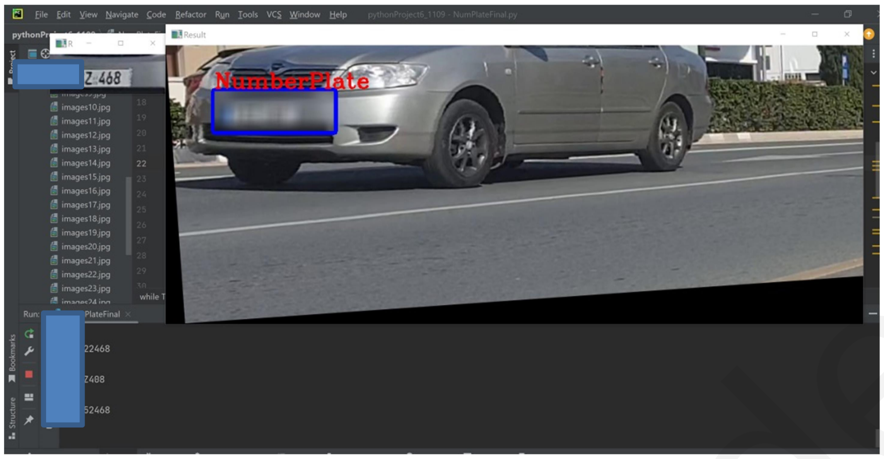
[redacted]

```
424

4

424
```

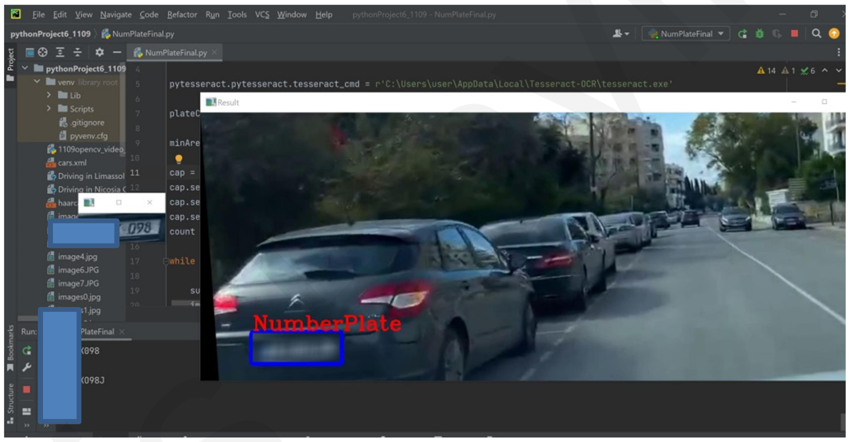
Figure 5-5 Result using video file 1 to run the code in Pycharm IDE(Letter M is mistaken with letter H)



```
while T
```

```
[redacted]22468
[redacted]Z408
[redacted]52468
```

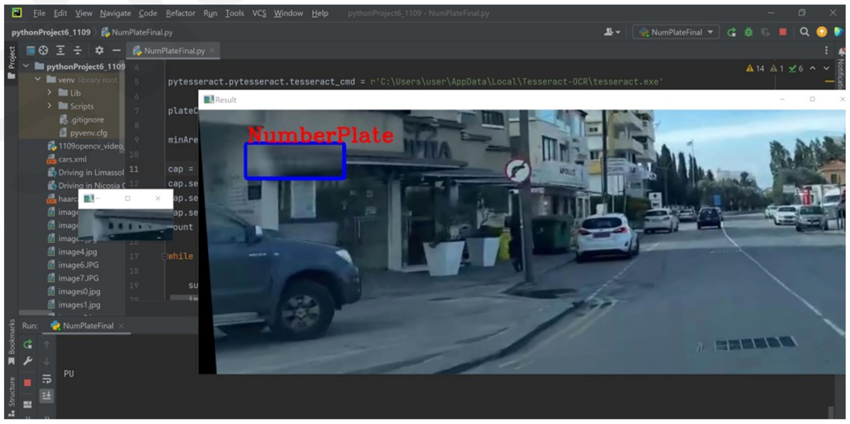
Figure 5-6 Result using video file 1 to run the code in Pycharm IDE(Letter “S” is mistaken with number “5”)



```
pytesseract.pytesseract.tesseract_cmd = r'C:\Users\user\AppData\Local\Tesseract-OCR\tesseract.exe'
```

```
[redacted]X098
[redacted]X098J
```

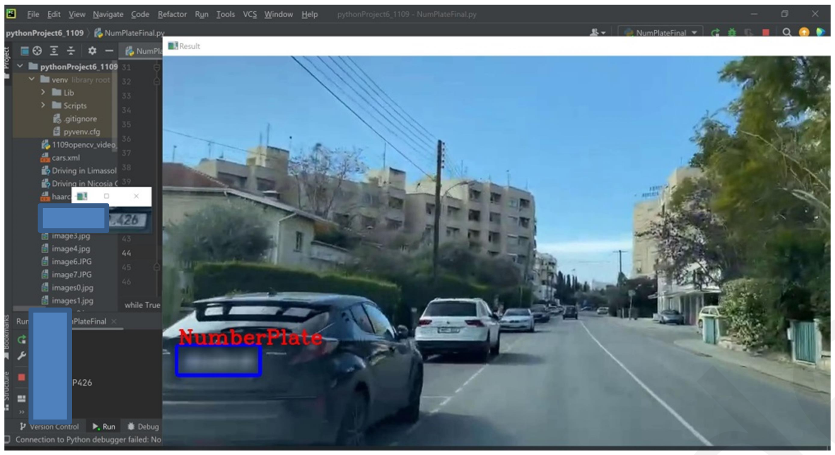
Figure 5-7 Result using video file 2 to the code in Pycharm IDE



```
pytesseract.pytesseract.tesseract_cmd = r'C:\Users\user\AppData\Local\Tesseract-OCR\tesseract.exe'
```

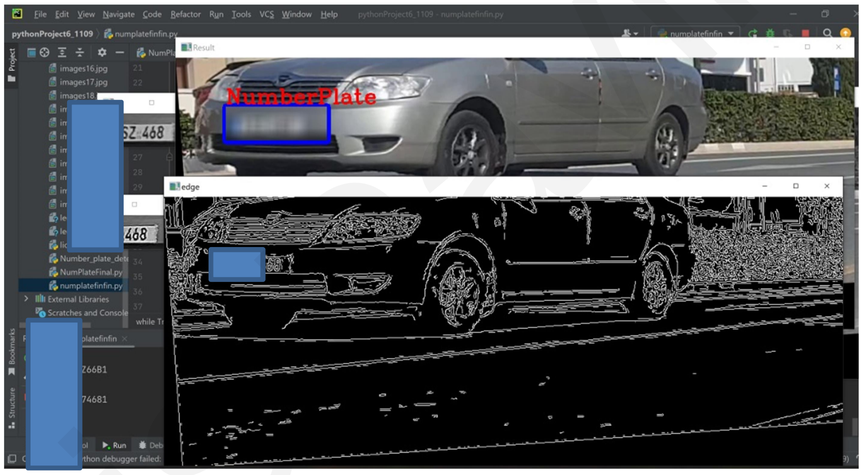
```
PU
```

Figure 5-8 Result using video file 2 to run the code in Pycharm IDE (Number plate detection fails)



```
[redacted]
P426
```

Figure 5-9 Result using video file 2 to run the code in Pycharm IDE



```
[redacted]
Z66B1
74681
```

Figure 5-10 Result using canny edge detection and video file 1 to run the code in Pycharm IDE

5.1 Approaching system accuracy as a two-stage problem

As seen in Figure 5-4,Figure 5-5,Figure 5-6,Figure 5-8 , although number plates were detected, the resulting text output from the OCR was failing to be accurate. In order to comprehend the reasons that affected the resulting accuracy, the system that is described in this project was chosen to be examined in two main stages:

a) automatic number plate detection

b) automatic number plate (OCR) recognition

The subsequent results' accuracy is given in the following chapters 5.2, 5.3, 5.4, and the accuracy is discriminated for the two video files that are used and for each stage of the system process.

5.1.1 First stage's accuracy results (Automatic number plate detection)

In order to evaluate each step involved in this process, two video files having 100 cars were used. The accuracy of the first stage was calculated by dividing A (the total number of accurate plate detections) with N (Total plate detections) which is the sum of F (Total Number of false detected Plates) and C (total number of car plates in the video). The equation for calculating automatic number plate detection accuracy is given in Figure 5-11:

Accuracy of automatic number plate detection (%) = $\frac{A}{F+C} * 100 = \frac{A}{N} * 100$

Where:

F: Total Number of false detected Plates (not cars)

C: Total Number of car plates in the video

A: Total Number of accurate plate detections

Figure 5-11 Equation for calculating automatic number plate detection accuracy

For example in Figure 5-8 the plate detection stage was giving a wrong part of an image as a license plate and that was counted in F (Total Number of False detected Plates). If a car plate was in the frame and the algorithm didn't detect it, that was counted in C (Number of Cars in the video). Only if the code

Arestis Pavlides

detected an actual plate of a car was counted in A (Total accurate plate detections).

The accuracy of the number plate detection stage was also tested for its connection with the parameters seen in chapters 5.3 Scale factor accuracy results and chapter 5.4 Minimum amount of neighboring rectangles accuracy results.

The accuracy results of the first stage for each parameter value and for each video file can be seen in Table 5-1,Table 5-2,Table 5-3.

5.1.2 Second stage’s accuracy results (automatic number plate (OCR) recognition)

The second stage’s accuracy which is OCR recognition output was measured in the videos by dividing O (Total accurate number of OCR recognitions) with S (Total number of OCR recognitions) which is the sum of W (Total number of false OCR detection of plates) and P (Total number of car plates in the video). The equation for calculating automatic number plate (OCR) recognition accuracy is given in **Error! Reference source not found.**.

Accuracy of automatic number plate (OCR) recognition (%) = $\frac{O}{W+P} * 100 = \frac{O}{S} * 100$

Where:

W: Total Number of false OCR detection of Plates (not cars)

Figure 5-12 Equation for calculating automatic number plate detection accuracy

Arestis Pavlides

The accuracy results of the second stage for each parameter value and for each video file can be seen in Table 5-1,Table 5-2,Table 5-3.

This stage was highly affected by the tool used to recognize the characters in the image, which is pytesseract library in this case. The quality of the video image and any modifications in the original image affected the accuracy of the OCR.

The accuracy of the OCR output was also tested for its connection with the parameter in 5.2 5.2 Page segmentation modes (PSM) accuracy results .

5.2 Page segmentation modes (PSM) accuracy results

In order to appraise the accuracy of the OCR library, a sample of about 100 cars were introduced to the developed software through two video files and the performance for each car sample was recorded by changing the parameter “psm” and keeping the other values like scale factor, picture size, minimum amount of neighboring rectangles constant and to the values seen in Table 6-1. In the table below one can see the accuracy scored for the given video file and for each option of the psm.

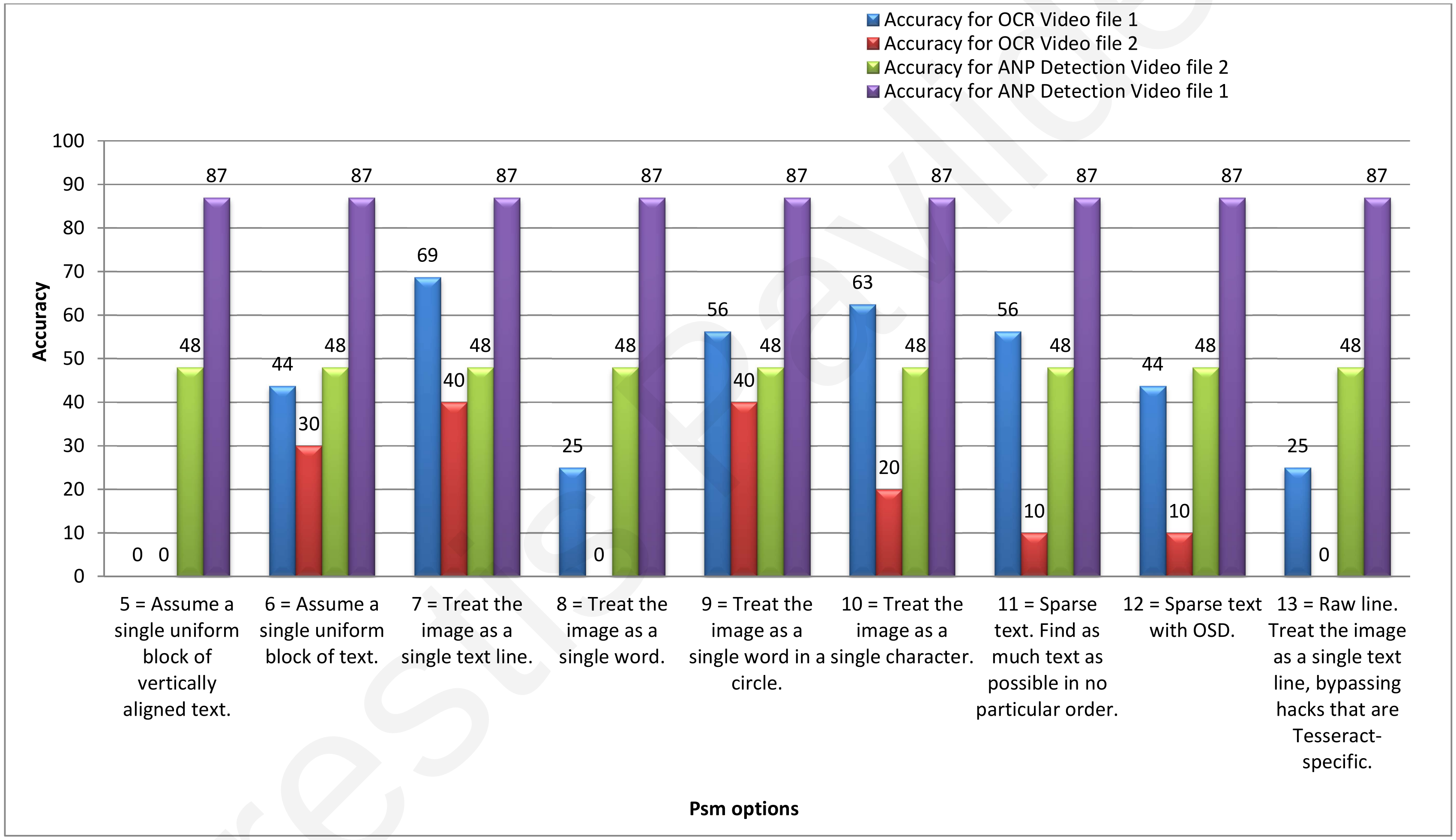
Table 5-1 Accuracy scored for the given video file for each option of the psm

--psm N with scale factor =1.2 and minNeigh=4				
N	Accuracy for number plate detection in Video file 1 (%)	Accuracy for number plate detection in Video file 2 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 1(%)	Accuracy of automatic number plate (OCR) recognition in Video file 2 (%)
5 = Assume a single uniform block of vertically aligned text.	87	48	0	0
6 = Assume a single uniform block of text.	87	48	44	30
7 = Treat the image as a single text line.	87	48	69	40
8 = Treat the image as a single word.	87	48	25	0

Arestis Pavlides

--psm N with scale factor =1.2 and minNeigh=4				
N	Accuracy for number plate detection in Video file 1 (%)	Accuracy for number plate detection in Video file 2 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 1(%)	Accuracy of automatic number plate (OCR) recognition in Video file 2 (%)
9 = Treat the image as a single word in a circle.	87	48	56	40
10 = Treat the image as a single character.	87	48	63	20
11 = Sparse text. Find as much text as possible in no particular order.	87	48	56	10
12 = Sparse text with OSD.	87	48	44	10
13 = Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific.	87	48	25	0

Arestis Pavlides



Psm option	Accuracy for OCR Video file 1	Accuracy for OCR Video file 2	Accuracy for ANP Detection Video file 2	Accuracy for ANP Detection Video file 1
5	0	0	48	87
6	44	30	48	87
7	69	40	48	87
8	25	0	48	87
9	56	40	48	87
10	63	20	48	87
11	56	10	48	87
12	44	10	48	87
13	25	0	48	87

Figure 5-13 Chart of the resulting Accuracy for a given psm option with constant scale factor =1.2 and minNeigh=4

Arestis Pavlides

5.3 Scale factor accuracy results

As with the previous appraisal, in order to measure the accuracy of the scale factor parameter a sample of about 100 cars were introduced in the program through two video files and the performance for each sample was recorded by changing the parameter scale factor and keeping the other values like psm minimum amount of neighboring rectangles constant and to the values seen in Table 6-1. In the table below, one can see the accuracy scored for the given video file for each value from 1-2 with an interval of 0.1.

Table 5-2 Accuracy scored for the given video file for each value from 1-2 with an interval of 0.1

Scale factor N with psm =7 and min Neighb=4				
N	Accuracy for number plate detection in Video file 1 (%)	Accuracy for number plate detection in Video file 2 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 1 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 2 (%)
1.01	7	3	0.0	0.0
1.1	64	12.5	62.5	30.0
1.2	87	48	69.0	30.0
1.3	75	22.5	69.0	20.0
1.4	50	15	25.0	10.0
1.5	53	12.5	38.0	10.0
1.6	47	10	25.0	0.0
1.7	35	7.5	19.0	0.0
1.8	30	5	25.0	0.0
1.9	30	5	25.0	0.0
2.0	45	17.5	31.0	20.0

Arestis Pavlides

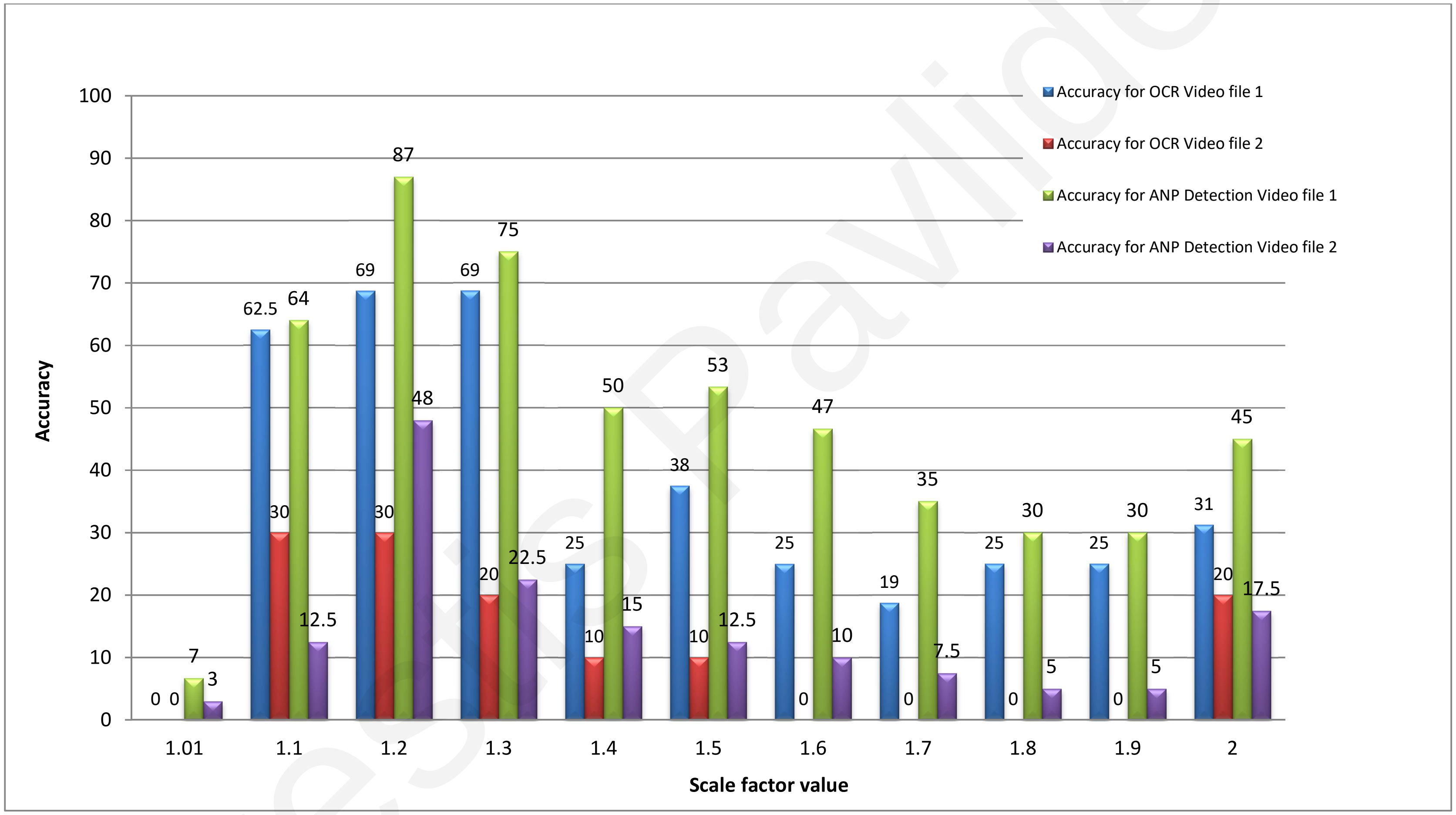


Figure 5-14 Chart of the resulting accuracy scored for the given video files for each Scalefactor value

Ares Pavlides

5.4 Minimum amount of neighboring rectangles accuracy results

In order to measure the accuracy of the minimum amount of neighboring rectangles parameter, a sample of about 100 cars were introduced in the program through two video files and the performance for each sample was recorded by changing the parameter minimum amount of neighboring rectangles and keeping the values of psm, scale factor to the values seen in Table 6-1. In the table below you can see the accuracy for the given video files with the parameter neighboring rectangles' value range from 1-5 with an interval of 1.

Table 5-3 Accuracy for the given video files with the parameter neighboring rectangles' value range from 1-5 with an interval of 1.

Minimum amount of neighboring rectangles N with psm=7 and scale factor=1.2				
N	Accuracy for number plate detection in Video file 1 (%)	Accuracy for number plate detection in Video file 2 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 1 (%)	Accuracy of automatic number plate (OCR) recognition in Video file 2 (%)
1	15	32	55	27
2	17	33	65	27
3	20	23	50	18
4	87	48	69	30
5	55	23	33	10

Arestis Pavlides

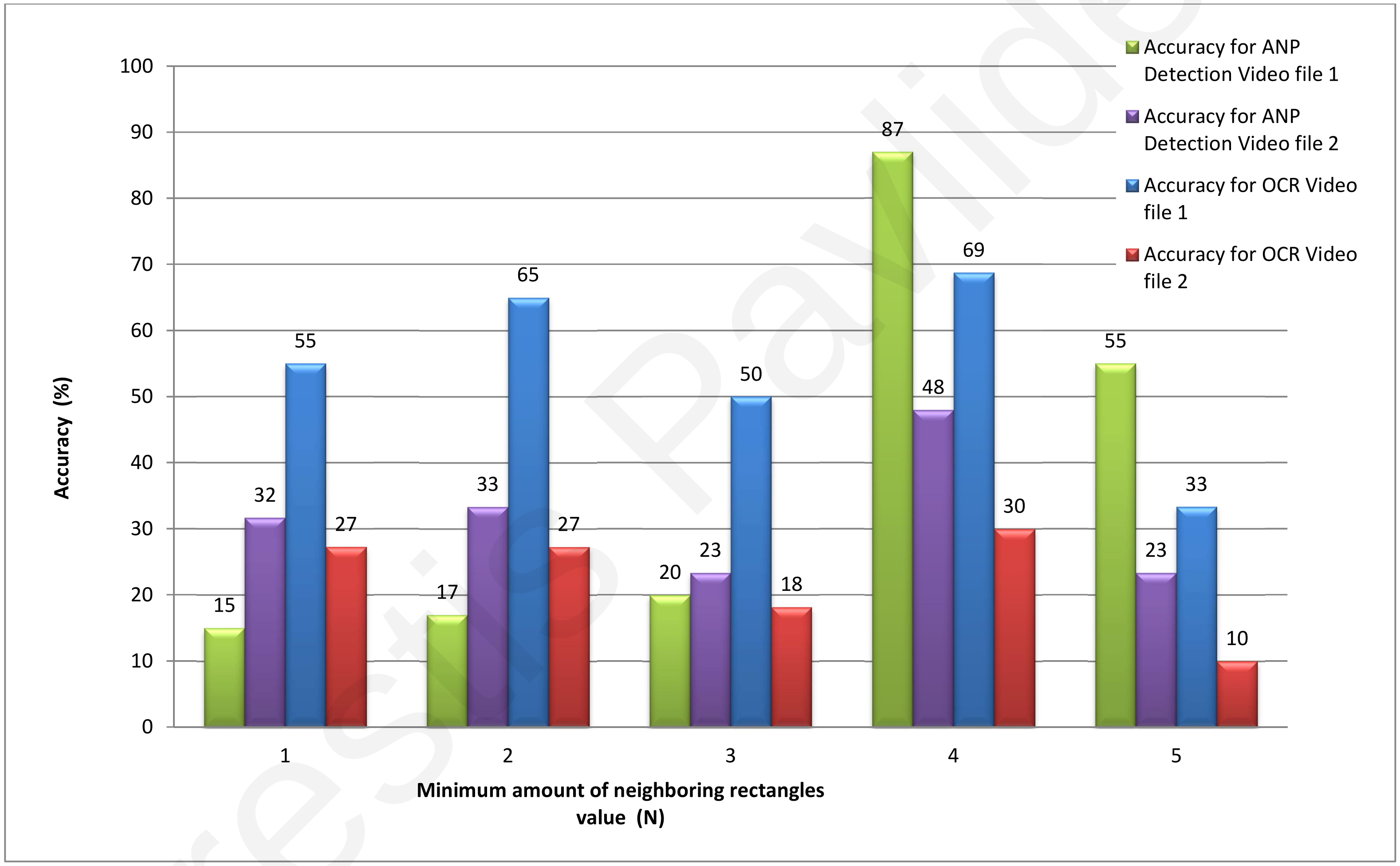


Figure 5-15 Chart of the resulting accuracy scored for the given video files for each Minimum amount of neighboring rectangles value

Arestis Pavlides

6. Results analysis

The overall system accuracy is highest if the parameters values are set as shown in Table 6-1 and the highest accuracy scores for the first stage (Accuracy of number plate detection) in Video file 1 & 2 and for the second stage (Accuracy of OCR recognition in Video file 1 & 2) are given in Table 6-2.

Parameter name	Value with the highest accuracy
PSM	7
Scale Factor	1.2
Minimum amount of neighboring rectangles	4

Table 6-1 Values which scored higher accuracy for the given parameters

Highest Accuracy scores for the parameters in Table 6-1	
Name	Resulting accuracy (%)
Accuracy for number plate detection in Video file 1	87
Accuracy for number plate detection in Video file 2	48
Accuracy of automatic number plate (OCR) recognition in Video file 1	69
Accuracy of automatic number plate (OCR) recognition in Video file 2	30

Table 6-2 Highest Accuracy scores using the parameter values in Table 6-1

6.1 Automatic number plate detection results

The code is able to detect license plates with 87% accuracy, as seen in Figure 5-13 Chart of the resulting Accuracy for a given psm option with constant scale factor =1.2 and minNeigh=4. Moreover in the aforementioned figure is been shown that PSM modes didn't affect the accuracy of ANP

Areti Pavlides

detection because they are a part of OCR library that is performing calculations after the license plate is detected. The accuracy of the code is highly depended on the haar cascade classifier file, the videos' image quality and image preprocessing used in this project. When the haar cascade classifier file is trained in a large sample of images containing license plates it performs with higher accuracy and can reach accuracies used in industrial applications. The first stage was affected by the aforementioned parameters. It gave the highest results for video file 1 and for video file 2 using the parameter values in the table below. For video file 1 gave 87% accurate plate detections and for video file 2 resulted in 48% accurate plate detections.

Parameter name	Value with the highest accuracy
Scale Factor	1.2
Minimum amount of neighboring rectangles	4

Table 6-3 Parameters that affected stage 1 and the values with the highest accuracy

The Video file 1 had a greater quality than video file 2 and this resulted in higher accuracy score. The parameters scale factor and minimum amount of neighboring rectangles are affecting the ANP detection as seen in from the results in Figure 5-14 and Figure 5-15 because they are preprocessing the image before haar cascade file performs the detection and therefore disturbing the detection process. By fine tuning the scale factor parameter the code was able to scale the image to a certain point that is more easily to trace the license plate in the image. By fine tuning the minimum amount of neighboring rectangles ,the plates detected by the haar cascade file had to reach a certain amount of detections before the code decide if they are indeed license plates.

Arestis Pavlides

The haar cascade file had its limitations which had to do with the certain file trained to detect the plates. These limitations are further expanded in chapter 7.1.

6.2 Automatic number plate (OCR) recognition results

By using *PyTesseract* library we were able to extract vehicle number text from vehicle license plate .The code gave up to 70% accuracy score which, even though high, is not recommended for industrial application. It presented the limitations of an ANPR system and ways to increase system accuracy if some factors such as image quality and transformation are improved.

The values of the parameters in Table 6-1 were adjusted in order to approximate the highest performance of the code.

The image orientation affected the OCR accuracy because the pytesseract's number 7 PSM mode was set to detect a text with horizontal alignment. An image rotation function was used and the value in the function was set to rotate the image in video file 1 by 4 degrees clockwise which made the text in the image appear horizontal .Since OCR was set to detect horizontal text it gave higher accuracy score. For higher rotation values, the OCR didn't perform as well. Therefore a horizontal alignment of the license plate text has to be obtained for increasing system accuracy ,by rotating the image to a certain value using a function, in order for the OCR recognize better the characters.

Arestis Pavlides

The results would have a positive impact on the method’s accuracy if these factors were improved. That is shown after multiple calibration settings done to the code shown in this research as to other codes used in other ANPR projects.

Arestis Pavlides

7. Discussion of results

In this project, the process of license plate recognition was examined through widely known algorithms and open source libraries. The code is able to detect the license plate using the Haar cascade classifier and extract the characters using *PyTesseract* with an accuracy highly depended on the quality of the video file and camera used to take the image. Lighting conditions were constant through the video and it could be more difficult for the code to perform under changing or low lightning conditions. For example as shown in Figure 5-2 the output of the ocr is false due to the different lightning conditions.

Furthermore, Haar cascades tend to be prone to false-positive detections, require parameter tuning when being applied for detection, and are not as accurate as the more industrial algorithms.

An ANPR system is a complex problem due to the number of functions it needs to perform and conditions to meet. The developed system has not achieved 100% overall accuracy in the stages examined. Certain factors like different illumination conditions, text orientation, vehicle shadow and non-uniform size of license plate characters, different font and background color affect the performance of ANPR and OCR recognition system. Some systems work in these restricted conditions only and might not produce good amount of accuracy in adverse conditions. Haar cascades are an important part of the computer vision and image processing literature and are still used with *OpenCV* still useful, particularly when working in resource-constrained devices as we cannot afford to use more computationally expensive object

Arestis Pavlides

detectors. Some of the systems are developed and used for a specific country. It is known that very few of the ANPR are developed in Cyprus and there is an increasing demand to develop such a system for a country like Cyprus. This thesis and the underlying research provides a comprehensive study of recent developments and future trends in ANPR.

7.1 Limitations of first stage and ways to increase system accuracy

Limitations of the ANPR detection stage are summed in the list that follows:

- The code doesn't do well with images affected by artifacts including partial occlusion, distorted perspective, and complex background.
- The accuracy of the program highly depends on the quality of the image taken, lightning of the scene
- The license plate shape poses a problem in its detection,
- number plate shape mistaken for a similar shape like a building window or rectangular road sign.
- The license plate cleanliness obstructs ANP detection.
- The accuracy of the program highly depends on the Haar cascade file and the number of trained images

and ways to increase ANP detection accuracy by:

- the use of a better camera.
- Use of better trained haar cascade file
- Rotation of the image

Arestis Pavlides

- Image denoising
- Good lightning conditions
- Image color manipulation
- Image pre-cropping to lessen computations and false positives.

7.2 Limitations of second stage and ways to increase system accuracy

Tesseract limitations:

- The text orientation poses a problem in its detection and recognition,
- The license plate cleanliness obstructs OCR recognition.
- The OCR tool doesn't do well with images affected by artifacts including partial occlusion, distorted perspective, and complex background
- The developed OCR and its underlying open-source libraries are not as accurate as some commercial solutions available to us.
- the confusion of the OCR for some letters and numbers that look alike, for example number “1” and the letter “I”
- It may find gibberish and report this as OCR output.
- If a document contains languages outside of those given in the -l LANG arguments, results may be poor.

Arestis Pavlides

- It is not always good at analyzing the natural reading order of documents. For example, it may fail to recognize that a document contains two columns, and may try to join text across columns.
- Poor quality scans may produce poor quality OCR.
- It does not expose information about what font family text belongs to.

and ways to increase OCR accuracy by:

- The use of a better camera.
- The use of better OCR tool
- Rotation of the image
- Image denoising
- Good lightning conditions
- Image color manipulation

7.3 How could traffic cameras be accurate with minimum possibilities of fail?

ANPR technology can be as accurate as over 90% of the occasions if some variables of the program are changed (Ganta, 2020). For example, if:

- the cascade model that is trained for finding license plate is exposed to a large number of samples containing plates.
- Better image quality is obtained. This done by better camera and lighting conditions.

Arestis Pavlides

- An OCR that further preprocesses the image and performs with higher accuracy is used.

Arestis Pavlides

8. Further applications

The procedure shown in this research can be repeated using tools that have open access and can be modified to serve different applications. Already traffic cameras can also be used as:

- Congestion charge cameras to detect vehicles inside the chargeable area which have not paid the appropriate fee.
- High-occupancy vehicle lane cameras to identify vehicles violating occupancy requirements.
- Level crossing cameras to identifying vehicles crossing railways at grade.
- Noise pollution cameras that record evidence of heavy vehicles that break noise regulations by using compression release engine brakes
- Parking cameras which issue citations to vehicles which are illegally parked or which were not moved from a street at posted times.
- Toll-booth cameras to identify vehicles proceeding through a toll booth without paying the toll.
- Turn cameras at intersections where specific turns are prohibited on red. This type of camera is mostly used in cities or heavy populated areas.

Arestis Pavlides

- Automatic number-plate recognition systems can be used for multiple purposes, including identifying untaxed and uninsured vehicles, stolen cars and potentially mass surveillance of motorists.

- Bus lane cameras that detect vehicles that should not be in the bus lane. These may be mounted on buses themselves as well as by the roadside

In Figure 8-1 an existing application of an ANPR system is shown where it allows authorized vehicles into controlled areas as well as for tracking the vehicles' movements.

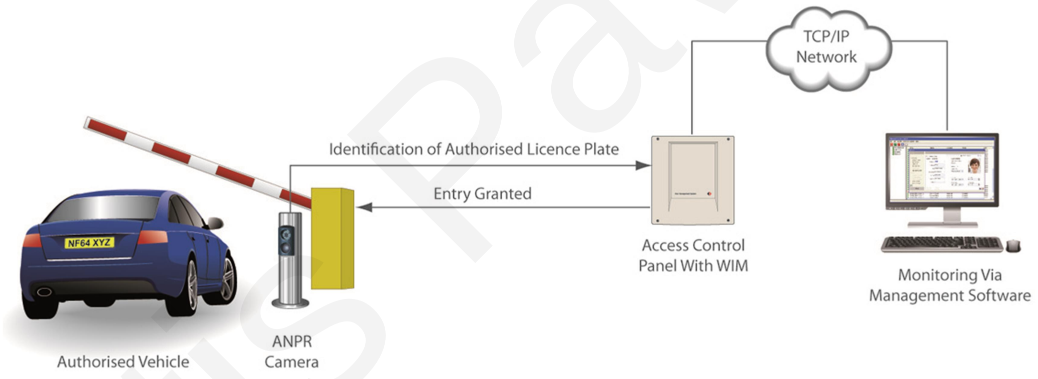

TCP/IP Network
Identification of Authorised Licence Plate
Entry Granted
NF64 XYZ
Access Control Panel With WIM
Monitoring Via Management Software
Authorised Vehicle
ANPR Camera

Figure 8-1 Vehicle license plates been used by ANPR camera system allowing authorized vehicles into controlled areas as well as for tracking the vehicles' movements. (Nortech, 2022)

In this project, we covered how to set up OpenCV and TesseractOCR (in the form of PyTesseract) within Python and how to use their powerful in-built functions to detect car license plates and extract the text from these number plates.

Arestis Pavlides

On top of that, we also discussed several theoretical concepts such as Haar Cascades, multi-scale detection parameters, image processing for optimized recognition, and TesseractOCR’s page segmentation modes (PSM).

This project serves as a stepping stone for larger scale (and more advanced) computer vision projects, such as bulk extraction of car license plate text from large image quantities and applying these concepts on video files or live feed.

Arestis Pavlides

Bibliography

(n.d.).

(1905). "Time Recording Camera for Trapping Motorists". In *Popular Mechanics. Vol. 7, no. 9. Hearst Magazines.* (p. p. 926). Chicago: Popular Mechanics Company.

About OpenCV. (2022). Retrieved 10 15, 2022, from Opencv.org: http://www.opencv.org/about

Astor, J. J. (1894). *From A Journey In Other Worlds.* United States: D. Appleton and Co.

Badr, A. M. (2011). "Automatic number plate recognition system.". *Annals of the University of Craiova-Mathematics and Computer Science Series 38, no. 1* , 62-71.

Chatzivasilis, M. (2022, 05 31). *Philenews*. Retrieved 06 01, 2022, from Άκυρα 11 χιλιάδες πρόστιμα - Δεν προχωρά η επόμενη φάση: https://www.philenews.com/koinonia/eidiseis/article/1478967/akyra-11-chiliades-prostima-den-prochora-i-epomeni-fasi

CV, O. (2022). *Open CV*. Retrieved 08 2022, from Tutorial_py_contours_begin: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

Cynthia Lum, J. H. (2011). License plate reader(LRP) police patrols in crime hot spots: an experimental evaluation in two adjacent jurisdictionss. *Journal of Experimel Criminology, Springer Netherlands*, pp. 312-345.

Dobesova, Z. (2011). Programming language Python for data processing. (pp. 4866-4869). *International Conference on Electrical and Control Engineering*, 4866-4869.

Draghici, S. (1997). A neural network based artificial vision system for licence plate recognition. *International Journal of Neural Systems*, 113-126.

Dubois, P. F. (2007). Guest editor's introduction: Python: batteries included. *Computing in Science & Engineering 9, no. 3*, 7-9.

Edmonton Police. (2011). *Web Archive*. Retrieved 10 20 , 2022, from Edmonton Police Service Intersection Safety Camera Ticket Cancellations: https://web.archive.org/web/20111105094905/http://www.edmontonpolice.ca/TrafficVehicles/IntersectionSafetyCameras/ISCCancellations.aspx

Edureka. (2022, 07). *Edureka*. Retrieved 07 2022, from Top 10 Features of Python You Need to Know: https://www.edureka.co/blog/python-features/

Freund, Y. a. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55, no. 1*, 119-139.

Arestis Pavlides

Ganta, S. a. (2020). A novel method for Indian vehicle registration number plate detection and recognition using image processing techniques. *Procedia Computer Science, 167*, 2623-2633.

Han, B. L. (2020). License plate image generation using generative adversarial networks for end-to-end license plate character recognition from a small set of real images. *Applied Sciences*, 10(8) p.2780.

Harris, C. M. (2020). Array programming with NumPy. *Nature*, 585(7825), pp.357-362.

Howse, J. (2013). OpenCV computer vision with python. *Birmingham: Packt Publishing*, 1-122.

Infratech, T. (2014, February 4). *ANPR is a very useful tool in traffic management,enforcement,tolling and security*. Retrieved 08 02, 2022, from Traffic Infratech magazine: https://www.trafficinfratech.com/anpr-is-avery-useful-tool-in-traffic-management/

Islam, Q. N. (2015). *Mastering PyCharm.* Birmingham - Mumbai: Packt Publishing Ltd.

Janowski, L. K. (2014). Quality assessment for a visual and automatic license. *Multimedia Tools and Applications*, 68(1), pp.23-40.

Leung, K. (2022, 12 28). *Towards Data Science*. Retrieved 08 02, 2022, from HANDS-ON TUTORIALS Russian Car Plate Detection with OpenCV and TesseractOCR: https://towardsdatascience.com/russian-car-plate-detection-with-opencv-and-tesseractocr-dce3d3f9ff5c

Maleehak. (2020, 09 10). *Car-number-plate-recognition-using-OpenCV*. Retrieved 08 02, 2022, from github.com: https://github.com/Maleehak/Car-number-plate-recognition-using-OpenCV/blob/master/detectvideo.py

Marosi, I. (2007). Industrial OCR approaches: architecture, algorithms, and adaptation techniques. . *Document Recognition and Retrieval XIV* , (Vol. 6500, pp. 11-20). SPIE.

Martinsky, O. (2007). Algorithmic and mathematical principles of automatic number plate recognition systems. . *Brno University of technology*, 20-23.

Millman, K. J. (2011). Python for scientists and engineers. *Computing in Science & Engineering*, 13(2), pp.9-12.

Mitra, D. a. (2016). Automatic number plate recognition system: a histogram based approach. *IOSR Journal of Electrical and Electronics Engineering vol .11*, 26-32.

Mufti, N. a. (2021). Automatic number plate Recognition: A detailed survey of relevant algorithms. *Sensors, 21(9)*, p.3028.

(1961). Netherlands: Precision Speed Trap. In *New Scientist Vol. 12 No 265* (p. p. 687). Netherlands: New Scientist.

Arestis Pavlides

Nortech. (2022, 08). *AUTOMATIC NUMBER PLATE RECOGNITION (ANPR)*. Retrieved 08 2022, from Nortech Control: https://www.nortechcontrol.com/solutions/vehicle/automatic-number-plate-recognition-anpr/

Oliphant, T. (2006). A guide to NumPy . In T. Oliphant, *A guide to NumPy* (pp. (Vol. 1, p. 85).). USA: Trelgol Publishing.

OpenCV. (2022, 07). *cv::CascadeClassifier Class Reference Public Member Functions*. Retrieved 07 2022, from Open Source Computer Vision: https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html

Patel, C. &. (2013). Automatic Number Plate Recognition System (ANPR): A Survey. *International Journal of Computer Applications (IJCA)*, 69. 21-33. 10.5120/11871-7665.

Pedregosa, F. G. (2011). Scikit-learn: Machine learning in Python. *The Journal of machine Learning research 12*, 2825-2830.

Python® – the language of today and tomorrow. (n.d.). Retrieved 09 2022, from Python Institute: https://pythoninstitute.org/about-python#:~:text=Python%20was%20created%20by%20Guido,called%20Monty%20Python's%20Flying%20Circus.

Sanner, M. F. (1999). Python: a programming language for software integration and development. *Journal of Molecular Graphics and Modelling (J Mol Graph Model 17), no. 1*, 57-61.

Shakya, S. (2020, 06 12). *Number_Plate_Detection*. Retrieved 08 02, 2022, from github.com: https://github.com/thesachinshakya/Number_Plate_Detection/blob/master/Number_plate_detection.py

Sharifara, A. R. (2014). A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection. *2014 International symposium on biometrics and security*, 5.

Sharma, P. R. (2019). Localisation of license plate and character recognition using Haar cascade. *In 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)IEEE.*, 971-974.

Smith, R. (2007). An overview of the Tesseract OCR engine. *Ninth international conference on document analysis and recognition (ICDAR 2007)*, Vol. 2, pp. 629-633 IEEE.

Srinath, K. (2017). Python–the fastest growing programming language. . *International Research Journal of Engineering and Technology, 4(12)*, pp.354-357.

Viola and Jones. (2001). Rapid object detection using a boosted cascade of simple features. *In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition.*, CVPR 2001 (Vol. 1, pp. I-I). Ieee.

Arestis Pavlides

Wilson, C. W. (2010). *Speed cameras for the prevention of road traffic injuries and deaths. Cochrane database of systematic reviews, (11).* Wiley Publishers.

Arestis Pavlides

APPENDIX A

In this Appendix, the code that was used in this project is shown along with the code that was influenced by. The changes made to perform the ANPR task are given later in this chapter . Most of the codes found are able to detect license plates but cannot recognize text. This task was made possible by combining commands from various files, that perform extraction of characters from an image, through Pytesseract.

The code bellow was used in this project and is responsible for the importation of a video file and plate recognition using a haar cascade file and also the blurring of the plates.

Explanation for every command in the code lines can be seen next to the" # "symbol.

The code used is given below:

```
import cv2 # Imports the library OpenCV version 4.6.0.66
import numpy as np # Imports the library numpy version 1.23.3 and calls it
with the letters "np"
import pytesseract # Imports the library Pytesseract OCR 5.2.2022


pytesseract.pytesseract.tesseract_ cmd =
r'C:\Users\user\AppData\Local\Tesseract-OCR\tesseract.exe' # Location of
the pytesseract file, in the system
```

Arestis Pavlides

```
plateCascade =
cv2.CascadeClassifier("haarcascade_russian_plate_number.xml")#Imports
the Haar Cascade file

minArea = 500 #Sets the value for minimum Area to be detected

cap = cv2.VideoCapture("leof strovol2.mp4") #Imports the Video File
cap.set(3, 640) #Sets the frame width to desired value (only for webcam)
cap.set(4, 480) #Sets the frame Height to desired value (only for webcam)
cap.set(10, 150) #Sets the frame brightness to desired value (only for
webcam)
count = 0

while True:

    success, img = cap.read() # read first frame as an image
   img = img[200:600, 0:1000]  # crops  the frame by desired value
   M = cv2.getRotationMatrix2D([0, 0], 4, 1.0) #rotates image by 4 degrees
clockwise
   img = cv2.warpAffine(img, M, (0, 0)) #Creates new rotated image
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Turn image to
Grayscale
numberPlates = plateCascade.detectMultiScale(imgGray, 1.2, 4) # Detects
the edges of a number plate from Haar Cascade file
```

Arestis Pavlides

```
    for (x, y, w, h) in numberPlates:
        area = w * h #Computes area of number plate
        if area > minArea: #if minimum area detected ,it extracts the Region of
Interest
            imgRoi = img[y:y + h, x:x + w] #Crop the ROI from the frame
            plate = cv2.blur(imgRoi, ksize=(20, 20)) #Blur the part of the image
that has the plates



            cv2.imwrite("images" + str(count) + ".jpg", imgRoi) # Save result image
            cv2.imshow('ROI', imgRoi) # Display ROI result image
            print(pytesseract.image_to_string(imgRoi,
                                              config=f'--psm 7 --oem 3 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
)#Detect and Show the Characters of the plate

            # Create Rectangle Around the Plates Plate
            cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 10)

           # Put text above the Rectangle
            cv2.putText(img, "NumberPlate", (x, y - 5),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)

            img[y: y + h, x:x + w] = plate # Replaces blurred part of image to the
original
```

Arestis Pavlides

```
cv2.imshow("Result", img) # Display result image


cv2.waitKey(10)#Time for displaying the result

count += 1 #Proceed to the next frame
```

Taken from (Maleehak, 2020) the following code recognizes plates , crops and blurs the part of the image that contain the plates.

```
import
cv2
import numpy as np
carPlatesCascade =
cv2.CascadeClassifier('haarcascades/haarcascade_russian_plate_number.xml')
cap = cv2.VideoCapture('carVideo.mp4')
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 80)

if (cap.isOpened()==False):
  print('Error Reading video')
while True:
  ret,frame = cap.read()
  gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
  car_plates = carPlatesCascade.detectMultiScale(gray,scaleFactor=1.2,
  minNeighbors = 5, minSize=(25,25))
  for (x,y,w,h) in car_plates:
    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
    plate = frame[y: y+h, x:x+w]
    plate = cv2.blur(plate,ksize=(20,20))
    # put the blurred plate into the original image
    frame[y: y+h, x:x+w,2] = plate
  if ret == True:
    cv2.imshow('Video',frame)

    if cv2.waitKey(0) & 0xFF == ord('q'):
      Break

  else:
    Break
cap.release()
```

```
        cv2.destroyAllWindows()
```

The code bellow is responsible for the importation of a video stream and plate recognition from a haar cascade file. Taken from (Shakya, 2020)

```
import
cv2
        import numpy as np
        frameWidth = 640    #Frame Width
        franeHeight = 480   # Frame Height
        plateCascade =
        cv2.CascadeClassifier("D:\SACHIN\haarcascade_russian_plate_number.xml")
        minArea = 500
        cap =cv2.VideoCapture(0)
        cap.set(3,frameWidth)
        cap.set(4,franeHeight)
        cap.set(10,150)
        count = 0
        while True:
            success , img  = cap.read()
            imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            numberPlates = plateCascade .detectMultiScale(imgGray, 1.1, 4)
            for (x, y, w, h) in numberPlates:
                area = w*h
                if area > minArea:
                    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                    cv2.putText(img,"NumberPlate",(x,y-
        5),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,255),2)
                    imgRoi = img[y:y+h,x:x+w]
                    cv2.imshow("ROI",imgRoi)
            cv2.imshow("Result",img)
            if cv2.waitKey(1) & 0xFF ==ord('s'):
                cv2.imwrite("D:\SACHIN\cascade\IMAGES"+str(count)+".jpg",imgRoi)
                cv2.rectangle(img,(0,200),(640,300),(0,255,0),cv2.FILLED)
                cv2.putText(img,"Scan
        Saved",(15,265),cv2.FONT_HERSHEY_COMPLEX,2,(0,0,255),2)
                cv2.imshow("Result",img)
                cv2.waitKey(500)
                count+=1
```

The code bellow is responsible for the extraction of text from an image. It performs character recognition using pytesseract and exports the resulting text: taken from (Leung, 2022)

```
# Display the text extracted from the car plate
print(pytesseract.image_to_string(carplate_extract_img_gray_blur, config = f'-
-psm 8 --oem 3 -c
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
)
```

Arestis Pavlides