BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

AUTOMATIC TRAFIC SCENE ANALYSIS USING IMAGE PROCESSING

AUTOMATICKÁ ANALÝZA SCÉNY V DOPRAVĚ PROSTŘEDNICTVÍM ZPRACOVÁNÍ OBRAZU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — LUKÁŠ VÁLEK
AUTOR PRÁCE

SUPERVISOR — prof. Dr. Ing. PAVEL ZEMČÍK
VEDOUCÍ PRÁCE

BRNO 2020

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

20774

Student:	**Válek Lukáš**
Programme:	Information Technology
Title:	**Automatic Trafic Scene Analysis Using Image Processing**
Category:	Image Processing

Assignment:

1. Study the available literature and solutions of automatic traffic scene analysis using image processing; study also libraries of functions suitable for traffic scenes analysis.
2. Select suitable methods for detection of vehicles, their tracking, and analysis of anomalies in their motion (turning around, obstacle avoidance, etc.)
3. Propose an implementation of the selected methods and discuss the features, advantages, and disadvantages of such implementation.
4. Implement the selected methods and demonstrate the results on some feasible example of traffic scene.
5. Discuss the achieved results and a possible future work.

Recommended literature:

- Dle pokynů vedoucího

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Zemčík Pavel, prof. Dr. Ing.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	May 28, 2020
Approval date:	May 26, 2020

Abstract

This thesis deals with the issue of scene analysis using computer vision methods. The aim of this work is to create a system capable of automatically detecting anomalies found in video recordings. The present thesis discusses object-detection and object-tracking systems. It also pursues graphical user interface and violation-detecting algorithm of user-defined rules. As a result of the present thesis, a web application is created that allows users to manage their videos, to define rules for scenes, to start the anomaly detection as well as display the results of the analysis. The system operates in real-time, notifies users when the operation is finished and stores the analysis results for further processing.

Abstrakt

Tato práce se zabývá problematikou analýzy scény pomocí metod počítačového vidění. Cílem této práce je vytvořit systém schopný automaticky detekovat anomálie nacházející se ve video záznamech. Práce se zabývá systémy pro detekci a sledování objektů v obraze, tvorbou grafického uživatelského rozhraní a algoritmem pro detekci porušení uživatelem definovaných pravidel. Výsledkem práce je webová aplikace, která uživateli umožňuje správu videozáznamů, definování pravidel pro scény, zahájení detekce anomálií a zobrazení výsledků analýzy. Systém pracuje v reálném čase, upozorňuje uživatele o dokončení operace a uchovává výsledky analýzy pro další zpracovaní.

Keywords

Image Recognition, Machine Learning, Python, Flask, YOLO, DeepSORT, Web Application, Object detection, Object tracking, Computer Vision

Klíčová slova

Rozpoznávání obrazu, Strojové učení, Python, Flask, YOLO, DeepSORT, Webová aplikace, Detekce objektů, Sledování objektů, Počítačové Vidění

Reference

VÁLEK, Lukáš. *Automatic Trafic Scene Analysis Using*
Image Processing. Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Dr. Ing. Pavel Zemčík

Rozšířený abstrakt

V posledních letech zaznamenaly bezpečnostní kamery obrovský nárůst popularity až do té míry, že se dnes vyskytují téměř všude a produkují miliony hodin videozáznamů. Tato videa vyžadují veliké množství úložiště a času na zpracování. Ve světě existuje mnoho pracovních pozic pro operátory, jejichž hlavní náplní práce je sledovat tyto záznamy a hledat v nich předměty zájmu nebo anomálie. V poslední době vyvíjí mnoho společností systémy, které tento proces automatizují a eliminují tím potřebu lidských operátorů.

Tyto systémy se do značné míry spoléhají na metody počítačového vidění a obvykle vyžadují extrémně vysokou výpočetní sílu k provedení analýzy videa. Cílem této práce je prozkoumat širokou škálu metod počítačového vidění pro sledování a detekci objektů. Práce se také snaží o vytvoření algoritmu schopného detekovat porušení pravidel, která byla definována uživateli. Dále se práce zabývá tvorbou systému, který se podobá systémům velkých společností. Tyto systémy umožňují takovu definici vlastních pravidel, aby bylo jejich využití co nejširší. Některé společnosti nabízejí vlastní hardwarové vybavení, jako jsou vlastní IP kamery a servery, jiné zase nabízí pouze softwarové řešení. Tyto systémy se především využívají pro zajištění bezpečnosti ve městech, budovách a na soukromých pozemcích. Dále se používají pro zpětné vyhledávání objektů ve videozáznamech. Systémy obvykle dokáží upozornit uživatele na objevený problém způsoby, jako je například upozornění e-mailem a SMS, vyskakovací upozornění a nebo kontaktování bezpečnostních sil.

Při návrhu aplikace byl kladen důraz na výběr vhodného systému pro detekování a sledování objektů v obraze, aby systém fungoval dostatečně rychle a přesně. Dále byly navrhnuty algoritmy pro extrahování trajektorií vozidel, pro tvorbu uživatelsky definovaných pravidel a pro detekci porušení těchto pravidel. Systém byl také rozšířen o webovou aplikaci, která uživateli umožňuje graficky pracovat s celým systémem. Uživatelé jsou schopni nahrávat a spravovat vlastní videozáznamy, vytvářet pro ně pravidla a spouštět pro ně analýzu, jejíž výstupy mohou prohlížet.

Systém byl převážné implementovám v jazyce Python. Dále byl použit jazyk JavaScript pro implementaci aplikace, které slouží k vytvoření uživatelkých pravidel. Webová aplikace využívá framework Flask, SQLAlchemy a databázi Postgres. Algoritmus pro extrahování trajektorií z videozáznamů využívá metodu YOLOv3 pro detekci vozidel a metodu DeepSORT pro jejich následné sledování.

Cílem práce bylo vytvořit systém, který automaticky detekuje anomálie ve videozáznamech dopravní scény. Tento cíl byl splněn. Výsledkem práce je program pro analýzu videa, který kombinuje YOLOv3 pro detekci objektů, DeepSort pro sledování objektů, algoritmus pro extrakci trajektorií a algoritmus pro porovnání trajektorií vozidel a uživatelsky definovaných pravidel. Byla také vytvořena webová aplikace, která uživatelům umožňuje definovat pravidla pro scény, nahrávat a spravovat videozáznamy, spouštět pro ně analýzu a také umožňuje výstupy analýzy prohlížet.

Automatic Trafic Scene Analysis Using Image Processing

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. prof. Dr. Ing. Pavel Zemčík. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Lukáš Válek
May 28, 2020

Acknowledgements

I would like to thank my supervisor prof. Dr. Ing. Pavel Zemčík for his kind guidance, patience and valuable advice. I would also like to thank my dear ones for their endless support.

Contents

Chapter 1

Introduction

In recent years, security cameras have seen a huge spike in popularity up to the point that nowadays, they are almost everywhere and produce millions of hours of video footage. These require a lot of storage and time to be processed. In the world, there are many work positions for operators whose main job is to watch hours of real-time or playback video footage and to find objects of interest or anomalies in them. Recently, many companies have been developing systems which automate this process of finding objects of interest in the video footage and eliminate the need of human operators.

These systems rely heavily on computer vision methods and they usually require extremely high computational power to perform the video analysis. This might have been a problem just a few years back but nowadays, it is not due to fast progress in GPU research.

The thesis aims to explore a wide variety of computer vision methods for object tracking and object detection. It also seeks to create an algorithm capable of detecting the violation of user-defined rules. Finally it strives to create a system similar to the ones produced by large companies.

In the thesis, I aimed to offer a solution to the problem mentioned above. I have chosen this task because I am interested in Computer Vision industry and I want to learn new information from this industry. Also, I intend to further pursue Computer Vision in my Master Degree studies.

The thesis is organised as follows. Chapter 2 introduces the field of object detection and tracking and it further summarises popular and historically significant methods in the field. Chapter 3 pursues the existing solutions of automated video analysis. Chapter 4 analyses the existing solutions and it sets the technical goals of the thesis. Chapter 5 proposes the final system architecture. Chapter 6 moves on to explore the implementation details of the system. Section 6.7 deals with the testing of the system. Finally, chapter 7 presents the accomplishments and it also discusses the possibilities of further extending the work.

Chapter 2

Object Detection And Tracking Methods Summary

In this chapter, the state-of-the-art of object detection and tracking is summarised. Various methods for object detection and object tracking are discussed. This chapter mainly focuses on methods which pursue these tasks; however, due to the scope of the thesis it is not possible to mention every available method.

2.1 Object Detection Methods Overview

Object Detection is a Computer Vision task which deals with locating and identifying objects which belong to a certain class. The object location can be interpreted in multiple ways [9]. Two most notable interpretations are:

- Bounding boxes - a rectangle around the found object
- Segmentation - marking every pixel that belongs to the found object

Objects detectors discussed in this section are:

- R-CNN
- Fast R-CNN
- Faster R-CNN
- F-RCN
- YOLOv3
- SSD

These methods are explained in more detail below.

R-CNN
R-CNN is one of the state-of-the-art CNN based Deep Learning methods for object detection. In an ideal scenario, there would be a bounding box for every possible position in the image which would be classified by the R-CNN [25]. This approach would be computationally impossible so a different approach must be used. Instead, the algorithm generates

around 2000 region proposals of various sizes and aspect ratios which are then classified [25].

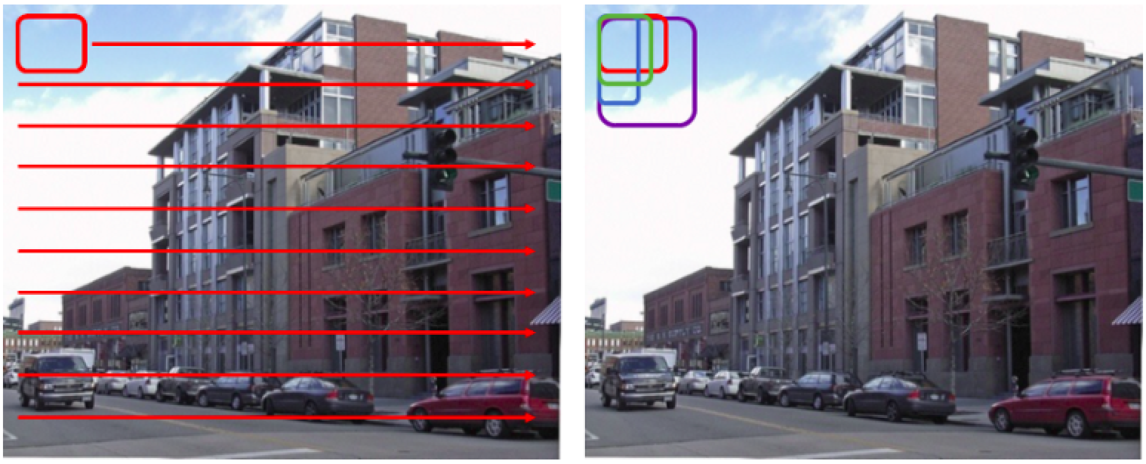


Figure 2.1: Region selection of the R-CNN detector[1]

Figure 2.1 illustrates the sliding window (left) and different aspect ratios (right). Bounding boxes are drawn around the detected objects and the algorithm ends. There are problems with this algorithm. Training of the CNN takes a very long time because each picture has to be detected 2000 times [25].

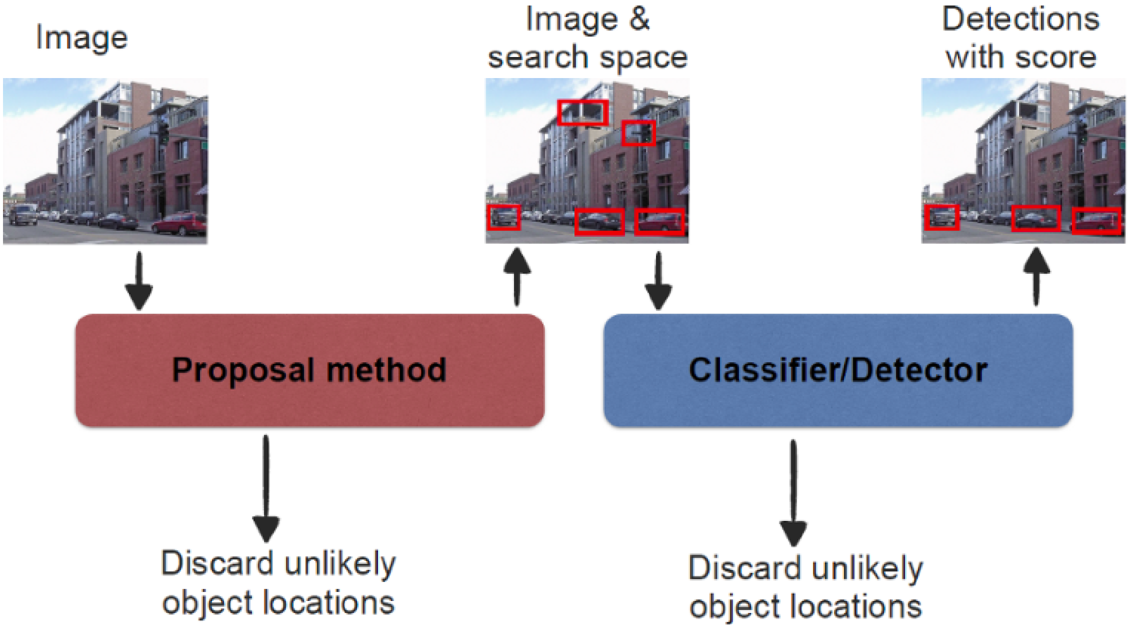


Figure 2.2: R-CNN detector algorithm[2]

[1] Taken from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

The architecture of this method can be seen in figure 2.2. This method cannot be implemented in real-time because each frame, using the current hardware, would take more than 30 seconds to process. The algorithm that detects region proposal is fixed and cannot be trained [25].

Fast R-CNN

Fast R-CNN works similarly to R-CNN but instead of classifying around 2000 proposed regions, the whole image is fed to CNN resulting in a feature map. Proposal region is extracted from the feature map, reshaped to squares and resized to a fixed size using RoI Pooling. RoI feature vectors are used as an input for a softmax layer to predict classes of the proposed regions and their bounding boxes coordinates [25].

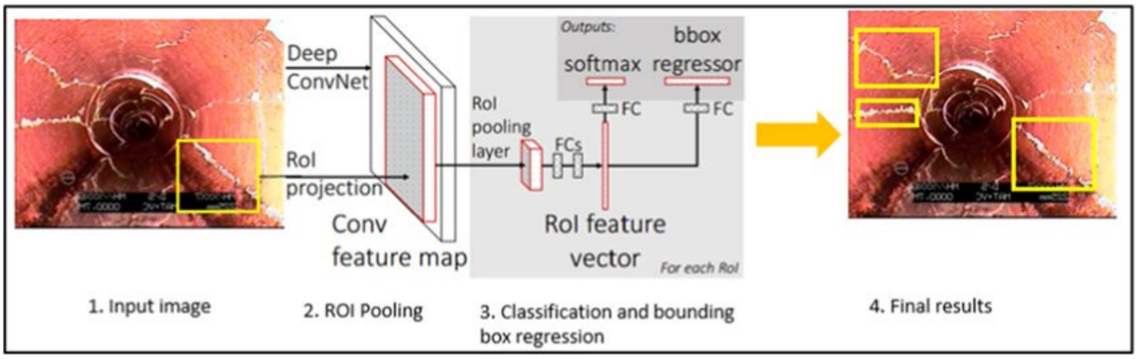


Figure 2.3: Fast R-CNN detector architecture[3]

The architecture of this method can be seen in figure 2.3. This method is way faster than the previous method because there are fewer proposed regions generated and the convolutional operation is done only once per image [25].

Faster R-CNN

Faster R-CNN is in a way similar to Fast R-CNN as it also feeds the whole image to CNN to extract a feature map. Instead of using a selective search algorithm for the proposed regions, it uses a special Region Proposal Network to get the proposed regions. These regions are then resized using RoI Pooling and fed to a classifier to get the object class and predicted bounding boxes for the given regions [25].

[2] Taken from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

[3] Taken from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

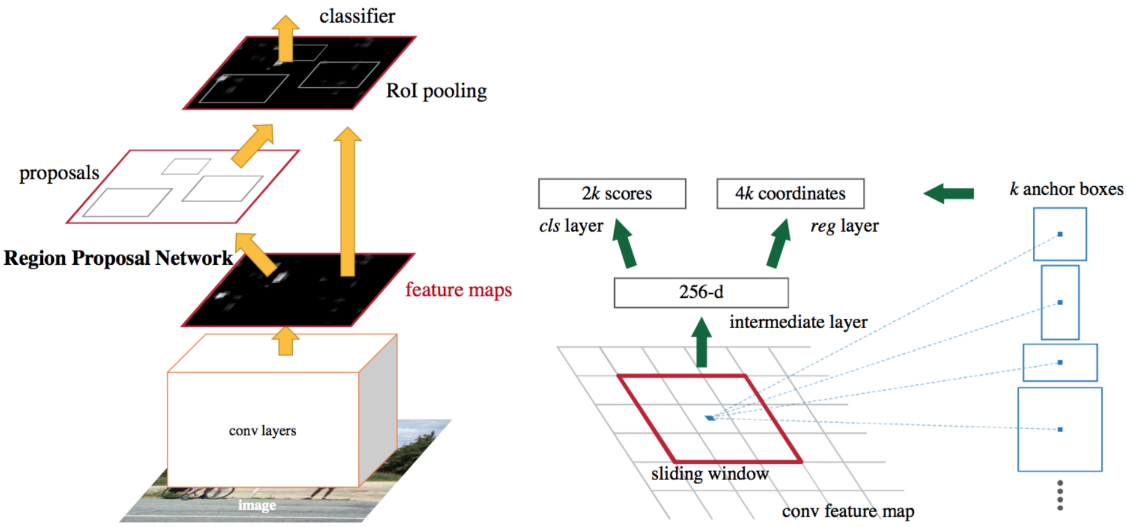


Figure 2.4: Faster R-CNN detector architecture[4]

The architecture of this method can be seen in figure 2.4. This method is very precise but due to its slow performance not suitable for real-time systems [25].

R-FCN

R-FCN just like R-CNN and its faster variants belongs to the two-stage detector category. It also uses RPN (region proposal network) to generate region proposals [25]. But fully connected layers after ROI pooling are removed. Instead, most of the major calculations are done before ROI pooling to generate the score maps. These score maps are used to perform average voting which greatly increases performance [15].

[4] Taken from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

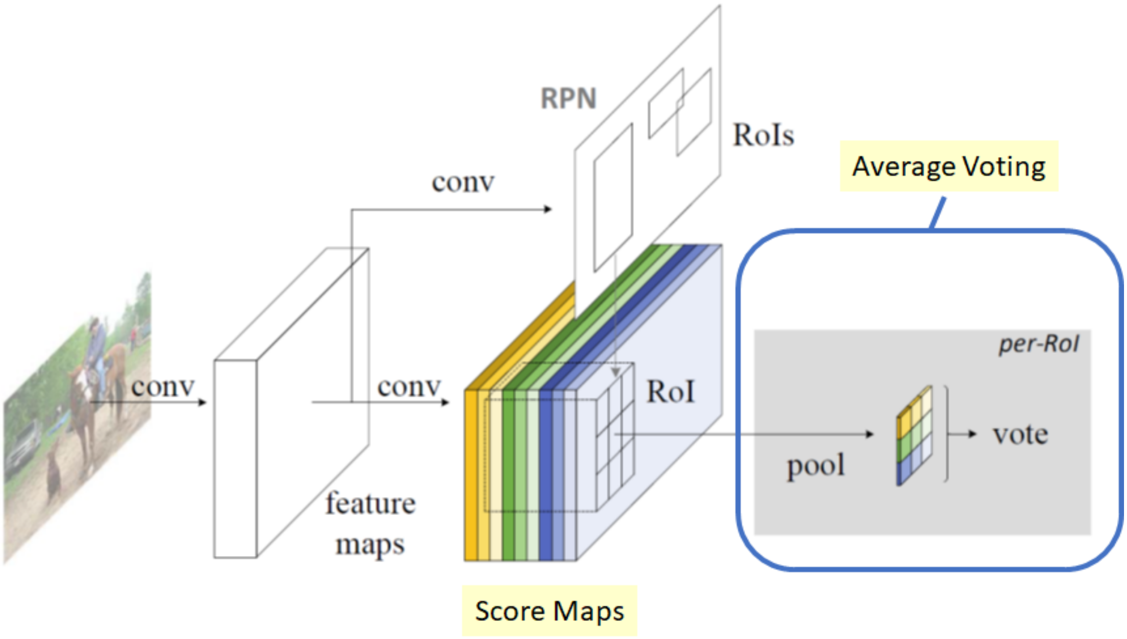


Figure 2.5: R-FCN detector architecture[5]

This results in a faster method than Faster R-CNN with similar mAP [29]. The architecture of this method can be seen in figure 2.5. Still, this method is not suitable for a real-time object detection due to its slow performance compared to SSD and YOLO which are discussed in the following sections.

YOLO

YOLO (You Only Look Once) is an extremely fast single CNN object detector. It predicts multiple bounding boxes and class probabilities for those boxes. YOLO is trained on full images and has optimised object detection performance. It runs at 45FPS (on Nvidia Titan X) while being more accurate than methods such as Fast R-CNN or DPM [25]. Unlike region proposal techniques, YOLO has the whole context of the image available. This means that YOLO makes fewer background errors when compared to region-based methods which only see a part of the object at once [25]. YOLO partitions the whole image into SxS grid. If the centre of an image falls into a grid cell that the grid cell is responsible for the detection of that object. Each grid cell predicts B bounding boxes and confidence score for those boxes. If there is an object in the grid cell the confidence score should be equal to IOU between the predicted box and the ground truth [22]. Each bounding box consists of 5 predicted values: x, y, w, h and confidence score. x and y represent the centre of the bounding box relative to the grid cell. w and h represent the size of the bounding box [22]. Each grid cell also predicts C conditional class probabilities. Only one conditional class probability is predicted per grid cell. At test time the conditional class probability is multiplied by the individual box confidence as such: $Pr(Classi|Object) * PR(Object) * IOU = PR(Classi) * IOU$ which computes a class-specific confidence score for each box.

[5] Taken from https://towardsdatascience.com/review-r-fcn-positive-sensitive-score-maps-object-detection-91cd2389345c

These scores indicate both the probability of the class appearing in the box and how the box fits the object [22].

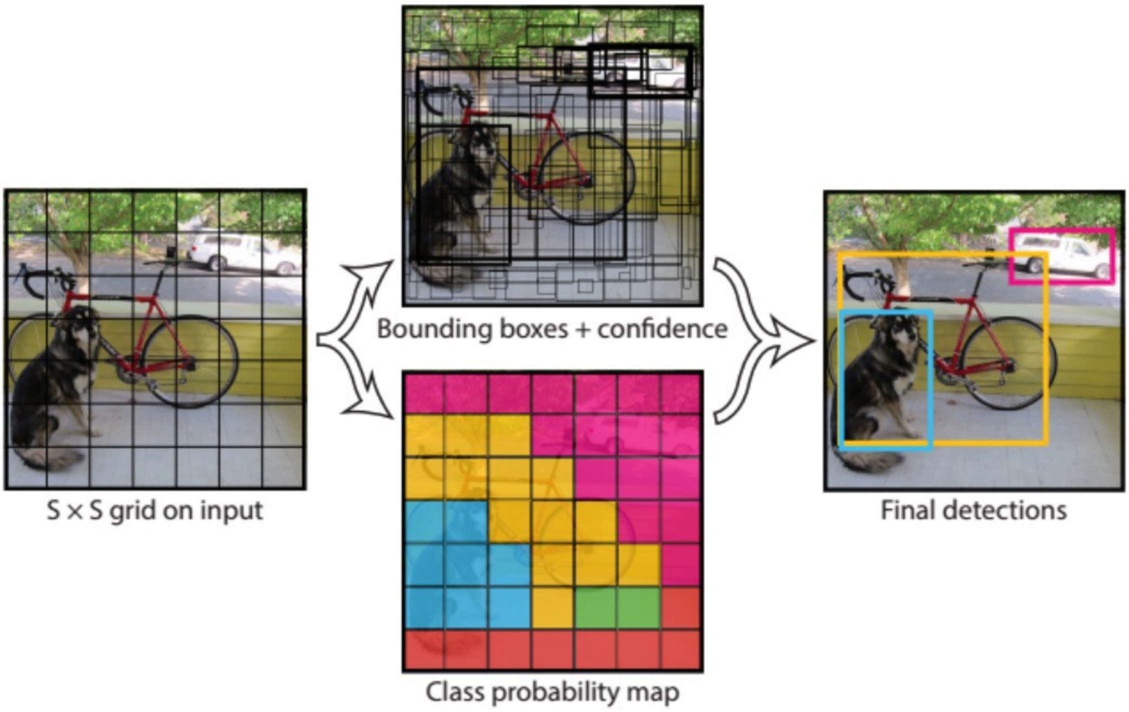


Figure 2.6: YOLO detector algorithm[6]

The algorithm of this method can be seen in figure 2.6. Because each grid cell only predicts two bounding boxes and one class, this method struggles with small objects which appear or objects which appear in groups such as flocks of birds [22].

SSD

Single Shot MultiBox Detector was released in 2016 and reached records in both speed and accuracy. Just like YOLO, both classification and localization happen in a single forward pass of the network. SSD consists of two main parts: the backbone model and SSD head [3]. The backbone model is a regular CNN which the fully connected layers were removed from . It is used as a feature extractor to gather the semantic meaning of the picture while maintaining a spacial structure at a cost of resolution degradation. The SSD head consists of another convolutional layer of different sizes which predict the bounding boxes and classes of objects in the spacial location [3]. This architecture can be seen in figure 2.7.

[6] Taken from https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375

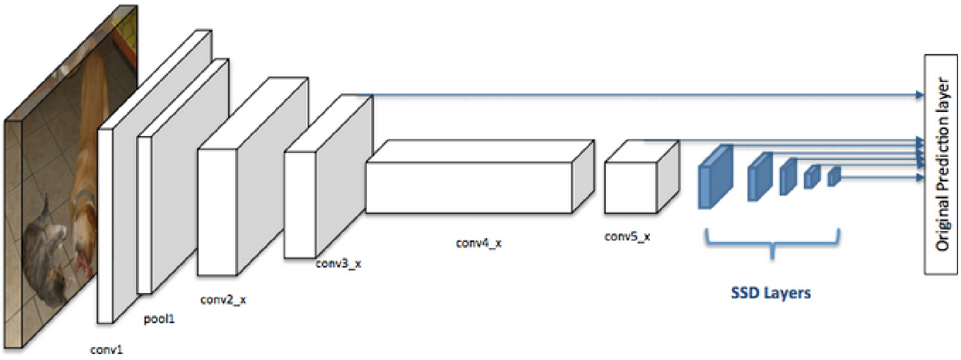


Figure 2.7: SSD detector architecture[7]

These layers separate the image into grid cells which contain default bounding boxes of different sizes and aspect ratios. Each layer separates the image into a different number of cells. These default bounding boxes are determined during the model training [13]. These bounding boxes can be seen in figure 2.8.

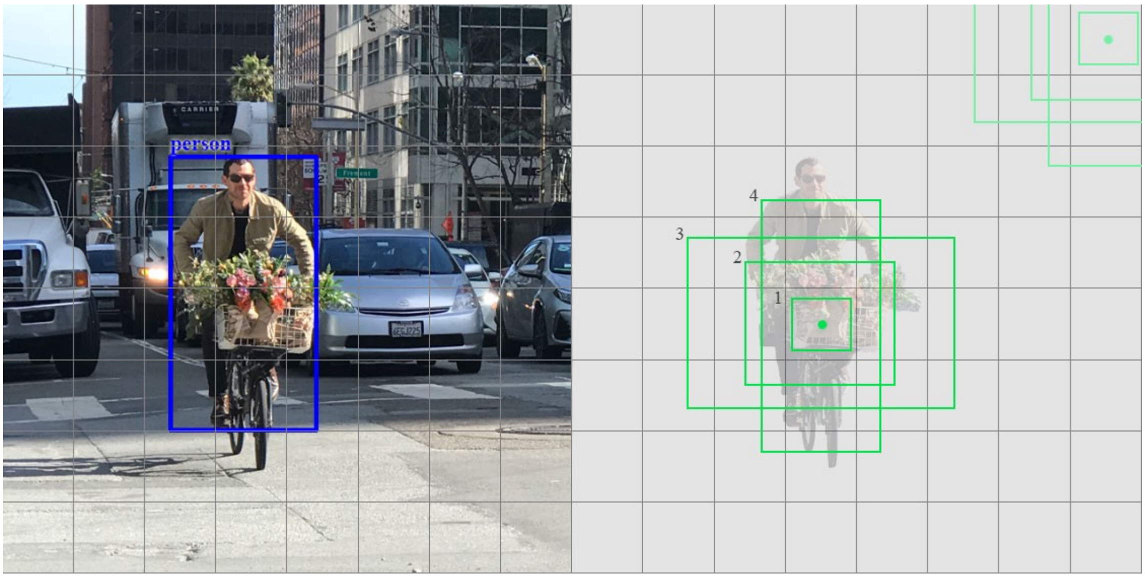


Figure 2.8: SSD detector bounding box selection[8]

Higher-resolution feature maps (layers) are responsible for the detection of small objects while lower resolution ones are used to detect larger objects. SSD predictions are classified as positive or negative matches. If the default bounding box has IoU greater than 0.5 with the ground truth, it is classified as a positive match. Otherwise, it is a negative match [13].

[7] Taken from https://developers.arcgis.com/python/guide/how-ssd-works/

[8] Taken from https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06

SSD is performing worse than Faster R-CNN for small-object detection but is considerably faster (around 8 times). It is around the same level in terms of speed and accuracy as YOLOv3 which makes it very suitable for real-time detection tasks [25, 13].

2.2 Object Tracking Methods Overview

Object tracking is a Computer Vision task which deals with reidentifying objects throughout multiple frames and about figuring out where the object is headed [1]. A tracking algorithm is initialised with a frame of a video and a bounding box to indicate the location of the object. It then attempts to keep track that object in consequent frames and output the bounding box for it. The task is quite difficult due to various obstacles like occlusion, orientation change, scale, and lighting variations [1]. Object tracking is used in applications such as video surveillance, human-computer interaction, robot navigation, activity recognition, anomaly detection, virtual reality, object navigation, and path detection. It is a great way to automate and optimize those processes [1]. Object trackers are divided into multiple categories:

- Detection based and Detection Free:
  - **Detection Based** trackers use a pre-trained object detector to initialise the tracking. This kind of tracker can detect objects which appear in the middle of tracking and immediately recognise when the object disappears from the frame. These methods either normally use the object detector, only use the tracking when the detection fails or they run the object detector every n-th frame to correct the tracking error. The latter variant is used for a long continuous tracking [26].
  - **Detection Free** trackers need a manual initialisation of the object which needs to be tracked. The tracker then localises those objects in the consequent frames. New objects which enter the frame after the initialisation are not tracked [26].
- Single and Multiple Object trackers:
  - **Single Object trackers** only track a single object in the frame even when there are multiple trackable objects in the frame. The object to be tracked is determined at the initialisation [26].
  - **Multi Object trackers** track every object which is found in the frame. If a detection-based tracker is used it also tracks each new object that appears after the initialisation [26].
- Online and Offline trackers:
  - **Online trackers** are used when there is not a possibility to predict the movement of the object from the future frames (eg. live video feed) [26].
  - **Offline trackers** are used for tracking objects in a recorded video. It uses future frames to predict the movement more accurately and minimise errors [26].
- Based on Learning strategy:
  - **Online Learning Trackers** use one or more frames during the initialisation phase to learn about the object. These trackers are more general-purpose because they do not need any pre-training and can learn to track any object on the fly [26].

- **Offline Learning trackers** need to be trained before trying to detect any object because they do not learn anything about the object during the runtime. These trackers can reach higher accuracy because they can be trained on thousands of examples [26].

This task requires detection based online multi-object tracker with offline training. Following text pursues some of the object trackers which are available in the OpenCV A framework.

Tracking methods of the OpenCV framework

BOOSTING tracker:
This classifier is trained at runtime with positive and negative examples of the object. The initial positive example is set by the user or another object classifier. The negative examples are selected in the parts of the image which are outside of the found bounding box [18]. The classifier is run for each frame on every pixel in the proximity of the previously found location. Each pixel is also assigned a value by the classifier. The next location is selected based on the highest score of the pixel. This new location is also used as a positive example for the classifier. This method is very old and does not recognise when the tracking fails [18].

MIL tracker
Multiple instance learning is similar in function to a BOOSTING tracker but instead of feeding the classifier with one positive example and many negative ones, it puts the positive example and other possible positive examples in the proximity to a positive 'bag'. When the current location of the tracked object is not accurate, there is a high chance that the 'shifted' location corresponds with some examples in the positive 'bag'. This results in a better tracking performance than the BOOSTING tracker and also works if the object is partially occluded [18].

KCF tracker
Kernel Correlation Filter further improves on the two previously mentioned methods. This method takes advantage of the fact that a lot of the examples in the positive 'bag' overlap to make the tracking more accurate and faster. This is currently the go-to method in the OpenCV framework [18].

TLD tracker
This tracker separates the term 'tracking' into 3 smaller parts.

- Tracking - The tracker follows the object from frame to frame.
- Detection - Detector locates the object and corrects the tracker if necessary.
- Learning - Learning estimates the detector's error and updates it to prevent errors in the future.

This tracker can deal with a lot of occlusions but due to a high number of false positives, it is almost unusable [18].

MEDIANFLOW tracker
This method tracks the object in both forward and backward direction and measures the discrepancies between these two directions. This approach enables the method to detect tracking failures. MEDIANFLOW tracker works best with small and predictable movements. [18].

GOTURN tracker
Generic Object Tracking Using Regression Networks takes advantages of the CNN (Convolutional Neural Network) trackers while being faster because of the offline training and no online fine-tuning. GOTURN receives the centred previous frame and the current frame (in which the object is most likely not centred) [18].

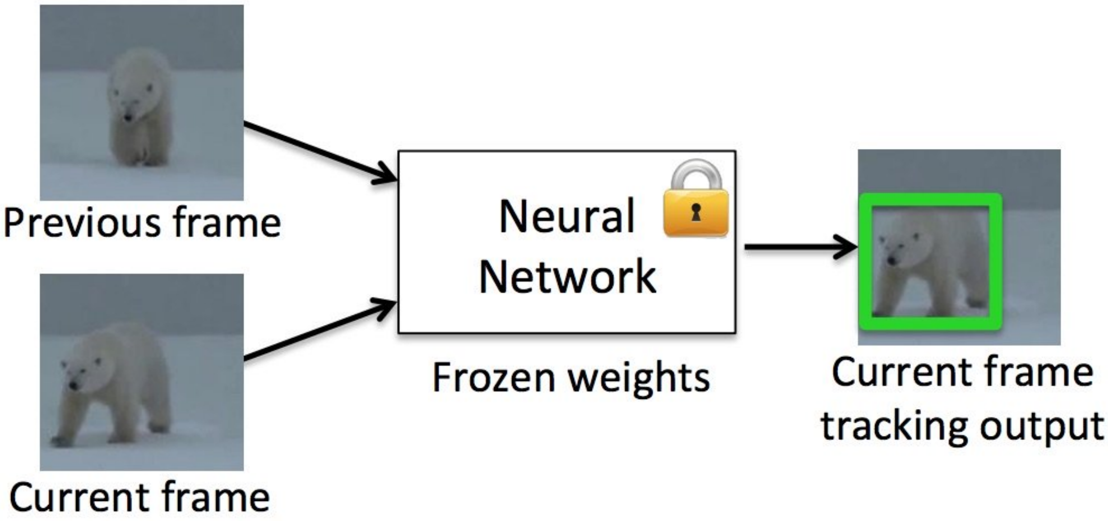


Figure 2.9: GOTURN tracker method[9]

As seen in figure 2.9 the tracker is given the bounding box of the object in the first frame and then tracks it throughout the video [19]. GOTURN doesn't handle occlusion but is very robust to viewpoint and lighting changes and various deformations. The performance of this tracking method is around 165FPS on a single Titan X. Both the previous and current frames are cropped and used as an input to the first 5 layers of the CafeeNet architecture [23]. The architecture of this method can be seen in figure 2.10.

[9] Taken from https://www.learnopencv.com/goturn-deep-learning-based-object-tracking/

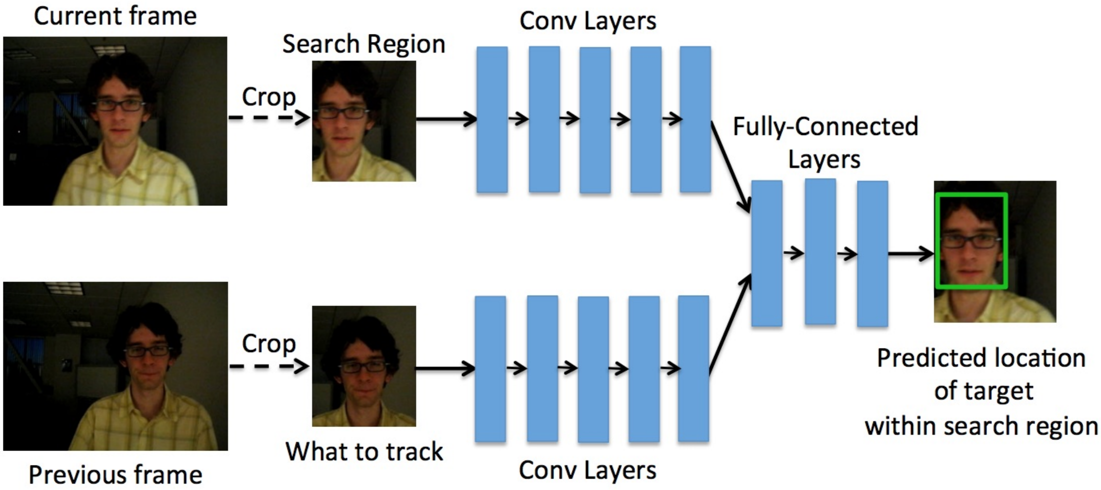


Figure 2.10: GOTURN tracker architecture[10]

The outputs of these convolutional layers are concatenated into a vector of the size of 4096. This vector is then used as an input to 3 Fully-Connected Layers which output 4 coordinates of the bounding box [11].

Other tracking methods

ROLO tracking method

ROLO stands for Recurrent YOLO and is an offline, single object tracker. It uses a CNN called YOLO to extract features of the tracked object and a bounding box of the object. Those features are used as input for the LSTM (Long Short Term Memory) which is an RNN (Recurrent Neural Network) [17].

[10] Taken from https://www.learnopencv.com/goturn-deep-learning-based-object-tracking/

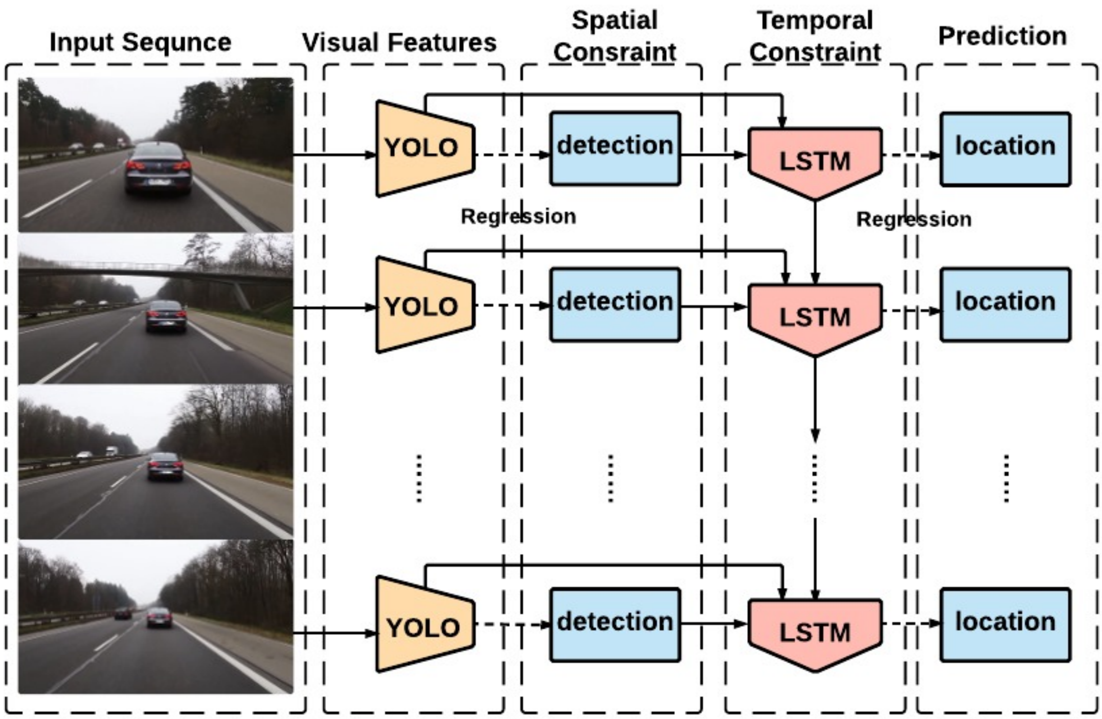


Figure 2.11: ROLO tracker architecture[11]

The architecture of this method can be seen in figure 2.11. The tracker deals well with occlusion and motion blur. As it is a single object tracker, it is not suitable for the goal of this thesis [21].

SORT tracking method

Simple Online and Realtime Tracking method is capable of tracking multiple objects in a video in real-time speed. It relies on an object detector to detect objects in the frame and associates them with previously detected objects based on the coordinates of detection results [20]. This means that this method can track objects of any type. It uses mathematical heuristics such as the IOU (Intersection-Over-Union) metrics between bounding boxes in neighboring frames. The assignment of those boxes is solved by the Hungarian Algorithm [6]. When the object which was present in the frame is not present in the current one, it assumes that the object has left the frame [20]. The result and speed of this tracker are dependent on the quality of the object detector which is used to find bounding boxes and classes of the objects. Due to this tracker not taking the object's visual features into the account, it is not suitable for video analysis in this thesis [6].

Deep SORT tracking method

This tracked, based on the SORT tracking method, is one of the most popular and used methods in the MOT (multiple object tracking) tasks. Deep SORT uses Kalman filtering to estimate the position of the object in the current frame based on the previous detections. The assignment is solved the same way as in the SORT method, using the Hungarian

[11] Taken from https://arxiv.org/pdf/1607.05781v1.pdf

algorithm. It uses Mahalanobis distance to incorporate the uncertainties from the Kalman filter. To further improve the tracking quality, another metric based on the appearance of the object is added [17]. A classifier with the final classification layer stripped is used to extract the features of the object in the bounding box. All bounding boxes of the detected objects in the image are passed to this classifier to get a vector of values for each object which represents the appearance of the object [33]. The distance metric is: $D = \lambda * D_k + (1 - \Lambda) * D_a$ Where λ is a weighting factor, D_k is the Mahalanobis distance and D_a is the cosine distance between the appearance feature vectors. This tracker is a perfect fit for this thesis because it extracts the features of the object which results in the ability to track the object across distant frames [33].

2.3 PyTorch Machine Learning Framework

PyTorch is a Python-based scientific computing package which takes advantage of the GPU as an accelerator for the computation. It was released in January 2016 and has been gaining a lot of attention since then. Nowadays it is one of the most used frameworks, right next to the TensorFlow by Google. On GitHub, it has over 35 000 starts and almost 9 000 forks [24]. PyTorch is also being used by large companies like Facebook, Twitter, NVIDIA, Uber for Natural Language Processing, machine translation, image recognition, neural networks and other AI areas [28]. PyTorch is mainly used to accelerate tensor computations with both CPU and GPU acceleration support and easy deep neural networks building. It contains many routines to accelerate scientific computations such as reductions, math operations, linear algebra, slicing and indexing. Among the things mentioned above, PyTorch is also getting a lot of popularity because of its simple interface, very Pythonic syntax and dynamic computational graphs [28]. PyTorch supports a technique called reverse-mode auto-differentiation which allows users to quickly modify the way the neural network behaves which creates a great environment for experimentation. PyTorch was not built as a Python binding into a monolithic C++ framework but as a part of the Python language. It can be easily combined with other popular scientific frameworks like NumPy A, SciPy A, scikit-learn 2.6 and others. PyTorch is even compatible with Python compilers such as Cython A or Numba A to further improve the already fast performance. PyTorch has various acceleration libraries such as cuDNN and NCLL from NVIDIA or Intel MKL integrated, making it as fast as the current hardware allows it to be [24].

2.4 TensorFlow Machine Learning Framework

TensorFlow is an open-source machine learning framework originally developed by Google Brain team. It first publicly appeared in late 2015 while the first stable release arrived in 2017. It runs on a variety of platforms like Windows, Linux, macOS, cloud services, iOS, Android and many others [10]. Currently, it is the most popular machine learning framework on GitHub with over 76 000 commits, 140 000 stars and 79 000 forks. It provides both Python and C++ APIs, as well as APIs for other languages but the framework logic is written in C++ to ensure the best possible performance while Python is used as a high-level wrapper of the C++ implementation [31]. TensorFlow contains tools for image recognition, word embeddings, recurrent neural networks, natural language processing, partial differential equations, based simulation and others. The biggest benefit to TensorFlow is that it provides an abstraction to the machine learning development. It enables developers to only focus on the application logic without worrying about small implementation details [10]. In October 2019, TensorFlow 2.0 was released further simplifying the use while improving the performance. Support for Keras 2.5 API for model training and experimentation was added. TensorFlow also supports distributed training across multiple GPUs and even multiple machines. Google has created its custom TPU(TensorFlow Processing Unit) accelerator which users can access through Google Cloud. Compared to competitors like PyTorch, TensorFlow has an advantage in large scale projects but lacks in fast prototyping and projects that need to go to production quickly [34].

2.5 Keras Machine Learning Framework

Keras is a high-level neural networks API, allowing its users to focus on fast experimentation. On GitHub, it has over 5000 commits, almost 50 000 stars and almost 20 000 forks [16]. The framework is written in Python and runs on top of other machine-learning frameworks like TensorFlow 2.4, Theano A or CNTK A. Recently, after the release of TensorFlow 2.0, Keras is only developed with TensorFlow in mind to allow tighter integration and uses TensorFlow as a default back-end. Keras supports both convolutional neural networks and recurrent neural networks and its combinations. It also supports GPU acceleration to speed up the training [30]. As mentioned above, Keras uses a backend for all low-level operations like tensor manipulation, convolutions and others. Keras is mostly used with TensorFlow as a back-end. The reason for the creating of Keras is to decrease the barrier of entry for newcomers, to ease model building, to enable fast model experimentation and to increase the production of researchers [12].

2.6 Scikit-learn Machine Learning Framework

Scikit-learn is a free machine learning library for Python which depends on other scientific modules for Python like NumPy, Pandas, iPython, Sympy and Matplotlib. This framework focuses on traditional machine learning algorithms and is a part of a module called SciPy A. As of today, Scikit-learn has over 25 000 commits on GitHub, almost 40 000 stars and almost 19 000 forks which makes it one of the most popular scientific modules for Python [27]. Scikit-learn was developed in 2007 by David Cournapeau as a project in Google Summer of Code. The first public release was made in the first half of 2010 and the stable release was made in May of 2019. The module contains many well-known algorithms for Regression - Logistic and Linear Regression, Clustering - K-Means, K-Means++, Preprocessing - Min to Max Normalization, Classification - K-Nearest Neighbors and others [2]. Scikit-learn focuses on machine learning so it does not support data loading, data handling, data manipulation and data visualisation. For that Scikit-learn has a very neat integration with other Python modules which handle these operations, especially NumPy [2].

Chapter 3

Object Detection and Tracking Solutions Summary

In this chapter, existing solutions for vehicle tracking and trajectory analysis are explored. This chapter mainly focuses on solutions which pursue these tasks; however, due to the scope of the thesis it is not possible to mention every available solution. Both Czech and foreign solutions were explored.

3.1 Certicon Company Offerings

Certicon is a Czech company dealing with innovation and development of software and hardware solutions for automotive, aerospace, telecommunication and healthcare industry. Certicon has a worldwide reach and tight cooperation with universities throughout Europe. The company was created in 1996 as a ČVUT project spin-off. Since then it has received many international rewards and gained the reputation of a cutting edge scientific and technical company of the 21st-century [7]. One of their main products is called CertiConVis(CCV). CCV is a software which analyses and processes camera feed in real-time or pre-recorded camera footage. The system recognises objects and situations in the scene and informs users about set events. It is used to secure public areas, to analyse traffic situations, to gather marketing statistics and to streamline the manufacturing process [7]. Users define objects and their behaviours which they want to track and let the system work independently from then on. An example of such behaviour is a car turning to the opposite direction which the system detects and informs users about the event. The system was, for instance, implemented in a SPEL's parking lot for 90 cars. The company has defined parking places in the video footage and the system detected whether the spot was free or occupied and the employees did not have to drive throughout the parking lot [8].

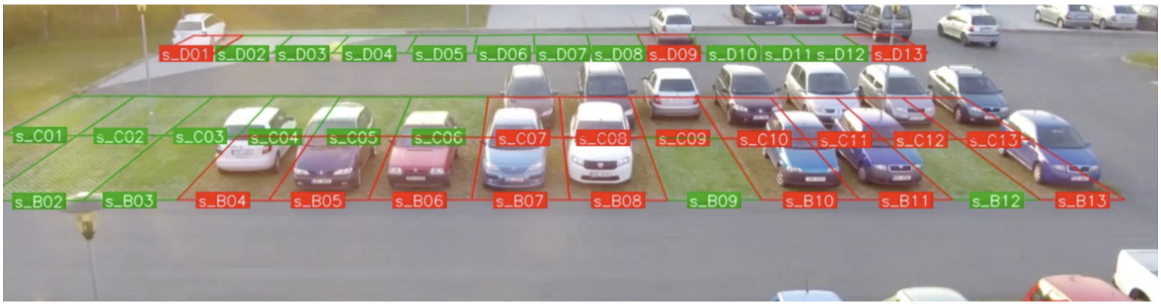


Figure 3.1: Parking assistance software[1]

The picture 3.1 shows a usage of the parking assistance software. Each parking spot has an identifier assigned to it which, if the slot is free, is shown to the employees looking for a parking spot.

3.2 IntelliVision Company Offerings

IntelliVision is a company which specializes in AI and Deep Learning-based video analysis software. The company was founded in 2002 and is headquartered in California with offices in the USA, Asia and Europe. Their software was deployed in over 5 million cameras for over 50 customers worldwide [14]. They provide solutions to smart homes, smart buildings, smart cities, smart retail applications and driver assistance. IntelliVision's AI adds intelligence to video cameras by analysing the video footage and alerting users in real-time. The analysis process can be integrated into cameras, on-premise servers or in the cloud. Their Video Cloud software provides management solutions and allows uploading the event data from a camera to the cloud and accessing it from a mobile device [14].

Smart City solution enables intelligent video motion detection like intrusion detection, pedestrian crossing set line, suspicious objects being left in the scene, crowd detection, loitering, vehicle and people counting, face recognition, license plate recognition, video search, video summary and many others [14].

Smart Transportation is a suite of video analysis products used in over 3000 intersections and many parking lots. It replaces expensive specialised solutions such as traffic light vehicle detection and inductive loops embedded into road. The system supports traffic analysis like congestion monitoring, vehicle speed, length and type, automatic plate recognition, people and vehicle counting, zone detection, zone intrusion and many others [14].

Smart Auto is a camera-based solution for OEMs and integrators. The analysis software turns a simple camera, mounted on a windshield of any vehicle, into a drivers assistance utility. The system is customisable and can also be built into dashcams. As of today, it supports forward collision detection, lane departure warning, pedestrian collision warning, driver monitoring, road sign detection and night vision enhancements while blind-spot detection and smart mirror features are coming [14].

[1] Taken from https://www.certiconvis.cz/pripadove-studie/pripadova-studie-2/

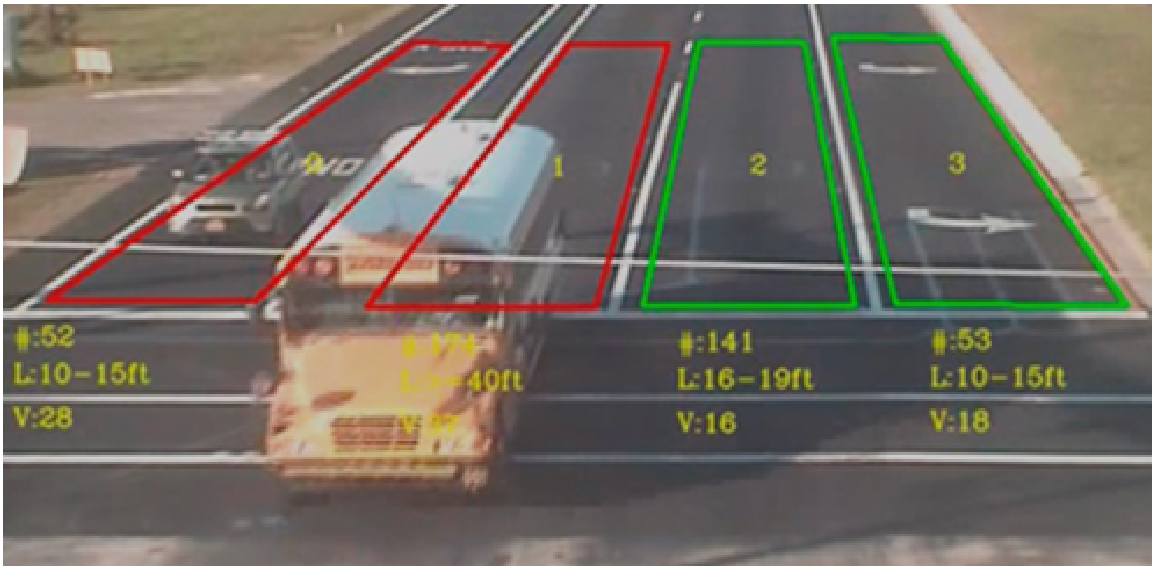


Figure 3.2: Smart Transportation system[2]

The picture 3.2 is an example of the Smart Transport system. Users only need to define traffic lanes and the system can automatically count vehicles and gather information about them like length and velocity [14].

3.3 Avigilon Company Offerings

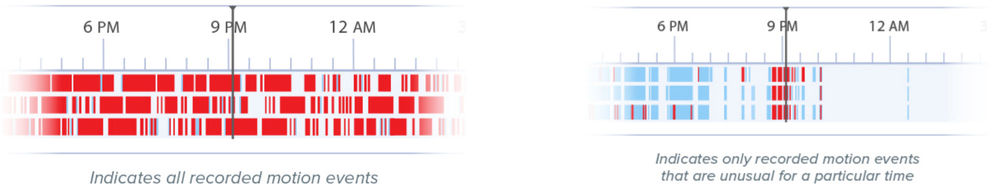
Avigilon is a company with a specialization on both hardware and software security solutions. The company was created in 2004 and its headquarters is located in Vancouver, Canada. Avigilon sales and distribution are based on a business-to-business model with Avigilon usually selling its products to integrators and dealers. Avigilon products include features such as electronic access control, automatic event detection, self-learning capabilities, pattern-based analysis, object search, appearance search and remote viewing via mobile devices [4].

Avigilon hardware products are usually capable of running their video analysis software which makes the analysis decentralised and puts less strain on servers. Avigilon Video Analytics software is capable of classifying and tracking over 50 objects, even when they are stationary. It can not only analyse people but also cars, trucks, buses, motorcycles and bicycles. It allows for a search for people and vehicles in both indoor and crowded environment. Due to its self-learning nature, the system is less prone to analysis errors and is more general [4].

Avigilon Appearance Search is a refined deep-learning search engine for video footage. It allows to easily process large quantities of video files to locate a particular object, be it a person or a vehicle. Operators can search for a person by specifying physical descriptors like the colour of the clothing, gender, age or even facial characteristics. This allows the system to track a person throughout multiple feeds of cameras even from different cameras. This feature helps with an investigation by enabling operators to compile evidence, create a powerful narrative of events, find out an objects route or last-known location [4].

[2] Taken from https://www.intelli-vision.com/smart-city/

Unusual Motion Detection (UMD) is a technology, that allows even more automation in the security department. The system learns what a typical movement in the scene looks like and detects anomalies which are then flagged. This helps to speed up the process of filtering through hours of mundane footage [4].



(a) Timeline without UMD enabled

(b) Timeline with UMD enabled

Figure 3.3: Comparison of a timeline with and without UMD enabled[3]

Figure 3.3 shows that the usage of UMD greatly reduces the number of events that the operators need to filter through.

Avigilon Licence Plate Recognition automatically reads license plate data from vehicles and saves it for later utilization. Operators can then specify a license plate information and the system will automatically find all occurrences of a given vehicle. A watch list can be created to automatically alert when a target vehicle is detected. The system can detect the license plate of most countries [4].

3.4 Agent Vi Company Offerings

Agent Vi is world-wide known company specialising in video analytics solutions. Agent Vi's systems offer real-time video analysis, business intelligence and video search, all of which easily integrates with a variety of 3rd party systems and can be deployed either like a cloud-based System as a Service (SaaS) or on-premise installations [32]. Agent Vi was created in 2003 and is headquartered in Rosh Ha'ayin, Tel Aviv District. Agent Vi's system enables automated video analysis which detects and alerts for the events of interest, searches through the video for a given target and gathers statistical information [32].

Agent Vi's product SavVi is an on-premise video analytics system which provides various functionalities using a single platform that is integrated with existing or new surveillance systems. SavVi features tools like real-time event detection, video search and business intelligence, all of which are demanded by multiple markets. These tools increase the productivity of operators who no longer have to filter through the whole video footage or watch the live video footage [32]. SavVi Real-Time Event Detection eliminates the need to rely on the alertness of the system's operator. The operator just needs to define the events of interest in advance and react to potential alerts in case of their occurrence [32]. SavVi Video Search offers an automatic retrieval and analysis of the recorded video by enabling users to define objects of interest, eliminating the need for tedious manual searches [32]. SavVi Business Intelligence automatically gathers statistics of traffic volumes, movement trends and motion patterns, helping business owners to make educated decisions [32]. SavVi offers two deployment options, optimised and flexible. Optimised offer means that the analysis system is integrated into the IP camera or an encoder device. On the contrary, the flexible

[3] Taken from https://www.avigilon.com/products/ai-video-analytics/umd

offering is used in combination with devices which have not undergone integration with savVi, so the analysis system is implemented on a proxy PC [32].

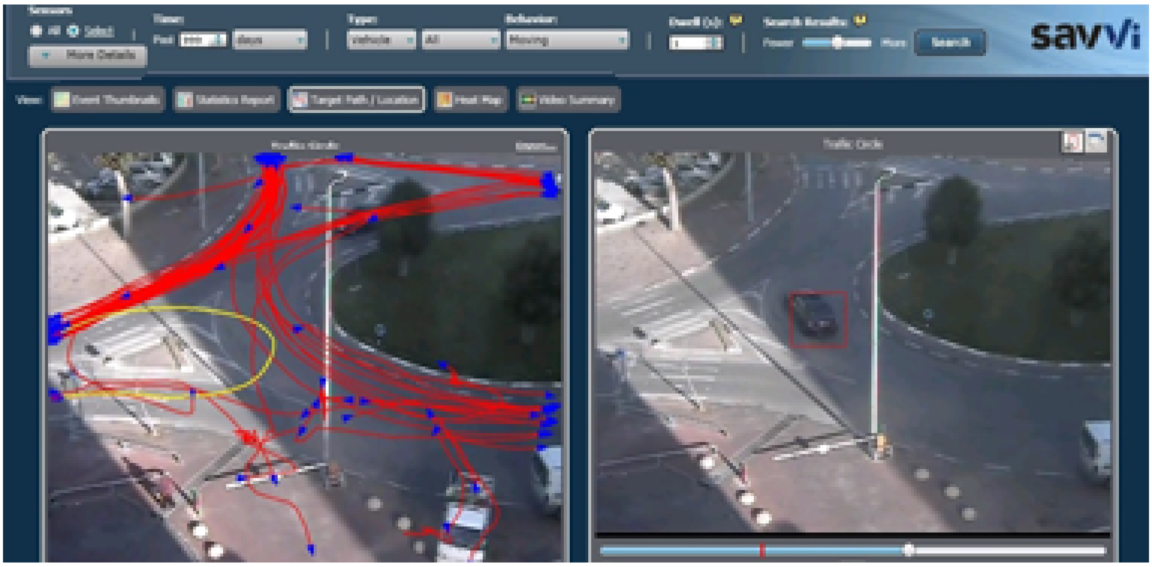


Figure 3.4: Interface of Agent Vi's savVi system[4]

Figure 3.4 shows the interface of Agent Vi's SavVi system analysing video footage of a traffic scene.

Another Agent Vi's product is called InnoVi. InnoVi is a video analysis solution powered by deep learning algorithms which can differentiate between people, cars, motorcycles, bicycles, buses, trucks and static objects. It transforms any camera into a smart video device helping to reveal otherwise hidden information [32]. InnoVi is a Software as a Service solution offering automatic real-time detections of security breaches, anomaly detections and other safety concerns in cities, public spaces and private facilities. InnoVi offers a united web interface accessible from any desktop or mobile computer, scalable service supporting an unlimited number of cameras and high service availability with automatic backups [32].

3.5 AxxonSoft Company Offerings

AxxonSoft is a company focusing on smart integrated security and video surveillance systems. AxxonSoft was founded in 2003 and its headquarters are located in Fremont, California. AxxonSoft's portfolio contains over 150 000 projects with 2.5 million cameras installed. The company cooperates with more than 5800 partners in over 100 countries which perform around 1500 installations per month [5]. AxxonSoft closely cooperates with IP camera manufacturers worldwide to offer integrations for over 10 000 devices. AxxonSoft is a big inventor in Forensic Search A technologies for recorded video and relies on them for their product line. AxxonSoft also develops video analytics based on deep learning. The company's neural networks are trained to perform customer-specific tasks from footage obtained from the customer's facility [5].

[4] Taken from https://www.agentvi.com/products/savvi/savvi-video-search/

Axxon Next is a video surveillance system used for security assurance and prevention of problems. Axxon Next is capable of basic features like a synchronous playback of video footage from several cameras, audio and video analytics, event-driven scenarios such as recording, alarm generation, notification generation, multi-level user rights and many more. Also, the system supports some special features like multiple criteria forensic search, face and license plate search, visual scene synopsis, reviewing hours of recorded video in minutes, merging multiple feeds of cameras into a panoramic one and more [5]. The system is highly scalable since there is no limit to the total number of servers per deployment. There are also no limits to the number of clients simultaneously connected to the system, the number of camera views displayed simultaneously, the number of license plate recognition channels and face recognition channels. The number of simultaneously tracked objects are limited to up to 25. Axxon Next is capable of running on personal computers and servers [5].

Axxon Intellect Enterprise is a security information management system which combines video analysis and event-driven automation in a single environment. The main features of the system include universality, simplicity, scalability, intelligence, cost-effectiveness, modularity and reliability. Just like Axxon Next, Axxon Intellect Enterprise supports a wide variety of smart or basic cameras. The system is implemented in C++ and is from the ground up designed as an object-oriented system. Users do not need to worry about the type of the camera, how it is connected or where it is mounted. They control each component the same way [5]. The system supports many video analytics features. Video detection tools process video and detect events which match defined criteria. These events can envoke actions like starting the recording, sending a notification, displaying the event on a separate screen or starting a custom script. Forensic search allows for quick look up of a video matching set criteria. People counting tools calculate how many people enter and leave an area. Queue management tools allow for the counting of people specifically waiting in line. Heat map tools determine where visitors stop and measure their lingering time in a specific area [5].

Axxon Traffic Control Suite is a small subset of AxxonSoft's product stack, containing only features relevant to monitoring and securing transport areas. Traffic Control Suite can recognise license plates on both stationary and moving vehicles, interacts with traffic lights and other barriers, monitors parking lots and public transportation areas and easily integrates with existing hardware and software [5].

Chapter 4

State-of-the-Art Analysis and Specification of Assignment

In this chapter, the aforementioned state-of-the-art software solutions are analysed. Also, a technical specification is made. This chapter also separates the research and the implementation part of the thesis.

4.1 Existing Software Offerings Analysis

Software offerings analysis:

- Certicon develops comercial products which offer:
  - both real-time and pre-recorded footage processing;
  - user notification about events;
  - user rule definition;
  - public area security, traffic situations analysis, marketing statistics gathering.
- IntelliVision develops comercial products which offer:
  - real-time footage processing;
  - user notification about events;
  - user rule definition;
  - smart homes, smart buildings, smart cities, smart retail and driver assistance solutions;
  - mobile access;
  - dashcamera integration.
- Avigilon develops comercial products which offer:
  - both real-time and pre-recorded footage processing;
  - user notification about events;
  - user rule definition;
  - indoor and outdoor surveillance;

  - both hardware and software solutions;
  - self-learning system;
  - mobile access;
  - person or vehicle search in video footage.

- Agent Vi develops comercial products which offer:
  - both real-time and pre-recorded footage processing;
  - user notification about events;
  - user rule definition;
  - city, public spaces and private facilites security;
  - cloud and local installations;
  - object search in video footage;
  - both hardware and software solutions.

- AxxonSoft develops comercial products which offer:
  - pre-recorded footage processing;
  - user notification about events;
  - user rule definition;
  - multiple criteria forensic search;
  - vehicle and traffic surveillance.

All of the companies mentioned in chapter 3 have developed object tracking systems to make cities smarter and more secure, to analyse vehicle and customer movement patterns and to detect suspicious behaviours. They allow a definition of custom rules to match every possible use case, making their systems highly flexible. Some of the companies offer (or need) custom hardware equipment like custom IP cameras and servers. The systems offer both online and offline processing, each with its use cases. Online processing is more suitable for real-time smarter cities, surveillance systems, driving assistants whereas offline processing is more suitable for the gathering of marketing statistics, the creation of heat maps and traffic analysis. The systems notify users about a discovered problem in various, predefined or custom ways like email and SMS notifications, pop-up alerts and contacting security forces.

4.2 Objectives and requirements for the resulting solution

Based on the analysis made in the previous section 4.1 I have proposed a system which automatically detects anomalies in a video feed of a traffic scene. The system will be similar to the ones described in the state-of-the-art solutions chapter 3 but will be freely available for the public. Also, I wanted to reveal which methods are suitable for such tasks.

The system should offer a software solution to a computer vision task of object detection and object tracking. The system should, based on the user-defined rules, detect violations of these rules and make them easily accessible for end-users. The goal of the system is to automate and quicken the process of traffic scene recordings analysis with better accuracy than a human operator could achieve.

4.3 Technical Specification of the Resulting System

Based on the analysis of state-of-the-art solutions we have agreed on the following specification:

- The implementation language of the system will be Python due to its wide variety of machine learning frameworks and popularity
- The object detector method used will be able to work in real-time
- The object tracker method used will be able to work in real-time
- The system will support a graphical creation of custom scene rules
- The scene rules will only need to be created once
- The system will automatically detect violations of the aforementioned rules
- The violations will be easily accessible for end-users
- Users will be notified when the analysis is done

The system needs to be tested to verify that the desired functionality is met. Also, verifying that the system is stable and runs fast enough is necessary. The following tests must be performed:

- performance test - the system needs to analyse input video in a shorter time span than the length of the video itself
- stability test - the system needs to work without any crashes for at least a month
- functionality test - the system has to precisely detect objects and find anomalies in the video footage
- usability test - the system has to be usable by average users

In upcoming chapter 5, individual components of the system are explained in detail.

Chapter 5

System And Components Outline

The proposed system is composed of 4 main parts (blue in figure 5.1):

- object detector which will detect vehicles in the scene - Vehicle Detection block in Figure 5.1
- object tracker which will track vehicles in the scene - Vehicle Tracking block in Figure 5.1
- rule violation detection algorithm which detects rule violations in the video footage - Anomaly Detection block in Figure 5.1
- graphical user interface which enables users to quickly create scene rules, start the analysis and view the analysis result - GUI block in Figure 5.1

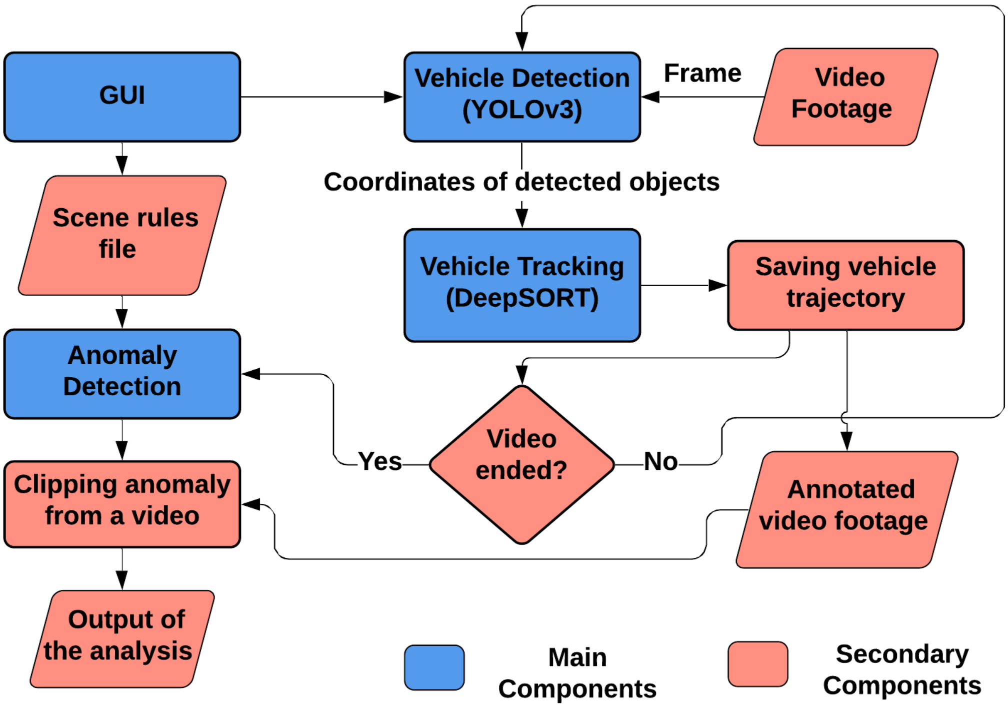
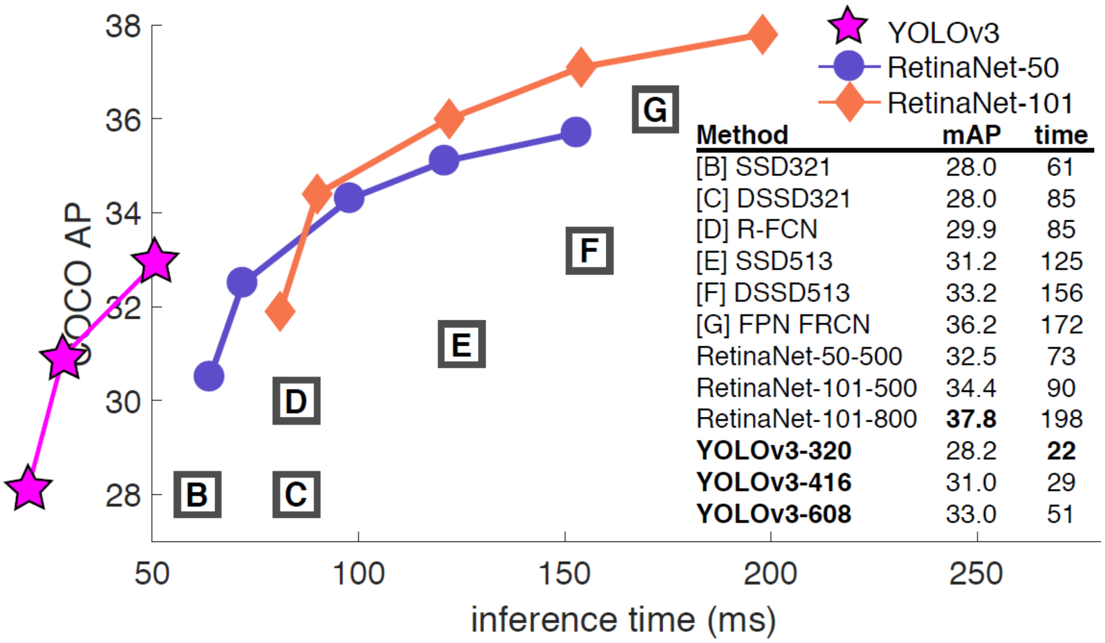


Figure 5.1: Outline of the system

The whole system will be controlled within the GUI without the need to tinker with command-line interface. Users will be able to upload own video footage which will be reformatted into internal codec and resolution, create scene rules for the said footage, manage uploaded videos and rules, start video analysis and observe the result. Data will be persisted on the local filesystem of the machine that runs the system and information about the files, users and settings will be stored in a relational database. After starting the analysis a new Python process will start in the background to process the video and find rule violations. When the analysis is done, users will be notified via e-mail used in the registration form. User can then inspect and manage found anomalies.

5.1 Object Detection Method Selection

This section discusses 'Vehicle Detection' block in Figure 5.1. As mentioned in section 2.1 there are many object detection methods available each with its advantages and drawbacks. From research, the best method currently seems to be YOLOv3 due to its combination of high performance and accuracy.



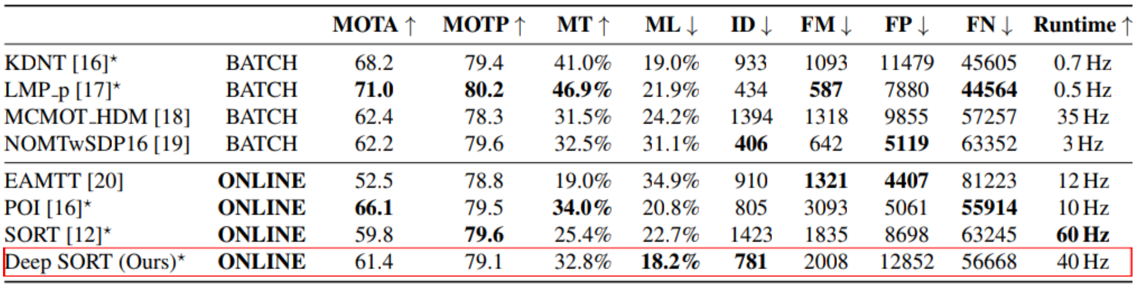
Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 5.2: Object detection methods comparison

Figure 5.2 shows that *YOLOv3-416* strikes the perfect balance between mAP (mean average precision) and time to detect an image. While *RetinaNet* based detectors and *FPN FRCN* detector have noticeably higher mAP values they are too slow for the goal of this thesis. *YOLOv3* is implemented in multiple frameworks but the variant made in PyTorch 2.3 under the MIT A license will be used for this thesis because it also supports *DeepSort* tracking method. For further information see section 5.2.

5.2 Object Tracking Method Selection

This section discusses 'Vehicle Tracking' block in Figure 5.1. There are many object tracking methods available each suitable for different tasks. This assignment requires a multi-object tracker to be able to reliably track all vehicles in the scene. From research made in section 2.2, the best currently available methods seems to be DeepSort which combines the Sort algorithm with a neural network to take visual features of the object into the account.



		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	**71.0**	**80.2**	**46.9%**	21.9%	434	**587**	7880	**44564**	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	**406**	642	**5119**	63352	3 Hz
EAMTT [20]	**ONLINE**	52.5	78.8	19.0%	34.9%	910	**1321**	**4407**	81223	12 Hz
POI [16]*	**ONLINE**	**66.1**	79.5	**34.0%**	20.8%	805	3093	5061	**55914**	10 Hz
SORT [12]*	**ONLINE**	59.8	**79.6**	25.4%	22.7%	1423	1835	8698	63245	**60 Hz**
Deep SORT (Ours)*	**ONLINE**	61.4	79.1	32.8%	**18.2%**	**781**	2008	12852	56668	40 Hz

Figure 5.3: Object tracking methods comparison

Figure 5.3 shows that even though the DeepSort tracker is one of the fastest methods tested and it also has one of the best tracking accuracies. While being a little slower than the SORT algorithm it has an additional feature in a form of visual features extraction which can be used to reidentify vehicles which have been lost for a large number of frames.

5.3 Anomaly Detection Algorithm Outline

This section discusses 'Anomaly Detection' block in Figure 5.1. The anomaly detector needs to be able to recognise if any of the vehicles in the video footage have violated any of the user-defined rules. It will compare the vehicle trajectory obtained by the trajectory analyser 5.6 and user-defined rules created in 5.7.

As mentioned above, the input for the anomaly detection will be a file containing user-defined rules and a file consisting of trajectories of vehicles found in the video footage. The system will then iterate over vehicle coordinates and check if it corresponds to any of the user-defined rules. If any of the vehicle coordinates are inside the 'Entry Forbidden' type rule, the vehicle is marked as a rule violator. This also applies to the rule type 'Direction Allowed' where a trajectory of the vehicle will be calculated and if it will not match the set direction of the rule (with deviation) the video will also be marked.

After determining which vehicles violated the user-defined rules, a small clip of the incidents will be acquired and saved in the database.

This algorithm will run in parallel and each process will be responsible for trajectories of a single vehicle. This will yield better resource utilization and faster detection.

5.4 Graphical User Interface Outline

This section discusses 'GUI' block in Figure 5.1. A graphical user interface needs to offer a simplistic way of video manipulation and output observing. It will be implemented in Flask A on the server-side and Bootstrap 4 A for the interface design. SQLAlchemy A will be used to translate class models into a relational database. Graphical User interface will be divided into multiple segments: Manage Users, Manage Videos, Manage Video, Manage Rules, Manage Rule.

Manage Users

Admins will in terms of user management be able to change permissions of other users, create new accounts, and change their profile settings. Users will have the option to change their passwords, e-mails, names and notification settings.

Manage Videos

Users will be able to upload new videos and find the already uploaded ones. By clicking on the existing videos, users will be redirected to a more detailed page about the particular video footage. Users will only see videos which were uploaded by them and will not be able to in any way manipulate videos they do not 'own'.

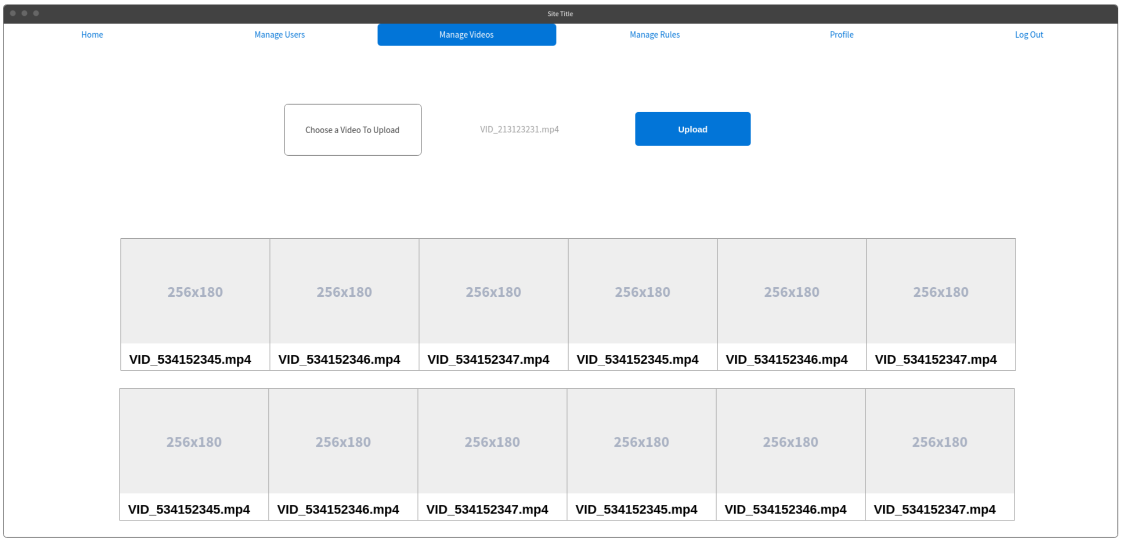
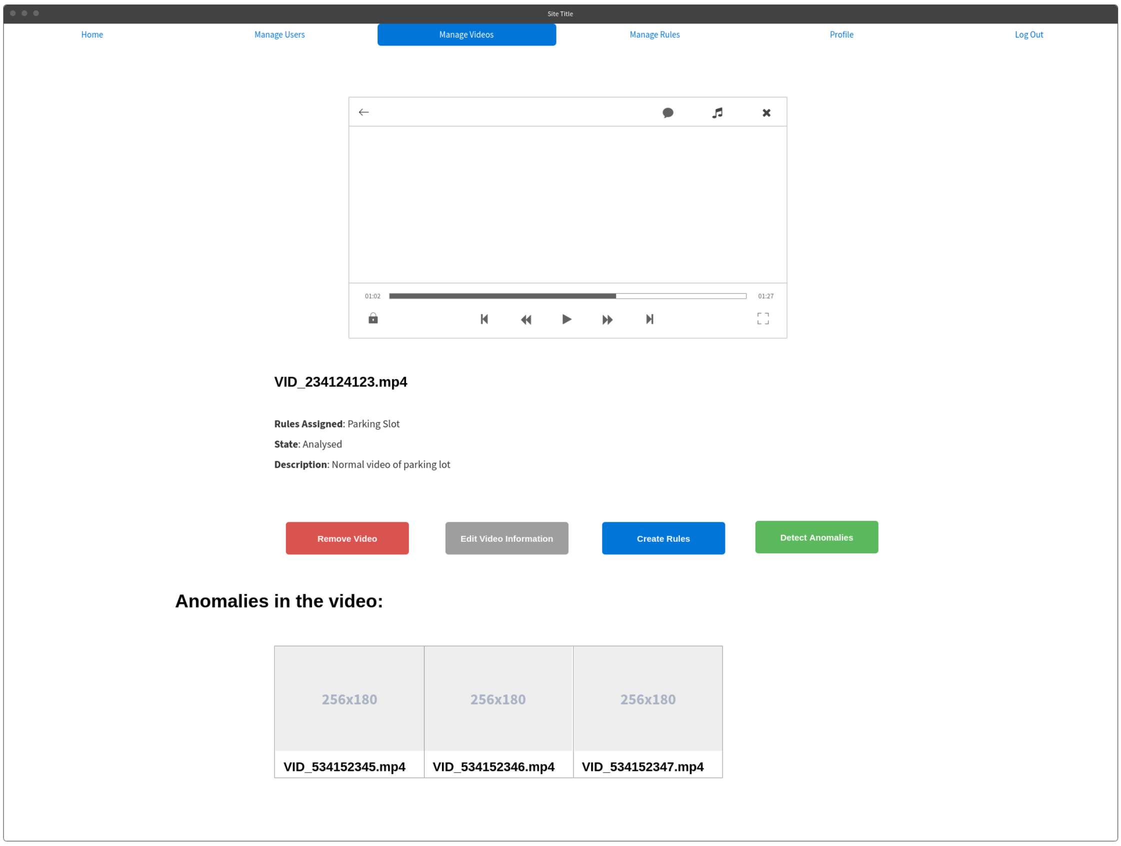


Figure 5.4: Outline of the web page for video upload

Figure 5.4 shows the outline of the web design. On the top of the page, users can click the white button to open a file explorer and select videos to upload or drag-and-drop them into the area. Below the upload section, all videos uploaded by signed users are displayed. Clicking those videos redirects users to a more detailed view with options to view and edit them. This page is discussed in paragraph *Manage Video* 5.4.

Manage Video

This page will allow users to see the detailed view of a particular video, edit its information and start the rule violation detection for the particular video.



VID_234124123.mp4

Rules Assigned: Parking Slot
State: Analysed
Description: Normal video of parking lot

Anomalies in the video:

VID_534152345.mp4
VID_534152346.mp4
VID_534152347.mp4

Figure 5.5: Outline of the web page for video manipulation

Figure 5.5 shows the design outline of the said page. At the top of the page, a video player will be located. Users will be able to view the whole video, fast forward it and maximise the video player window. Below the video player, there will be a column with information about the video like its name, description, rule it is assigned to and its state. Below the information, there will be buttons which can remove the video from the system, allow for modification of the information, redirect to the rule creator page and start the rule violation analysis. At the bottom of the page, there will be a list of all anomalies found in the particular video. They will only appear after the rule violation analysis is complete and will be removed if the rules for the video change.

Manage Rules

This page will be responsible for displaying existing user-defined rules, redirecting users to a particular rule and redirecting users to the rule creator 5.7.

More detailed information about the outline is availabe in appendix B.

Manage Rule

This page will be graphically similar to the *Manage Video* 5.4 one but instead of managing a particular video, it will control a particular scene rule. It will also enable users to edit the rule information and start the rule violation detection for every video assigned to it.

More detailed information about the page is availabe in appendix B.

5.5 Programming Language And Frameworks Selection

There are many programming languages suitable for machine learning nowadays but some of them have the edge whether for the variety of available frameworks, the execution speed or the comfort of use. The three main languages which meet some or all of these criteria are Python, C++ and C#. Python is extremely easy and safe to use due to its high level of abstraction and lack of pointers. Also, most of the main machine learning and computer vision frameworks like TensorFlow 2.4, PyTorch 2.3, Keras 2.5, Scikit-Learn 2.6 and OpenCV A are available for Python and are implemented as a C or a C++ module. As many of these popular frameworks are natively implemented in C or C++ and Python implementation acts as a wrapper for the code, there is no need to use C++ directly since the feature set and execution speed are equal while Python is more pleasant to use.

As mentioned above there are many scientific frameworks available for Python. The choice of the machine learning framework depends mostly on the available implementation of the most suitable object detectors and trackers since this thesis does not deal with the implementation of these methods and is discussed in section 5.1 and 5.2.

OpenCV is a framework of choice for video manipulation and visualisation of vehicle detection and vehicle tracking results.

Flask is a framework of choice when it comes to small-sized web sites as it provides plenty of tools which simplify web development.

Postgres A and SQLAlchemy A will be used for the database layer as it allows for seamless conversion of Python classes to SQL tables and vice versa.

5.6 Trajectory Analysis Method Outline

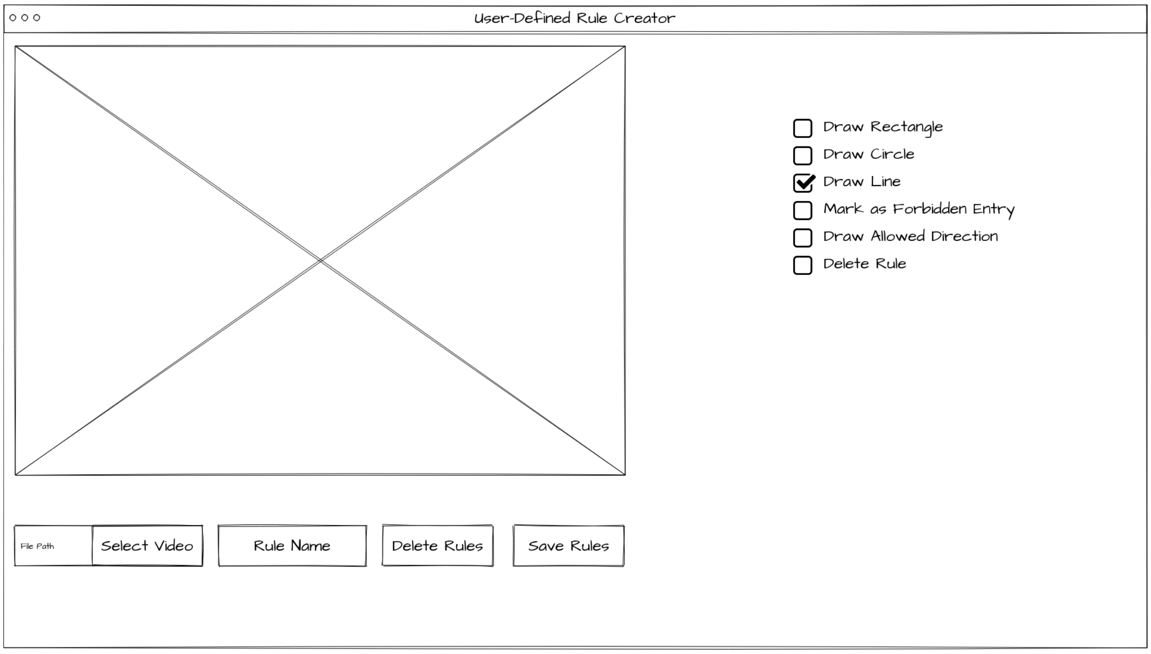
The trajectory of each vehicle in the scene will be persisted to perform the rule violation detection, which is done after the trajectory analysis, as mentioned in chapter 5.

The trajectory of each detected vehicle will be internally saved in a Python dictionary data structure periodically (each frame, every other frame, etc.). The position of a vehicle and identifiers of a vehicle will be obtained by the tracker which will return two x coordinates and two y coordinates indicating the position of the vehicle in the frame. The position used from this rectangle will be its centre coordinate or a different coordinate if testing reveals a more suitable one. The first and last appearance of each vehicle will also be stored for the creation of clips of vehicles that cause a rule violation. After the analysis of the whole video, the internal structure will be converted into a file on the servers filesystem and the path will be saved into the database.

5.7 User-Defined Rule Creator Outline

The user-defined rule creator will be a web application using Javascript and PaperJS A to draw on the top of the canvas, dynamically process user input and dynamically display feedback and changes to the user.

The outline of the user interface can be seen in figure 5.6. The left top part of the screen will be occupied by the canvas displaying a frame of an uploaded video, right part will be used for the selection of the drawing mode and bottom part below the canvas will host buttons for video upload, rule naming and rule manipulation.



□ Draw Rectangle
□ Draw Circle
☑ Draw Line
□ Mark as Forbidden Entry
□ Draw Allowed Direction
□ Delete Rule

Figure 5.6: Outline of the rule creator

Drawing will be done by clicking and dragging on the canvas with a drawing mode selected. Each shape will be represented in the PaperJS A object and will have a rule (Direction allowed, Forbidden access) assigned to it. Clicking the *Save Rules* button will convert the PaperJS objects into JSON A file and it will send it to the main web application.

Chapter 6

Proposed System Implementation

In this chapter, the implementation of the proposed system is described. It consists of sections about an object tracker and object detector implementation 6.1, trajectory analyser implementation 6.2, rule creator implementation 6.3, graphical user interface implementation 6.5, system overview 6.6 and system testing 6.7.

6.1 Object Tracker and Object Detector Implementation

Object Detector Implementation
As a foundation of the system, an already existing solution of an object detector was used. The solution implements YoloV3 object detector in Python's framework Pytorch 2.3 and wraps it in a class called /YOLOv3|. It supports CUDA acceleration which results in much faster inference time than using a CPU. The detector takes an image in NumPy array form and returns bounding boxes and object class identifiers. The detector is trained on a COCO dataset and can distinguish between eighty common objects. The existing solution has a public GitHub repository [35].

The object detector is used in the class `VideoTracker` located in the file `VideoTracker.py`. It is initialised in the constructor `__init__` of the class with configuration and uses CUDA, if available in the system. Configuration and pre-trained weights for the detector are stored in the files *detector/YOLOv3/cfg/** and *detector/YOLOv3/weights/**. Which configuration files and pre-trained weights are used is determined by the configuration file *configs/yolov3.yaml* which needs to be supplied to the system at the start-up of each analysis. When the detector is initialised, an input video is loaded by the OpenCV framework. The *run* method starts the video analysis and cycles through each x frame in the video, where x is passed as a parameter when running the video analysis. Each cycle, a frame of the video is retrieved and converted into a NumPy array:

```
_, ori_im = self.vdo.retrieve()
im = cv2.cvtColor(ori_im, cv2.COLOR_BGR2RGB)
```

Then the image array is passed as an argument to YOLOv3 to do the detection. The detector returns found bounding boxes, class configuration and class identifiers of the found classes:

```
bbox_xywh, cls_conf, cls_ids = self.detector(im)
```

If there were any objects found in the frame, they are filtered to keep only the objects of interest which include bicycles, cars, motorbikes, buses and trucks. This filtered array of found objects of interest is then passed to the object tracker, which is discussed in the paragraph below.

Object Tracker Implementation

As a foundation of the system, an already existing solution of an object tracker was used. The solution implements DeepSORT object tracker in Python's framework Pytorch 2.3 and wraps it in a class called `DeepSort`. It supports CUDA acceleration which results in much faster inference time. The tracker takes found bounding boxes as well as class probabilities and the original image as an input and returns identified bounding boxes.

The object detector is used in the class `VideoTracker` located in the file `VideoTracker.py`. It is initialised in the constructor `__init__` of the class with configuration and uses CUDA, if available in the system. Configuration is stored in the file *configs/deep_sort.yaml* and pre-trained weights for the tracker are stored in the file *deep_sort/deep/checkpoint/ckpt.t7*. The configuration file needs to be passed to the system at start-up. Each time objects in the video frame are detected by the object detector, they are passed to the object tracker:

```
outputs = self.deepsort.update(bbox_xywh, cls_conf, im)
```

The tracker compares the newly found object positions and tries to reidentify the vehicles in the frame. If there is a new vehicle in the frame a new identifier is assigned to it. After the reidentification, if the system was able to detect any vehicle, the trajectory analysis starts 6.2.

6.2 Trajectory Analyser Implementation

The trajectory analysis is responsible for storing information about the vehicle positions throughout the whole video footage. It is implemented as a Python dictionary where the vehicles identifier, the one assigned to it by the object tracker, is used as a key to the dictionary. The value of an item is a list of coordinates that represent the position of the vehicle in the video. The algorithm is implemented in the method `run` of the class `VideoTracker` located in the file `VideoTracker.py`. The trajectory analysis happens after the tracking of the objects is done; in other words when the vehicles in the frame have an identifier assigned to them. The algorithm also needs to remember when the vehicles have appeared for the first time and the last time. This information is used later for the extraction of the vehicle from the video during the anomaly detection.

The algorithm is run for every vehicle identified in the current frame. If a particular vehicle has already been detected by the system, its „last seen“ attribute is updated to the index of the current frame, otherwise its „first seen“ attribute is updated:

```
if identity not in self.trajectories:
  self.trajectories[identity] = []
  self.vehicle_occurence[identity] = {"first": idx_frame,
                                      "last": idx_frame}
else:
  self.vehicle_occurence[identity]["last"] = idx_frame
```

Then a coordinate which represents the position of the vehicle in the frame is calculated. The most accurate representation of the position of the vehicle is the centre of the bounding box's x axis and two-thirds of the bounding box's y axis from the top. Subsequently, the calculated coordinate is added to the trajectory of the vehicle.

```
if idx_trajectory % self.args.trajectory_interval == 0:
    box = bbox_xyxy[i]
    current_position = (int((box[0] + box[2]) // 2),
                        int(box[3] - abs(box[1] - box[3]) // 3))
    if len(self.trajectories[identity]) != 0:
        if not self.is_too_close(self.trajectories[identity][-1],
                                 current_position):
            self.trajectories[identity].append(current_position)
    else:
        self.trajectories[identity].append(current_position)
```

Two modifications have been done to the algorithm:

The first one helps to smooth out the trajectory by only taking into account every x frame. The problem was that sometimes the detector, especially on slowly moving vehicles, inaccurately detected the vehicle, so it seemed that the vehicle suddenly changed direction. This caused a lot of problems in the rule violation analysis where many false positives occurred. Therefore, the parameter *trajectory_interval* was added.

The second modification was not of a correctness nature but rather a performance one. When there were many idle vehicles in the scene, typically vehicles in the parking lot or in intersections, each time a coordinate was captured it was almost identical. This resulted in a performance issue during the rule violation analysis because the system had to unnecessarily calculate with the same coordinates. A method `is_too_close` solves this issue by comparing the previous and current coordinates of the vehicle. The threshold which decides if the vehicle has moved enough or not is calculated as 1% of the video resolution. As an example with 1280x720 resolution, the vehicle has to move at least 12 pixels on the x axis and 10 pixels on the y axis for the coordinate to be captured.

When the trajectory analysis is done, the dictionary is converted into a JSON file. The file also contains information about the video footage like its resolution, path on the filesystem and name.

6.3 Rule Creator Implementation

The rule creator is a Javascript application which allows users to define custom scene rules. It uses PaperJS framework which handles the creation of objects that represent shapes. The applications implementation is located in files *AnomalyDetectorFront/static/rule_creator/canvas_manipulation.js* and *AnomalyDetectorFront/static/rule_creator/controls.js*. The first file handles HTML canvas manipulation, key presses and shapes object creation while the second file takes care of the control buttons like switching a drawing mode, removal of all rules, conversion of PaperJS objects to JSON file.

The application supports six types of input:

- Draw a line
- Draw a box
- Draw a circle

- Draw an allowed direction for the shape/rule
- Forbid an entry for the shape/rule
- Delete the shape/rule

The polygons are internally represented as PaperJS objects. The canvas has a set size of 1280x720 pixels which makes the coordinates consistent across different display resolutions and web browsers. The aim was to have an image from the input video as a background of the canvas and to draw shapes on top of it using PaperJS. Unfortunately, either a picture or a PaperJS layer can be visible at the same time so there are two different canvases stacked on the top of each other. The bottom one shows a frame of the video footage and the top one serves as the drawing layers. The bottom canvas is controlled by functions in the file *AnomalyDetectorFront/static/rule_creator/controls.js*. When users select a video using a file picker, the video is loaded into a *video* hidden HTML tag. By moving the slider, a different frame is selected from the video so that users can find a situation which represents the scene the best. This file also takes care of all the control buttons on the page. Pressing the save button, the user-defined rules are converted into a JSON format by the `saveRules` function. This function also sends the converted JSON file to the web application which saves the file on the server's filesystem and adds it to the database.

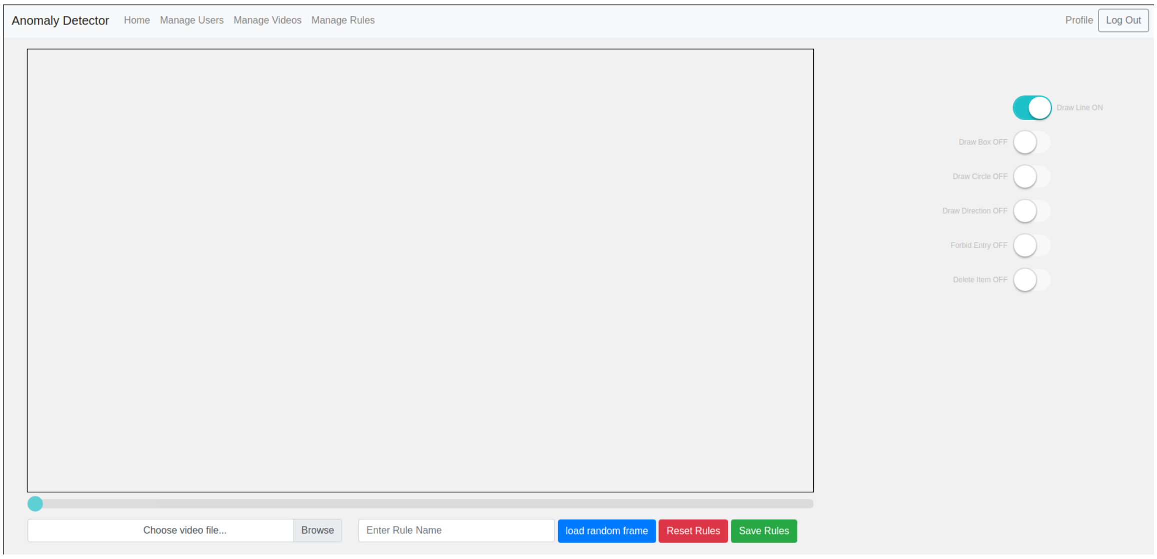


Figure 6.1: Final design of the Rule Creator

As can be seen in figure 6.1, users can select a particular input mode on the right side of the web page. The most flexible drawing mode is the 'Draw a line' one because it enables users to define precise shapes to fit any situation. The 'Draw a line' mode helps users to easily close the polygon by snapping the last line to the starting point if the mouse cursor moves too close to it. If the previous input mode was 'Draw a line', the polygon was not closed and users switch to a different one, the shape is automatically finished. Each rule has a random colour assigned to it to visually differentiate the created rules.

6.4 Rule Violation Analysis Implementation

The rule violation analysis is responsible for the detection of vehicles which break the user-defined rules in video footage. It accomplishes that by comparing the file containing the user-defined rules created in 6.3 and the file containing trajectories of the vehicles occurring in the video created in 6.2. The analysis is implemented in the class *AnomalyDetector* in the file *AnomalyDetectorBack/AnomalyDetector.py*. The rule violation analysis is started by the web interface of which implementation is discussed in section 6.5.

When the analysis is run, it first analyses the input videos using the systems discussed in sections 6.1 and 6.2. This is implemented in the method `traffic_scene_analysis`. Each input video is processed in a separate process using Python's built-in multiprocessing tool Pool. To prevent overloading of the system only `cpu core amount / 2` number of processes can run at the same time. Before the start of the video analysis and once it finishes, the system notifies the web application which uses the information to update its database. When the video analysis is done and a file containing trajectories is generated, the method `detect_anomalies` is called.

Trajectory files are processed sequentially but the particular vehicle trajectories are processed in parallel. The method `anml_dtc_proc_spawner` first loads up all the necessary files, parses the rule file and corrects the coordinates from the rule file (Rule Creator works in 1280x720 resolution while videos have variable resolution). For each trajectory, a process for method `anomaly_detector_process` is created which has user-defined rules and a particular vehicles trajectories at the input. This method iterates over the coordinates and checks if any of them have entered the used-defined rule. If the entered rule is of an „Entry forbidden“ nature, the vehicle is marked as a rule violator. However, the entered rule is of a „Direction Rule“ type, the trajectory of the vehicle is calculated and compared to the allowed one. If it deviates from the allowed value too much, the vehicle is marked as a rule violator:

```
for _, rule in rules['direction_rules'].items():
  if inside_polygon(coordinate[0], coordinate[1], np.array(rule['rules'])):
      current_direction = calculate_direction(trajectory[0], trajectory[1])
      if math.fabs(current_direction - rule['direction']) > MAX_DEVITATION:
          queue.put(vehicle_id)
for _, rule in rules['entrance_rules'].items():
  if inside_polygon(coordinate[0], coordinate[1], np.array(rule['rules'])):
      queue.put(vehicle_id)
```

The `inside_polygon` method checks whether the vehicle is inside the polygon of the rule. It takes the vehicles x and y coordinates and the coordinates of the rule's polygon. It returns true if the vehicle's coordinate is located inside the rule's polygon, otherwise it returns false. The `calculate_direction` method is called when the vehicle enters the „Direction Allowed“ type of rule. The method returns the angle difference of two line segments with the origin in $[0, 0]$ and ending the coordinates passed to the method. If this angle is greater than the allowed deviation, which reduces false positives, the vehicle is marked as a rule violator. When all of the trajectories are analysed and vehicles which broke the user-defined rules are found, a clip of the rule violation is created for each vehicle. The clip is then reformated, using ffmpeg A program, into an MP4 format with H264 CODEC, which results in a smaller size and ability to play the clip in the web application (OpenCV high-quality formats can not be played on the web). When the reformatting is done, the web application is notified to store the clips into the database. Users are notified when the analysis is done by an e-mail.

The output clips of the analysis can be viewed in the web application and is further discussed in paragraph 6.5.

6.5 Web Application Implementation

The web application is responsible for enabling users to explore the different parts of the system which were mentioned in previous sections. It enables users to control the whole system, to upload videos, to create rules and to start the rule violation analysis. The application is built in Python's web framework Flask A, uses Postgres and SQLAlchemy for the database layer and utilises Docker for the deployment of the database. Bootstrap 4 is used for styling most of the HTML tags.

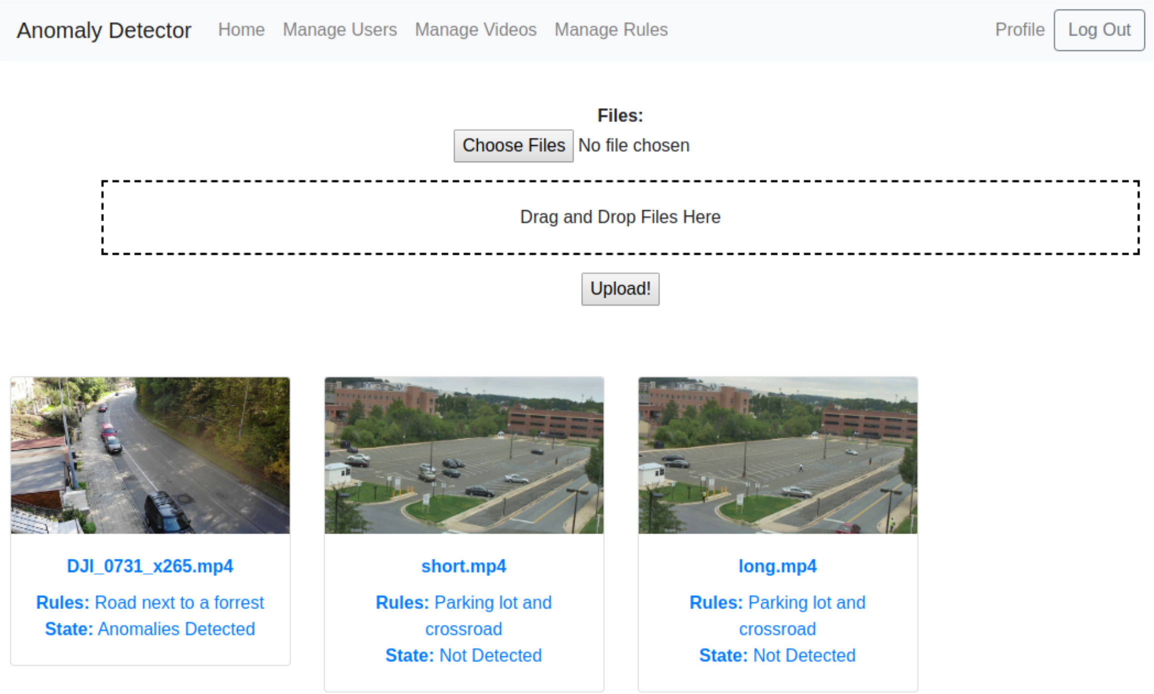
An existing solution was used as a base for the web application. Some generic functions and templates for user manipulation, login system and forms were reused from the IIS school assignment. My colleagues have allowed me to use the code of the assignment. Files that contain the reused code are marked in the header.

The website is separated into multiple pages, each providing different functionality. The functions which handle redirecting and template rendering are located in the file *AnomalyDetectorFront/routes.py*.

Manage Videos Page

Function related to this web page are:

- *manage_videos* - Queries for all videos uploaded by logged-in users and shows them in a grid layout with basic information.
- *upload_video* - Handles the video upload.
- *upload_complete* - Saves the video information to the database and reformats it if needed.



DJI_0731_x265.mp4

Rules: Road next to a forrest
State: Anomalies Detected

short.mp4

Rules: Parking lot and crossroad
State: Not Detected

long.mp4

Rules: Parking lot and crossroad
State: Not Detected

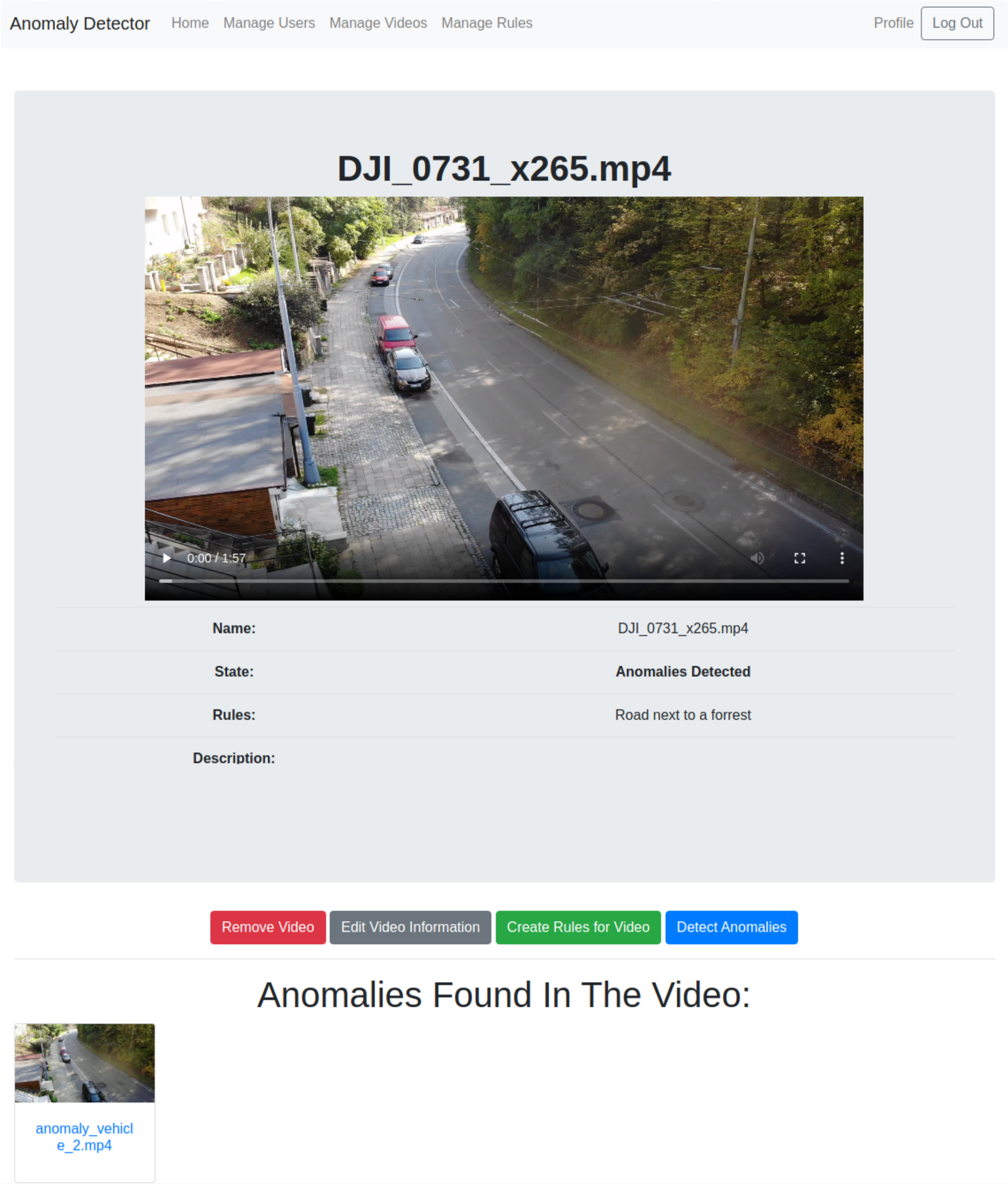
Figure 6.2: Final design of the Manage Videos page

As can be seen in figure 6.2, users can use the top part of the page to upload new videos into the system by drag-and-dropping them to the dashed line or by using the file selection pop-up window. When users upload a video, its format is checked and the video is reformated if the resolution is higher then FullHD or the CODEC is not H264. Also, each video uploaded by users, after the reformatting is done, is displayed in a grid layout with basic information attached to it. Clicking on the video thumbnail or the information below it, users are redirected to a more detailed view of the particular video.

Manage Video Page

Functions related to this web page are:

- *manage_video* - Queries for a particular video uploaded by a logged-in user and renders the page with information about the video.
- *edit_video* - Renders and changes information about a particular video uploaded by a logged-in user.
- *remove_video* - Removes a particular video and all related files from the system.



DJI_0731_x265.mp4

Name:	DJI_0731_x265.mp4
State:	**Anomalies Detected**
Rules:	Road next to a forrest
Description:	

Anomalies Found In The Video:

anomaly_vehicle_2.mp4

Figure 6.3: Final design of the Manage Videos page

The page shows all available information about a particular video and allows users to manage the video.

As shown in figure 6.3, the top of the page is used to display and play the video while interesting information about the video is below the media player. Below the section containing the video information, there are buttons which from the left:

- remove the video with all its connected files
- allow the information editing
- redirect users to a rule creator page for that particular video
- start the rule violation detection if possible

At the bottom of the page, anomalies found in the video are displayed. Clicking on these anomalies redirects users to a detailed page about a particular anomaly.

Manage Rules Page

The button at the top of the page redirects users to the rule creator page. Below the button, there are rules created by a logged-in user displayed in a grid layout with basic information about the particular rule. Clicking them redirects users to a detailed view of the particular rule.

More detailed information about the page is availabe in appendix C.

Manage Rule Page

The top of the page contains a picture of the scene used during the creation of the rule. Information about the rule and buttons are located beneath the picture. The first button deletes the rule with all associated files. The second one redirects users to a page for information editing. The last button starts the rule violation analysis for all assigned videos to the particular rule.

More detailed information about the page is availabe in appendix C.

Manage Anomaly Page

This page is very similar to the manage video one. The top of the screen is occupied by the media player which allows users to observe found anomaly. Information about the anomaly and the remove button are located below the media player.

More detailed information about the page is availabe in appendix C.

User management pages

These pages allow users and admins to manage user accounts. They are all based on basic forms where some fields are mandatory and some optional. The e-mail assigned to the user's account is used for notification when the video uploading, video reformatting or rule violation analysis is finished. Users can only edit their profile while admins can manage every account.

More detailed information about the page is availabe in appendix C.

Database layer

The models which represent database tables are located in files in the *AnomalyDetector-Front/models* folder. These models are automatically converted by SQLAlchemy to relational tables in the Postgres database. Items only use integer values as primary keys to the database. Video thumbnails and video footage is stored on the server filesystem of the server while their paths are stored in the database. Working with the database works as follows:

```
admin = RegisteredUser(name='admin', email='admin@admin.cz',
                       phone_num='123456678', user_type='admin')
admin.set_password('admin')
db.session.add(admin)
db.session.commit()
```

First, an SQLAlchemy object has to be created. Secondly, a password is set for the user. When the `add` method is called, SQLAlchemy converts the objects to a relational representation and inserts it to the database. The `commit` method confirms the changes made in the database. This way each object is added to the database or updated.

The removal of the items from the database is done similarly:

```
user = RegisteredUser.query.filter_by(email=email).first()
if user is None:
    return redirect(url_for('index'))
db.session.delete(user)
db.session.commit()
```

Firstly, the item has to be retrieved from the database. Secondly, using the `delete` method, the item is removed from the database. Lastly, the changes made in the database are confirmed. This way each object is removed from the database.

6.6 System Overview

The system is separated into multiple folders and files based on the functionalities it implements. Systems backend components:

- *AnomalyDetectorBack/detector/YOLOv3* - a folder containing YOLOv3 implementation, configuration and pre-trained weights
- *AnomalyDetectorBack/deep_sort* - a folder containing DeepSORT implementation, configuration and pre-traied weights
- *AnomalyDetectorBack/configs* - a folder containing information about filepaths needed for object detector and tracker
- *AnomalyDetectorBack/utils* - a folder containing functions used for parsing, displaying detected images, web requests
- *AnomalyDetectorBack/AnomalyDetector.py* - a file containing an implementation of the user-defined rules violation detector
- *AnomalyDetectorBack/VideoTracker.py* - a file containing an implementation of vehicle detector, vehicle tracker and trajectory analysis
- *AnomalyDetectorBack/requirements.txt* - a file containing dependencies needed to run the backend system

Systems frontend components:

- *AnomalyDetectorFront/models* - a folder containing database models
- *AnomalyDetectorFront/static* - a folder containing static files like pictures, css styles, Javascript code, uploaded video footage
- *AnomalyDetectorFront/templates* - a folder containing html templates for the web application
- *AnomalyDetectorFront/DetectorState.py* - a file responsible for manipulation the state of the system
- *AnomalyDetectorFront/forms.py* - a file containing forms for the web application
- *AnomalyDetectorFront/routes.py* - a file containing routing and business logic of the web application
- *AnomalyDetectorFront/utils.py* - a file containing utility functions for checking if a file is a video, e-mail sending, path manipulation

Deployment components:

- *docker/docker-compose.yml* - a file containing instruction about the deployment of the system
- *include/flask_permissions* - a folder containing an edited version of flask_permissions module for Python
- *migrations* - a folder containing migrations of the database
- *anomaly_detector.py* - a file that needs to be run by flask to run the system
- *config.py* - a file containing configuration for the flask application
- *Dockerfile* - a file responsible for building the Docker image of the application

6.7 System Testing

It is necessary to test that the system can quickly and precisely analyse large quantities of video footage. Unfortunately, both of these criteria are variable as the precision of the rule violation detection is dependent on how accurately are the scene rules created. The performance is also dependent on the model of the GPU if it is even available. What can be tested is the precision of the vehicle detector and the vehicle tracker as well as the performance of the system using one specific GPU which I have available. Another criterion for testing is the stability of the system because it is meant to be installed on a remote powerful server and users should not be able to access the server. Lastly, the web application has to be working on major web browsers.

Performance Testing
The system analyses videos at 40-60 FPS depending on the number of vehicles in the scene. The analysis of one-minute long video takes about 30-45 seconds. The system was run on Nvidia GTX 1070 GPU. Also, the video reformatting is about 2.7 times faster than the real-time playback of the video which results in one-minute long video being reformated

in about 22 seconds. The reformatting is done on the CPU which in this case was Intel i7-7700k running at 4.8Ghz. The performance information can be seen in the terminal output of the system during the analysis.

Web Application Testing

The web application was tested on 3 popular web browsers. These are Firefox, Google Chrome and Chromium. The design looked identical on each browser. There was a difference in fluidity in the video playback and scrollbar in the rule creator. When the operating system of the user was Windows 10 rather than a Linux distribution, the video seemed smoother. This does not affect functionality in any way and it is probably caused by the lack of GPU acceleration in Linux browsers.

Video Analysis Testing

The system was tested with video footage of parking lots from the top and the side perspective[1].

An example of a frame of the video with a top-down perspective is shown in figure 6.4. The system was unable to detect any vehicles in this perspective. This is caused by the object detector which is not pre-trained on images of the roofs of vehicles. This limitation could be resolved by the training of the object detector on thousands of annotated pictures of roofs of vehicles.

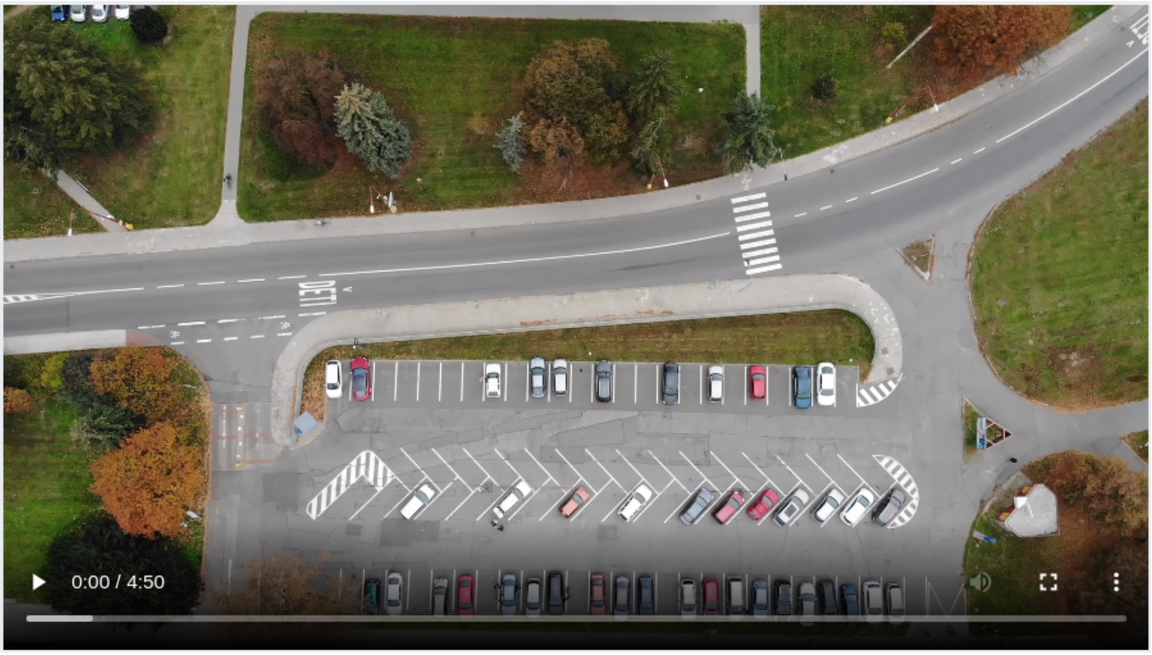


Figure 6.4: Top view of a parking lot

The frame is from the testing video with a side perspective and it is shown in figure 6.5. In the video, there are 20 unique vehicles but the system detected 24 vehicles (the last

[1] Taken from: https://data.kitware.com/#collection/56f56db28d777f753209ba9f/folder/56f581ce8d777f753209ca43

vehicle is next to a tree on the bottom right of figure 6.5). This is caused by the vehicle identified by the number '18' in figure 6.5. This vehicle is stationary throughout the whole video and is partly occluded by another vehicle and a street lamp. The object detector was unable to consistently identify this vehicle which caused assigning of new identifiers to the vehicle.

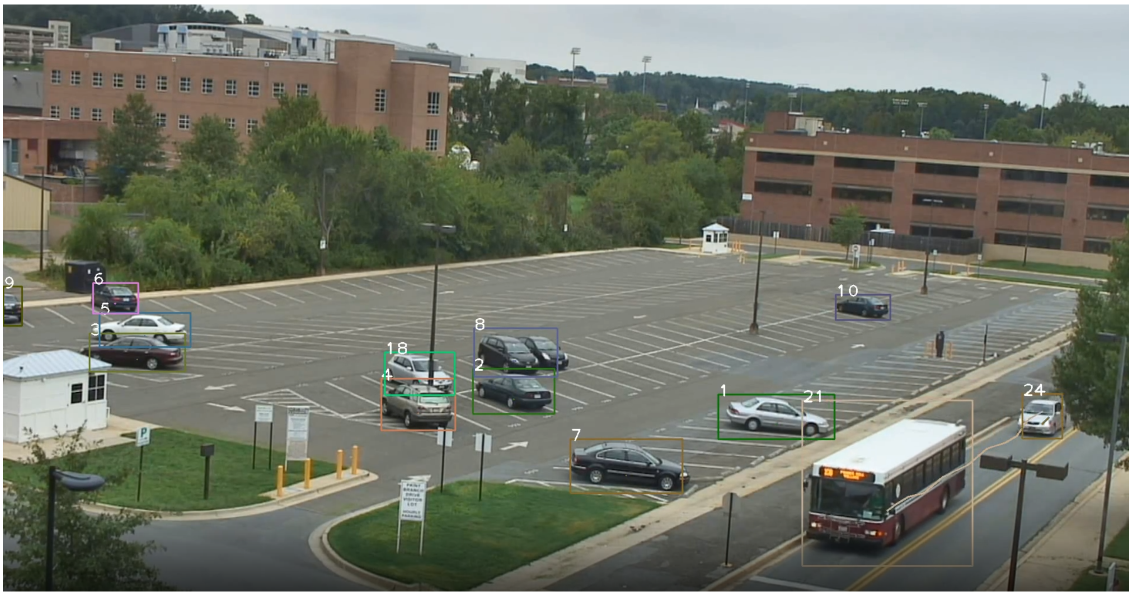
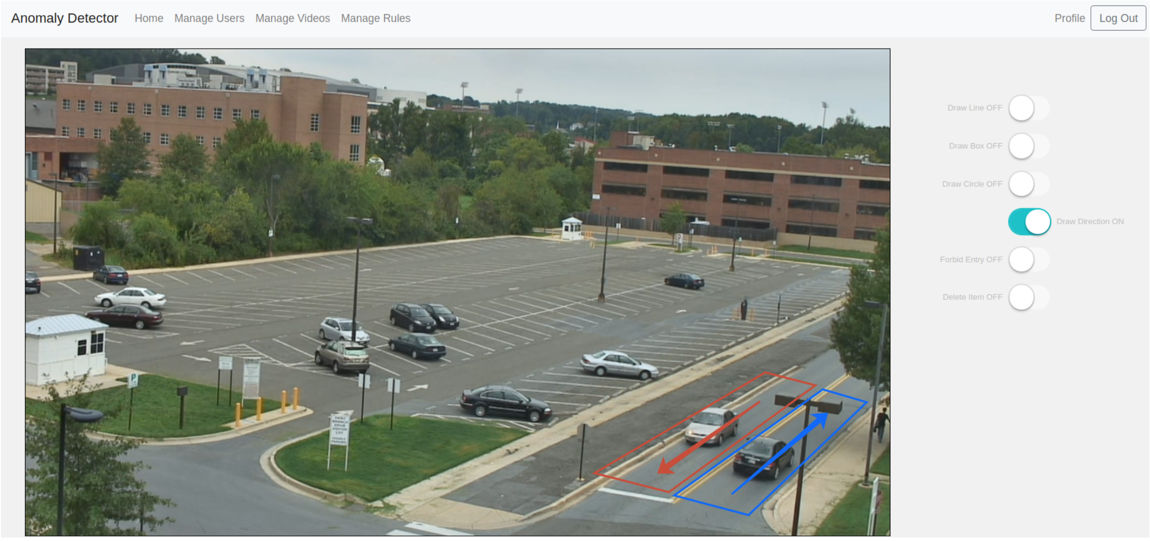


Figure 6.5: Last frame of the testing video footage

Figure 6.6 shows rules created for the testing video. It checks if vehicles follow their traffic lane. Only one rule violation was found which was caused by the bus in figure 6.5. The trajectory of the bus has exceeded the allowed deviation of the allowed direction. This was also caused by the object detector which sometimes detects bounding boxes which do not perfecly define the vehicles.



Draw Line OFF
Draw Box OFF
Draw Circle OFF
Draw Direction ON
Forbid Entry OFF
Delete Item OFF

Figure 6.6: User Rules for the testing video footage

Both presented examples show situations in which the system has problems to function correctly. A more precise object detector and a different dataset need to be used to mitigate these issues.

System Stability Testing

The stability of the system was due to a limited time only tested for a day. The system was responsive throughout the whole day and never crashed. If the system were to be publicly and commercially released, extensive stability testing would need to be carried out.

Chapter 7

Conclusions

The goal of the thesis was to create a system which automatically detects anomalies in the video footage of a traffic scene. This goal was accomplished.

I have studied literature and the state-of-the-art solutions and my gained knowledge is discussed in chapter 2. Then, I have analysed the state-of-the-art solutions and compared them in chapter 4. The outline of the system was created which is described in chapter 5. The outlined system was implemented and is covered in chapter 6. Lastly, the system was tested which is discussed in section 6.7.

The outcome of the thesis is a video analysis program which combines YOLOv3 for object detection, DeepSort for object tracking, an algorithm for trajectory extraction, and an algorithm for a comparison of vehicle trajectories and user-defined rules. This program takes a video file and user-defined rules as an input, analyses the video file to extract the vehicle trajectories and detects if any of the vehicles violated any of the user-defined rules. A web application was also created which encloses the analysis program, enabling users to define scene rules, upload and manage video footage, start the aforementioned video analysis program and view the output of the analysis was created.

During the elaboration of this work, I learned a lot of new information about machine learning, computer vision and web application development industry. Especially, I have improved coding in Python, explored scientific libraries like NumPy, PyTorch, Scikit-learn, OpenCV and learned about modern computer vision methods and techniques.

Future work includes adding more features to the user-defined rules creator to further expand the possibilities of the system. Also, a faster and more precise detection and tracking methods could be used to eliminate some of the drawbacks of the current system. Moreover, the web application could be improved to be more seamless to use, to offer more features and to provide more clear feedback on user actions. Lastly, the web API could be rewritten with authentication in mind.

The systems similar to the one described in the thesis will only gain in popularity as will rise the number of surveillance cameras which monitor every human step.

Bibliography

[1] Agren, S. *Object tracking methods and their areasof application: A meta-analysis.* Umea, Sweden, None. Masters Thesis. UME AUniversity. Available at: http://www8.cs.umu.se/education/examina/Rapporter/SannaAgrenFinal.pdf.

[2] Amigos Maker. *What is scikit learn?* [online]. DEV Community, november 2019 [cit. 2020-5-15]. Available at: https://dev.to/amigosmaker/what-is-scikit-learn-5ddl.

[3] Arcgis. *How single-shot detector (SSD) works* [online]. Arcgis, 2019 [cit. 2020-5-15]. Available at: https://developers.arcgis.com/python/guide/how-ssd-works/.

[4] Avigilon. *Know what is happening. Act with certainty.* [online]. January 2020 [cit. 2020-5-15]. Available at: https://www.avigilon.com/.

[5] AxxonSoft. *AxxonSoft Video Surveillance and Security Solutions* [online]. 2020 [cit. 2020-5-15]. Available at: https://www.axxonsoft.com/.

[6] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. *SIMPLE ONLINE AND REALTIME TRACKING* [online]. July 2017 [cit. 2020-5-15]. Available at: https://arxiv.org/pdf/1602.00763v2.pdf.

[7] CertiCon. *CertiCon členem Svazu průmyslu a dopravy ČR* [online]. 2020 [cit. 2020-5-15]. Available at: https://www.certicon.cz/.

[8] CertiCon. *Úvodní stránka* [online]. 2020 [cit. 2020-5-15]. Available at: http://www.certiconvis.cz/.

[9] Ganesh, P. *Object Detection : Simplified* [online]. Towards Data Science, october 2019 [cit. 2020-5-15]. Available at: https://towardsdatascience.com/object-detection-simplified-e07aa3830954.

[10] Guru99. *What is TensorFlow? Introduction, Architecture & Example* [online]. [cit. 2020-5-15]. Available at: https://www.guru99.com/what-is-tensorflow.html.

[11] Held, D., Thrun, S. and Savarese, S. *Learning to Track at 100 FPS with Deep Regression Networks* [online]. 2016 [cit. 2020-5-15]. Available at: http://www8.cs.umu.se/education/examina/Rapporter/SannaAgrenFinal.pdf.

[12] Heller, M. *What is Keras? The deep neural network API explained* [online]. InfoWorld, january 2019 [cit. 2020-5-15]. Available at: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html.

[13] Hui, J. *SSD object detection: Single Shot MultiBox Detector for real-time processing* [online]. December 2018 [cit. 2020-5-15]. Available at: https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06.

[14] IntelliVision. *AI Video Analytics - Face Recognition, ALPR/ANPR, Retail Analytics, Traffic* [online]. December 2019 [cit. 2020-5-15]. Available at: https://www.intelli-vision.com/.

[15] Jonathan, H. *Understanding Region-based Fully Convolutional Networks (R-FCN) for object detection* [online]. Medium, Apr 2019 [cit. 2020-5-15]. Available at: https://medium.com/@jonathan_hui/understanding-region-based-fully-convolutional-networks-r-fcn-for-object-detection-828316f07c99.

[16] Keras Team. *Keras: Deep Learning for humans* [online]. November 2019 [cit. 2020-5-15]. Available at: https://github.com/keras-team/keras.

[17] Maiya, S. R. *DeepSORT: Deep Learning to track custom objects in a video* [online]. AI & Machine Learning Blog, april 2020 [cit. 2020-5-15]. Available at: https://nanonets.com/blog/object-tracking-deepsort/#multiple-object-tracking.

[18] Mallick, S. *Object Tracking using OpenCV (C /Python)* [online]. Feb 2017 [cit. 2020-5-15]. Available at: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/.

[19] Mallick, S. *GOTURN : Deep Learning based Object Tracking* [online]. Jul 2018 [cit. 2020-5-15]. Available at: https://www.learnopencv.com/goturn-deep-learning-based-object-tracking/.

[20] Nikolenko, S. and Gaydashenko, A. *Tracking Cows with Mask R-CNN and SORT* [online]. July 2018 [cit. 2020-5-15]. Available at: https://medium.com/neuromation-blog/tracking-cows-with-mask-r-cnn-and-sort-fcd4ad68ec4f.

[21] Ning, G., Zhang, Z., Huang, C. and Hea, Z. *Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking* [online]. Columbia: [b.n.], june 2016 [cit. 2020-5-15]. Available at: https://arxiv.org/pdf/1607.05781v1.pdf.

[22] Open Data Science, O. *Overview of the YOLO Object Detection Algorithm* [online]. Medium, Sep 2018 [cit. 2020-5-15]. Available at: https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0.

[23] OpenCV. *Cv::TrackerGOTURN Class Reference* [online]. OpenCV, Oct 2019 [cit. 2020-5-15]. Available at: https://docs.opencv.org/3.4/d7/d4c/classcv_1_1TrackerGOTURN.html.

[24] Pytorch. *Pytorch/pytorch* [online]. May 2020 [cit. 2020-5-15]. Available at: https://github.com/pytorch/pytorch.

[25] Remanan, S. *Beginner's Guide to Object Detection Algorithms* [online]. Medium, april 2019 [cit. 2020-5-15]. Available at: https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375.

[26] Sachan, A. *A Quick Guide to Object Tracking: MDNET, GOTURN, ROLO* [online]. April 2019 [cit. 2020-5-15]. Available at: https://cv-tricks.com/object-tracking/quick-guide-mdnet-goturn-rolo/.

[27] Scikit Learn. *Scikit-learn* [online]. May 2020 [cit. 2020-5-15]. Available at: https://github.com/scikit-learn/scikit-learn.

[28] Shetty, S. *Tracking Cows with Mask R-CNN and SORT* [online]. September 2018 [cit. 2020-5-15]. Available at: https://hub.packtpub.com/what-is-pytorch-and-how-does-it-work.

[29] Sik Ho, T. *Positive-Sensitive Score Maps (Object Detection)* [online]. Towards Data Science, may 2019 [cit. 2020-5-15]. Available at: https://towardsdatascience.com/review-r-fcn-positive-sensitive-score-maps-object-detection-91cd2389345c.

[30] Team, K. *Simple. Flexible. Powerful.* [online]. [cit. 2020-5-15]. Available at: https://keras.io/.

[31] Tensorflow. *TensorFlow* [online]. May 2020 [cit. 2020-5-15]. Available at: https://github.com/tensorflow/tensorflow.

[32] Vi, A. *Agent Vi's Comprehensive Video Analytics Range* [online]. 2020 [cit. 2020-5-15]. Available at: https://agentvi.com/.

[33] Wojke, N., Bewley, A. and Paulus, D. *SIMPLE ONLINE AND REALTIME TRACKING WITH A DEEP ASSOCIATION METRIC* [online]. April 2017 [cit. 2020-5-15]. Available at: https://arxiv.org/pdf/1703.07402v1.pdf.

[34] Yegulalp, S. *What is TensorFlow? The machine learning library explained* [online]. InfoWorld, Jun 2019 [cit. 2020-5-15]. Available at: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html.

[35] ZQPei. *ZQPei/deep_sort_pytorch* [online]. May 2020 [cit. 2020-5-15]. Available at: https://github.com/ZQPei/deep_sort_pytorch.

Appendices

List of Appendices

Appendix A

Links to Mentioned Technologies

This chapter shows links to mention technologies in the thesis.

Links

Used existing solution
Used Solution

NumPy
NumPy

SciPy
SciPy

Cython
Cython

Numba
Numba

Theano
Theano

CNTK
CNTK

Forensic search
Forensic Search

OpenCV
OpenCV

PostgreSQL
PostgreSQL

SQLAlchemy
SQLAlchemy

MIT License
MIT License

Paper.js
Paper.js

JSON
JSON

Bootstrap 4
Bootstrap 4

Flask
Flask

FFmpeg
FFmpeg

Appendix B

Complete Graphical User Interface Outline

A graphical user interface needs to offer a simplistic way of video manipulation and output observing. It will be implemented in Flask A on the server-side and Bootstrap 4 A for the interface design. SQLAlchemy A will be used to translate class models into a relational database. Graphical User interface will be divided into multiple segments: Manage Users, Manage Videos, Manage Video, Manage Rules, Manage Rule.

Manage Users
Admins will in terms of user management be able to change permissions of other users, create new accounts, and change their profile settings. Users will have the option to change their passwords, e-mails, names and notification settings.

Manage Videos
Users will be able to upload new videos and find the already uploaded ones. By clicking on the existing videos, users will be redirected to a more detailed page about the particular video footage. Users will only see videos which were uploaded by them and will not be able to in any way manipulate videos they do not ’own’.

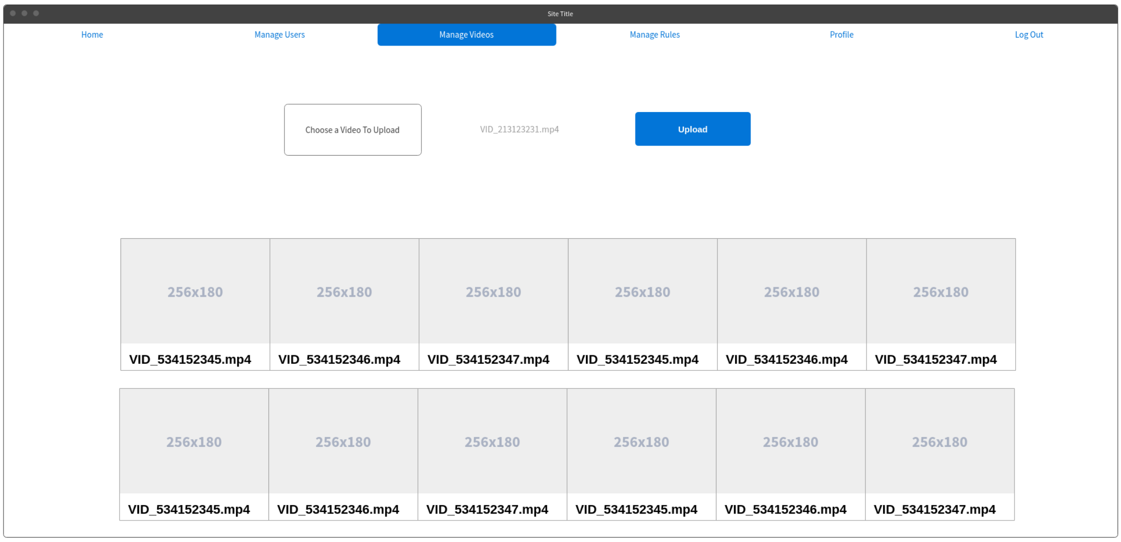
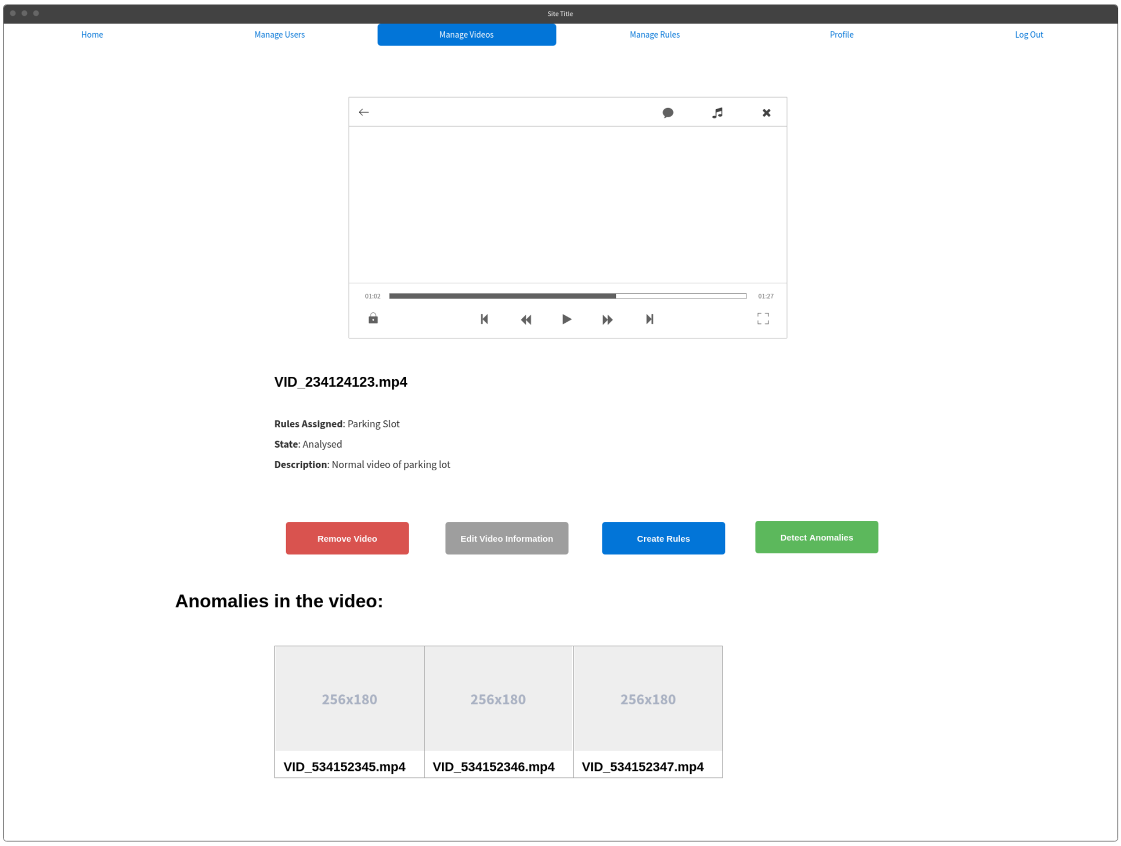


Figure B.1: Outline of the web page for video upload

Figure B.1 shows the outline of the web design. On the top of the page, users can click the white button to open a file explorer and select videos to upload or drag-and-drop them into the area. Below the upload section, all videos uploaded by signed users are displayed. Clicking those videos redirects users to a more detailed view with options to view and edit them. This page is discussed in paragraph *Manage Video* B.

Manage Video

This page will allow users to see the detailed view of a particular video, edit its information and start the rule violation detection for the particular video.



VID_234124123.mp4

Rules Assigned: Parking Slot
State: Analysed
Description: Normal video of parking lot

Anomalies in the video:

VID_534152345.mp4
VID_534152346.mp4
VID_534152347.mp4

Figure B.2: Outline of the web page for video manipulation

Figure B.2 shows the design outline of the said page. At the top of the page, a video player will be located. Users will be able to view the whole video, fast forward it and maximise the video player window. Below the video player, there will be a column with information about the video like its name, description, rule it is assigned to and its state. Below the information, there will be buttons which can remove the video from the system, allow for modification of the information, redirect to the rule creator page and start the rule violation analysis. At the bottom of the page, there will be a list of all anomalies found in the particular video. They will only appear after the rule violation analysis is complete and will be removed if the rules for the video change.

Manage Rules

This page will be responsible for displaying existing user-defined rules, redirecting users to a particular rule and redirecting users to the rule creator 5.7.

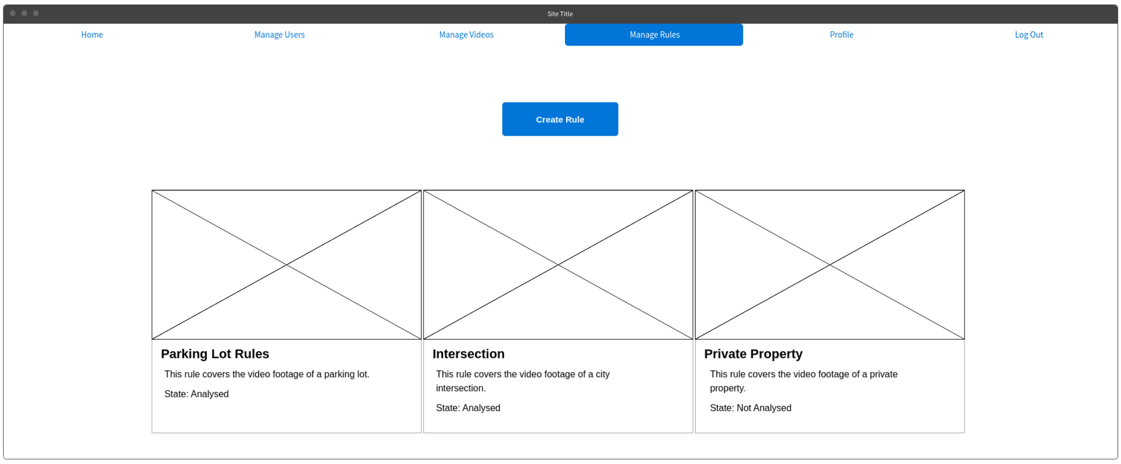
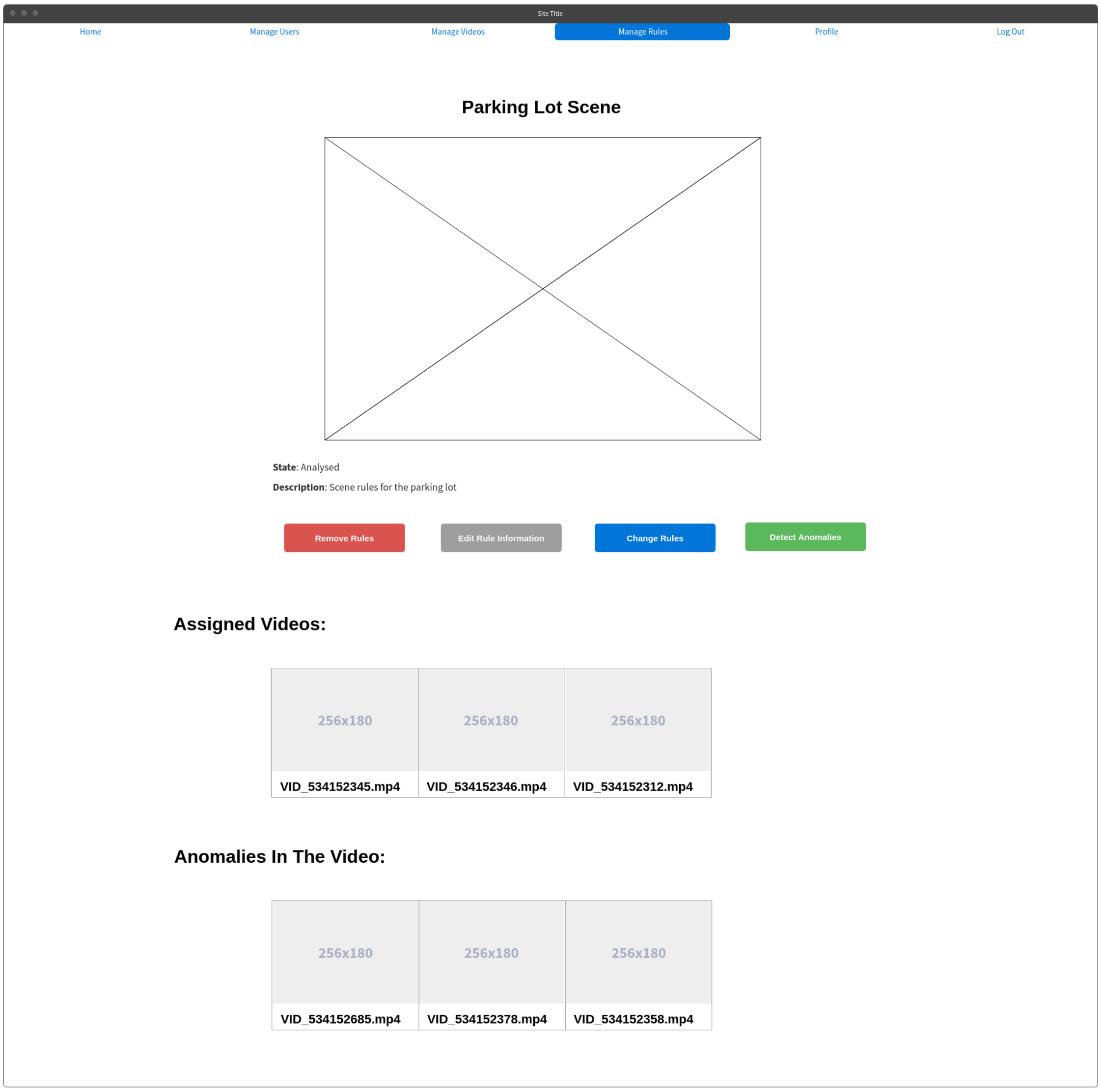


Figure B.3: Outline of the web design for the rules management

Figure B.3 shows the outline of the design of the page. At the top of the page, there will be a button that redirects users to the rule creator 5.7. Below the button, all the rules created by users will be displayed with an icon and brief information about the rule. Clicking on a rule will redirect users to a more detailed view of that particular rule.

Manage Rule

This page will be graphically similar to the *Manage Video* B one but instead of managing a particular video, it will control a particular scene rule. It will also enable users to edit the rule information and start the rule violation detection for every video assigned to it.



Parking Lot Scene

State: Analysed

Description: Scene rules for the parking lot

Assigned Videos:

VID_534152345.mp4 VID_534152346.mp4 VID_534152312.mp4

Anomalies In The Video:

VID_534152685.mp4 VID_534152378.mp4 VID_534152358.mp4

Figure B.4: Outline of the web design for rule manipulation

As seen in figure B.4, the top of the page holds the image preview of the rule with additional information below that. The centre of the screen is occupied by the control buttons which will remove the rule, enable the alteration of the information, enable the modification of the rule and start the analysis for every video assigned to the rule. At the bottom of the page, there will be a list of all videos assigned to a particular rule as well as a list of all anomalies found in videos assigned to the given rule. Clicking any of these videos will redirect users to a page where the video can be viewed and manipulated.

Appendix C

Complete Web Application Implementation

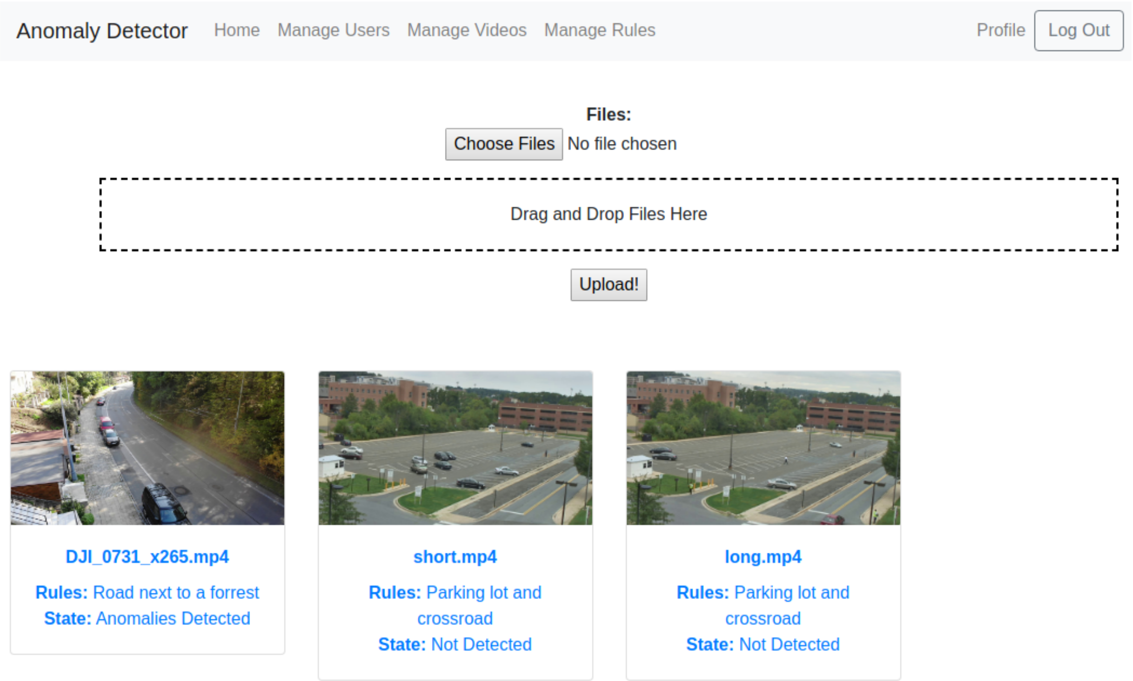
The web application is responsible for enabling users to explore the different parts of the system which were mentioned in previous sections. It enables users to control the whole system, to upload videos, to create rules and to start the rule violation analysis. The application is built in Python's web framework Flask A, uses Postgres and SQLAlchemy for the database layer and utilises Docker for the deployment of the database. Bootstrap 4 is used for styling most of the HTML tags.

The website is separated into multiple pages, each providing different functionality. The functions which handle redirecting and template rendering are located in the file *AnomalyDetectorFront/routes.py*.

Manage Videos Page

Function related to this web page are:

- *manage_videos* - Queries for all videos uploaded by logged-in users and shows them in a grid layout with basic information.
- *upload_video* - Handles the video upload.
- *upload_complete* - Saves the video information to the database and reformats it if needed.



DJI_0731_x265.mp4
Rules: Road next to a forrest
State: Anomalies Detected

short.mp4
Rules: Parking lot and crossroad
State: Not Detected

long.mp4
Rules: Parking lot and crossroad
State: Not Detected

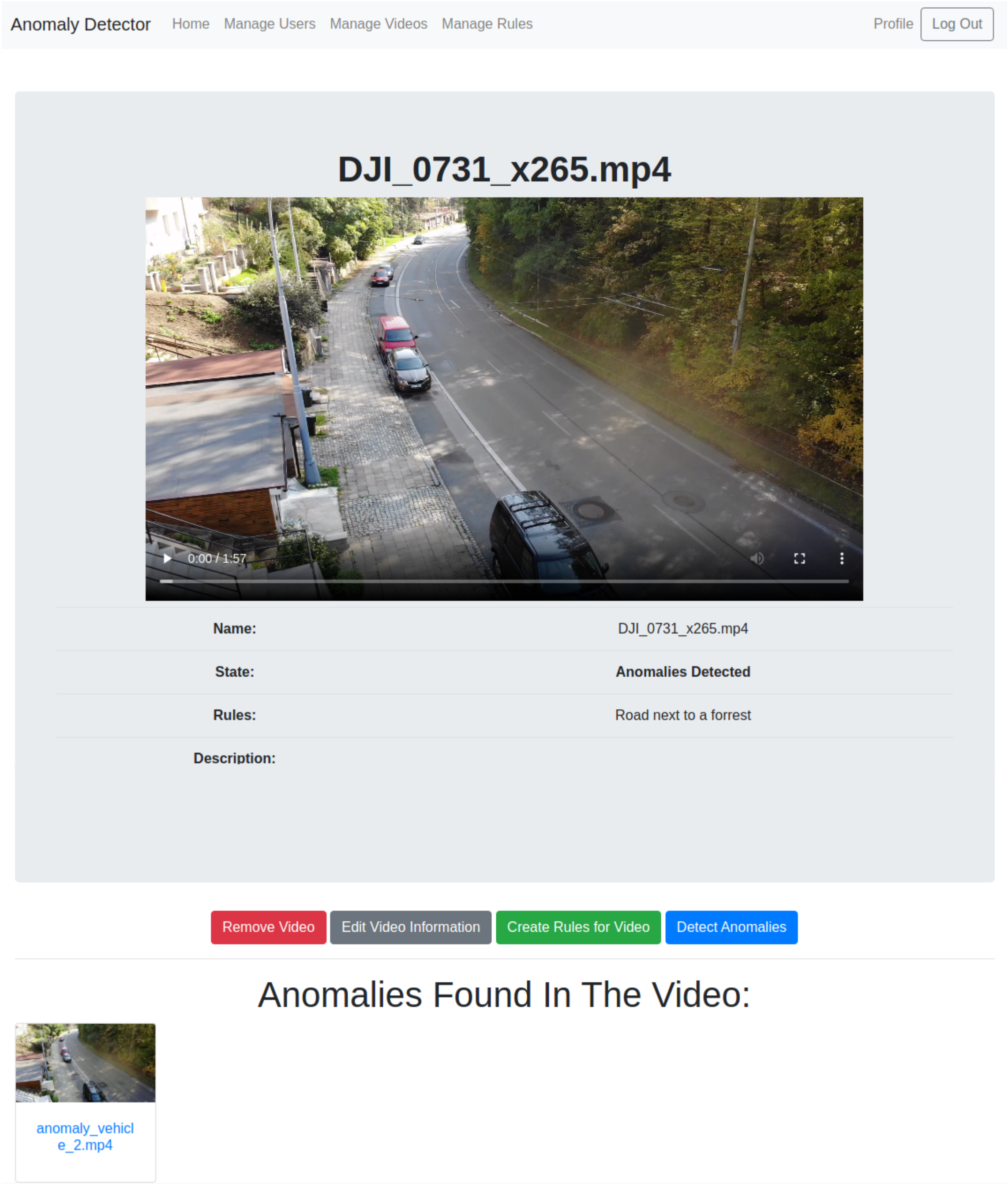
Figure C.1: Final design of the Manage Videos page

As can be seen in figure C.1, users can use the top part of the page to upload new videos into the system by drag-and-dropping them to the dashed line or by using the file selection pop-up window. When users upload a video, its format is checked and the video is reformated if the resolution is higher then FullHD or the CODEC is not H264. Also, each video uploaded by users, after the reformatting is done, is displayed in a grid layout with basic information attached to it. Clicking at the video thumbnail or the information below it, users are redirected to a more detailed view of the particular video.

Manage Video Page

Functions related to this web page are:

- *manage_video* - Queries for a particular video uploaded by a logged-in user and renders the page with information about the video.
- *edit_video* - Renders and changes information about a particular video uploaded by a logged-in user.
- *remove_video* - Removes a particular video and all related files from the system.



DJI_0731_x265.mp4

Name:	DJI_0731_x265.mp4
State:	**Anomalies Detected**
Rules:	Road next to a forrest
Description:	

Anomalies Found In The Video:

anomaly_vehicle_2.mp4

Figure C.2: Final design of the Manage Videos page

The page shows all available information about a particular video and allows users to manage the video.

As shown in figure C.2, the top of the page is used to display and play the video while interesting information about the video is below the media player. Below the section containing the video information, there are buttons which from the left:

- remove the video with all its connected files

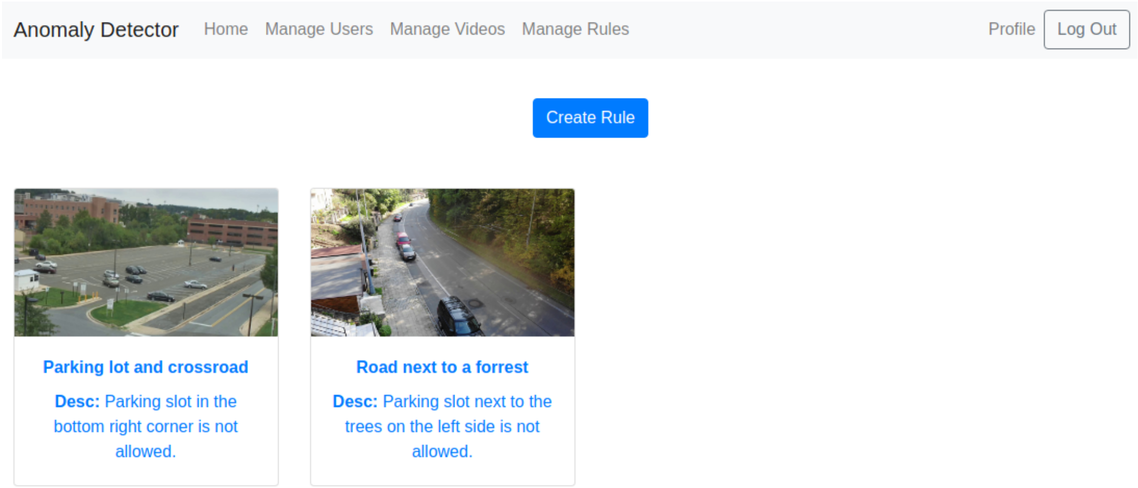
- allow the information editing
- redirect users to a rule creator page for that particular video
- start the rule violation detection if possible

At the bottom of the page, anomalies found in the video are displayed. Clicking on these anomalies redirects users to a detailed page about a particular anomaly.

Manage Rules Page

The function related to this web page is:

- *manage_rules* - Queries for all rules created by a logged-in user and displays them in a grid layout with basic information.



Parking lot and crossroad

Desc: Parking slot in the bottom right corner is not allowed.

Road next to a forrest

Desc: Parking slot next to the trees on the left side is not allowed.

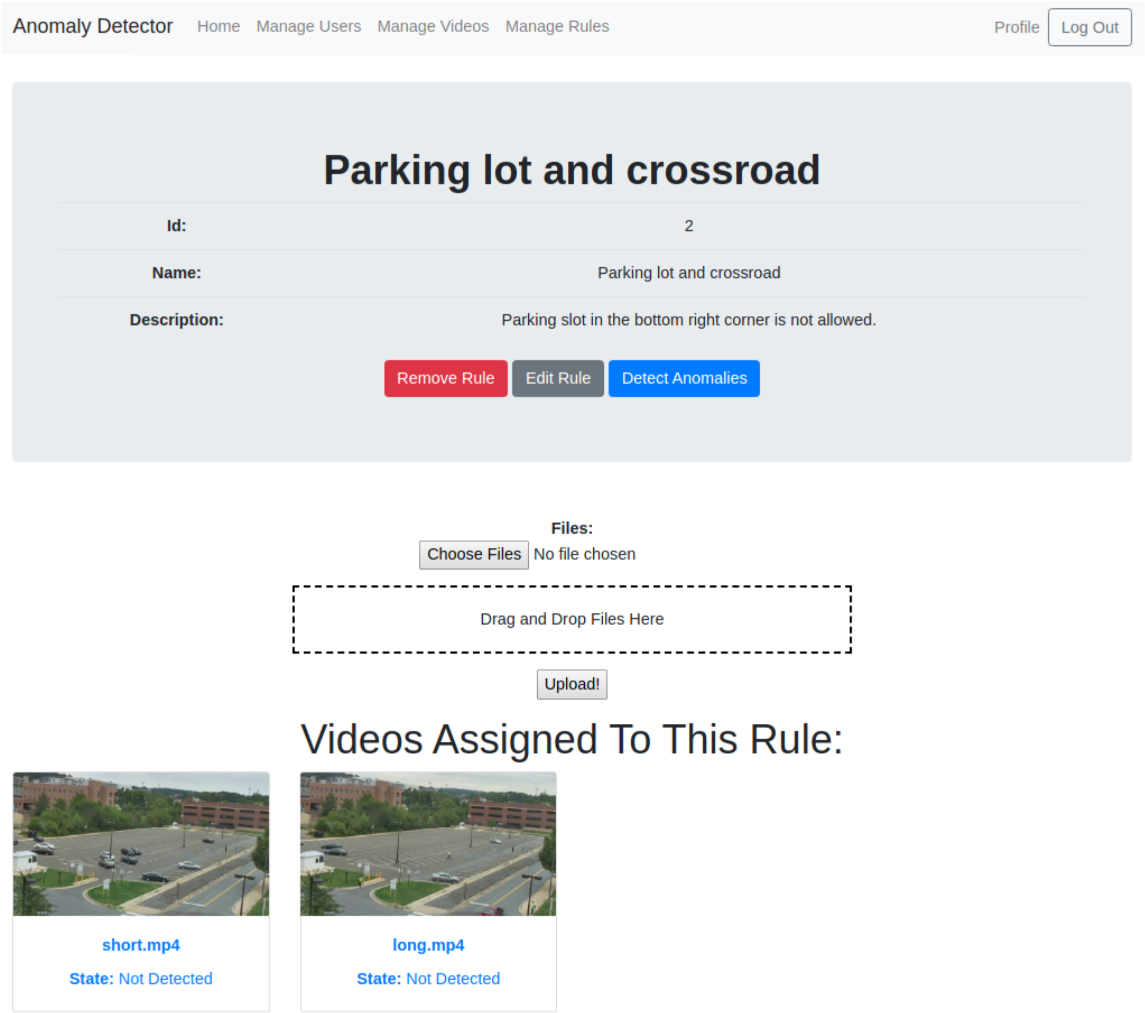
Figure C.3: Final design of the Manage Rules page

As can be seen in figure C.3, the button at the top of the page redirects users to the rule creator page. Below the button, there are rules created by a logged-in user displayed in a grid layout with basic information about the particular rule. Clicking them redirects users to a detailed view of the particular rule.

Manage Rule Page

Functions related to this web page are:

- *manage_rule* - Queries for a particular rule created by a logged-in user and renders the page with information about the rule.
- *edit_rule* - Renders and changes information about a particular rule created by a logged-in user.
- *remove_rule* - Removes a particular rule and all related files from the system.



Parking lot and crossroad

Id:	2
Name:	Parking lot and crossroad
Description:	Parking slot in the bottom right corner is not allowed.

Videos Assigned To This Rule:

short.mp4
State: Not Detected

long.mp4
State: Not Detected

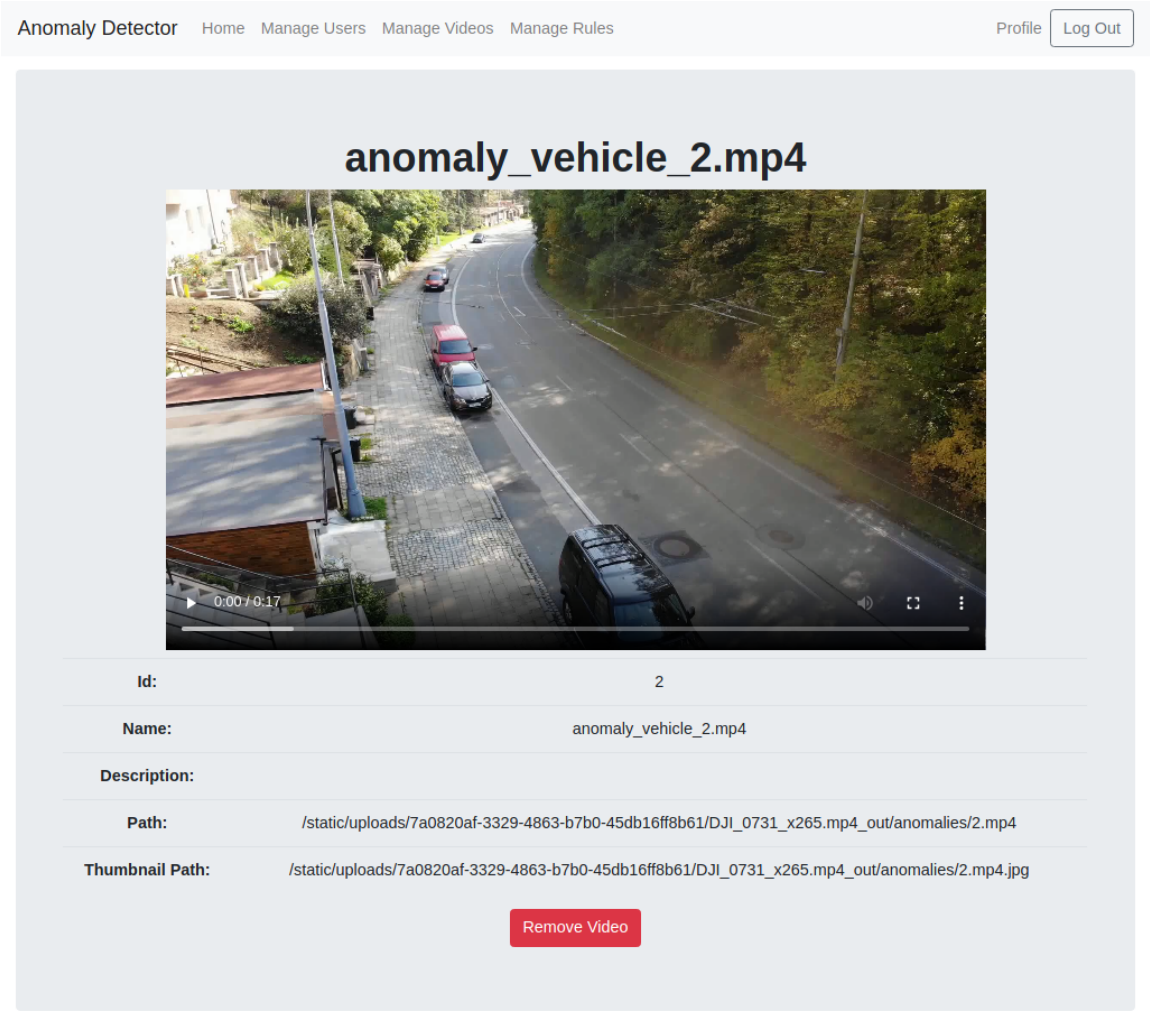
Figure C.4: Final design of the Manage Videos page

As can be seen in figure C.4, the top of the page contains the of information about the rule. Buttons are located beneath the information. The first button deletes the rule with all associated files. The second one redirects users to a page for information editing. The last button starts the rule violation analysis for all assigned videos to the particular rule.

Manage Anomaly Page

Functions related to this web page are:

- *manage_anomaly* - Queries for a particular anomaly by id and renders the page with video preview and information about the anomaly
- *remove_anomaly* - Removes a particular anomaly and all related files from the system.



anomaly_vehicle_2.mp4

Id:	2
Name:	anomaly_vehicle_2.mp4
Description:	
Path:	/static/uploads/7a0820af-3329-4863-b7b0-45db16ff8b61/DJI_0731_x265.mp4_out/anomalies/2.mp4
Thumbnail Path:	/static/uploads/7a0820af-3329-4863-b7b0-45db16ff8b61/DJI_0731_x265.mp4_out/anomalies/2.mp4.jpg

Figure C.5: Final design of the Anomaly Page

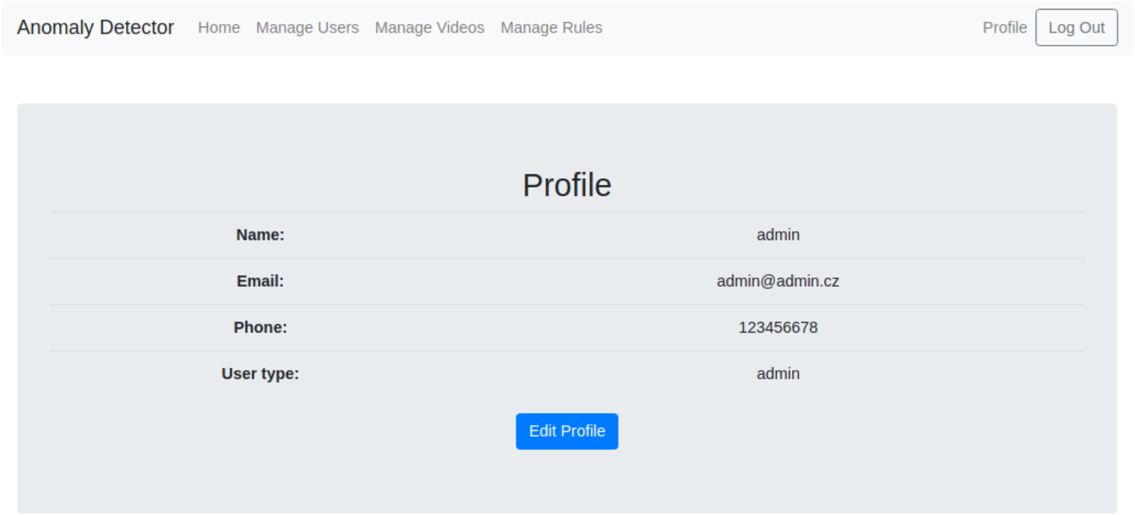
This page is very similar to the manage video one. As can be seen in figure C.5, the top of the screen is occupied by the media player which allows users to observe found anomaly. Information about the anomaly and the remove button are located below the media player.

User management pages

Functions related to this web page are:

- *manage_users* - Queries all registered users in the database for the admin users only.
- *manage_one_user* - Renders a profile of a particular registered user for the admin users only.
- *profile* - Renders a profile page of for a registered user.
- *edit_profile* - Renders the form that enables users to edit their profile.
- *logout* - Logouts a logged-in user.

- *login* - Renders a login page and allows users to log-in to the system.
- *register* - Renders the registration page and allows users to register to the system.
- *add_user* - Renders the registration page and allows the admin to register new users to the system.
- *edit_user* - Renders the profile edit page and allows the admin to edit user's profiles in the system.
- *remove_user* - Allows the admin to remove a user profile from the system.



Profile

Name:	admin
Email:	admin@admin.cz
Phone:	123456678
User type:	admin

Figure C.6: Final design of the user management pages

An example of the profile page can be seen in figure C.6. These pages allow users and admins to manage user accounts. They are all based on basic forms where some fields are mandatory and some optional. The e-mail assigned to the user's account is used for notification when the video uploading, video reformatting or rule violation analysis is finished. Users can only edit their profile while admins can manage every account.