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

SURVEILLANCE VIDEO SEARCH

VYHLEDÁVÁNÍ V ZÁZNAMECH BEZPEČNOSTNÍCH KAMER

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **LUKÁŠ PIWOWARSKI**
AUTOR PRÁCE

SUPERVISOR **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM)

Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Piwowarski Lukáš**
Programme: Information Technology
Title: **Surveillance Video Search**
Category: Artificial Intelligence

Assignment:

1. Analyze existing approaches to surveillance video processing and methods for recognizing basic characteristics of people and vehicle movements (their speed and direction).
2. Survey relevant datasets usable for continuous testing of results.
3. Design and realize a system for efficient search in surveillance video through semantic queries (for example, relating the trajectory to the monitored area).
4. Run and evaluate experiments on representative data.
5. Create a poster presenting your work, its goals and results.

Recommended literature:

- Raval, R. M., Prajapati, H. B., & Dabhi, V. K. (2019). Survey and analysis of human activity recognition in surveillance videos. *Intelligent Decision Technologies*, *13*(2), 271-294.

Requirements for the first semester:

- A prototype of the search system

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

Supervisor: **Smrž Pavel, doc. RNDr., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: May 28, 2020
Approval date: March 31, 2020

Abstract

With the growing number of video recordings produced by security cameras, the demand for systems that are able to search them is growing as well. This work examines such systems and methods that are behind them.

The introduction of this thesis describes the scheme of surveillance video search systems together with the methods that these systems use to store information they obtain during the video analysis. Algorithms for object detection (YOLO) and object tracking (DeepSort) are also introduced. These algorithms are then used in a system created for the practical part of this thesis.

The end of the thesis describes the created system, which uses the trajectories of detected objects in searched video recordings. To specify the searched events, the proposed query language within this work is used. This language consists of so-called search blocks, the composition of which can be used to define situations such as: "a person got out of a car" or "a car stopped in a parking space".

Abstrakt

S rostoucím množstvím video záznamů produkovaných bezpečnostními kamerami roste poptávka po systémech, které jsou schopné je prohledávat. Tato práce zkoumá právě takové systémy a metody, které za nimi stojí.

V úvodu práce je popsáno schéma systémů pro vyhledávání záznamů z bezpečnostních kamer společně s metodami, které tyto systémy používají pro uchovávání informací, které získávají při analýze videí. Představeny jsou také algoritmy pro trasování objektů (DeepSort) a detekci objektů (YOLO), které jsou využity v systému vytvořeného v rámci této práce.

Závěr práce je věnován vytvořenému systému, který využívá trajektorií detekovaných objektů v prohledávaných záznamech. K specifikování hledaných událostí systém využívá v rámci této práce navrženého dotazovacího jazyka. Tento jazyk se skládá z tzv. vyhledávacích bloků, jejichž skládáním mohou být definovány situace jako: „člověk vystoupil z auta“ nebo „auto zastavilo na parkovacím místě“.

Keywords

surveillance video search, object detection, object tracking, object trajectory-based video search, homography

Klíčová slova

vyhledávání v záznamech bezpečnostních kamer, detekce objektů, trasování objektů, vyhledávání objektů podle jejich trajektorie, homografie

Reference

PIWOWARSKI, Lukáš. *Surveillance Video Search.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. RNDr. Pavel Smrž, Ph.D.

Rozšířený abstrakt

Každým rokem přibývají na světě nové bezpečnostní kamery. S tím, jak roste jejich počet, roste také poptávka po systémech, které jsou schopné efektivně prohledávat velké množství záznamů vyprodukované těmito kamerami. Tyto systémy jsou vyhledávány jak soukromým sektorem, tak policejními složkami. Čas, který pak tyto systémy ušetří, může být efektivně využíván například právě policejními složkami k vyšetření většího množství kriminálních případů či přestupků. Text této bakalářské práce představuje právě takové systémy a jednotlivé části, ze kterých se skládají.

Existuje více přístupů k prohledávání kamerových záznamů, ale obecně se schéma systémů pro prohledávání kamerových záznamů napříč těmito přístupy příliš neliší. Lze proto představit tři hlavní pilíře, které se nacházejí u většiny těchto systémů. Každý z těchto pilířů je také detailně popsán v textu této práce.

První z nich je databáze, která uchovává informace extrahované z analyzovaného videa. Mezi takové informace lze zařadit například polohu detekovaných objektů nebo také jejich barvu či tvar.

Další neméně důležitou části je analyzátor videa, který je zodpovědný za získávání informací, které jsou pak uchovávány v databázi. Tato práce se zaměřuje na vytvoření systémů, který využívá informace o pohybu objektů detekovaných ve videu (člověk nebo automobil). Proto jsou v práci popsány principy detektorů objektů a algoritmů pro trasování objektů.

Poslední klíčovou částí je uživatelské rozhraní, které uživatel využívá pro specifikování hledaných objektů či událostí. Uživatelské rozhraní může být přizpůsobeno prostředí, pro které byl systém vytvořen (rozhraní pro vyhledávání automobilů bude jiné než rozhraní pro vyhledávání lidí) nebo může nabývat obecnějších rozměrů, tak aby bylo možné jeho nasazení v různých prostředích. Narazit tak můžeme například na systém, ve kterém specifikujeme hledanou událost pomocí kreslení čar určující pohyb hledaných objektů nebo také systémy, ve kterých můžeme specifikovat hledanou událost za pomocí přirozeného jazyka. Způsob prohledávání extrahovaných informací však zůstává stejný.

V druhé části práce je popsán systém, který byl implementován v rámci praktické části. Jedná se o systém, který je schopen prohledávat kamerové záznamy. Systém je založen na sledování trajektorií pohybu jednotlivých objektů v záznamech. Informace o polohách detekovaných objektů v čase je uchovávána v databázi, která je pak následně prohledávána.

Pro specifikování hledaných událostí či objektů byl vytvořen jednoduchý dotazovací jazyk, která se skládá z tzv. vyhledávacích bloků. Skládáním vyhledávacích bloků pak uživatel může definovat události jako je například: „člověk vystoupil z automobilu". Takový požadavek na vyhledávání může být vytvořen složením tří bloků: „objekt", „objevit se", „do vzdálenosti od objektu". Výsledný dotaz na vyhledávání po tom, co jsou jednotlivé vyhledávací bloky naplněny dodatečnými informacemi, vypadá takto: „Osoba se objevila ve vzdálenosti do 3 metrů od automobilu". Jakmile systém obdrží tento dotaz na vyhledání jsou z databáze vybrány kandidátní objekty typu „osoba". Tyto kandidátní objekty jsou pak následně testovány, zda se některý z nich nenacházel ve vzdálenosti do 3 metrů od objektu typu automobil.

Testováním bylo zjištěno, že systém je schopen kromě výše zmíněného příkladu nalézt situace jako „automobil přijel na parkoviště" či „člověk vstoupil do budovy" nebo také „automobil zastavil na konkrétním parkovacím místě". Jazyk pro vytváření dotazů je postaven tak, aby bylo možné do budoucna přidávat nové vyhledávací bloky. Proto by mohly přibýt například blok pro specifikování dominantní barvy objektu nebo případně jeho výšky a další.

Surveillance Video Search

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Doc. RNDr. Pavel Smrž, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Lukáš Piwowarski
May 28, 2020

Acknowledgements

I would like to thank my supervisor Doc. RNDr. Pavel Smrž, Ph.D. for his guidance and willingness. Also, I would like to thank my family and friends who supported me and helped me to review this thesis.

Contents

Chapter 1

Introduction

Searching a large number of video recordings from security cameras that are produced every day requires systems that are able to search them effectively. Such systems use different methods for extracting information from a video, for storing that information and subsequently for searching through that extracted information to retrieve searched events. This work aims to examine these different methods and to explain processes which are behind the surveillance video search systems.

Chapter 2 of this thesis explains what surveillance video search systems are and briefly describes their architecture. Also, existing programs which are nowadays used for surveillance video search are introduced. The end of the chapter gives insight into different approaches that these systems use for storing extracted information from video recordings.

In the following chapter 3, the principle of operation of object detectors is explained as they play an important role in many video search systems. Methods for object detections such as the Viola-Jones algorithm, HOG features based object detection or object detection using convolutional neural networks are also discussed there.

The homography matrix and its function in the calculation of object's position in the ground plane is explained in chapter 4. Chapter 5 gives a theoretical basis for the description of the functioning of object tracking algorithms, especially for the DeepSort algorithm that is used in the program created for the practical part of this thesis.

Chapter 6 describes the operation of object tracking algorithms. In the beginning of this chapter, a basic overview of object tracking algorithms is given. The rest of the chapter introduces the reader to object tracking algorithms such as DeepSort, TLD or GOTURN.

In the second part of the thesis, the implementation of the system for surveillance video search is described. The system enables the user to specify searched events such as "a person got out of a car" or "object moved with a certain speed in a certain direction". The search is based on the idea of basic search blocks which, when combined, enable the user to create more complicated queries or to define their own search blocks.

Chapter 2

Surveillance video search

This section deals with the research of existing surveillance video search systems. At the beginning, a brief overview of a scheme of such systems is presented. Further different approaches in surveillance video search are presented, and the end of this chapter gives insight into the current possibilities of existing surveillance video search systems.

2.1 What is surveillance video search?

As every day the number of newly installed security cameras grows so does the amount of created video recordings by these cameras and demand for systems that would be able to search through this vast number of recordings. Such systems are sought by both police and private sector with the main goal of saving time spent by people searching through recordings from security cameras. This time can be used more effectively and in the case of the police, this may lead to solving more crimes in a shorter time [2].

Systems for surveillance video search differ in the way in which they work and complexity of queries they allow. Therefore, we can find systems that are only capable of finding anomalies, such as motion in an area was detected but the type of the object remains unknown, through systems that are capable of finding situations when a person moved from place A to place B to the most complex ones which are capable of finding a person with red t-shirt who ran in a specific area with a specific direction of movement [11] [33] [18].

It is obvious that user requirements differ by the type of recordings they want to analyze. For example, a user with recordings from a highway will look for a different system than a person who is interested in analyzing video recordings from a mall. In the case of these two users, an interface of the system which they use is the thing that differs, but the way in which the information extracted from the recordings are saved is same for both systems. When it comes to the highway recordings, a colour of cars, license plate or potentially the type of car could be the main objects of the search. On the contrary, when it comes to the mall recordings, a person's height, age or sex could be the main objects of the search. Therefore the database of these different systems will contain different information, but the search method remains the same.

We can say that systems for surveillance video search are such systems which enable extraction of information from video recordings, storing it in a database and allowing the user to search through the database effectively to find events or objects of interest.

2.2 General scheme of video surveillance systems

There are several types of systems for surveillance video search, but their topology does not vary dramatically. Therefore it is possible to present one general system which would represent all of them (figure 2.1). Such a system would contain these parts:

- Camera
- Video analyzer
- Database of extracted information
- Video database
- Interface for searching

Camera is the most crucial element because the quality of the camera recordings we can give to our system influences the final functioning of the system. The higher the quality, the more information can be extracted from the video. The quality of the recording is determined by its resolution, whether the footage is grey-scale or colourful. A codec used by the camera also plays a significat role [29]. Because the amount of produced recordings can be high, a balance between the compression ratio of the codec and the resulting recording quality must be found. Today the H.264 codec is used by the majority of cameras [34].

A Video analyzer is another element which can be found in surveillance video search systems. Its primary purpose is to extract information from the recordings. It may be an object detector (chapter 3) together with an object tracker (chapter 6). When an object detector detects an object other features of the object might be extracted such as its colour or shape. This extracted information is then stored in a database of extracted information. It is also possible that while the recording is being analyzed the events of interest might be detected during this process and stored in the database, too.

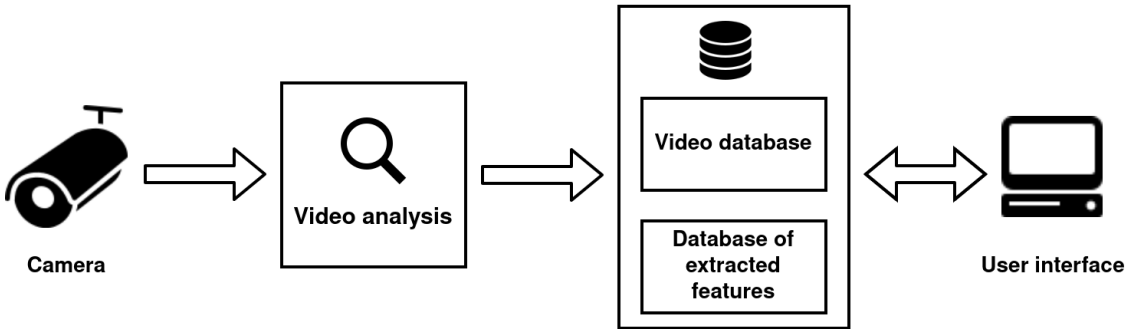

Figure 2.1: General scheme of surveillance video search systems.

We can divide surveillance video search systems according to the approach which they use during the analysis of a video into two main groups. The first group includes systems that detect events in a short time after their formation. This requires the user to specify events of interest before the start of the analysis and speed of analysis of each video frame under $1/fps$. This is often called *a real-time alerting* [26]. Systems which need more time to analyze a video belong to the other group. These systems might be slower but on the other hand, they may provide a more in-depth analysis of the recording.

There is a wide variety of **databases that are used for storing extracted information** from the recordings. The database used for the storing of extracted information reflects the approach that is used for searching through the database. We can come across systems which use a relational database [26] but also a simple plain text file [14].

Video database is used for storing video recordings, which are referred to by detected events or objects stored in the database of extracted information. When searched event or object is found in the database of extracted information, then the video containing the searched event or object is retrieved from the video database and presented to the user.

Interface for the search is used for specifying events or objects which the user wants to find. Interface may take various forms depending on the use case of the system. We can find systems where the user specifies the searched event using a query language [33]. However, there are also systems where the user draws lines to the video frame to specify searched event or object [5].

2.3 Examples of video search systems

In this section existing surveillance video search systems are presented.

2.3.1 IMB Video Analytics

IBM Video Analytics [16] offers offline video search but also real-time event detection. In the case of offline video search, it offers a search of people or cars. It is possible to specify different attributes of the searched objects such as colour, age of a person, the colour of skin, or whether a person is wearing glasses or not. As an example, a situation when the user tries to search for a person wearing a red t-shirt may be used. All that must be done is to specify object type, which is in this case person and then the dominant colour of the upper part of the person's body, which is red. When the search button is pressed a set of video recordings is displayed sorted by the probability of satisfaction of the query.

As mentioned above, it is also possible to specify rules for real-time event detection. When the rule is satisfied, then the user is alerted that it happened so. For example, if a user wants to be warned about a big number of people piling up in a specific area, the following steps should be made:

1. New rule must be created.
2. Camera capturing the event must be specified.
3. Region of interest must be selected.
4. Constant called Motion area must be set.

The constant Motion area specifies the number of pixels in the region of interest at which a motion is detected (figure 2.2). This way of definition is used because it is more effective when it comes to videos with lower resolution. Also, it does not require such computational power to detect motion at pixel as it does to detect a person.

In a similar way as in the previous two examples, it is possible to define rules for the detection of long lines at a cashier or analyse traffic on a highway.

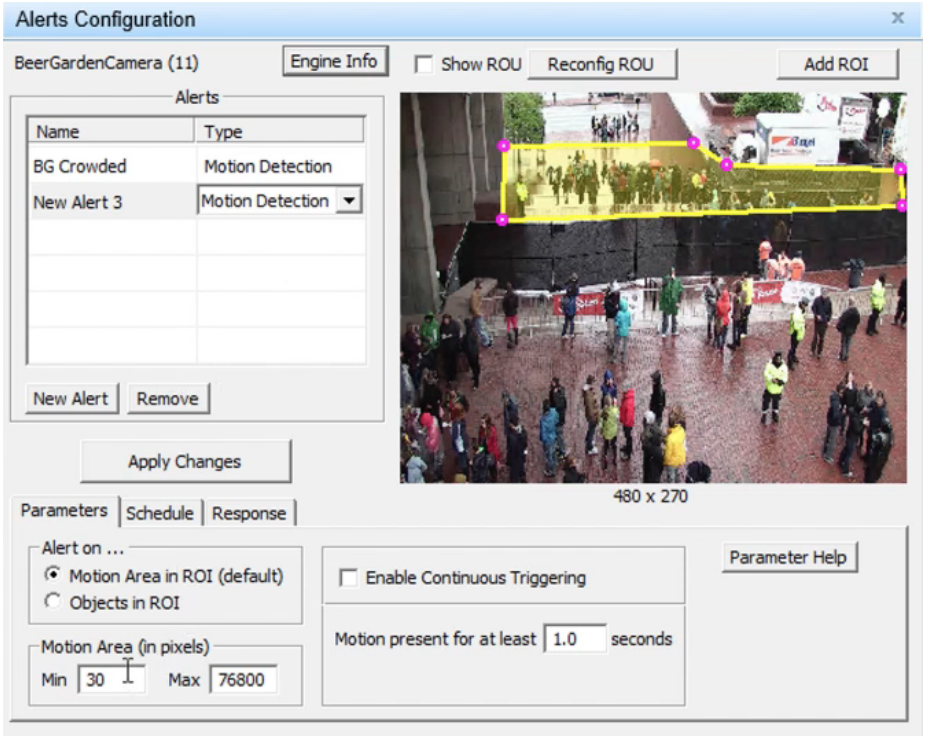

Figure 2.2: Crowd search in IBM Video Analytics. [40]

2.3.2 Axxonsoft

Axxonsoft [5] is a system which offers real-time detection of events specified by the user and offline video search. An example of using this system is a search when we try to find a car leaving a parking lot. If we have one exit from the parking lot, as it is shown in figure 2.3, then we need to specify a line and direction that determines in what way the object (car) should cross the line. Similarly, the system offers a search of parts of the videos in which a certain amount of people is located in a specific area or parts of the videos in which an object moved from place X to place Y.

Noteworthy is also a feature that cleverly allows the user to show in one video all moving objects which were recorded at different times. When the user clicks on a specific object, the part of the video with the object is then presented.

Another capability which this system offers is to search for a person by face. If a user owns a picture with the searched person, then all that must be done to find the person is to upload the image to the system, and after that, videos containing the person are listed.

In the case of real-time event detection, the user can specify situations such as:

1. Object entered into an area
2. Object crossed over a line
3. Lingering object
4. Left luggage

Figure 2.3: Search of a car leaving or entering a parking lot. [4]

2.3.3 Camio

Camio [10] differs from the two systems mentioned above. It is unique with its user interface and its simplicity of usage, which may reduce the time needed for learning how to work with such a system but at the cost of reducing the space of possible queries. To start to use this system, the user must connect a camera to the software produced by the company, which then sends video footage to the cloud for analysis if a camera captures a movement.

To search for an event, the user writes a sentence containing the specifics they are looking for. For example, the system allows the following query: "people in red or blue Liverpool". This query lists all videos in which a person wearing red or blue clothes was captured on camera which the user named Liverpool.

Camio system also enables detection of situations called tailgating. It is a situation when two people enter a guarded building or room of which only one uses a security card for opening a door. For this purpose, ground plane homography is calculated for tracking people in 3D space. If two people appear nearby the door, move away from it, and at the same time, one security card is used, then tailgating is detected, and an owner of the system is alerted.

2.4 Approaches in storing video events

In this section, three different approaches in storing found events in videos are presented.

2.4.1 Storing events in tables

One of the most straightforward approaches in storing events extracted from videos is to store them in a table. For this purpose, relational databases can be used together with existing popular query languages. However, there are also query languages specially created for surveillance video search.

For example, the system mentioned in work of Hampapur et al. [26] is based on obtaining an object's position from an object detector and object tracker. When a new object is detected, then its location is tracked across the frames, and along with it other features of

the object are calculated such as colour, size or speed. Then when the object disappears from the field of view, a new record in the database is created (figure 2.4).

Camera ID:	23
Unique event ID:	2345678
Start:	2020-09-03T03:04:22Z
End:	2020-09-03T05:24:42Z
Keyframe:	234567.jpg
Video:	/videos/parking_lot.wmv
Object type:	Car
Additional fields:	Trajectory, color, ...

Figure 2.4: Example of an object record in a database. [26]

Later when a user wants to search for a specific part of the video they can search for it using a query that is similar to the following one:

```
FIND ALL, Object Type="VEHICLE", Object Size > X1 Object Size < X2,
Object Color=Yellow, Object Speed > S1
```

Such a query searches for all vehicles whose size is in the range $(x1, x2)$, have a yellow colour and speed higher than $S1$.

Another way of storing events and objects captured in a video is to use two tables, one for events and another one for objects, as shown in [33]. Each event is defined by id, name, contained objects, and time interval. An object is defined by id, type, position and bounding box sizes. For the video retrieval SQL like language is used (VSQL) which is demonstrated in the following two examples:

```
SELECT video-frames FROM * WHERE ((p: PERSON) AND (i: SubImage) AND
(i color-similarity p))
```

This query would return parts of videos in which a person was captured and whose colour histogram is similar to a colour histogram of a given image (SubImage).

```
SELECT video-frames FROM Video-Database WHERE ((e: Events) AND
(e's Name = "Close-to-gate"))
```

would return parts of videos from video database named „Video-database“ where objects appeared nearby a gate.

2.4.2 Indexing events using spatio-temporal and-or graphs

Approach described in this section is inspired by the article “Semantic video event search for surveillance video” by Tae Eu Choe et al. [14]. It is based on searching through spatio-temporal and-or graphs produced for each analyzed video.

A few steps are involved before the production of the graph. First primitive data – set of records of detected objects, are obtained from a video analyzer. Each record contains the location of an object, size of the bounding box and other. These data are passed to the semantic analyzer whose main purpose is to detect primitive events associated with an object such as:

- Object stopped
- Object started to move
- Object accelerated/decelerated

Objects and events obtained from the semantic and the video analyzer are then arranged into spatio-temporal and-or-graph (figure 2.5). A spatio-temporal graph is a graph with nodes whose location in space takes on meaning [17]. If we let $G = (V, E)$ be a graph where V is set of vertices and E be set of edges then we can define a spatio-temporal graph as $G_s = (V_s, E_s)$ where E_s is set of edges and V_s be set of triads (x_n, x_t, x_s) where x_n is notation of a node, x_t are temporal coordinates of the node and x_s are spatial coordinates of the node.

As shown in the simplified figure 2.5 where x-axis would denote temporal coordinates and y-axis would denote space coordinates. *Human disappeared* event is within the time close to the *vehicle start-to-move* event. On the contrary, *vehicle stopped* event is within the time far from the *vehicle start-to-move* event.

Such a graph is then used for the creation of a graph that encompasses relationships between nodes in the spatio-temporal and-or graph. The nodes of this graph represent events and objects and vertices define the relationship between them. If we want later to find an event in the video, we have to find a matching graph for this event in our database. That is if we want to find an event in the database, we have to first transform the query from its original form to the graph and then find matching one in our database.

The problem of finding the graph in the database is solved as follows. The graph which is produced from the spatio-temporal and-or graph is divided into set of subgraphs that are stored in a lookup table. When a new query is created, then a graph representing this query is created together with the subgraphs of which it is composed. These subgraphs are then used for searching in the lookup table.

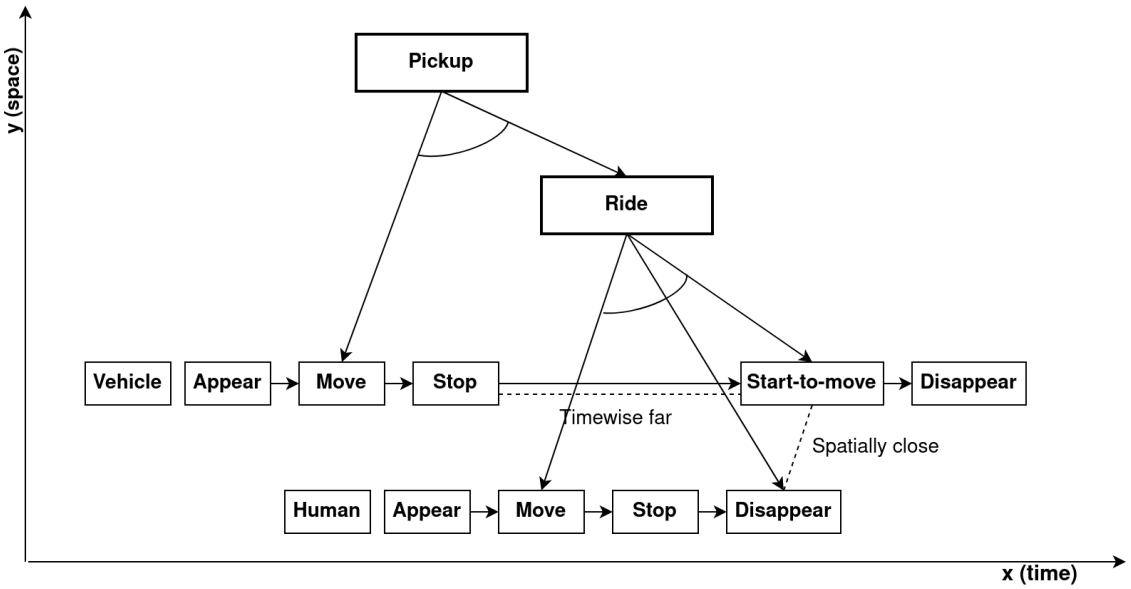

Figure 2.5: Example of a spatio-temporal and-or graph. [14]

2.4.3 Indexing events using feature trees

This section introduces a search approach for the videos mentioned in [11]. The approach is based on so-called feature trees that are created for each part of an analyzed video. The analyzed video is divided into a series of blocks called *documents* where each *document* can contain from 30 to 100 frames of the recording. Each such a block is subdivided into other sub-blocks called *atoms* from which the features of interest are extracted (figure 2.6). Feature of interest can be colour, direction of movement, shape and other.

The *atoms* of each *document* are sorted into a tree structure in which each node has four children. The value stored in a non-leaf node is the result of the aggregated values of its children. We can write that the value of a non-leaf node is given by the aggregate function:

$$\vec{x}_f^{(i)} = \psi_f\left(\vec{x}_f^{(1)}, \vec{x}_f^{(2)}, \vec{x}_f^{(3)}, \vec{x}_f^{(4)}\right), \tag{2.1}$$

where ψ_f is an aggregate function and $\vec{x}_f^{(n)}$ are either vectors of extracted features or already aggregated vectors from the previous level of the tree. The resulting tree is then saved to the database and index of feature trees using locality sensitive hashing is created. Locality sensitive hashing (LSH) is similar to standard hashing, except that with standard hashing, values that are close to each other before hashing may get assigned hash values that are far apart. Using LSH it holds that if we have two keys k_1 and k_2 whose similarity is high and $H(k_1)$ and $H(k_2)$ are their hash values then $|H(k_1) - H(k_2)|$ is low and vice versa if the similarity of the keys k_1 and k_2 is very low then $|H(k_1) - H(k_2)|$ is high [25].

When searching for events, the user specifies a region and features it should contain. From the user query, an empty *document* is then created, which is filled with searched features. It is then converted to a tree using the aggregate function. For that tree, a hash value is calculated using LSH, which points to the part of the database in which similar trees are found. This space of the database is finally searched for resulting videos.

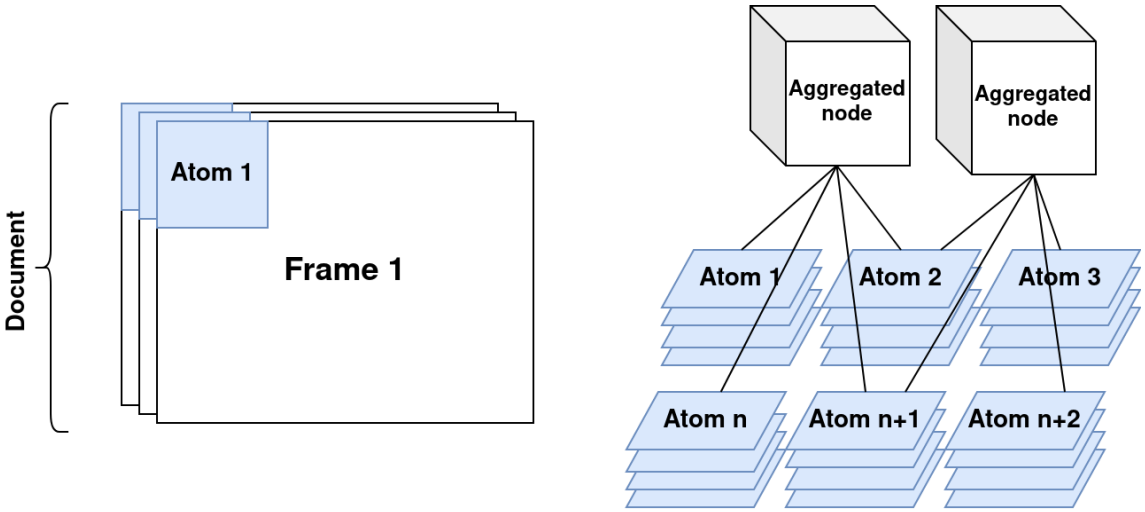

Figure 2.6: Example of storing extracted features from a video into a feature tree. [11]

Chapter 3

Object detection

This section explains what object detection is and gives an overview of the basic object detection approaches.

3.1 Introduction to object detection

Object detection is a process of finding objects in an image belonging to some class C [23]. The system that performs detection usually scans the image first and proposes candidate regions in which the searched objects could be located. These regions can be found by various methods. For example, a simple sliding window algorithm can be used when we move rectangles of different sizes across the image. This is of course enormously time-consuming therefore different methods are used such as selective search that iteratively groups together regions with similar features (colour, texture, size) [46]. For each proposed region, the system for object detection checks whether it contains the object of class C or not (image classification).

The image classification can be done by extracting some features (colour, shape, ...) from the proposed region. Using these features, the probability that the region contains an object belonging to searched class C is calculated. The resulting probability is compared to a threshold. If it is higher than the threshold, then the detection is evaluated as successful. The system for image classification needs to be trained to be able to find a boundary between objects belonging to class C and objects which do not.

The training data is needed for training the system. It consists of a set of training images I containing two sub-sets. The O_c set that contains training data containing objects from the class we want to detect and the O_w set, which is the complement of O_c.

Then we try to detect objects from class C in training images I. Based on the results of the detection, the system may be modified to minimize false-positive errors and false-negative errors. False-positive error is an error when the detected object $o \notin C$ and vice versa false-negative error is an error when the rejected object $o \in C$ (figure 3.1). It may be useful to minimize different types of errors for different uses. For example, for weapon detection at an airport, we will prefer greater false-positive error than false-negative error.

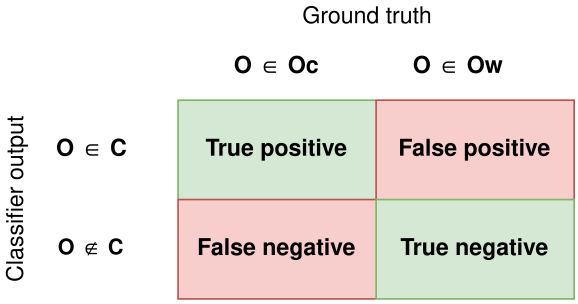

Figure 3.1: Error types.

Finally, when we have trained a classifier which meets our requirements, we can use it to detect objects in an image.

3.2 Evaluating performance of an object detection algorithm

An object detector's output is a position of an object in an analyzed image. The output may take various forms. We can come across polygon bounding box or, most often, a rectangular bounding box that is defined by the coordinates of its two corners.

When working with detectors, it is necessary to evaluate how well the detection system works. To evaluate how accurately an object is detected, IOU is used, which stands for Intersection Over Union and can be calculated as follows:

$$IOU = \frac{area\ of\ overlap}{area\ of\ union}, \tag{3.1}$$

where the *area of overlap* is an area of overlap of the bounding box obtained from the object detector with the bounding box from the training data and *area of union* is a unified area of both the bounding box from the detector and the bounding box from the training data [23].

If we want to have a more global view of how good an object detector is, we look at a metric called mAP (mean Average Precision) [28]. To calculate mAP we need to know how the precision and recall values for given object detector are calculated:

$$Precision = \frac{True\ positives}{True\ positives + False\ positives} \tag{3.2}$$

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives} \tag{3.3}$$

The mAP is then calculated by testing our system with data from a training set, where the detection success is defined according to some IOU threshold, e.g., 0.5. We will sort the detections according to their confidence level. For each record in the table, we calculate the precision and recall values similarly, as shown in figure 3.2. If we plot the table into a graph and calculate integral for such a graph, we get *AP@0.5*. For calculation of mAP we calculate an average of APs for different IOU levels [28].

Confidence	Correct detection	Precision	Recall
1	Correct	1 / 1 = 1	1 / 5 = 0.2
0.98	Correct	2 / 2 = 1	2 / 5 = 0.4
0.97	False	2 / 3 = 0.67	2 / 5 = 0.4
.	.	.	.
.	.	.	.
.	.	.	.

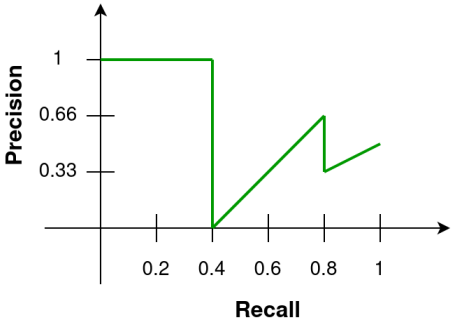

Figure 3.2: Example of calculation of AP for object detection algorithm. [28]

3.3 The Viola-Jones algorithm

This section introduces the Viola-Jones algorithm [47]. It was the first algorithm which in its time was capable of detecting faces in videos with relatively reasonable speed of 15 fps. It is possible to use the algorithm to detect any object, but it is mostly used for face detection.

The algorithm is based on the idea that some parts of a face are lighter than others. If we learn which areas of the face are lighter or darker during the training of a system for face detection, then we can use this knowledge later to find faces in images. The algorithm looks for Haar-like features. These are found by placing rectangles from figure 3.3 on the analyzed image. After the rectangle is placed, values of pixels under the white area are added together and subtracted from the values of pixels under the black area. Depending on the resulting value, we then decide whether the feature is in the image or not. Calculating values for a large number of Haar-like features in this way is unnecessarily computationally expensive. Therefore, an integral-image is introduced (figure 3.3).

20	30	10
5	20	10
5	15	5

Image

20	50	60
25	75	95
30	95	120

Integral image

Haar-like features

Figure 3.3: Example of integral image calculation ($ii(x,y) = \sum_{x'<x,y'<y} i(x,y)$) and Haar-like features.

Using the integral image, it is necessary to add or subtract a maximum of four numbers to calculate the sum of any area in the analyzed image. This greatly speeds up the process of detecting Haar-like features.

Thus, in order to detect faces, we need to find a set of Haar-like features that is present in the majority of faces, and that best separates images with faces from images without faces. To obtain such a set, the AdaBoost algorithm is utilized.

The AdaBoost algorithm works so that each image from the training set receives a weight w_i. Subsequently, so-called weak classifiers are trained on the training set for each possible Haar-like feature. From all the trained classifiers, one with the lowest error rate is selected.

The error rate is calculated as:

$$e = \sum_i w_i |h(x) - c_i| \tag{3.4}$$

Where w_i is a weight of an image, $h(x)$ is the result of weak classifiers which ranges from 0 to 1, and c_i is 1 if the image contains a face or 0 if the image does not contain a face. Then the weights of the images are adjusted so that the weights of images that the selected classifier detects poorly are increased, and the weights of images that the selected classifier detects correctly are decreased. We repeat this process until we have enough classifiers (weak classifiers). From these weak classifiers, we then assemble a strong classifier.

Face detection is performed by passing an image through a series of strong classifiers. If all of them confirm that the image contains a face, then the detection is successful.

3.4 Histogram of Oriented Gradients based object detection

Histogram of Oriented Gradients (HOG) is a feature descriptor that represents an image as a vector using its colours of pixels. The vector can be then passed to a classifier, which decides whether the image contains an object or not. The following section introduces how this vector is extracted from a grayscale image and outlines the operation of a support vector machine (SVM) that uses the HOG feature to determine whether or not the image contains an object.

Histogram of Oriented Gradients splits the image into 8 x 8 blocks. For each pixel in the block, two values are calculated - direction and magnitude. To calculate these values it is necessary to know the concept of pixel gradient. The gradient together with direction and magnitude, is calculated as follows:

$$\nabla g(x,y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} f(x+1,y) - f(x-1,y) \\ f(x,y+1) - f(x,y-1) \end{bmatrix} \tag{3.5}$$

$$m(x,y) = \sqrt{g_x^2 + g_y^2} \tag{3.6}$$

$$d(x,y) = tan^{-1}(\frac{g_y}{g_x}) \tag{3.7}$$

Where $\nabla g(x,y)$ is the gradient, $m(x,y)$ is the magnitude and $d(x,y)$ is the direction of a pixel. After calculating the direction and magnitude for each pixel in the 8 x 8 block, we get the matrix of directions and magnitudes for given block. The block is then represented by one vector, which is created by summing the magnitudes of all pixels with a direction in a certain range. This means adding up all the magnitudes separately for pixels with a direction in the range of 0 - 20 degrees, 20 - 30 degrees,..., 160 - 180 degrees[1] [39]. These vectors for each block are then combined into a single vector that represents the entire image.

A trained **Support Vector Machine** takes the acquired HOG feature vector and decides whether or not the object is in the image. Training of SVM starts by obtaining HOG vectors for images from the training data set. These are projected into space, in which we try to find a hyperplane that separates vectors that represent images with the object we want to detect from vectors that do not represent images with the object we want to detect.

[1] when we calculate HOG feature, we want to have a direction in the range of 0-180 degrees, therefore the direction is calculated with values ($|g_y|$ a $|g_x|$)

The resulting hyperplane must separate the data so that the margin that is displayed in figure 3.4 is as large as possible [13].

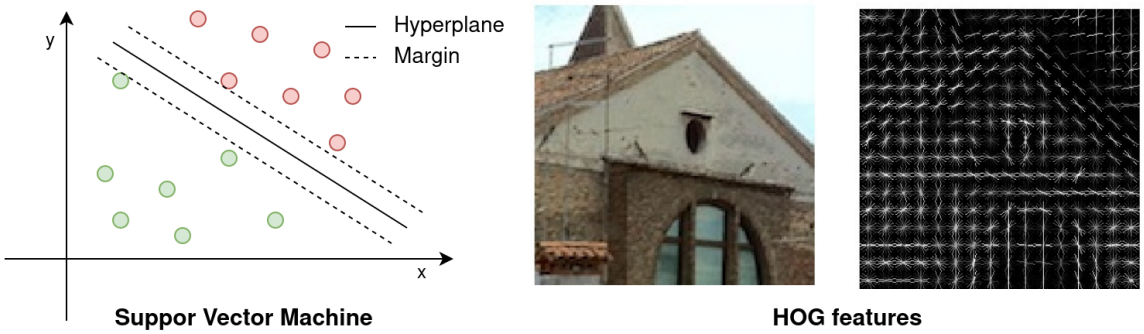

Figure 3.4: Example of Support Vector Machine and HOG features. [3]

3.5 Scale invariant feature transform based object detection

HOG features from section 3.4 have an issue with detection of objects at different angles and objects that are scaled differently. This problem can be solved by using so-called scale-invariant features [44], which are, as the name suggests, scale-invariant and also rotation invariant. For finding these features SIFT is utilized.

The main idea of SIFT is finding so-called keypoints. Each keypoint is represented as a vector in such a way that when detecting an object with the same keypoint but at a different angle or different scale, we get the same vector.

The first step of calculating a feature for an image using SIFT is, therefore, finding candidate keypoints. For this purpose, the image is blurred several times using Gaussian function each time with different variance. Then we create an image which is half of the size of the previous image, and the process of blurring is repeated (this happens several times). The result of this process is scale space. Using pictures from this space, difference of Gaussian (DOG) images are created. The images are created by subtracting from each other two adjacent images (with different blurring but same sizes) in the scale space. In search of the keypoints, these DOG images are utilized.

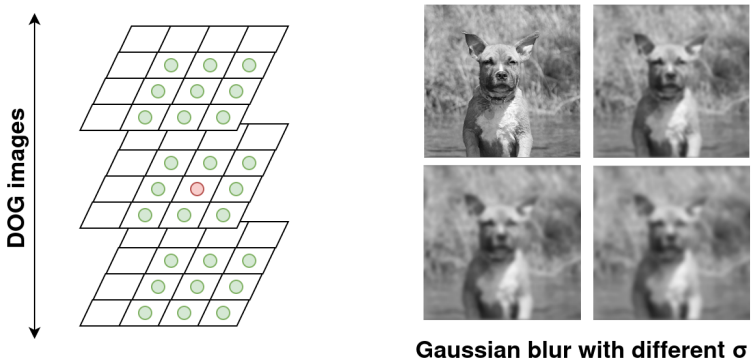

Figure 3.5: Example of keypoint (red dot) with its surrounding (green dots) and Gaussian blur.

Candidate keypoints are found among DOG images by evaluating each pixel in the DOG image. If a pixel is evaluated as a pixel with the lowest or highest value in its vicinity, then it is selected as a candidate keypoint. Vicinity of a pixel is defined as the unification of three areas - an area defined by 3 x 3 square surrounding the evaluated pixel in given DOG image and two areas defined by 3 x 3 squares that are located on the same positions but in the adjacent DOG images (figure 3.5).

Found candidate keypoints then go through the process of elimination in which keypoints whose absolute value is lower than a selected threshold are discarded. So are keypoints that are not located in the "corner". A point is considered to be in a corner if gradients g_x and g_y of this point are high.

If a keypoint passes through the previous step, then it is transformed into a vector. The resulting vector holds information about keypoint's surrounding. To create the vector, magnitudes and directions of surrounding pixels are calculated similarly as in the calculation of HOG feature (section 3.4). However, before the calculated directions and magnitudes are put into the resulting vector, keypoint's direction is subtracted from each calculated direction to make the resulting vector invariant to rotation.

Thus obtained vectors are used for object detection. If we then later encounter an object with keypoints that are represented by a similar vector, we can say that the object we detected is the searched object.

3.6 Using convolutional neural network for object detection

In this section, neural networks and their subclass convolutional neural networks are introduced. For an informal description of **neural networks**, a scheme similar to the one in figure 3.6 is often used. A neural network consists of so-called neurons which are also depicted in figure 3.6. Each neuron has several inputs a_i with weights w_i, output y and bias b. The output of a neuron can be calculated with these parameters as follows[2] [30]:

$$y = \sigma(A^T W + B), A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}, W = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}. \tag{3.8}$$

σ is an activation function, which may be different for various types of neural networks. For example, it is possible to encounter a sigmoid function. If we pass a vector X to a neural network, we then expect the neural network to output another vector or scalar Y. Training of neural network then consists of the modification of weights w_i in such a way that the output Y of the neural network is as close as possible to our expected output for any given input X.

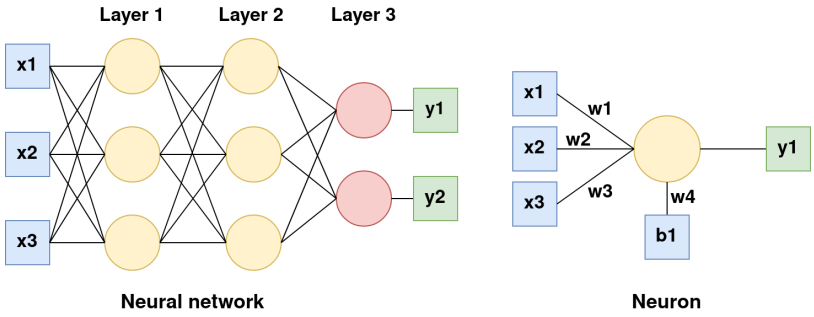

Figure 3.6: Interconnected neurons form a neural network.

[2] Biases can be treated as special neurons with output value 1.

Convolutional neural networks (CNN) are neural networks that have at least one layer which performs convolution (figure 3.7). CNN for object detection takes an tensor as an input which represents the image and outputs a vector. This vector may contain, for example, probabilities of detection of different types of objects. As already indicated, training of a neural network consists of adjusting the values of weights w_i. The weights are set to a random value at the beginning of the training of CNN. When the CNN returns a vector for an image from a training set, we calculate the error for this output using loss function L. Using the error, the weights are adjusted in such a way that the loss function L approaches zero. Formally we can write this as:

$$W_{i+1} = W_i - \eta \nabla L(W), \nabla L(W) = \frac{\partial L}{\partial W}, \tag{3.9}$$

where η is the learning rate and $\nabla L(W)$ is the gradient of $L(W)$ [24]. Learning rate should be set so that it is neither too high nor too low. If η is too big, it is possible that during the training of neural network we would miss the minimum of function $L(W)$, on the other hand, if η is too small, it could take too long to approach the minimum of $L(W)$ [9].

The trained CNN can be used in combination with object proposal algorithm which proposes regions in an image where potential objects of interest may be located, and the CNN then checks whether the region contains an object of interest or not.

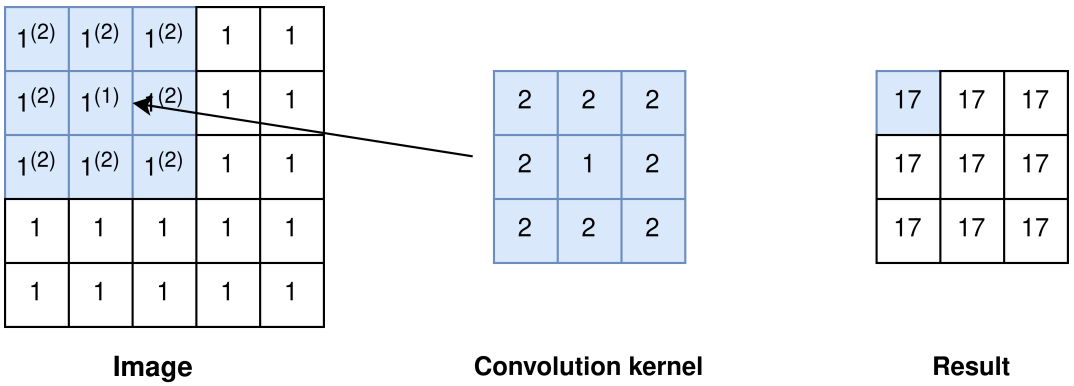

Figure 3.7: Calculation of convolution by placing convolution kernel on an image.

3.7 You Only Look Once

In this section, the YOLO convolutional neural network is discussed [43]. The approach of YOLO differs from the approaches mentioned in the previous sections. The previous approaches searched for potential regions of interest and then ran object classifier, which would decide whether the proposed region contains the searched object or not. YOLO, on the other hand, looks at the whole image at once. Hence the name You Only Look Once is used. This enables an increase of speed for object detection.

YOLO divides the analysed image into $S * S$ grid. Then for each box of the grid, B bounding boxes are predicted with confidence scores P. Where P denotes the probability that the object is contained within the bounding box. The goal is P to be equal to $P(Object) * IOU$. If the bounding box does not contain an object, the P should be equal to zero; otherwise, it should be equal to IOU of the ground truth and the detected bounding box. Also, vector C is assigned to each grid cell which contains probabilities of

occurrence of different types of objects in the bounding box (dog, car, ...). If we combine all the outputs, we conclude that the final output of the CNN is $S * S * (B * 5 + C)$ tensor[3].

Because the number of bounding boxes which proposes the CNN is very high and many of them overlap, it is necessary to remove duplicate bounding boxes. For this, the non-max suppression algorithm is used (algorithm 1). Resulting bounding boxes of non-max suppression are then considered as detected objects whose type is selected from the vector C.

```
Initialise list L;
Remove all bounding boxes with probability p_c lower than threshold t;
while there are any unprocessed bounding boxes do
    B = bounding box with the highest value p_c;
    Put B to list L;
    Remove all bounding boxes that have IOU >= 0.5 with bounding box B;
end
```

Algorithm 1: Non-max suppression

During the training of the neural network, a fairly complicated lost function is used. However, it can be broken down into smaller, simpler sections. Each such a section, simply said, calculates the sum-squared error for each output of the CNN (height, width, coordinates, ...). Sum-squared error is calculated as follows:

$$\sum_{i=1}^{N}(x_i - x_c)^2 \tag{3.10}$$

where in our case x_i is the output of the CNN (e.g., height of the bounding box) and x_c is the ground truth (the correct height according to the ground truth). If the neural network returned wrong height, the sum would be some high number which we would try to lower in the next round of the training.

If we look for an object detector which is fast and quite accurate then YOLO is a good choice. On the contrary, if we do not mind the object detector to take more time to detect an object, then it might be useful to look somewhere else, as the YOLO may have for example issue with detecting small objects.

[3]The prediction of the bounding boxes has five components: coordinates (x,y), width and height (w,h), and confidence score.

Chapter 4

Computer vision and homography

In this section homography matrix and its usage is explained.

4.1 Homography matrix

When tracking objects, we get (x, y) coordinates of objects on the computer screen (image plane). Following the trajectory of the object's movement using only such points would neglect the perspective, therefore it is necessary to transfer the points to a plane that follows the plane on which the objects move (ground plane) - figure 4.1.

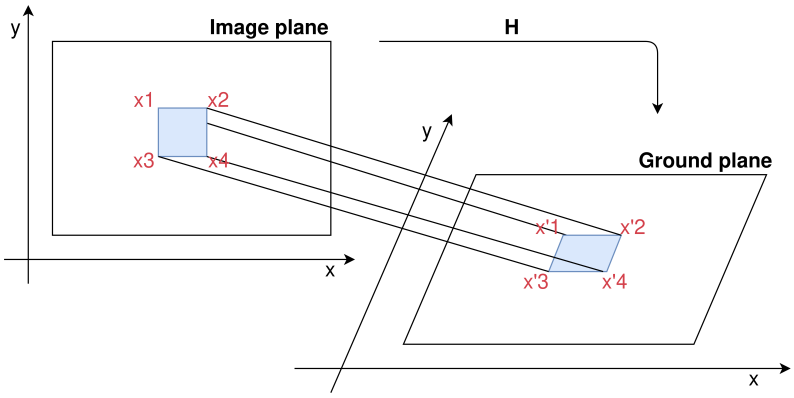

Figure 4.1: Transformation of coordinates from an image plane to a ground plane.

For the transformation from image plane to ground plane a matrix must be found. The matrix which needs to be found is called homography matrix. If there is a square in the image plane, then the homography matrix allows us to map it to any quadrilateral located in the ground plane [38].

The equation for the transformation from an image plane to a ground plane can be written as:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ z \end{bmatrix}, H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \tag{4.1}$$

where $\begin{bmatrix} x' & y' & z' \end{bmatrix}^T$ are homogeneous coordinates in ground plane and $\begin{bmatrix} x & y & z \end{bmatrix}^T$ are homogeneous coordinates in image plane. Thus, for successful remapping we have to find matrix H.

The following equations (4.2, 4.3) apply to the conversion of homogeneous coordinates into Cartesian coordinates: $x_c = x/z$ and $y_c = y/z$. Also we can assume that z is equal to one. Therefore we can calculate ground plane coordinates for any point with coordinates x_n and y_n in an image plane as follows [32]:

$$x'_n = \frac{h_{11}x_n + h_{12}y_n + h_{13}}{h_{31}x_n + h_{32}y_n + h_{33}} \Rightarrow x'_n(h_{31}x_n + h_{32}y_n + h_{33}) = h_{11}x_n + h_{12}y_n + h_{12} \tag{4.2}$$

$$y'_n = \frac{h_{21}x_n + h_{22}y_n + h_{23}}{h_{31}x_n + h_{32}y_n + h_{33}} \Rightarrow y'_n(h_{31}x_1 + h_{32}y_1 + h_{33}) = h_{21}x_1 + h_{22}y_1 + h_{23} \tag{4.3}$$

H is generally normalized with $h_{33} = 1$ [45]. Therefore this equation must be solved to get H for specific transformation.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 & -y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 & -y'_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.4}$$

For finding the solution we must specify eight points (four points that are in the image plane and four points that are their images in the ground plane). If we have more matching points from ground plane and image plane, we can use the least-squares method [32]. If we rewrite the previous equation as:

$$Ah = 0, \tag{4.5}$$

then we would try to find $min||Ah - 0||^2$. If we want even more precise and stable method, the RANSAC can be used, which eliminates anomalies between picked points for homography estimation [19].

4.2 Estimating homography matrix

We need at least eight points to derive the homography matrix, as shown in the previous section. Ideally, to get these eight points, we would use a reference object placed on the ground plane. Knowing the shape or dimensions of the object, we would then get four points in the image plane and four corresponding points in the ground plane. If the coordinates of the blue square in figure 4.1 were $x_1 = (100, 100)$, $x_2 = (150, 100)$, $x_3 = (100, 50)$, $x_4 = (150, 50)$, then we would determine ground plane coordinates from the knowledge of shape of a square as $x'_1 = (0, 0)$, $x'_2 = (1, 0)$, $x'_3 = (0, -1)$, $x'_4 = (1, -1)$ [1]. However, we often do not have such a possibility and therefore it is necessary to use other methods.

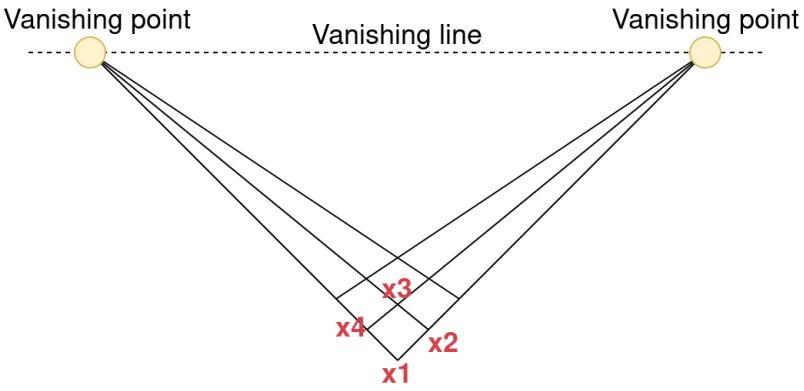

Figure 4.2: Example of vanishing points and vanishing line.

Such methods will vary depending on the use case. For example, if we want to find a homography matrix for cars that move along the highway, we can find two so-called vanishing points (figure 4.2) and use a similar method to the one mentioned in [20]. A vanishing point is a point in the image plane where two lines that are parallel to each other in the ground plane meet in the image plane. Assuming that vehicles move in one direction (i.e. along parallel lines), we can find a vanishing point as the intersection of lines along which analysed vehicles move. The second vanishing point can be found as the intersection of lines perpendicular to the lines on which the vehicles move. The points thus obtained are placed on a line called vanishing line.

Using vanishing points, we find such points x_1, x_2, x_3, x_4, which will be reflected in the ground plane as a square. Knowing the average size of a car, we can then rescale the homography matrix so that the coordinates in the ground plane correspond to the metric system (this can be used for example for calculating the speed of movement of the cars).

Chapter 5

A theoretical basis for object tracking

In this chapter, the theoretical foundations necessary for the explanation of object tracking algorithms (chapter 6) are introduced. Primarily this information is required for explanation of DeepSORT algorithm.

5.1 Kalman filter

The Kalman filter is an efficient algorithm that enables more precise estimation of some unknown variable X based on the knowledge of the measured value of X (M) and the estimated value of X (E). X, M, and E are defined as:

$$X \sim \mathcal{N}(\mu_t, \sigma_t^2) \tag{5.1}$$

$$M \sim \mathcal{N}(\mu_m, \sigma_m^2) \tag{5.2}$$

$$E \sim \mathcal{N}(\mu_e, \sigma_e^2) \tag{5.3}$$

How the filter works can be shown in an example of estimation of the position of a moving vehicle (figure 5.1). In this example, we know the location of the vehicle X_t, and we try to estimate its new location X_{t+1} using:

- **Estimation** (E) of the current position of the vehicle using its speed and direction of movement, which we know from the onboard computer.
- **Measured** (M) position of the vehicle using GPS.

Knowing E and M, we can calculate a new estimate of the position (X_{t+1}) of the car as follows [6]:

$$X_{t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2) \tag{5.4}$$

$$\mu_{t+1} = \mu_e + K_g(\mu_m - \mu_e) \tag{5.5}$$

$$\sigma_{t+1}^2 = (1 - K_g)\sigma_t^2 \tag{5.6}$$

$$K_g = \frac{\sigma_e^2}{\sigma_e^2 + \sigma_m^2} \tag{5.7}$$

Notice that μ_{t+1} will be located somewhere between values μ_e and μ_m. K_g, therefore, defines whether μ_{t+1} is closer to μ_e or μ_m. If both values M and E have the same error σ^2 then the resulting X_{t+1} with μ_{1+1} is located exactly in the middle of μ_m and μ_e.

Newly calculated X_{t+1} is used for estimation of the car's position (μ_{t+1}). If the vehicle moves again, then X_{t+1} will become X_t and the process repeats.

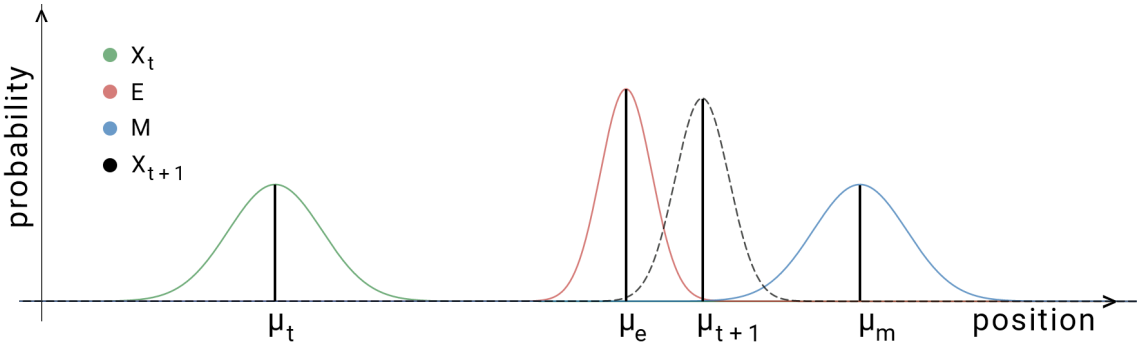

Figure 5.1: Estimation of the car's position in one dimension using Kalman filter.

The Kalman filter is widely used. It can be found in the already mentioned GPS. However, it is also used for an error correction when tracking objects. If we have an object's motion history, then we can predict its future location (E). Then if later an object detector detects the object (M), we can use the Kalman filter to calculate more precise position of the object.

5.2 Mahalanobis distance

The use of Mahalanobis distance can be explained on a problem which arises when we try to measure the distance between a normal distribution and some point x [42]. The first possibility is to measure the distance from the mean of the normal distribution to the point x. If we look at the picture in figure 5.2 we can notice that Euclidean distance for point $x1$ and point $x2$ from the mean of the distribution would be the same. It does not take into consideration how the distribution is spread out.

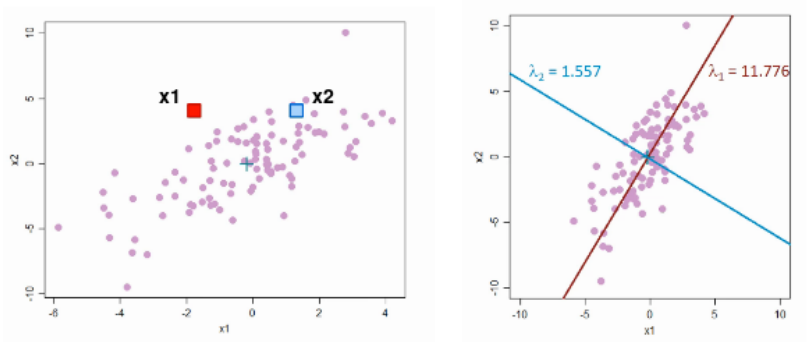

Figure 5.2: Graph of normal distribution and highlighted eigenvectors (red and blue) of its covariance matrix. [15]

In some cases, the Euclidean distance would be enough, but there are some situations in which we would like to take into consideration how the normal distribution is spread

out. The "shape" of a normal distribution is defined by the covariance matrix. We can use the eigenvalues and eigenvectors of the covariance matrix as visualized in figure 5.2 to find direction in which there is the most significant variance. Eigenvalues define how much spread out the distribution is along the eigenvector. If we use the eigenvectors as the axes of a new coordinate system (eigenvalues would define how much the axis is shrunken or stretched), then we can measure the distance of the point from the normal distribution using this new coordinate system. This is the idea behind the Mahalanobis distance.

For its calculation we can use following formula:

$$D(m, v) = \sqrt{(m - v)^T S^{-1} (m - v)} \tag{5.8}$$

Where m is the mean of the normal distribution and v is the point whose distance from the distribution we want to calculate, and S^{-1} is the inverse covariance matrix.

5.3 Cosine similarity

Feature vectors were introduced in chapter 3 alongside with various methods for obtaining such a vector. If we detect a different object in the future, and we would like to know how similar it is to the previously detected object, then we need some metric to measure the similarity between the vectors. For that, the cosine similarity can be used.

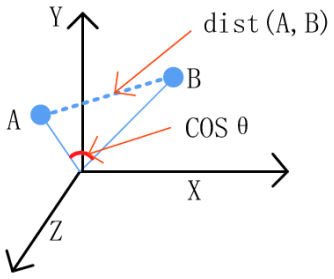

Figure 5.3: Example of cosine distance and Euclidean distance.[36]

The cosine similarity calculates the cosine of an angle between two vectors (figure 5.3). We can calculate it as follows:

$$S(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||} \tag{5.9}$$

Such a metric is particularly useful if we measure the distance between two unnormalized vectors. If we use Euclidean distance to measure the similarity between the vectors, then it may happen that the vectors point in the same direction, but the distance between them is large [21]. This might not be useful for some systems, including some systems for object detection and tracking.

It is possible to come across a term cosine distance. Cosine distance is basically the same thing as cosine similarity except that cosine distance is calculated as:

$$cosine\ distance = 1 - cosine\ similarity$$

Chapter 6

Object tracking

Following chapter explains the term object tracking and introduces some algorithms, which implement object tracking such as DeepSort or GOTURN.

6.1 What is object tracking?

Object detectors introduced in chapter 3 enable us to detect the position of some object in an image. However, if we want to track the same object across many consecutive frames, the need for another type of algorithms, called object tracking algorithms, arises.

Object tracking algorithms assign a unique identifier to the detected object, which is then associated with it for the whole time of its detection (figure 6.1). Their principle of operation can be based on the extraction of features from the detected object whose value is then compared to a newly detected object. If the values are similar, then the conclusion that it is the same object is made.

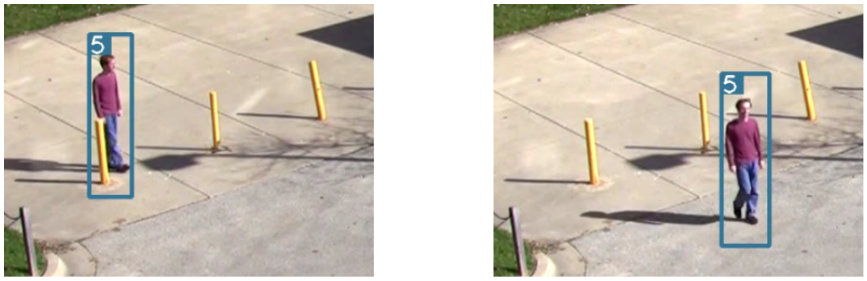

Figure 6.1: Person is being tracked across multiple video frames.

Also, it is possible to encounter more straightforward methods—for example, a method based on measuring the Euclidean distance between two detected objects. If a detected object is at a certain distance from a previously detected object, then it is likely it is the same object. Similarly, there are methods based on measuring IOU of two consecutive detected objects. Again, if the IOU is higher than the selected threshold, then it may be the same object. Such methods, of course, encounter problems when there is a large number of objects of the same type at a short distance from each other (e.g. crowded square).

Object trackers can also be viewed based on what information they use to track objects [37]:

- **Online trackers** must be able to track objects based only on the information obtained from the frames that are located before the currently analyzed image.
- **Offline trackers** can use information from frames that are located after the currently analyzed frame.

Each type of tracker encounters different kinds of problems. One of the most frequent problems is occlusion which happens when an object disappears for a few frames. The object may be occluded only partially or in some cases the object may completely disappear. In such situation methods based on Kalman filter may be used to predict the motion of the object using a history of the object's movement [35].

6.2 DeepSORT algorithm

DeepSORT is an effective algorithm for object tracking [48]. It is based on its successor - SORT (Simple Online Realtime Tracking) [7]. It uses the Kalman filter in combination with feature vectors to track the detected objects. For each newly detected object a structure (track) containing the following information is created and stored in a list L:

- location of the centre of the bounding box (u,v),
- aspect ratio of the bounding box (a),
- height of the bounding box (h),
- velocities of u,v,a and h (velocities are defined as derivatives of u,v,a and h with respect to time).

When DeepSORT recieves a newly detected object O it tries to assign it to some object which is already contained in the list L. The process of assigning the detected object to the object in the list L can be broken down into four steps.

In the **first step**, the Kalman filter is used to predict the positions l_i of the objects in the list L. For this calculation, the velocities stored in L for each object are used.

In the **second step** distance of the newly detected object O from each object contained in L is calculated. The distance can be calculated as follows:

$$D = \lambda * D_m + (1 - \lambda) * D_c, \tag{6.1}$$

where λ is the weighting factor, D_m is the Mahalanobis distance between the newly detected object O and the predicted location l_i and D_c is cosine distance between the feature vector of the object O and the feature vector of the object from the list L.

In the **third step**, the object O is assigned to the object in L whose distance D is the lowest. If the distance is higher than some threshold for each object in L then structure (track) representing the object is created and stored in L. However, in reality, more than one objects at the same time would be detected. In such a case the Hungarian algorithm is used for assigning the detected objects to objects in L using the distances D.

In the **fourth step**, the information about the detected objects is stored in L. If some object was not updated for a longer period of time, meaning there was no detected object associated with it, then it is discarded. Also, information about the objects in the list L is updated using Kalman filter.

6.3 GOTURN

A different approach of object tracking offers the GOTURN algorithm, which is based on a different idea than the previously mentioned DeepSORT [27]. GOTURN uses pre-trained CNN (section 3.6) to predict bounding boxes of moving objects. Using such a CNN we can track the object with speed up to 100 fps.

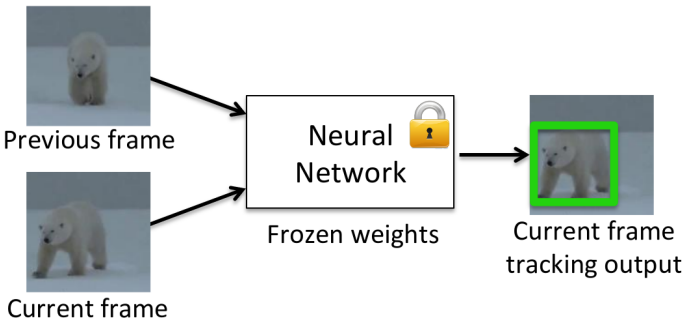

Figure 6.2: Object tracking using GOTURN. [27]

The input of the pre-trained CNN is a tuple of pictures (figure 6.2). The first picture is a cut region from the frame of the analyzed video at a time T with the tracked object in its centre. The width and the height of the cut region are scaled with respect to the width and height of the bounding box of the tracked object ($k_1 * h$, $k_2 * w$). The constants k_1 and k_2 may be set to different values depending on how fast the tracked objects move. The second picture is the same cut area, but this time at time $T + 1$. The output of the network is then coordinates of the detected object at time $T + 1$.

6.4 TLD

Another approach offers TLD algorithm [31]. The TLD stands for Training, Learning and Detection. The algorithm is based on the idea that both the detector and the tracker may collaborate together to improve its performance. The collaboration is ensured by a newly introduced component of the tracker. The component takes as input results of the tracker and detector. Then based on these inputs it tries to detect false positive and false negative errors of the detector. Using the knowledge about the errors the component then offers to the detector new learning data which should help the detector to improve its performance and to avoid making similar mistakes in the future.

Chapter 7

Implementation

In this section, the program which was created as a practical part of the thesis is described. It is a program that enables the user to search through video recordings from surveillance video cameras. The goal was to create a system which would be able to search through the recordings effectively and also a system that is easy to use.

The system is focused on offline search. When the user uploads a video to the system, the objects in it are detected, and their trajectories are stored to the database. Because object detection and object tracking are computationally expensive, the system is implemented as a web application. All the computation is then done on a server which the user selects and the system can be accessed immediately from anywhere.

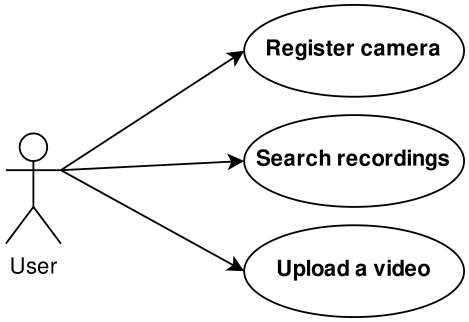

Figure 7.1: Use case diagram of the implemented program.

Before it is possible to search or upload a video to the system, a camera must be registered (figure 7.1). For successful completion of the registration, a homography matrix is needed, which converts the coordinates from image plane to the metric ground-plane coordinate system.

When the camera is registered, and a video is uploaded to the system, then the video search is possible. The video search is based on simple custom query language which uses predefined blocks:

- object moved in a direction,
- object was in a certain area,
- object moved from point A to point B,
- object appeared at a certain distance from an object.

These building blocks can be used to build more complicated queries such as: “person got out of the car” or “car entered a parking lot”.

7.1 Overview of the system

As already indicated, the system is divided into two main parts: client-side and server-side. The client-side is responsible for accepting queries from the user and displaying responses from the server. The server is responsible for storing the analysed videos and retrieving videos which contain searched events or objects.

After the user uploads a video to the system, an object detector (YOLO - chapter 3) together with object tracker (DeepSort - chapter 6) extracts information about objects in the video. Every second frame of the video is analysed, and the result of the analysis is stored in the database (for each detected object is created a record in the database).

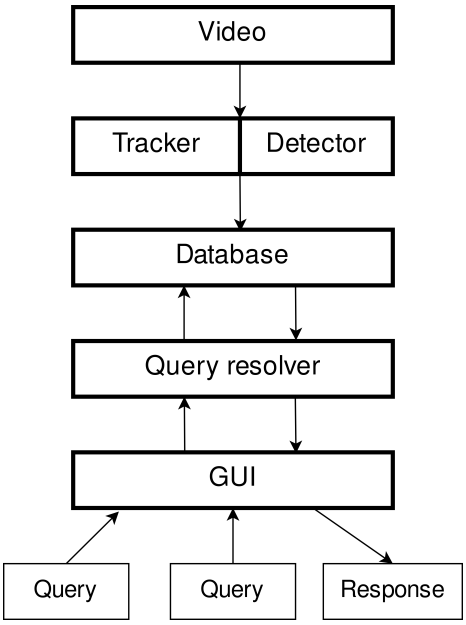

Figure 7.2: Overview of the system.

The database is later searched when the user specifies searched object using predefined search blocks mentioned in previous section 7. From the database, candidate objects are then retrieved, which are tested whether they meet the requirements which the user specified in the query. If an object satisfies all the requirements, then the corresponding video with the object in it is retrieved from a video database. The video is cut so that it only contains the part with the searched object and stored to the temporary video database. From the temporary database, the video is then downloaded by the client-side (figure 7.2).

7.2 Tools used for the implementation

A programming language chosen for the implementation of the server part of the system is **Python**. The language was selected especially because of the framework Flask, which was created for this language and for its simplicity to add libraries necessary for the im-

plementation of other parts of the system. For example, a library for communication with time-series database InfluxDB.

The **InfluxDB** time-series database is used for storing the location of detected objects. A time-series database differs from a relational database in the way of indexing stored data. The time-series databases focus on the processing of the time-stamped data, on the other hand, the relational databases are more general-purpose, and therefore they do not guarantee such quick processing of time-stamped data as time-series databases do.

For communication between the GUI and the server-side of the system, framework **Flask** is used. Flask enables us, among other things, to define endpoints of the server-side of the system which are used to obtain the state of the running processes in the background of the system and to obtain the HTML pages together with JavaScript code from the server. How does the framework works can be explained by a code sample in figure 7.3.

```
@app.route("/foo")
def foo():
    return render_template("foo.html")
```

Figure 7.3: Creation of endpoint in Flask framework. When the webpage /foo is requested, an HTML page is rendered using Jinja2 language and sent to the client-side.

Library **OpenCV** is also used in the application. OpenCV is a library which offers a wide variety of preprogrammed functions for video and image analysis. In the created application, it is used especially for obtaining information about the analysed video such as its resolution or length. But the library offers many more functions. For example functions implementing object tracking algorithms like GOTURN (section 6.3) or TLD (section 6.4) can be found in the library too.

The library mentioned above is also used by the implementation of object tracker, which is used for tracking objects in analysed videos[1]. The implementation uses YOLO algorithm (section 3.7) for object detection, which passes its output to DeepSort algorithm (section 6.2). Together these two algorithms are able to track position of objects with speed up to ~15 fps. This speed has been reached when tested on *NVidia2080TI* graphic card and when the analysed video contained no more than ten objects to track simultaneously.

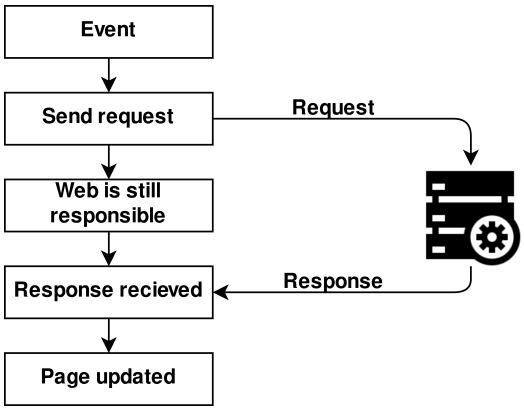

Figure 7.4: Example of sending request to a server using AJAX.

[1] https://github.com/ZQPei/deep_sort_pytorch

The GUI is created using JavaScript together with HTML and CSS. The GUI communicates with the server-side of the application using AJAX, which enables to retrieve new information from the server-side and update the GUI without the need of reloading the whole website (figure 7.4).

7.3 The architecture of the system

This section contains the description of the architecture of the server-side and the client-side of the application.

7.3.1 The server side of the application

The server part of the system can be divided into four main parts: part for handling incoming messages from the client-side, database (contains extracted information from videos, defined blocks by the user and video recordings), part for analysis of uploaded videos and part responsible for searching through video recordings.

The first part which is responsible for handling incoming messages and communication with the client-side has several predefined endpoints:

- `/add_camera` and `/delete_camera` add cameras to the system and delete them. For the creation of the camera homography matrix and name of the camera is needed.
- `/get_cameras` and `/get_cameras_frame` return information about already existing cameras and pictures containing view from specific cameras.
- `/search_request` accepts user requests to search for an object or an event.
- `/add_object`, `/get_objects` and `/delete_object` are used for creation, retrieval and deletion of defined search blocks by the user.
- `/upload_video` accepts uploaded video by the user. After the video is uploaded to the video database of the system, it is passed to the part of the application that is responsible for its analysis.

During the analysis of each video, each frame of the recording is analyzed, and the positions of the objects together with their identifiers, which are obtained from the object tracker, are stored in the database. Each record in the database contains:

- normalized position of the centre of the bounding box in the ground plane,
- normalized height and width of the bounding box,
- position of the centre of the bounding box in the image plane,
- unique identifier of the detected object within the analyzed video,
- UUID of the video that is assigned to it when uploaded to the system,
- name of the camera that captured the video,
- date and time of the moment when the object was captured by the camera.

User-defined blocks are stored in a database that consists of JSON files. Each file contains a description of the block, which consists of its name and the parameters of the sub-blocks that define the given block.

When a search request arrives at the system, objects are created that represent all potential searched objects (SearchBlobs). Subsequently, a list of objects (search blocks – section 7.5) is created in which each object is responsible for analyzing a SearchBlob and deciding whether the SearchBlob satisfies the query sent by the user or not.

Each SearchBlob is passed to every search block in the list. If a search block in the list evaluates the SearchBlob as an object that does not match to the query sent by the user, then the SearchBlob is discarded. SearchBlobs that pass this process of elimination are passed to the part of the system that is responsible for cutting out the parts of the video that contain searched event.

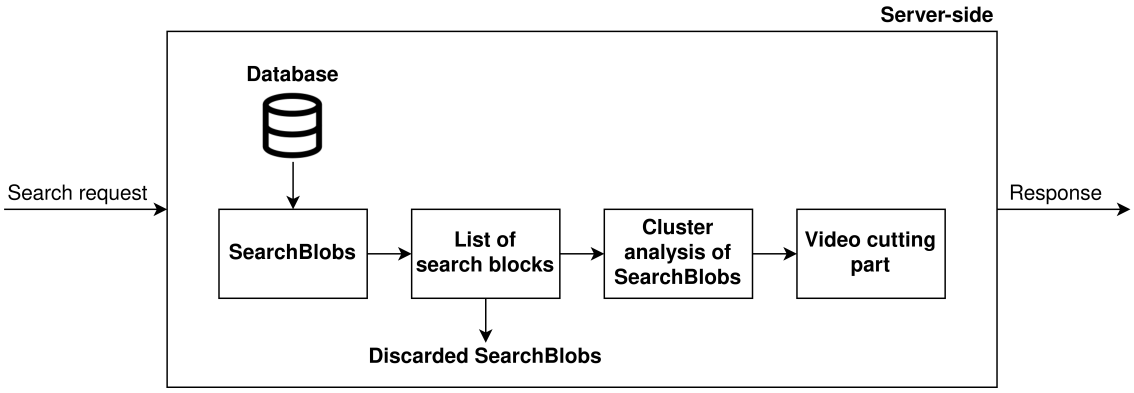

Figure 7.5: Scheme of the search process.

Sometimes the same part of the video may be presented to the user several times. This can happen, when resulting SearchBlobs are within time close to each other. To prevent this from happening, at least partly, a cluster analysis is performed, which tries to merge these SearchBlobs into one. After the cluster analysis, the SearchBlobs are sent to a video cutting section of the application, which prepares videos containing the searched event. The video is then stored in the temporal video database, from which it is deleted after one hour, and sent to the client-side (figure 7.5).

7.4 The client-side of the application

The search for events in the records in the database is performed, as already indicated in the section 7, using predefined blocks. These blocks are divided into three main categories:

- **Object block** defines type of the searched object *(car or person)*.
- **Action blocks** define actions which the object performs *(appear, disappear, moved from X to Y, moved in direction, stopped)*.
- **Condition blocks** put additional constraints on the searched event *(date and time, speed of movement, distance from an object, camera, in an area)*.

Each block is part of the language that is described by a finite state machine in figure 7.6.

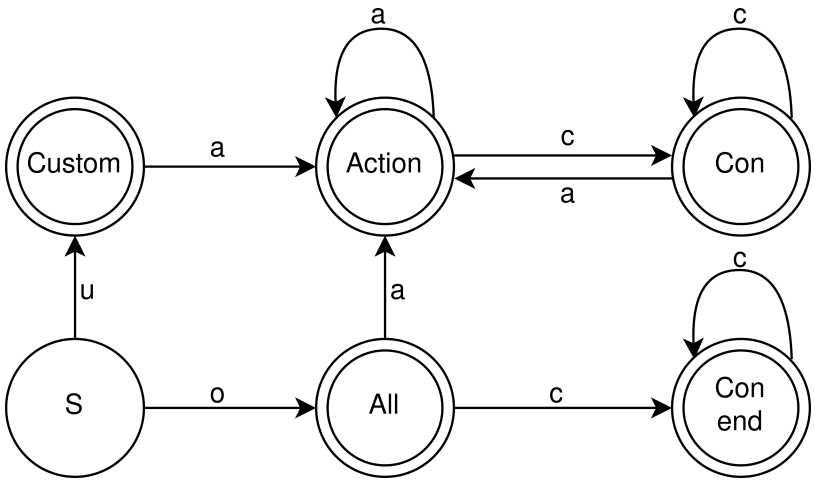

Figure 7.6: The finite state machine describing the query language. *(a = action block, b = condition block, u = custom block, o = object block)*

The search section of the GUI consists of a search bar and help section, which proposes to the user blocks that they can use for the search. When the user clicks on the proposed block, a new object is created that represents it and is stored in a list in the background (figure 7.8). Also, when the user selects one of the proposed blocks, the state of the finite state machine is changed, based on which the help section is updated for the user. After the user clicks the search button then all objects in the list are checked whether all the necessary information is defined for each search block. If not, then the user is prompted to complete the necessary information.

Once all the necessary information for executing the query is available, then the query is converted to JSON format (figure 7.7)[2].

```
{
    "request": [
        ...
        {
          'name': 'DATE_TIME',
          'block_description': 'Date and time',
          'params': {
            'from_timestamp_mili': '1577836800000',
            'to_timestamp_mili': '1609459200000',
            ...
          }
        },
        ...
    ]
}
```

Figure 7.7: Example of search request containing DateTime search block.

[2]If the query contains a user-defined block, then the parameters for the given block are added on the server-side.

The serialized query is then sent to the server, where a search is performed, the results of which are then sent back. The results, among other things, contain paths to videos that contain the events the user is looking for. These are then displayed to the user.

A similar process is followed when creating user-defined blocks. Except that, after the user selects the blocks to form the new block, and it is checked that all the necessary information is filled in, the definition of the new block is sent to `/create_block`.

To upload new videos for analysis, the user selects a video in the upload section and writes the time when the video was taken. After pressing the upload button, the video is uploaded to the server, where it is analyzed. The status bar informs the user about the status of the performed analysis, which every second asks at the endpoint `/analysis_progres` what proportion of the video has already been analyzed and whether any error occurred during the analysis.

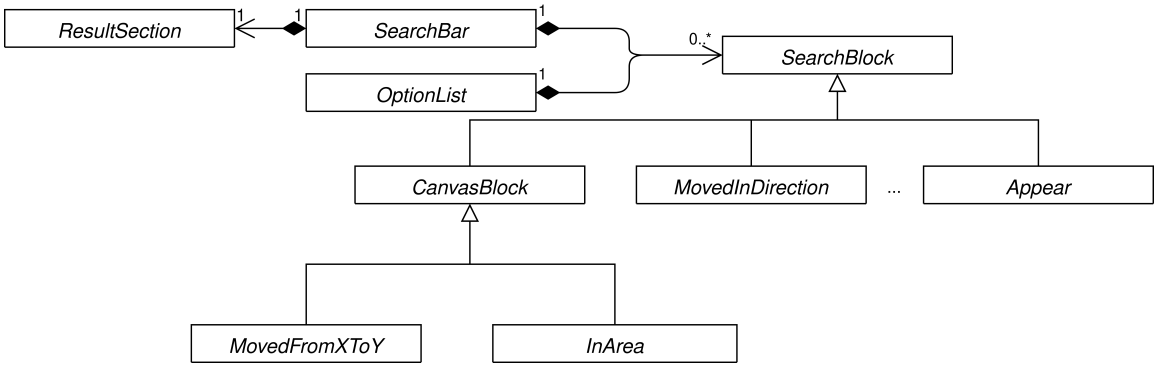

Figure 7.8: Class diagram of the client-side.

7.5 Description of the search blocks

This section describes the search blocks which test data stored in SearchBlobs (section 7.3.1) whether it meets the conditions specified by the query. Each SearchBlob stores inside records from the database describing the position across time of the object it represents. This data is stored in two variables, `db_data` and `db_data_processed`. `Db_data` stores the original data from the database and `db_data_processed` the original data filtered by individual search blocks (figure 7.9).

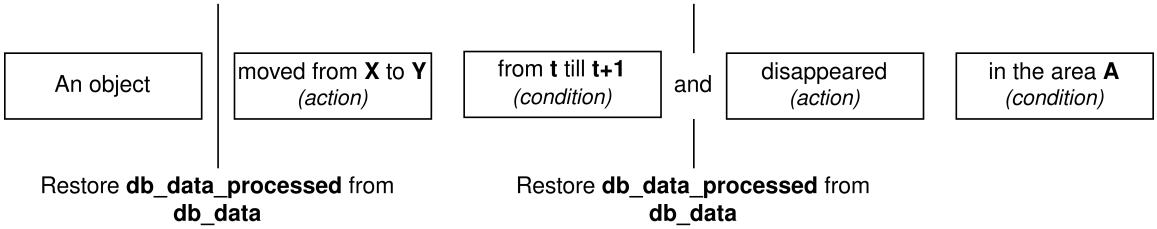

Figure 7.9: Example of a query. Each SearchBlob is analyzed sequentially by the search blocks as they appear in the query.

If the query contains multiple action blocks with constraints, then the records in `db_data_processed` are restored from `db_data` each time SearchBlob is processed by new action block.

The first simplest search blocks are *appear* and *disappear* blocks. These blocks works on a simple principle. They take data from `db_data` and insert the last or the first record into `db_data_processed` variable.

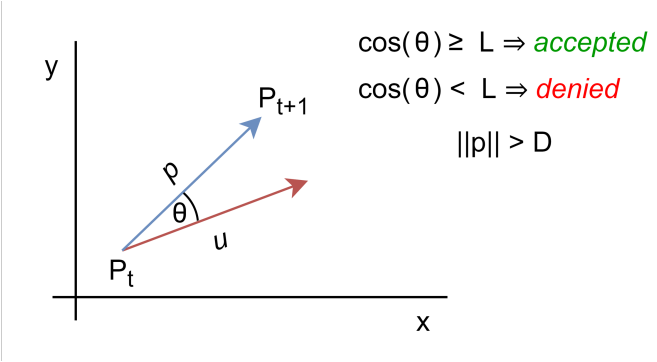

Figure 7.10: Filter points (P_t,P_{t+1}) by the angle that the vector defined by these points makes with the vector defined by the user. *(u = user-defined vector, p = vector defined by the points from the database, D = minimal distance, L = acceptance limit)*

Another action block is *moved from X to Y block*. This block searches for an object that has moved from the location with coordinates X in the image plane to the location with coordinates Y. For this purpose, the records in the `db_data_processed` variable are checked. If the block encounters a record with a position near point with X coordinates, then it starts searching for a record with coordinates near point with coordinates Y. If it finds such two records, then all records which are not timewise between these two records are deleted from `db_data_processed` variable.

The *moved in direction* block searches for the parts of the object trajectory where the object moves in the user-selected direction. It sequentially goes through the records in SearchBlob, looking for pairs of points whose spatial distance is greater than the specified limit, and which indicate that the object was moving in a direction similar to the direction defined by the user (figure 7.10).

The *stopped* block works in a similar manner. If there are two points in `db_data_processed` which are spatially close to each other but timewise are more than 1 second apart then these points are considered to represent part of the object's trajectory on which the object was not moving.

The approach of other blocks which were not mentioned here does not differ a lot. Thanks to the architecture which is based on the search blocks it should not be difficult to add new search blocks to the language in the future, if needed.

7.6 Testing the system

Three aspects of the system were tested. The first was the usability of the system, especially how difficult it is for a new user to learn to work with it. Usability was tested on six people who were given a series of task focusing on creating queries and using the system.

Another aspect that has been tested was the ability of the system to find in the videos the basic situations as: "a person got out of/in a car", "a person entered the building" or "a car arrived in the parking lot".

The last but not least of the important aspects is the speed with which search blocks process individual SearchBlobs.

When testing the system, a 10-minute recording from the VIRAT video dataset was used, which contains enough different events necessary for testing the system [41]. The videos from this dataset were used mainly because, together with the videos, it also offers homography matrices for each video. However, when creating the system and especially when testing object tracking and object detection algorithms, such dataset as [12] or [22] were also used.

7.6.1 Usability

The usability testing of the system was performed with the help of six people[3]. It was explained to each person how the system works. In particular, it was necessary to explain what is homography detectormatrix and how object detector and object tracker work. The tested users were also shown a video that contained the detected objects to give them a better idea of what principle the system works on.

After the brief explanation, it was tested how intuitive the application is. The tested users were asked questions such as: "If you had to upload a video to the system, how would you do it?" or "If you had to register a new camera, where and how would you do it?". The full questionnaire and the described testing procedure are available in appendix A.

Subsequently, on the example of the query "a person came out of the building", users were introduced to the search part of the system and the query language. The knowledge from this example was then used for creating a new queries such as "a person got out of the car at 5 o'clock."

The results of testing the usability of the application are as follows. Users usually did not have major reservations about the user interface. The only reservation that users mentioned was the inability to delete individual search blocks using backspace. On average, the tested users rated the UI with a mark of 1.83 (1 – good, 1 – bad). When testing the query language, it turned out that users usually need to try to create from two to three search queries with different search blocks to understand how the query language works. Therefore, when testing, users often tried more queries than are found in the test protocol in appendix A. Users rated the difficulty of learning the language on average by number 2.5 (1 – good, 1 – bad). The most common opinions about the language that appeared were:

- The language can be understood. However, it is necessary to understand what the individual blocks do. I was not sure how does the "distance from" block works.
- If I take time to understand what each block does, then using the language is quite easy.

7.6.2 Testing the capability of the query language and its speed

A total of four different queries were selected to test the ability of the query language to find searched events. These queries were chosen to show the capabilities of individual search blocks. Along with testing the language's capability to find specific situations, the speed of finding the searched events was also measured[4].

[3] Tested users: three men (20 y.o. - advanced computer skills, 24 y.o. - advanced computer skills, 19 y.o. - advanced computer skills), three women (47 y.o. - basic computer skills, 23 y.o. - intermediate computer skills, 18 y.o. - advanced computer skills)

[4] tested on *Intel (R) Core (TM) i5-825OU CPU @ 1.60Hz*

The first query looks for situations in which a person got out of a car (figure 7.11).

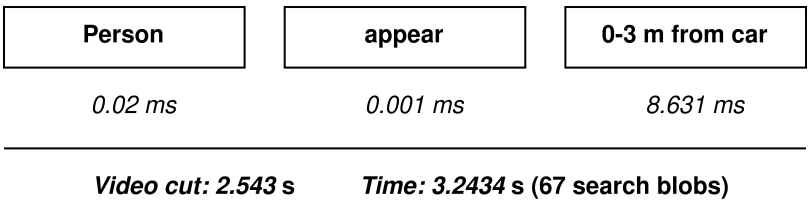

Figure 7.11: A query that looks for situations in which a person gets out of a car.

The result of this query was three recordings. The system discovered all parts of the video used for testing in which a person got out of the car.

In figure 7.12 is query which searches for situations where a car arrived at a certain speed in the parking lot.

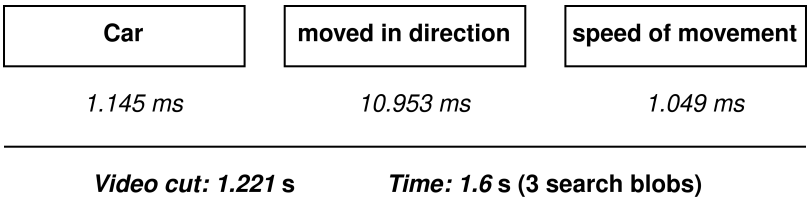

Figure 7.12: A query that looks for situations in which a car arrived in a parking lot with a certain speed of movement.

For this query, the system discovered the searched car that arrived in the parking lot with the specified speed.

And finally the last query (figure 7.13) was used to find the situation when the car stopped in a specific parking space.

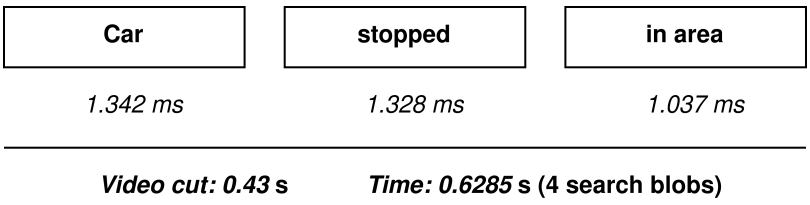

Figure 7.13: A query that looks for situations in which a car stopped in a certain area.

For this query, the system successfully discovered the one situation in which a car stopped in a parking space.

Looking at the times in figures 7.11, 7.12 and 7.13, it is clear that a large proportion of the search time takes the time required for cutting out the parts of videos which contain the searched event.

7.7 Future development

In order to improve the resulting system, I suggest omitting the option of adding a homography matrix when registering a new camera. When uploading a new video, the user would only choose the name of the already registered camera, and subsequently, if a video was loaded into the system, the homography matrix would be calculated similarly as in [8].

To reduce the error rate when tracking objects, it would also be useful to try to create a custom object detector and object tracker, as the system is built so that they are easily replaceable. The currently used combination of object tracker and object detector sometimes mark the same object with different id, because of that, the system treats some objects as several separate objects.

And finally, the current version of the system also converts the uploaded video to *.webm* format to make it possible to display it in most web browsers. This conversion increases the time needed by the system to analyse the video. Therefore, it would be useful to convert the video in the background after the end of the video analysis.

Chapter 8

Conclusion

One of the main goals of this work was to find out the principle on which surveillance video search systems are based. This question was answered at the beginning of this work, where a general scheme of such systems was presented, together with ways of storing information extracted from the videos to be searched.

This work also aimed to create a system that will be able to search for objects in video recordings based on their trajectory and speed. Therefore, it was necessary to find out how the homography works and how we can calculate it.

To track the trajectory of objects in video recording, it is necessary to find objects in video recordings. For this reason, algorithms for object detection were introduced.

The chapter on object detectors was followed by a chapter on object trackers, which were necessary for tracking objects across consecutive frames. The most important thing was to explain the DeepSort algorithm, that was used in the implementation of the resulting system.

In the second part of the work, the theoretical knowledge gained from the previous part was used to implement a system that is able to search records based on the trajectory of movements of objects detected in the records. This system is based on a simple query language, which builds on so-called search blocks, with which the user specifies the searched event or object. During testing, it was found that the system is able to detect simple situations, such as “a person got out of the car“ or “the car stopped in a particular parking place“. This system is not perfect, but by using more accurate methods for object tracking, it is possible to achieve better results. In the future, it is also possible to expand the system with new search blocks. For example, a search block to specify the colour of a searched object can be added, or an option that would enable specification of the number of searched objects can be added – such option would allow us to create a queries like: “three people have moved from place A to place B“.

Bibliography

[1] Anon.. *Tutorial Camera Calibration*, 24. january 2020. Available at: https://boofcv.org/index.php?title=Tutorial_Camera_Calibration.

[2] Ashby, M. The Value of CCTV Surveillance Cameras as an Investigative Tool: An Empirical Analysis. *European Journal on Criminal Policy and Research* [online]. september 2017, vol. 23, p. 441–459, [cit. 2020-04-29]. DOI: https://doi.org/10.1007/s10610-017-9341-6. Available at: https://link.springer.com/article/10.1007/s10610-017-9341-6.

[3] Authors, T. V. Basic HOG computation. *VLFeat.org* [online]. 2007. Available at: https://www.vlfeat.org/overview/hog.html.

[4] AxxonSoft, I. *MomentQuest* [online]. 2019 [cit. 2020-04-04]. Available at: https://www.youtube.com/watch?v=__x-Da7bfdc.

[5] AxxonSoft, I. *AxxonSoft* [online]. 2020 [cit. 2020-04-01]. Available at: https://www.axxonsoft.com/.

[6] Becker, A. *Kalman Filter* [online]. 2020 [cit. 2020-04-20]. Available at: https://www.kalmanfilter.net/default.aspx.

[7] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)* [online]. Phoenix, AZ, USA: IEEE, September 2016, p. 3464–3468 [cit. 2020-04-17]. ISBN 978-1-4673-9961-6. Available at: https://ieeexplore.ieee.org/document/7533003.

[8] Brouwers, G., Zwemer, M., Wijnhoven, R., With, P. de, Gang, H. et al. *Automatic calibration of stationary surveillance cameras in the wild*. 2016. ISSN 0302-9743.

[9] Brownlee, J. How to Configure the Learning Rate When Training Deep Learning Neural Networks. *Machine Learning Mastery* [online], 23. january 2019. Available at: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks.

[10] Camiolog, I. *Camio* [online]. 2020 [cit. 2020-04-01]. Available at: https://www.camio.com/.

[11] Castañón, G., Saligrama, V., Caron, A. L. and Jodoin, P. Real-Time Activity Search of Surveillance Video. In: *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance* [online]. Beijing, China: IEEE,

September 2012, p. 246–251 [cit. 2020-04-04]. ISBN 978-0-7695-4797-8. Available at: https://ieeexplore.ieee.org/document/6328024.

[12] Chavdarova, T., Baqué, P., Bouquet, S., Maksai, A., Jose, C. et al. WILDTRACK: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* [online]. IEEE, 2018, p. 5030–5039 [cit. 2020-05-05]. Available at: https://ieeexplore.ieee.org/document/8578626.

[13] Chen, L. The simplistic illustration of basic concepts in Support Vector Machine. *Support Vector Machine — Simply Explained* [online], 6. january 2019. Available at: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496.

[14] Choe, T. E., Lee, M. W., Guo, F., Taylor, G., Yu, L. et al. Semantic video event search for surveillance video. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)* [online]. Barcelona, Spain: IEEE, November 2011, p. 1963–1970 [cit. 2020-04-01]. ISBN 978-1-4673-0063-6. Available at: https://ieeexplore.ieee.org/document/6130489.

[15] Clapham, M. E. *23: Mahalanobis distance* [online]. [cit. 2020-04-20]. Available at: https://www.youtube.com/watch?v=spNpfmWZBmg.

[16] Corporation, I. *IBM Video Analytics: AI-infused video can increase the effectiveness and speed at which businesses communicate and operate* [online]. The Weather Company, an IBM Business1 New Orchard RoadArmonk, NY 10504: IBM Corporation, july 2019 [cit. 2020-04-01]. Available at: https://www.ibm.com/downloads/cas/2KDPOBLP.

[17] Dale, M. R. Graphs as Structure in the Ecological Context. In: *Applying Graph Theory in Ecological Research*. Cambridge University Press, 2017, p. 1–36. DOI: 10.1017/9781316105450.002.

[18] Dasig, S. *Introducing MV32, Motion Search 2.0, and Motion Recap* [online]. 2019 [cit. 2020-04-29]. Available at: https://meraki.cisco.com/blog/2019/04/introducing-mv32-motion-search-2-0-and-motion-recap/.

[19] Derpanis, K. G. Overview of the RANSAC Algorithm. [online]. 1.2. may 2013, [cit. 2020-04-29]. Available at: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf.

[20] Dubská, M., Sochor, J. and Herout, A. Automatic Camera Calibration for Traffic Understanding. In: *Proceedings of BMVC 2014* [online]. The British Machine Vision Association and Society for Pattern Recognition, 2014, p. 1–10 [cit. 2020-04-29]. ISBN 1-901725-52-9. Available at: https://www.fit.vut.cz/research/publication/10682.

[21] Emmery, C. *Euclidean vs. Cosine Distance* [online]. 2017 [cit. 2020-04-29]. Available at: https://cmry.github.io/notes/euclidean-v-cosine.

[22] Fleuret, F., Berclaz, J., Lengagne, R. and Fua, P. Multicamera People Tracking with a Probabilistic Occupancy Map. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence [online]. IEEE. 2008, vol. 30, no. 2, p. 267–282, [cit. 2020-05-05]. DOI: 10.1109/TPAMI.2007.1174. ISSN 1939-3539. Available at: https://ieeexplore.ieee.org/document/4359319.

[23] Ganesh, P. *Object Detection : Simplified*, 12. august 2019. Available at: https://towardsdatascience.com/object-detection-simplified-e07aa3830954.

[24] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning* [online]. 1st ed. MIT Press, 2016 [cit. 2020-04-14]. Available at: http://www.deeplearningbook.org.

[25] Gupta, S. An effective way of reducing the dimensionality of your data. *Locality Sensitive Hashing* [online], 29. june 2018 [cit. 2020-04-04]. Available at: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134.

[26] Hampapur, A., Brown, L., Feris, R., Senior, A., Chiao-Fe Shu et al. Searching surveillance video. In: *2007 IEEE Conference on Advanced Video and Signal Based Surveillance* [online]. London, UK: IEEE, September 2007, p. 75–80 [cit. 2020-04-01]. ISBN 978-1-4244-1695-0. Available at: https://ieeexplore.ieee.org/document/4425289.

[27] Held, D., Thrun, S. and Savarese, S. Learning to Track at 100 FPS with Deep Regression Networks. In: Leibe, B., Matas, J., Sebe, N. and Welling, M., ed. *Computer Vision – ECCV 2016* [online]. Cham: Springer International Publishing, September 2016, p. 749–765 [cit. 2020-04-19]. ISBN 978-3-319-46448-0. Available at: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_45.

[28] Hui, J. *MAP (mean Average Precision) for Object Detection* [online], 7. march 2018. Available at: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

[29] Iqbal, A., Arif, F. and Minallah, N. Analyzing impact of video codec, encapsulation methods and streaming protocols on the quality of video streaming. In: *Eighth International Conference on Digital Information Management (ICDIM 2013)* [online]. Islamabad, Pakistan: IEEE, September 2013, p. 182–186 [cit. 2020-04-29]. Available at: https://ieeexplore.ieee.org/document/6693983.

[30] Jordan, J. *Neural networks: representation.*, 28. june 2017. Available at: https://www.jeremyjordan.me/intro-to-neural-networks/.

[31] Kalal, Z., Mikolajczyk, K. and Matas, J. Tracking-Learning-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. IEEE. December 2012, vol. 34, no. 7, p. 1409–1422, [cit. 2020-04-19]. DOI: 10.1109/TPAMI.2011.239. ISSN 1939-3539. Available at: https://ieeexplore.ieee.org/document/6104061.

[32] Kriegman, D. *Homography Estimation* [online]. San Diego, USA: UC San Diego [cit. 2020-04-07]. Available at: https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf.

[33] Le, T.-L., Thonnat, M., Boucher, A. and Brémond, F. A Query Language Combining Object Features and Semantic Events for Surveillance Video Retrieval. In: Satoh, S., Nack, F. and Etoh, M., ed. *Advances in Multimedia Modeling* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 307–317 [cit.

2020-04-01]. ISBN 978-3-540-77409-9. Available at: https://link.springer.com/chapter/10.1007/978-3-540-77409-9_29.

[34] Leavitt, A. *Why Use The H.265 Video Codec For Surveillance Systems?* [online]. 2018 [cit. 2020-04-29]. Available at: https://getsafeandsound.com/2018/06/h-265-video-codec/.

[35] Lee, B. Y., Liew, L. H., Cheah, W. S. and Wang, Y. C. Occlusion handling in videos object tracking: A survey. *IOP Conference Series: Earth and Environmental Science* [online]. IOP Publishing. February 2014, vol. 18, [cit. 2020-04-16]. DOI: 10.1088/1755-1315/18/1/012020. Available at: https://doi.org/10.1088%2F1755-1315%2F18%2F1%2F012020.

[36] Leilei Wang, J. W. *The difference between Euclidean distance and cosine similarity.* Available at: https://www.researchgate.net/figure/The-difference-between-Euclidean-distance-and-cosine-similarity_fig2_320914786.

[37] Luo, W., Zhao, X. and Kim, T. Multiple Object Tracking: A Review. *CoRR* [online]. may 2014, abs/1409.7618, [cit. 2020-04-29]. Available at: http://arxiv.org/abs/1409.7618.

[38] Mallick, S. Image Alignment (ECC) in OpenCV (C++ / Python). *Learn OpenCV* [online], 1. june 2015. Available at: https://www.learnopencv.com/image-alignment-ecc-in-opencv-c-python/.

[39] Mallick, S. Histogram of Oriented Gradients. *Learn OpenCV* [online], 6. december 2016. Available at: https://www.learnopencv.com/histogram-of-oriented-gradients/.

[40] Niessen, M. *How to Perform Crowd Alerting with IBM Video Analytics* [online]. [cit. 2020-04-01]. Available at: https://www.youtube.com/watch?v=YA1Eja7M7qQ.

[41] Oh, S., Hoogs, A., Perera, A., Cuntoor, N., Chen, C.-C. et al. A large-scale benchmark dataset for event recognition in surveillance video. In: *CVPR 2011*. IEEE, 2011, p. 3153–3160. ISBN 9781457703942. Available at: https://viratdata.org/.

[42] Prabhakaran, S. *Mahalonobis Distance – Understanding the math with examples (python)* [online]. 2019 [cit. 2020-04-29]. Available at: https://www.machinelearningplus.com/statistics/mahalanobis-distance/.

[43] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. Las Vegas, NV, USA: IEEE, June 2016, p. 779–788 [cit. 2020-04-14]. ISBN 978-1-4673-8851-1. Available at: https://ieeexplore.ieee.org/document/7780460.

[44] Sinha, U. SIFT: Theory and Practice. *SIFT* [online]. Available at: https://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/.

[45] team, O. *Basic concepts of the homography explained with code* [online]. OpenCV team, april 2020 [cit. 2020-04-07]. Available at: https://docs.opencv.org/master/d9/dab/tutorial_homography.html#lecture_16.

[46] Uijlings, J., Sande, K. van de, Gevers, T. and Smeulders, A. Selective Search for Object Recognition. *International Journal of Computer Vision* [online]. Boston: Springer US. 2013, vol. 104, no. 2, p. 154–171, [cit. 2020-04-11]. ISSN 0920-5691. Available at: https://link.springer.com/article/10.1007/s11263-013-0620-5.

[47] Viola, P. and Jones, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* [online]. Kauai, HI, USA, USA: IEEE, December 2001, p. I–I [cit. 2020-04-11]. ISBN 0-7695-1272-0. Available at: https://ieeexplore.ieee.org/document/990517.

[48] Wojke, N., Bewley, A. and Paulus, D. Simple Online and Realtime Tracking with a Deep Association Metric. *CoRR* [online]. Cornell University. March 2017, abs/1703.07402, [cit. 2020-04-17]. Available at: http://arxiv.org/abs/1703.07402.

Appendix A

Test protocol

- ☐ **Get the following information about the tested person:**
  - ☐ Age
  - ☐ Level of computer skills *(basic - intermediate - advanced)*
- ☐ **Explain the theory behind the system.**
  - ☐ What is object detection?
  - ☐ What is object tracking?
  - ☐ What is a bounding box?
  - ☐ What is a homography matrix?
- ☐ **Show the system and let the tested person use it for 5 minutes.**
- ☐ **Give the tested person following tasks and mark whether they completed them or not.**
  - ☐ Try to register a new camera. *(YES - NO)*
  - ☐ Try to upload a video. *(YES - NO)*
  - ☐ Try to delete the registered camera. *(YES - NO)*
- ☐ **Test the usability of the query language.**
  - ☐ Show the tested person the query which searches for a person who came out of a building.
  - ☐ Let user make query which searches for: „a person got into a car at HH:MM on YYYY-MM-DD“. *(YES - NO)*
  - ☐ Let user make query which searches for: „a car which arrived in a parking lot“. *(YES - NO)*
- ☐ **Make the tested user rate the system.**
  - ☐ It was difficult for me to find things in the UI. *(1 - easy, 10 - very difficult)*
  - ☐ Using the UI was a pleasant experience. *(1 - agree, 10 - disagree)*
  - ☐ It was difficult for me to understand how does the query language work. *(1 - agree, 10 - disagree)*
  - ☐ Is there something you would like to add?

Appendix B

Installation

Requirements:

- The application was tested on Fedora OS (version 30) and ManjaroLinux (version 19.0.2), but it should work on most Linux distributions.
- `docker` and `docker-compose` for quick preparation of InfluxDB database.
- python 3.6.9
- python package `virtualenv`

Installation steps:

1. Change the directory to the root directory of the project.
2. Create a virtual environment and install requirements
   (`virtualenv .venv &&`
   `. .venv/bin/activate && pip install -r requirements.txt`)
3. Change the directory to `docker_influxdb` directory (`cd docker_influxdb`)
4. Run container with InfluxDB database (`docker-compoe up`)
5. Change the directory to `surveillance_video_search` directory
   (`cd ../surveillance_video_search/`)
6. Run the flask application
   (`flask run` or `flask run --host=<ip_address> --port=<port_number>`)
7. The application can be viewed with any web browser at `127.0.0.1:5000` or at the IP address and port defined in the previous step.

Appendix C

Poster

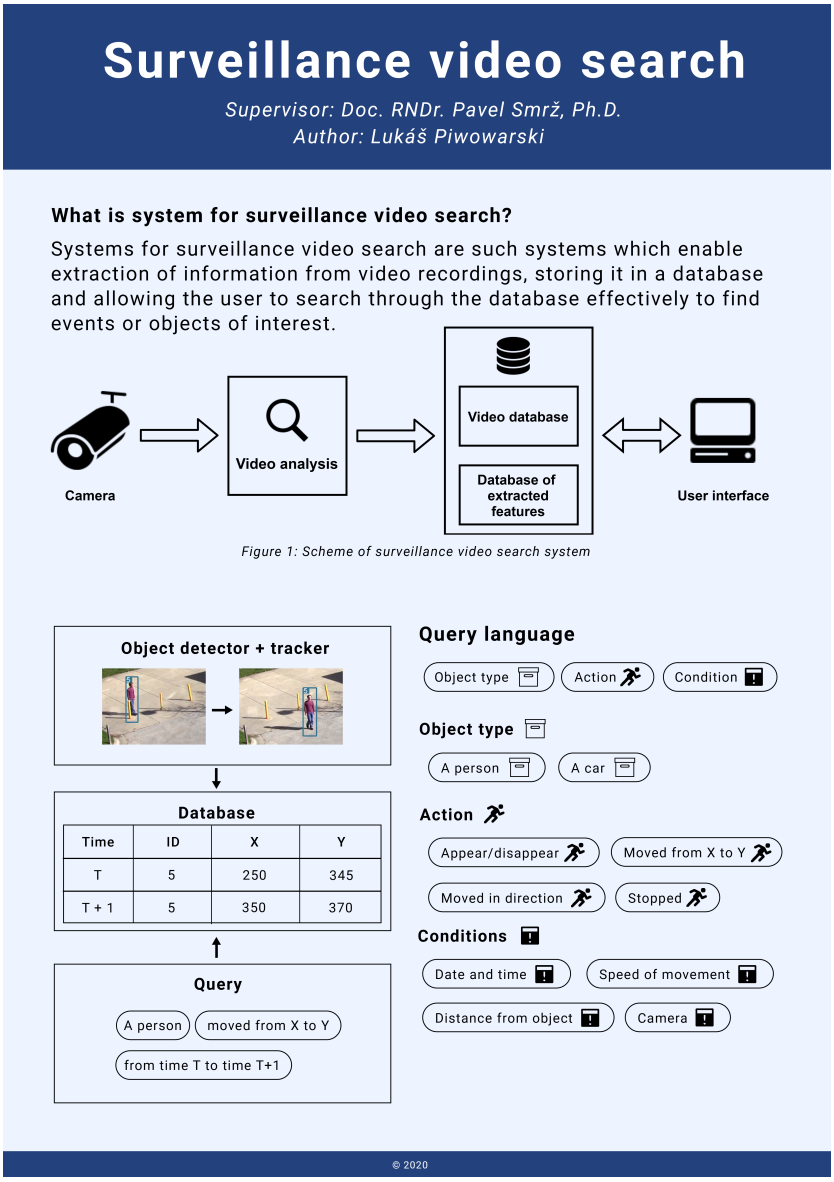

Figure C.1: The poster presenting this thesis *(in full resolution on the enclosed sd card)*