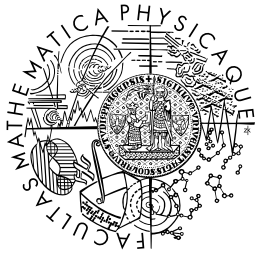

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**BACHELOR THESIS**

Dominika Ďurišková

# Presentation interface for analytical module of the Videolytics system

Department of Software Engineering

Supervisor of the bachelor thesis: prof. RNDr. Tomáš Skopal, Ph.D.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  .....

Author's signature

I would like to express my gratitude to my supervisor, prof. RNDr. Tomáš Skopal, Ph.D., for his patient guidance and the useful remarks he has given me during my journey of working on this thesis.

I would also want to thank my family, my boyfriend, and my friends who supported me throughout my studies, especially during the time of writing this thesis.

Title: Presentation interface for analytical module of the Videolytics system

Author: Dominika Ďurišková

Department: Department of Software Engineering

Supervisor: prof. RNDr. Tomáš Skopal, Ph.D., Department of Software Engineering

Abstract: With the sharp increase in the number of surveillance cameras in public spaces in recent years, there is a rapidly increasing need for video processing and analysis without the necessity of human assistance. Computers are able to process several times more information in much less time than humans. In addition, thanks to the impressive progress in the field of machine learning algorithms and artificial intelligence, computer-based video analysis is becoming a common part of everyday life and is steadily finding its way in various fields. In this thesis, we design and implement a graphical user interface for the analytical module of the Videolytics system. We aim to design a graphical user interface consisting of two parts that is user-friendly and simple. The input part of the interface allows users to enter complex visual queries and modify query parameters. The second, presentation part, is focused on the process and logic of working with the results and their rendering. Additionally, it also allows the export of this data for further processing by external applications and the import of the post-processed data. Finally, we show the module in practice and its ways of application in practical life on the enclosed examples.

Keywords: video surveillance, video analysis, web application, graphical user interface

# Contents

# Introduction

Due to the recent development of camera hardware, the importance of video surveillance cameras is increasing rapidly. Nowadays, they can be found in almost every public or private space. However, it becomes almost impossible for people to keep up with these very large amounts of footage. Analyzing data manually could take hours upon hours and require manpower which is often unavailable and expensive. That is why the demand for video analytics tools is on the rise.

Moreover, thanks to a significant advancement in the field of machine learning and neural networks, the task of analyzing a video has become much simpler. The usage of neural networks replaced old systems utilizing traditional algorithms and heuristics like Pfinder [1]. Computer vision algorithms are evolving and improving every year - from traditional histograms of oriented gradients [2] to Fast Feature Pyramids [3] to Deep Neural Networks (DNNs).

Video analysis can be done either online or offline. The difference between the two lies in the time of its execution:

- *Online analysis* is performed in real time, often using a pre-trained neural network or other machine learning algorithms. A common source of video footage are the Internet Protocol (IP) cameras, which allow streaming data via an IP network.

- *Offline analysis* uses precomputed features of a given video for analyzing requested patterns. The analysis is not performed in real time and can be done over any video at any time.

Every analytical tool is based on *object detections*. Unlike image classification, in which the main task is to recognize an object on the given image, object detection is more complex. The algorithm must decide not only about the class of the object but also about the position within the image. The position of the detected object in the image is defined by a *bounding box* - a rectangle drawn around the given object.

Naturally, detections open up possibilities for motion analysis. One example is modeling *trajectories* - a path of detected objects in time as a sequence of their detections. Trajectories can be clustered and analyzed for understanding behavioral patterns, the prediction of future movements, or monitoring traffic [4].

Many commercial companies offer analytic tools but we didn't find many open-source projects that do so. The main goal of the Videolytics project is to provide an open-source online analysis, and with the newly implemented analytical module, offline analysis as well.

## Goals

The goal of this thesis is to propose and implement GUI (Graphical User Interface) for the analytical module of the Videolytics project.

We aim to create an interface that is easy to work with and allows users to easily place visual queries, define query parameters, review, export, and import results data.

## Structure of the work

We start with a review of some already existing commercial and non-commercial solutions for video analysis in Chapter 1.

Because the analytical module is a part of a more complex project Videolytics, in Chapter 2 we present an overview of the Videolytics framework and modules.

Chapters 3 and 4 are dedicated to describing the input interface and the presentation interface of the analytical module from the user's point of view. It does not contain implementation details, rather it explains how queries are entered and how results can be handled.

In Chapter 5 we present several practical examples of using the analytical module. We start with a simple example and gradually work our way up to the harder ones, including exporting results and their post-processing by external applications.

Finally, in Chapter 6 we discuss the implementation details of our solution.

# 1. Related works

Before we begin describing our architecture and its implementation, we first take a look at related software and projects, which deal with the analysis of video input.

## 1.1 Commercial projects

The most prevalent type of software which we were able to find was commercial, either requiring one-time payment or subscription fees, for which the user receives a copy of the software, or is eligible to send and request analysis of their video footage.

### 1.1.1 Sentinel

Sentinel[1] technology by Accuware offers a camera tracking system for human tracking, counting, and searching in real-time. They only provide API for developers which has to be integrated into customers' servers. Artificial intelligence and neural network algorithms have been used for the development of their system.

The system can be used for queue length monitoring, analyzing people's behavior, people searching, people counting and people tracking. They also provide cross-camera re-identification of people. The recognition of a person is done by assigning him a unique person ID and a feature vector for encoding his appearance. The actual detection crop is saved into the database along with additional information as a tracklet with a unique tracklet ID. Feature vectors are used for the re-identification across multiple cameras (see Figure 1.1).

It is also possible to define geo-fences - imaginary lines. This can be used for detecting people in restricted areas or monitoring dwell times in some areas. In case of breaching the defined fence, alerts can be triggered to highlight suspicious activities.

The output of the system is a CSV file that contains records for each tracklet. Each CSV record includes a timestamp, frame number, person ID, feature vector, and coordinates. CSV files are created periodically according to the settings. These files can be further processed, e.g. the results can be displayed on a heatmap.

### 1.1.2 TimeRethink

TimeRethink[2] is a commercial technology that offers an offline analysis of video footage. Customers send them data and they produce a report according to given parameters. Longer-term cooperation with customers is expected to improve the results of the analysis even more.

[1] `https://www.sentinelcv.com/`
[2] `https://timerethink.com/`

Their service is aimed at business owners, retailers, or hospitals. The video archives' analysis should help businesses increase revenue, minimize loss and theft, or reinforce health and safety measures.

Their analysis includes detecting and tracking objects and people, monitoring employee attendance, recognition of individuals, analyzing the movement of people in some area, or analyzing waiting times. The analysis is performed using unique machine learning algorithms. Screenshots of their system can be found in Figure 1.2.

### 1.1.3 Senstar

Senstar[3] is a commercial company that provides video analysis in real time. The data is collected from public IP camera and analyzed by machine learning algorithms. They offer a large variety of analytical tools, including:

- **Indoor people tracking.** Typical application for indoor people tracking is detecting people in restricted areas, wrong-way detection, or customer behavior analysis. It is possible to draw an alarm zone and an alarm is triggered in case someone is detected within the area. The results of the analysis could be further processed into heatmaps to visually display patterns.
- **Outdoor people and vehicle tracking.** The main advantage of outdoor tracking is that it ignores changes in the scene caused by vegetation movement, shadows, rain, or snow. As well as indoor people tracking, it can be used for wrong-way detection or to avoid break-ins.
- **Crowd detection.** Monitoring crowds is essential to ensure security at public places such as subways or shopping malls. It is also possible to monitor occupancy levels and staff can be notified when the limits are exceeded.

The analysis is based on advanced machine learning algorithms and can be run locally on each camera, centrally on a video server or hybrid models are also possible. Screenshots from their products are shown in Figure 1.3.

[3] `https://senstar.com/products/video-analytics/`

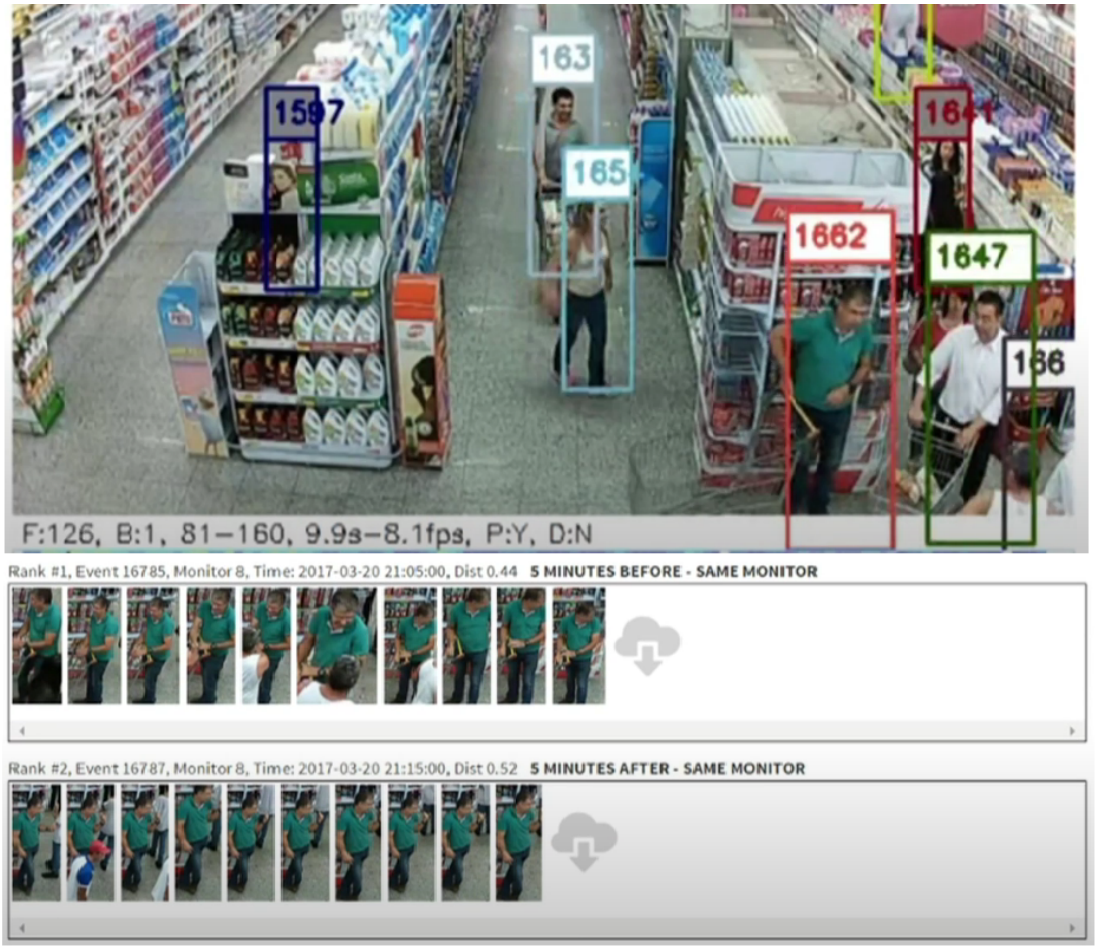


Figure 1.1: An example of the Sentinel system used in a supermarket. Each recognized person has a unique person ID. Below are displayed search results for a man with ID 1662. Screenshots from `https://www.youtube.com/watch?v=nuhBnlHKAK0`.

Figure 1.2: Screenshots showcasing the possible usage of the TimeRethink technology from their website. The first picture shows the detection of people while recognizing them by name. The second one analyzes the cleaning schedule of the given area. The third one is a heatmap based on the movement of people.

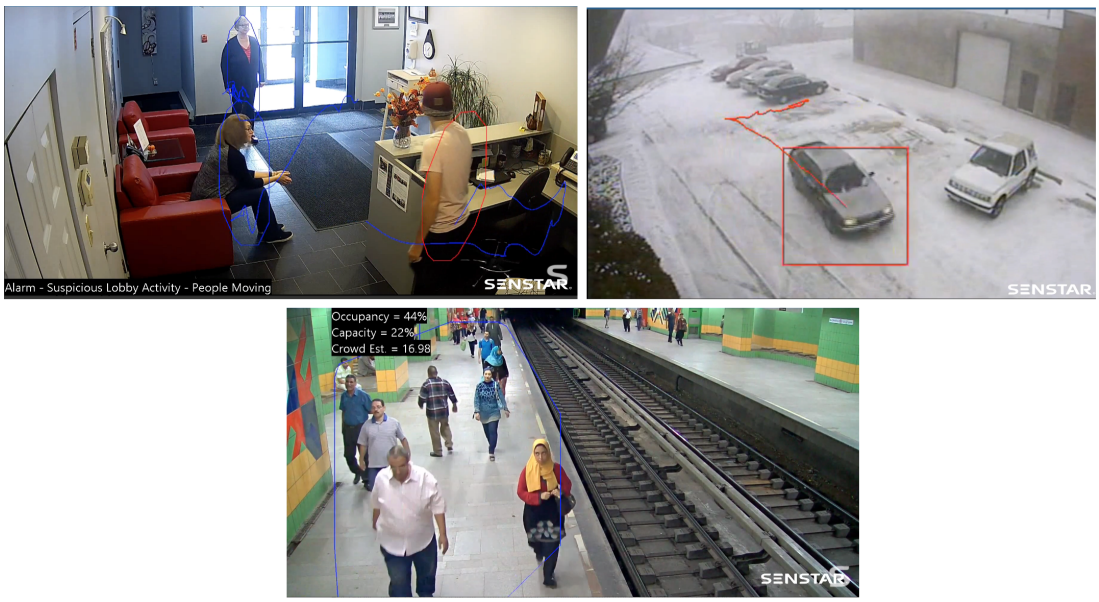


Figure 1.3: Screenshots of the Senstar products from the promotional videos found on their website. The first picture shows the usage of the indoor people tracking along with an alert set to trigger when movement is detected. The second one shows vehicle tracking. The last one demonstrates crowd monitoring in a subway.

## 1.2 Non-commercial projects

The other part of our findings were non-commercial projects. These were most often funded by European Union or similar institutions.

### 1.2.1 P-REACT

P-REACT[4] (Petty cRiminality diminution through sEarch and Analysis in multi-source video Capturing and archiving plaTform) project [5] is a video surveillance system. The project was partially funded by the European Union's Seventh Framework Programme for research, technological development and demonstration. Its development lasted 2 years and was finished in 2015.

The project's main objective is to develop a low-cost and effective surveillance system that analyzes footage and audio to detect petty crimes. It has modular architecture consisting of local embedded service and cloud service, including an interface for end-users and a mobile app. Video analysis is done on both, embedded and cloud levels, and it is focused on detecting abnormal behavior such as fighting, chasing, or running. Moreover, the video analysis results are enhanced by an audio analysis. Audio recordings cover a 360-degree area which means that surveillance system coverage can be extended beyond a camera's field of view.

The analysis is performed in two steps: motion is detected first and based on certain parameters, it is evaluated whether it is unusual or not. If the activity is considered unusual, an alert is triggered. The system also generates evidence for these situations that can be viewed by the operators of the system. A screenshot of the P-REACT user interface is shown in Figure 1.4.

### 1.2.2 SAVASA

The main goal of the SAVASA (Standards Based Approach to Video Archive Search and Analysis) project [6] is to develop an interactive video search platform. It was also funded by the European Union and the project itself took 2.5 years to develop. The project participated in the TRECVid 2012[5] in the task of the interactive surveillance event detection (SED).

The core of the system is formed by algorithms for detecting and tracking objects, scenario recognition, and event detection. Three main events are supported - ObjectPut, PersonRuns, and Pointing. These events are identified by two methods. The first one is based on descriptors from motion trajectories. The second one use region-based identification with two different configurations.

The algorithm also predicts the most probable place where some event may occur and create a heatmap accordingly.

The end-user interface (see Figure 1.5) allows to place queries over the video data and filter the results by different parameters like confidence, level of motion, or number of people. The results are displayed as animated GIF file pictures.

Unfortunately, the system proved to be very slow for practical use. On the bright side, it provided authors a new direction to follow in the future.

[4] `http://p-react.eu/`

[5] `https://www-nlpir.nist.gov/projects/tv2012/index.html`

### 1.2.3 VICTORIA

VICTORIA[6] (Video analysis for Investigation of Criminal and TerrORIst Activities) [7] is a project devoted to reasonably speeding up video analysis for Law Enforcement Agencies.

The system has a modular architecture with one core module which interconnects all the other modules and data storage. It is designed to be able to support an increasing number of videos - it uses so-called big data technologies. The processed video is divided into smaller parts which are processed in parallel.

For object detection, several complementary approaches for the deep neural network called YOLO have been used. The YOLO detector has been re-trained and optimized to detect criminal activities on the modified COCO dataset with added 81 new classes of weapons.

The VICTORIA system should also support multi-class multi-target tracking of detected objects across different frames. Since most deep neural networks are dedicated only to single object tracking, multi-target tracking is hard. Only little work has been done since the authors encountered several difficulties, e.g. the lack of relevant training data. They employ two-tracker modules - one is dedicated to tracking people while the second one tracks cars. Rather than developing a new multi-class multi-target tracker, they run several instances of the same single-class tracker module with different parameters. For a practical example, see Figure 1.6.

Both object detection and target tracking are integrated into the Connected Vision modular framework [8] which allows processing video analysis tasks distributively. Each module is an independent web service that collects and processes data. The system can process real-time camera feed as well as already recorded videos.

[6] `https://www.victoria-project.eu/`

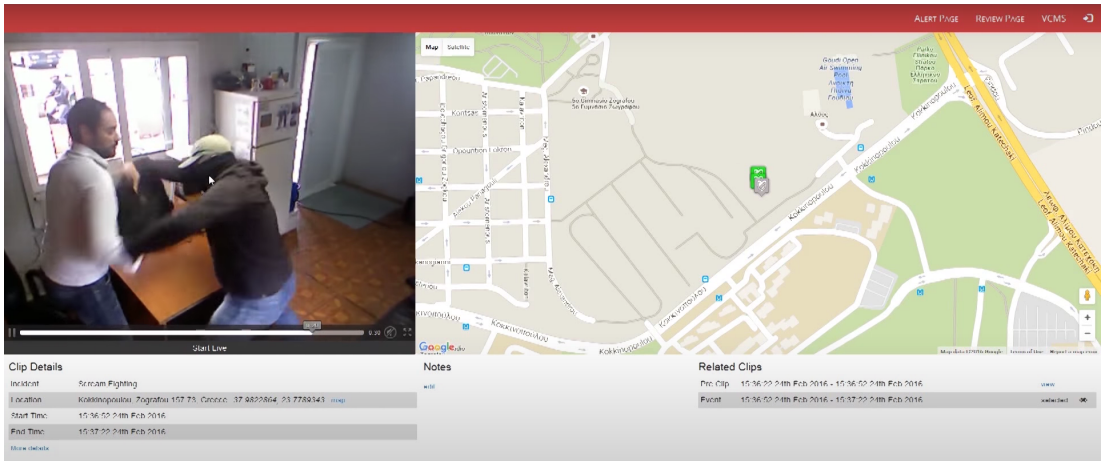


Figure 1.4: A screenshot of the P-REACT user interface shown in their promotional video, showcasing fighting indoor detection. An alert is sent to the operator of the surveillance system who decides what to do next. The original video can be found at `https://www.youtube.com/watch?v=ktCY87EwjF0`.

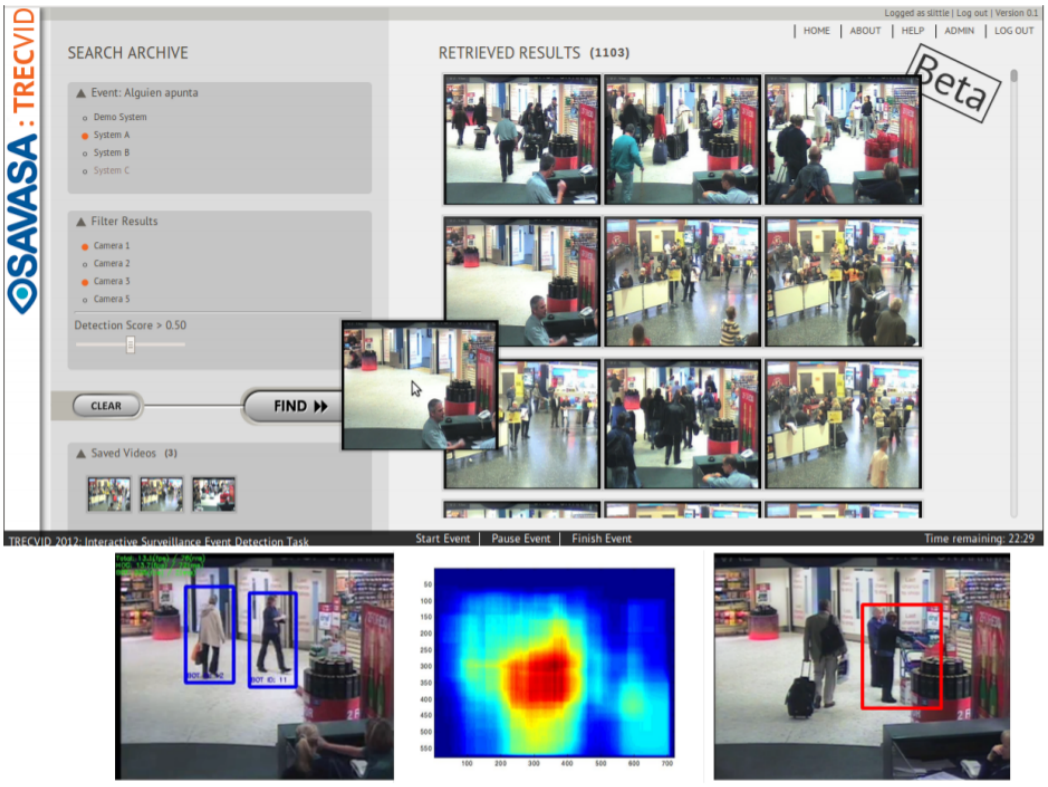


Figure 1.5: A screenshot of the SAVASA search user interface is displayed at the top while the bottom pictures show examples of object tracking, heatmap, and Pointing event detection, respectively. Source: [6].

Figure 1.6: Figure showing one use case of the VICTORIA system - tracking a person that carries some object of interest. The first picture is the queried person, the second one is the results of the query - the same person in different places, tracked by multiple cameras. The original picture can be found at `https://www.victoria-project.eu/fileadmin/websites/victoria/documents/180517-victoria-flyer_6-faces_Final.pdf`.

# 2. Videolytics overview

The Videolytics[1] system was created in 2019 at Charles University by a group of students under the leadership of prof. RNDr. Tomáš Skopal, Ph.D. It is still under development at the time of writing this thesis.

The goal is to provide a complex open-source application for the analysis of video streams. Unlike other similar systems, Videolytics does not rely on large neural networks focused on a specific problem. Instead, it uses pre-trained detectors and hand-engineered visual descriptors to build a hierarchical structure of models to extract features of different types of abstraction [9].

## 2.1 Framework

The Videolytics software employs a bottom-up design. This vision framework for feature fusion as described in [10] is constructed in multiple, hierarchically structured layers. Each layer fuses lower-level features into higher-level features.

The lowest layer denoted the *L0 model* consists of two sub-modules. Firstly, the extraction of learned features (e.g. bounding boxes) from detected objects by deep convolutional neural networks. Secondly, other features that can be extracted from the raw data (e.g. color histograms).

The layer above the *L0 model* fuses features from the *L0 model* into higher-level *L1 model* features (e.g. detections are aggregated into trajectories). This process can be done recursively until we reach the top layer, which represents the final analytics over features.

All features are stored in the central database which also serves as a source for the extraction of higher-level features. The full schema is illustrated in 2.1.

This approach has several advantages. It is easily extensible, and a single pipeline that fuses features based on previous features is enough to build any level of abstraction we need. Unlike big neural networks, the cost of this process can be considerably lower.

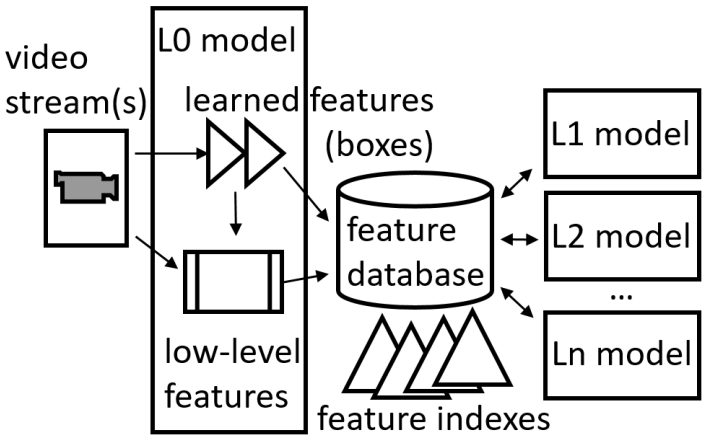


Figure 2.1: Videolytics vision framework for feature fusion architecture.

[1] http://videolytics.ms.mff.cuni.cz/stream.html

## 2.2 Database

A crucial part of Videolytics is its central database. It is a PostgreSQL[2] open-source relational database with native support for geometric types and functions.

It consists of more than twenty tables. Every table has a single-column primary key and can have foreign keys.

We will briefly describe tables that are essential for our analytical module:

- **Frame.** Table with information about all frames.
- **Detection.** Table of all detections from all frames, consisting of identification numbers, classes, coordinates, crops, confidence numbers, features, and frame identification numbers.
- **Traj_detection.** Table of assignments of detections into trajectories.
- **Traj.** Table holding detections while grouping them by trajectories, in which rows correspond to trajectories. Each row contains a set of detections bound to each trajectory.
- **Traj_model.** Table of all trajectories models, having IDs, descriptions, and feature types.

The schema of the database is shown in Figure 2.2.

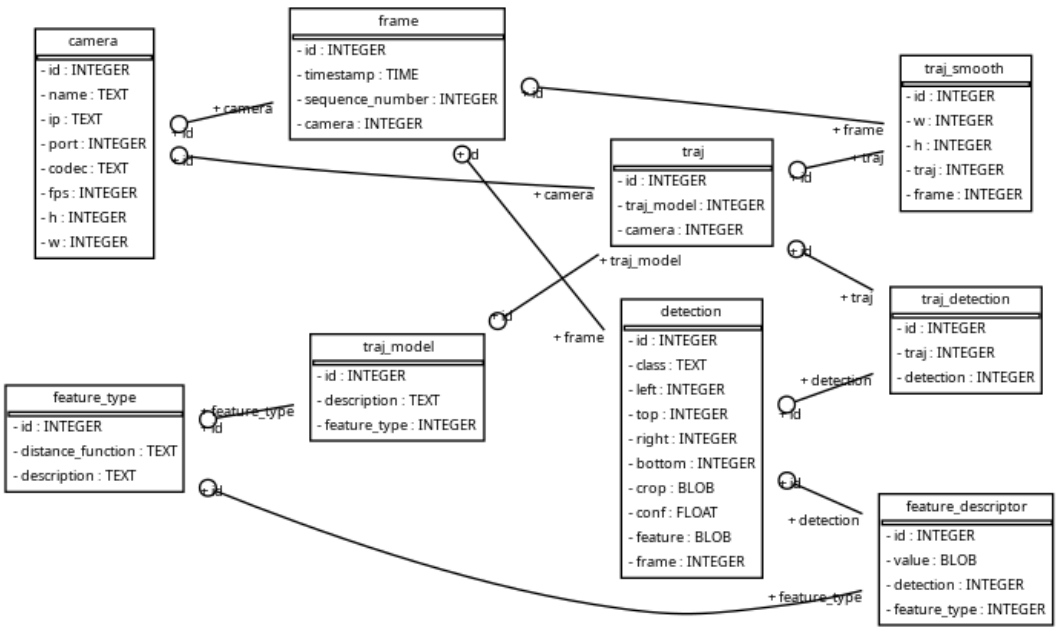


Figure 2.2: The ER schema of the database.

[2] https://www.postgresql.org/docs/

## 2.3 Object detection

An essential part of video analysis is object detection. As we mentioned in the introduction, object detection involves locating and determining the class of an object within an image. Detection, therefore, consists of two parts - the *object class* itself and the so-called *bounding boxes*, rectangles that define the space where the object in the image is located. Algorithms for detecting objects are called *object detectors*.

Over the decades, object detectors based on deep neural networks have taken a big leap forward. Initially, *two-stage detectors* had been used. The neural network first proposed regions of interest - a section of an image, where an object could potentially be found. After that, the regions were processed by the image classifier. A few years later, *one-stage detectors* were discovered. This approach skips over the first step. Regions of interest are no longer precomputed and the detector runs on the whole image, exploring all different possibilities of the object's positions [11].

Because Videolytics should be able to process video streams in real time, the choice of detector is very important. There are only a few detectors that meet our criteria. The most famous are YOLO (You Only Look Once) [12] and SSD (Single Shot Detector) [13].

Videolytics uses Single Shot MultiBox Detector. It is a one-stage detector, which is noticeably faster, and the accuracy of detections is higher. Detailed description and analysis can be found in [14].

## 2.4 Modules

Videolytics consist of five modules, namely:

- **LivED.** Live (realtime) entity detector from streamed video.
- **TrajAn.** Trajectory analyzer.
- **ReID.** Object/person reidentification module.
- **WebClient.** Video stream management, visualization of features from the database and management of server processes, and other modules.
- **Analytical Module.** Analytics over data in the database.

Each module (except the analytical one) is independent of the others, and they are connected via WebClient. All the modules are communicating with the central database. Thanks to this modular approach, the system is easy to maintain and each part of it can be optimized effectively.

### 2.4.1 LivED

The LivED module aims to implement an efficient pipeline for detecting objects in real-time using an already trained SSD detector. Frames of the video on the input are decoded, colors are converted to a suitable format and the data is passed to the detector. Processed data are then written to the database. It uses

a generic approach and can be easily adapted to newly added functionality. For more information about the implementation please refer to [15].

### 2.4.2 TrajAn

TrajAn is a module that aggregates detections into trajectories by applying various heuristic methods. The decision-making process is statistical, based on computing weights and confidence that the detection belongs to the given trajectory. Centroids are generated from the detections, depending on which it is decided whether the given detection belongs to the trajectory or not. Frame IDs and directions are also taken into account.

### 2.4.3 ReID

The main task of the re-identification module is to identify the same entities across multiple frames and cameras. For this purpose, it uses already generated detections - cropped image data of detected entities stored in the database. For more details about the implementation please refer to [16].

### 2.4.4 WebClient

The goal of the WebClient is to provide a web client module that manages video stream, visualizes data from the database, and manages server processes and other modules [17]. It serves both as the main access point for users as well as the basis for all extension modules, which are integrated into its codebase. A screenshot of the WebClient app is shown in Figure 2.4.

It runs on the Apache HTTP Server[3] and server-side processes are managed by PHP[4] scripts. The client-side app is written in JavaScript and consists of the main class App and multiple manager classes, including:

- **Detection manager.** Manages detections fetching and drawing.
- **Detection color manager.** A part of the detection manager which manages the choice of colors for different classes.
- **Trajectory manager.** Manages trajectories fetching and drawing.
- **Stream manager.** Manages initialization, starting, or stopping of the stream.
- **Button manager.** Manages button elements on the website.
- **Analytics manager.** Manages the whole analytical module. It is a part of the analytical module.

The Videolytics WebClient app supports two video playback modes, differing in how the generated detections and trajectories are handled:

[3] `http://httpd.apache.org/`
[4] `https://www.php.net/`

- **Offline mode.** Videos are stored on the server disk from which they are run at the user's request. Moreover, they have been already processed by the object detection module, and learned features along with detections are stored in the database. If trajectories are available, they are stored in the database as well.

- **Real-time simulation mode.** Even though the videos are still stored on the server disk, the application treats it as it is an actual stream happening in real-time. Detections and trajectories are not exported from the database but are generated at the moment of playback. Even though they are stored in the database, they are removed after the user leaves.

### 2.4.5 Analytical Module

Once detections and trajectories are all stored in the central database, it is desirable to be able to further process and filter these based on additional rules and restrictions. Additionally, we prefer to construct these filters visually via a GUI rather than writing text-based queries, such as database requests. Finally, we would like to also visualize the results in a user-friendly format.

The analytical module of Videolytics software aims to tackle exactly this problem by offering a specialized GUI for configuring and sending queries to the database as well as rendering the responses (see Figure 2.3).

The analytical module (Analytics tab) is a part of the WebClient described in 2.4.4. GUI of the analytical module allows user to:

- Place visual queries by drawing spatial predicates (geometrical shapes).

- Specify their parameters (detected objects classes, time interval...).

- Visualize results in different forms (animation, heatmap...).

Because the analysis is performed on the data in the database, it is available only in the Offline mode of the WebClient. The analytical module will be described in more detail in the following chapters.


(a) (b)

Figure 2.3: An example of a detection query. The image on the left shows a screenshot from a stream while the image on the right shows detection crops as query results.

## 2.5 Use cases

Videolytics can be used in a variety of situations where public cameras can be used, including monitoring suspicious activities, monitoring crowds, or theft prevention.

Thanks to the analytical module, the range of practical applications expand even more. For example, it is possible to analyze human behavior, movement of detected objects in time, and more. Practical use examples of the analytical module and more use cases are described in the Chapter 5.

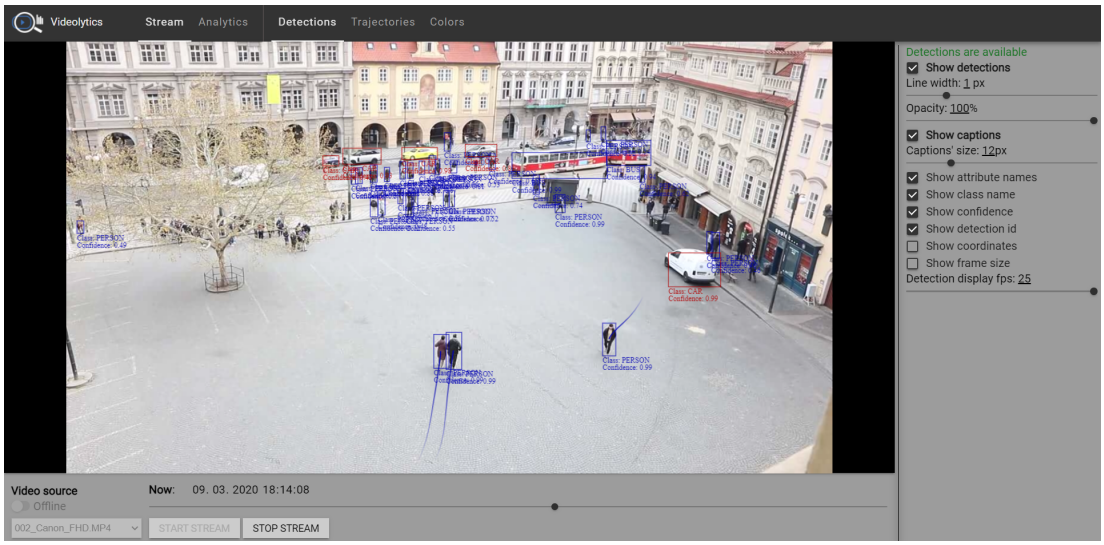


Figure 2.4: GUI of the Videolytics WebClient app, displaying precomputed detections and trajectories while playing selected video.

# 3. Input interface

The analytical module's input interface allows users to visually define queries by drawing multiple types of geometrical shapes - we will refer to them as *spatial predicates*. The query can be specified in more detail by choosing desired output arguments, object detection classes, or time interval. A screenshot of the whole input interface is shown in Figure 3.3.

In the following sub-chapters, we will describe the operation of the user interface step by step.

## 3.1 Video selection

Firstly, a video must be chosen. Currently, 8 sample videos are listed and more are expected in the future. After clicking on the name of the video, a reference frame is loaded from the server, which serves as a guide for drawing predicates and showing results.

## 3.2 Query type

The analytical module works in two modes:

- **Detections mode.** The results of the queries are detections.
- **Trajectories mode.** The results of the queries are trajectories.

Modes can be switched using radio buttons. Visual comparison between these two modes can be found in Figure 3.1.

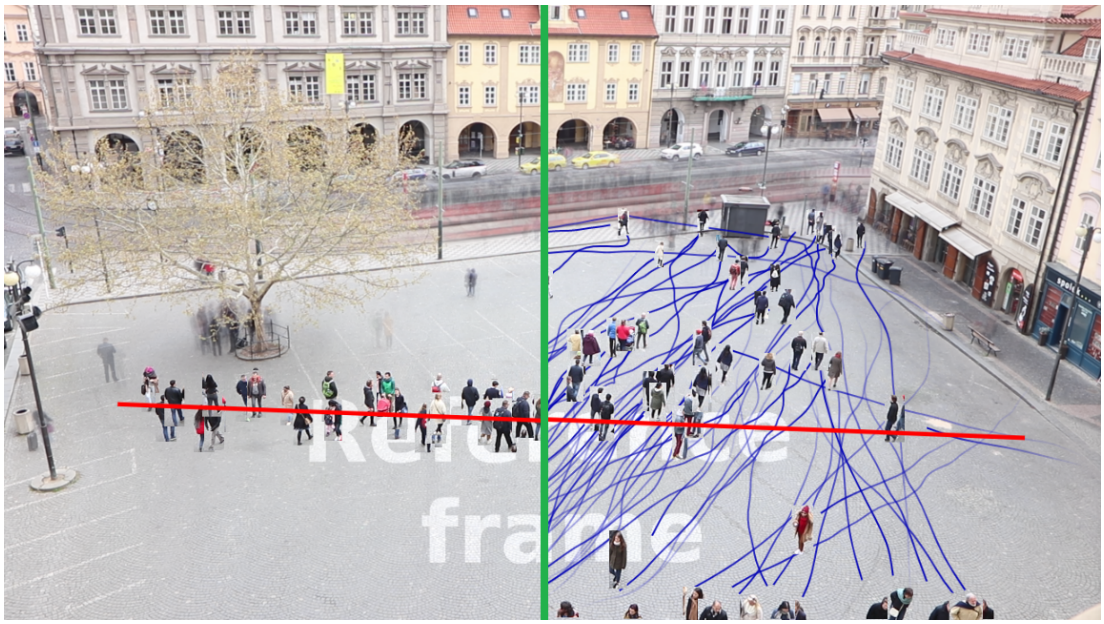

Figure 3.1: Visual comparison between the results of the detections (left) and the trajectories (right) mode.

## 3.3 Entering predicates

Each query consists of spatial predicates - basic building blocks of the analytical module.

### 3.3.1 Drawing predicates

The first step in entering a visual query is to draw a spatial predicate. One can choose from multiple shapes - lines, polygons, or circles. A directed line is also supported in trajectories mode.

Drawing is started by clicking on the reference frame image. Locking angles is possible by holding the Shift key while dragging the mouse on the image.

New predicates are automatically listed and selected in the predicates table. Selected predicates are highlighted with a different background color. Selecting multiple predicates can be done by clicking while holding the Ctrl or Shift key.

### 3.3.2 Removing predicates

Selected predicates can be removed by clicking on the *Remove selected predicates* button. To delete all predicates, click on the *Remove all predicates* button.

### 3.3.3 Actions

Each predicate has its own *action*. The goal of action is to define the role of the predicate in a query. Available actions depend on the mode of the module and drawn shape.

Available actions in detections mode:

- **Contains.** Return all detections contained in the given predicate.
- **None.** Given predicate will not be used in a query.

Available actions in trajectories mode:

- **Intersects.** Return all trajectories that intersect the given predicate.
- **Contains.** Return all trajectories that are contained by the given predicate. Not available for a line.
- **Go_in.** Return all trajectories that start outside the predicate and end inside. Not available for a line.
- **Go_out.** Return all trajectories that start inside the predicate and end outside. Not available for a line.
- **Active.** Action associated with directed line, querying all trajectories in the given direction.
- **None.** Given predicate will not be used in a query.

### 3.3.4 Combining predicates

The next step is to decide how drawn predicates should be combined. This is done by setting an appropriate logical operator. An example is shown in Figure 3.2.

By default, *and* (logical conjunction) is used, meaning that the results should include detections/trajectories in the intersection of the predicates.

It can be switched to *or* (logical disjunction), representing a union of the results from every predicate.

(a) Predicates combined with **and**.

(b) Predicates combined with **or**.

Figure 3.2: Examples of the same predicates but combined with different operators.

## 3.4 Query parameters

Next, we define query parameters. By defining parameters, we specify more precisely what interests us - classes of detected objects, time interval, or columns from the database that should be retrieved.

### 3.4.1 Choosing select arguments

By select arguments, we mean names of columns from the database. In the case of only showing the number of results, it is sufficient to check the **Count** checkbox element. Otherwise, one can choose which pre-defined argument should be included in the results:

- **Class.** Show classes in the results.
- **Frame ID.** Show IDs of frames.
- **Timestamp.** Show timestamp.
- **X.** Show x coordinate of detection centroid.
- **Y.** Show y coordinate of detection centroid.

Since trajectories are made of thousands of detections, it is impossible to return information about all detections. Therefore in the trajectories mode, only class argument is supported. On the other hand, trajectory model, first frame and last frame are returned automatically for every trajectory query.

The choice of arguments is particularly important if the results are to be further processed.

### 3.4.2 Choosing classes

By classes, we mean the class of detected objects. In this part, the user can choose which classes are relevant for his query and select them accordingly.

Common classes used in our system are:

- **Person**
- **Motorcycle**
- **Car**
- **Truck**
- **Train**
- **Bicycle**
- **Bus**

At least one class must always be selected to create a valid query.

Please note that classes are dependent on the video. Checkbox with class options is dynamically loaded when selecting a video.

### 3.4.3 Time interval

Lastly, it is possible to determine the time interval by specifying the start and end time by dragging sliders on the HTML range element.

## 3.5 Database communication

Because we are focusing only on the user interface, we do not work directly with the database. Query generation and execution are performed by scripts written by another student. Therefore, we will not describe generating queries or technical details of communication with the database.

### 3.5.1 Passing data to the database

After clicking on the *Generate & Execute query* button, the internal state of the module is updated and sent as a JSON to a script generating a query.

### 3.5.2 Prefetch mode

Due to long response and downloading times in case of bigger queries, we added a functionality called prefetch mode, which is used to fetch all the necessary data instead of requesting it on demand. This results in a longer initial waiting time but enables the user to use the requested data without any delay once it is downloaded and saved into memory.

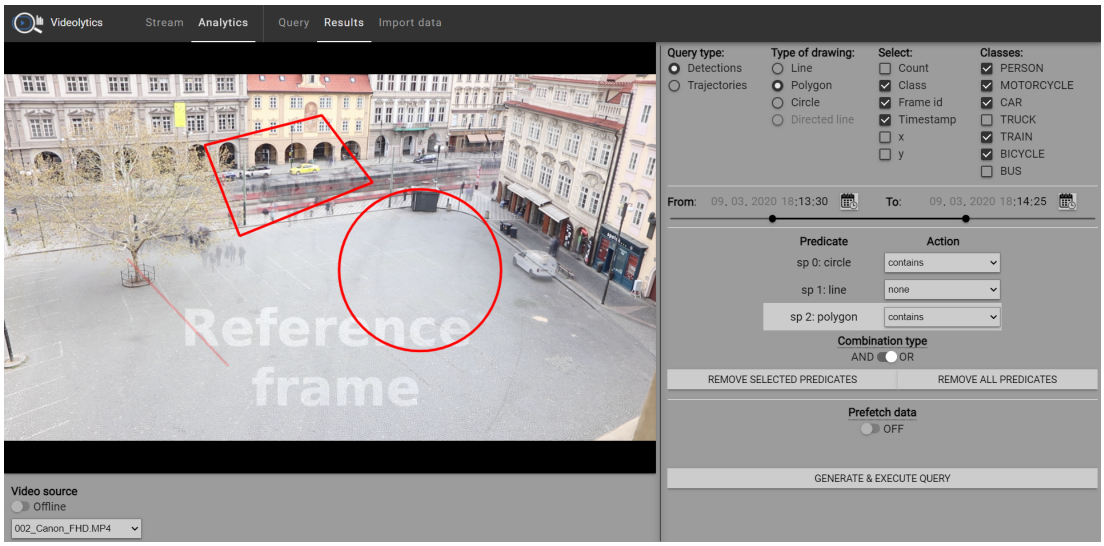


Figure 3.3: Whole input interface of the analytical module embedded into the Videolytics WebClient.

# 4. Presentation interface

The more interesting part of the analytics module is the visualization of query results.

After loading results, a generated query is displayed in the textbox area and results are listed in the results table. A screenshot of the whole input interface is shown in Figure 4.3.

## 4.1 Draw results mode

The first step is to determine in what form the results should be displayed. There are two options:

- **Draw bounding boxes**. While drawing only detection bounding boxes is easy and less data demanding, it does not provide all the information about given detection. However, in some situations it is enough e.g. if a user is only interested in the placement of detected objects.

- **Draw actual crops**. Drawing actual crops of detected objects is more complex and provides additional information about the appearance of the object. On the other hand, it demands more resources to store all the information.

One can choose whether actual crops from the original video or only bounding boxes of detections should be drawn by setting the *Draw crops* switch accordingly. For visual comparison see Figure 4.1.

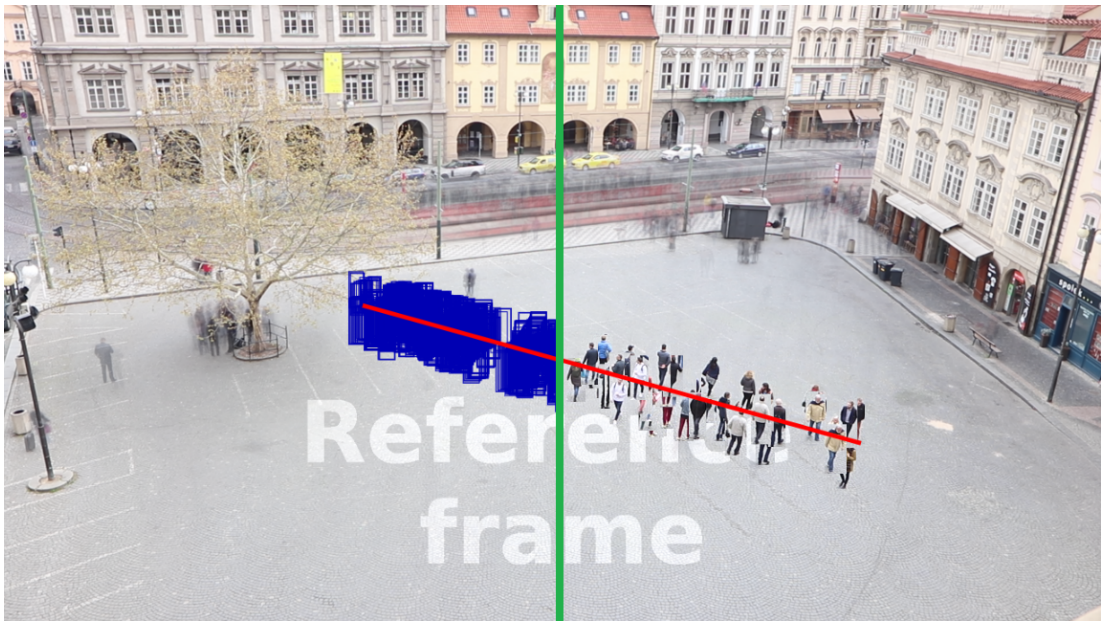


Figure 4.1: Difference between the crops mode (left) and the bounding boxes mode (right) for detections.

## 4.2 Results visualization

There are several methods to visualize the results. These methods can be static (drawing results on canvas) or dynamic (animation). Supported visualization methods are described in the following sections.

### 4.2.1 Visualization of all results

Visualization of all results is done by clicking on the *Draw all results* button. If crop mode is off, all of the bounding boxes are drawn on top of each other. Because the number of results is usually high, it is more efficient and visually pleasing not to draw everything in the crop mode. That's why users can choose a maximum number of crops to draw, ranging from 10 to 500. Then a representative set of crops is created and only these crops are drawn.

### 4.2.2 Visualization of selected results

It is common for a user to want to see only some specific results. Hence clicking on the results table row automatically plots the given result. Multiple results can be drawn at the same time by holding the Shift key. Again, the appearance of the drawn result depends on the crop mode turned on/off.

### 4.2.3 Clearing drawings

Drawn results can be removed by clicking on the *Clear drawings* button.

### 4.2.4 Animating results

Another form of visualization of results is its animation - creating the impression of movement from static pictures. As these pictures should create a comprehensive animation, it is necessary to order them by the time of appearance (in our case the ID of the frame) and in the trajectories mode by trajectory IDs as well.

### 4.2.5 Results aggregation

So far we have described drawing of the individual detections/trajectories based on the data from the database.

It is also possible to aggregate the results in some way and create new information about the data. One example is a *heatmap* - an image that uses different color intensities to show frequencies of an event.

Heatmap is a great tool to get an idea of the density and way of moving entities within the video. In combination with spatial predicates, it creates a very powerful tool for quick and easy analysis of specific areas or time slots on video. In addition, the heatmap is transparent, i.e. it provides this information based on a reference frame, thus providing an intuitive view of the situation without restrictions. For example, one can immediately see the busiest sections of the square (see Figure 4.2), the most visited restaurants, or the most attractive monuments, without any additional data analysis.

Heatmaps for both detections and trajectories are supported and can be drawn by clicking on the *Draw heat map* button. In addition, a heatmap with a reference frame can be exported.

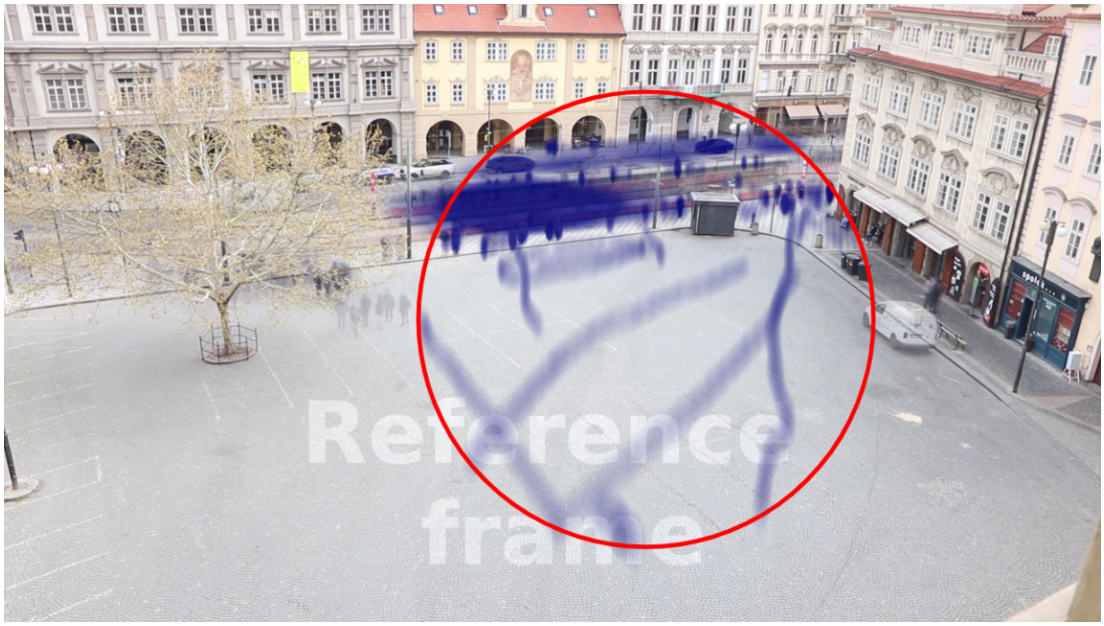


Figure 4.2: An example of a detections heatmap. The highest density of detections is at the tram stop.

## 4.3 Exporting results

We are aware that our analytics module does not provide answers to every users' question. That's why it could be useful to export results and process them with external tools. One can choose from three options:

- **Export CSV.** Results are exported as a CSV file. CSV files can be easily imported as SQL tables to a database which creates space for even more complex post-processing of the data.

- **Export JSON.** Results are exported as a JSON file. JSON is another common data format that is supported by many data processing tools.

- **Export image.** Drawn results along with drawn predicates are exported as a PNG file. Can be used as a visual reference of the exported data in JSON or CSV files.

Export of the files is done by clicking on the *Export (desired format)* buttons. Along with the results, a file with metadata containing information about the internal state of the analytical module is automatically exported. This file is used to restore the internal state of the analytical module when importing results.

## 4.4 Importing results

Importing results can be useful when re-examining the results of a query that has been generated in the past or visualizing results that have been post-processed by external applications.

There are two ways to import results:

- **Import results only.** If only the results file is uploaded, it is not possible to determine for sure what type of results it contains, and also it is not possible to draw predicates that belong to the results. If one does not supply the metadata file, the video from which the results are must be selected manually.
- **Import results along with metadata.** When importing metadata along with results, metadata makes it possible to redraw predicates of the results query. Examining the imported results is therefore much more enjoyable.

Data can be imported by clicking on the *Import data* tab in the Analytics menu.

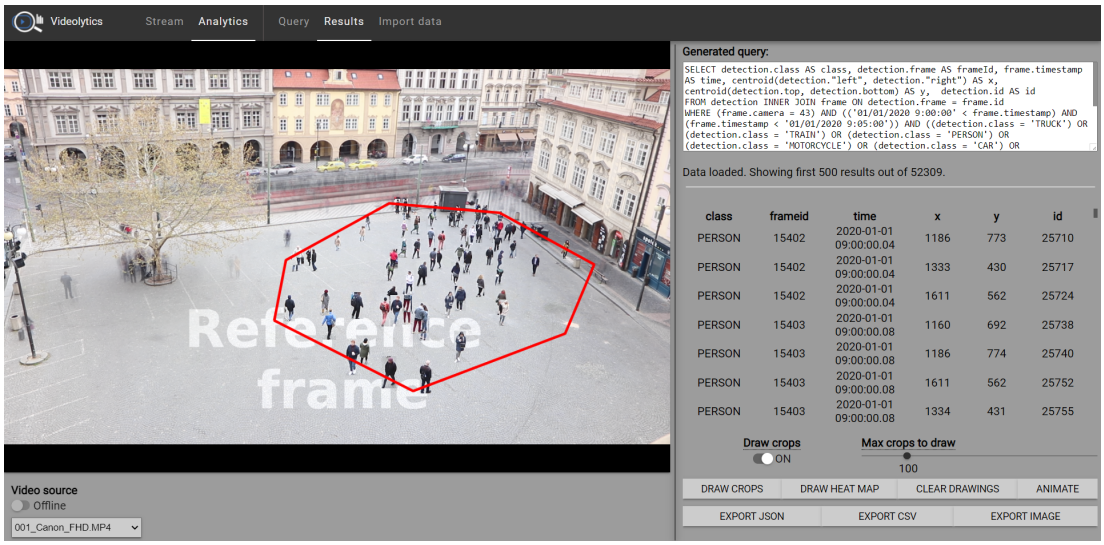


Figure 4.3: Whole presentation interface of the analytical module embedded into the Videolytics WebClient.

# 5. Usage evalation

In this chapter, we will demonstrate the practical use of our analytical module on example queries.

## 5.1 People crossing a square

In this example, we will show multiple queries regarding people crossing a square.

### 5.1.1 Trajectory count query

We start with a simple question: **How many people cross the square in the time interval of three minutes?**

#### Placing a query

As described in Chapter 3, firstly we have to select a video. We chose a video that was filmed in Malostranske sq., Prague. After the reference frame has loaded, we can start drawing predicates. In our case, only one single **line** across the square with **intersects** action will do the trick.

Next, we set parameters for the query. Because we want to know how many people crossed the square, simple detections do not provide enough information and we have to use **trajectories**. Since we only want to know the number of people, we check **count**. We are interested in trajectories that belong to people. That's why we check only **person** class. Lastly, we set the time interval from 10:00 to 10:03 as the required three-minute interval.

The parameters of the query are now set and we can proceed to its generation and execution.

#### Analysis of results

After the query is generated and executed, the given query is displayed in the textbox area and the results are listed below.

In our case, the answer to our query is number 64. We can conclude that during the three-minute interval, specifically from 10:00 to 10:03, 64 people crossed the square. Parameters and results are displayed in Figure 5.1.

### 5.1.2 Trajectory visualization query

We have already found out that 64 people crossed the square. Now we are interested in the **visualization of the actual trajectories.**

#### Placing a query

To be able to draw trajectories, we need to uncheck the count checkbox and check **class** checkbox. Then we can generate and execute this new query.

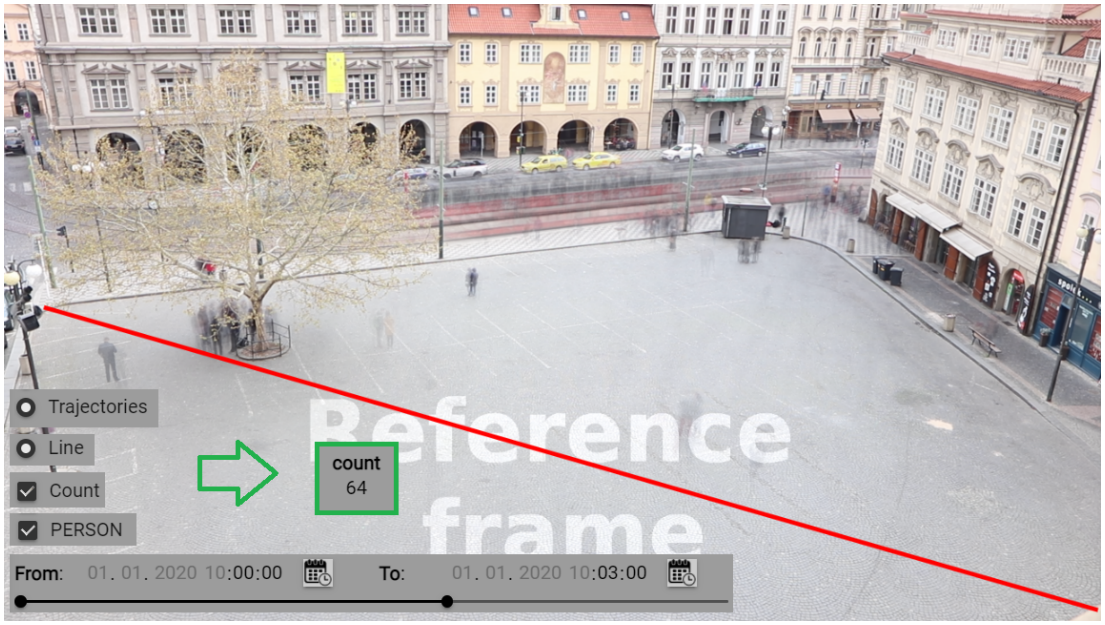


Figure 5.1: Visualization of query parameters and the results for the number of people that crossed the square.

### Analysis of results

Now we will focus on what can we do with the listed results.

- **Visualizing given trajectories.** Firstly, we will examine individual results separately by clicking on their row. That results in selecting the given trajectory and its visualization.
- **Animation.** One might ask whether the trajectories can be visualized more interactively. The answer to that question is animation. By clicking on the *Animate* button, an animation is played based on the selected results.
- **Visualizing all.** After that we would like to display all trajectories by clicking on the *Draw all results* button. All results of our query are displayed in Figure 5.2.
- **Aggregating results.** After we have seen all trajectories, we might ask questions about the patterns that people make while walking. **Is there any spot they stop on for a while? Which part of the square will most people walk through?**

  Answers to these questions can be provided by a **heatmap**. A screenshot of the described heatmap can be found in Figure 5.3. In our case, we can see that multiple people stayed for a while near the tree on the left side. In the screenshot, this place is marked with a green rectangle. From the heatmap we can conclude that people tend to pass the square uniformly.

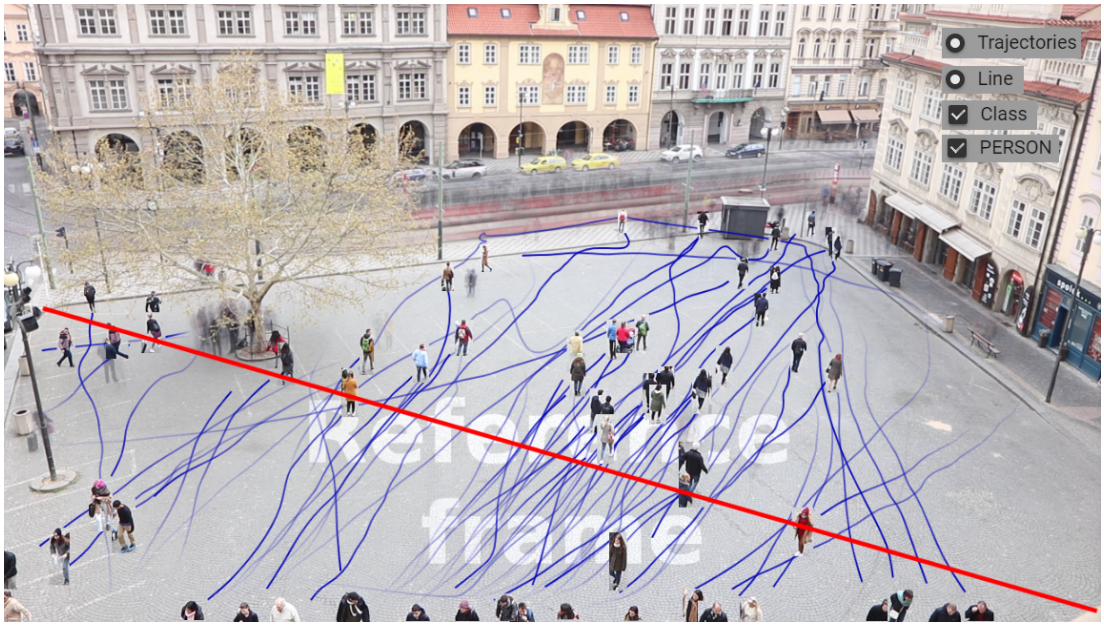


Figure 5.2: Visualization of all trajectories from the results.

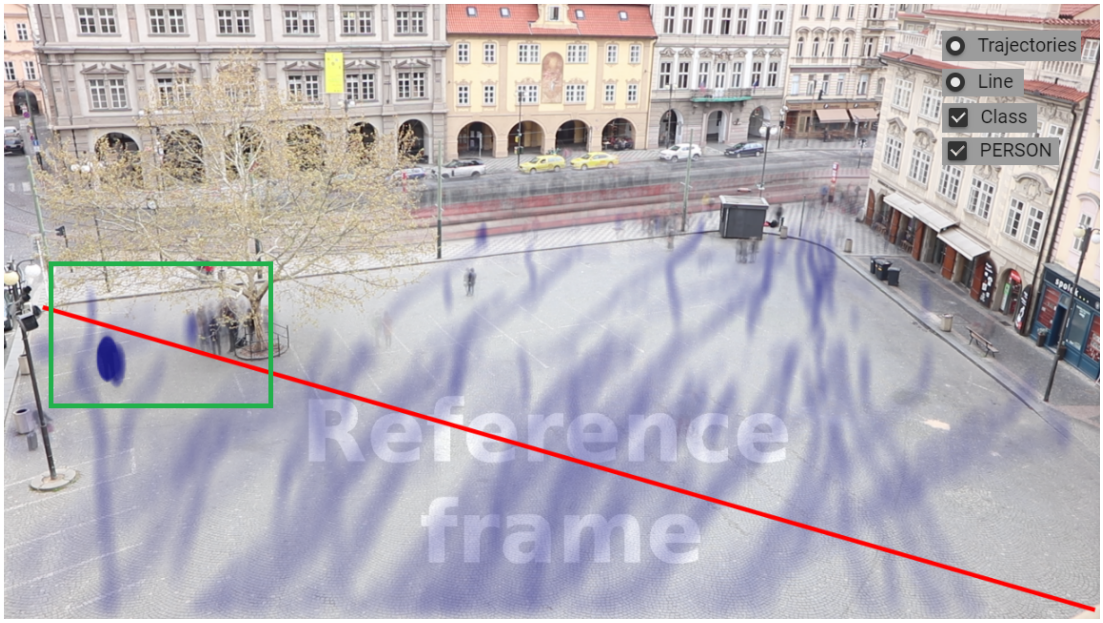


Figure 5.3: Visualization of trajectory heatmap. The green rectangle indicates an area where people often stop.

### 5.1.3 Directed trajectory query

For the last query, we want to show **people who cross the square in the middle and walk towards the Charles Bridge.** The Charles Bridge is not in the field view of our camera but we know that it is situated on the right, towards the upper right corner.

#### Placing a query

For this query, we have to delete the line predicate and draw **a directed line** in the given direction and set its action to **active**. All other parameters can stay the same.

#### Analysis of results

As a result, we obtain five suitable trajectories in the direction that are close enough to the drawn predicate line. The results can be examined the same way we did in the previous query. A screenshot of the results is shown in Figure 5.4.

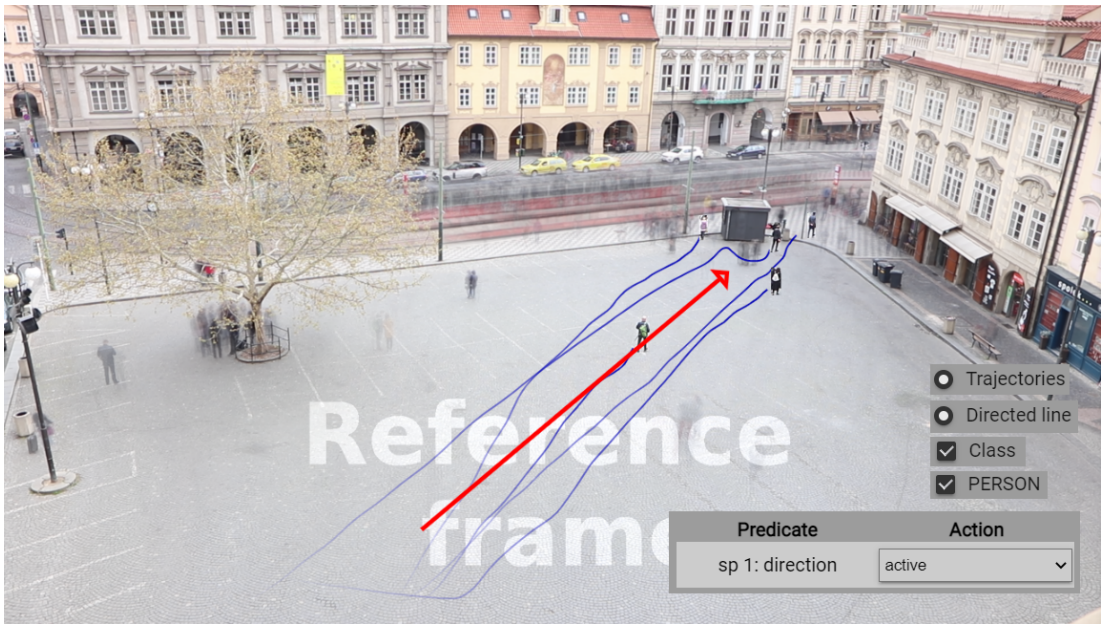


Figure 5.4: Visualization of trajectories in the given direction.

## 5.2 Coffee shop and the tram stop nearby

So far we have demonstrated placing simple queries with only one single predicate. However, it is possible to form more complex queries with more predicates.

Again, we will work with the video of Malostranske sq. This time, imagine that the cafe on the corner of the building on the right required analysis on **how many people get off the tram and go straight to buy a coffee.**

### Placing a query

The answer to this question can be modeled by examining **trajectories**. For this task, we have drawn two predicates. Firstly, a **polygon** which delimits the area in front of the cafe. We want to display trajectories that start somewhere else and end in this area, therefore the action is set to **go_in**. The second predicate is a **line** drawn along the tram stop. We set the action for the line to **intersects**. Because we have more than one predicate, we need to decide which logical operator to use to combine them. In our case, we have to use **and** in order to formulate our query correctly. The procedure for selecting parameters is the same as in the previous queries except that we set the time interval for five minutes (the maximum length of the video).

### Analysis of results

We generate and execute our query to find out that two people meet the requirements (see Figure 5.5). However, we cannot guarantee that these two people really bought a coffee. Ideally, there should be done further analysis from the cameras in the cafe. Regardless it gives us some estimate that the cafe can take advantage of.

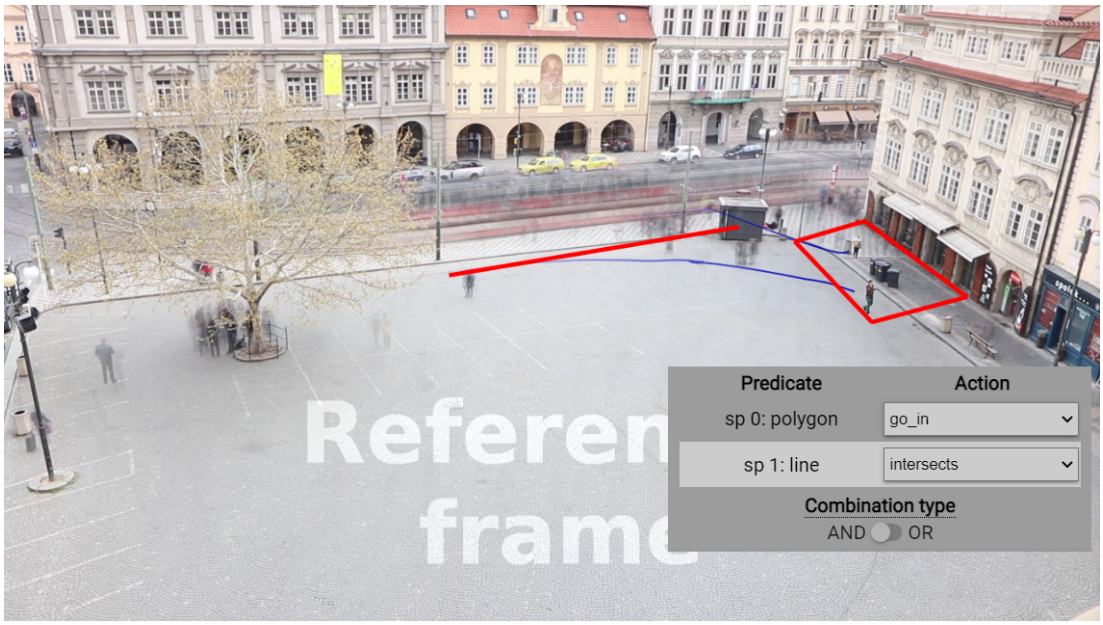


Figure 5.5: Results of the coffee shop query - two people that came from the tram stop to the area in front of the cafe.

## 5.3 A popular time to visit a monument

For the last example, we want to know the busiest times when most people visit a certain monument. In our case, we are interested in **analyzing the number of people detected in front of the Prague Astronomical Clock at different times.** Our analysis will cover only a small time span because we have available only a 30-minute video that was recorded in the evening.

In this example, we will demonstrate data acquisition using the analytical module, their export, and their post-processing by other applications.

### Placing a query

Firstly, we select a video from the Old Town sq., Prague where the Astronomical Clock is located. This time we work with **detections**. We draw a **polygon** in front of the Clock and set the action to **contains**. For parameters, we check all the checkboxes for select arguments and we choose **person** class. Since we want to analyze the busiest times throughout the day, we set the time interval for the entire duration of the video. Visualization of the query parameters is shown in Figure 5.6.

Of course, we could place multiple queries querying the number of people while manually shifting the time interval and writing down the results. However, we assume that we are analyzing a large amount of data and manual interval shifting would take lots of manual work.

### Working with results

After the query is generated and executed, we obtain 215768 detections. We export these results as a CSV file and import the data to an external SQL tool.

We use the following SQL statement which outputs counts of grouped data by **timestamp** in chronological order:

```
SELECT time, COUNT(*) AS count
FROM data
GROUP BY time
ORDER BY time
```

Next, we copy the output of this SELECT statement and process it by Google Sheets[1]. By creating a histogram from the aggregated data, we can easily visually observe at which times does the number of people present peaks, and when, on the other hand, it is the lowest. The histogram is shown in Figure 5.7.

[1] https://www.google.com/sheets/about/

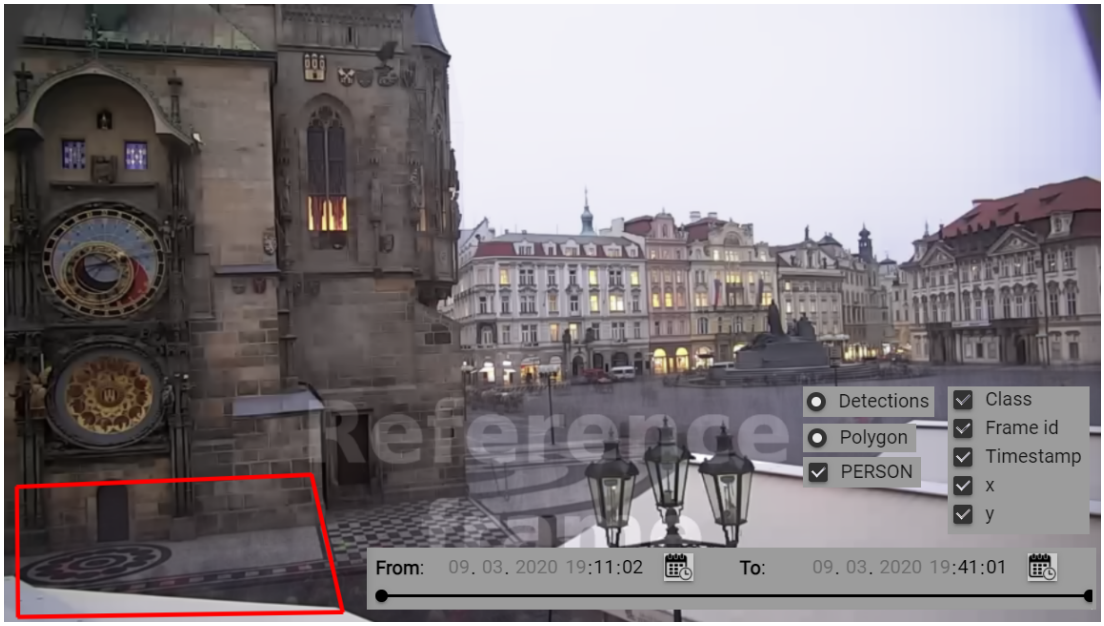


Figure 5.6: The polygon predicate indicating the place in front of the Astronomical Clock and query parameters used for the visit analysis query.

| | time | count |
|---|---|---|
| 1 | 2020-03-09 18:11:02.1652520 | 6 |
| 2 | 2020-03-09 18:11:02.1830150 | 6 |
| 3 | 2020-03-09 18:11:02.1846000 | 7 |
| 4 | 2020-03-09 18:11:02.1861260 | 7 |
| 5 | 2020-03-09 18:11:02.1876000 | 6 |
| 6 | 2020-03-09 18:11:02.1888960 | 6 |
| 7 | 2020-03-09 18:11:02.1900720 | 5 |
| 8 | 2020-03-09 18:11:02.1912280 | 6 |
| 9 | 2020-03-09 18:11:02.9803180 | 7 |
| 10 | 2020-03-09 18:11:02.9815550 | 7 |
| 11 | 2020-03-09 18:11:02.9828160 | 6 |
| 12 | 2020-03-09 18:11:02.9841140 | 6 |
| 13 | 2020-03-09 18:11:02.9852580 | 5 |
| 14 | 2020-03-09 18:11:02.9863410 | 5 |
| 15 | 2020-03-09 18:11:02.9874560 | 6 |
| 16 | 2020-03-09 18:11:02.9885460 | 6 |
| 17 | 2020-03-09 18:11:03.3100460 | 6 |

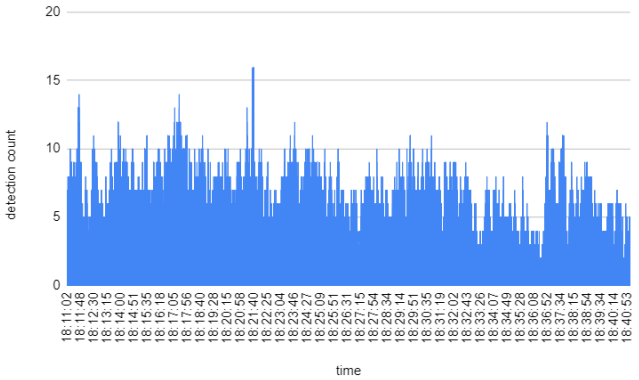


Figure 5.7: Post-processed query results and the histogram based on this data. We observe that the most detections were recorded at around 18:21:33. The hour difference between the web page and the results occurred because the time on the web page is adjusted according to the user's time zone, while the time records in the database are determined by the actual time when the video was recorded.

## 5.4 Other practical use cases

To complete this chapter, we present questions that could be answered either by the analytical module itself or by post-processing the exported results in the SQL environment:

- Which restaurants/bars are the most visited? Which restaurants/bars are the most visited during the evening?
- Which roads/crossroads/traffic lights often form traffic jams?
- Show critical places where traffic accidents often occur (i.e. some cars are not moving while others pass by).
- Are there overcrowded parking lots? Are there unused parking spaces?
- Is there a public transport connection that is useless at a certain stop (i.e. hardly ever someone gets on/gets off)?

# 6. Solution analysis

In this chapter, we will analyze our implementation of the graphical user interface for Videolytics.

## 6.1 Used languages

The graphical user interface of the analytical module is a part of the Videolytics WebClient and shares the same architecture. Therefore the choice of the programming language was pretty straightforward - it is written in JavaScript.

### Javascript

Javascript[1] is a programming language commonly used for developing web pages but it is also used in many non-web applications.

JavaScript runs on the client's web browsers. This reduces the load on the server because many user requirements can be handled by JavaScript, without the need to be sent to the server.

It is an interpreted language - there is no need to wait for the source code to be compiled. Instead, an interpreter in the web browser reads the code and interprets it line by line. In more modern browsers, JIT (Just In Time) compilers are used - the compilation of the JavaScript code is done during the execution of the code.

While no compile time is a huge advantage, interpreted language has its downside - the code cannot be optimized in compile time. Unlike server-side scripting, JavaScript is strongly dependent on the client's hardware. Computationally intensive web pages could become too difficult to handle for the client's machine.

Another disadvantage of JavaScript is that different browsers can interpret JavaScript code differently. To ensure proper execution everywhere, it must be tested on each browser.

### JavaScript libraries

There exist over 1 million libraries which makes JavaScript versatile for every task. In our implementation, we use:

- **jQuery[2].** jQuery is a library that simplifies HTML document traversal and manipulation as well as event handling and much more. It is the most used JavaScript library and it is used on almost every web page.

- **jQuery UI[3].** jQuery UI is a library built on top of jQuery providing GUI (Graphical User Interface) widgets for easy creation of responsive web pages.

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[2] https://api.jquery.com/
[3] https://api.jqueryui.com/

## 6.2 Internal structure

Designing a web user interface means working with various HTML elements - radio buttons, check-boxes, switches, intervals, and buttons. These elements have to react to changes made by the user accordingly. Every change of state fires an *event*. Functions, that respond to events are called *event handlers*.

The Videolytics WebClient consists of one HTML web page. It is built around the main class *App* which has several *managers* - classes, that manage different parts of the web application. A brief description can be found in 2.4.4. We will focus on the *Analytics Manager*, which takes care of everything related to video analysis.

### 6.2.1 Drawing predicates

Firstly, we will describe the process of drawing predicates and its implementation.

#### Canvases

The first task is to create an environment for drawing. We use the HTML canvas element which is commonly used for drawing graphics. Namely, we use multiple stacked canvases:

- **Image canvas.** Image canvas is a canvas that is used for displaying reference frame images. Reference frames are stored on the server and loaded on every selection of a video.

- **Interactive canvas.** It serves as an intermediate layer that only reflects the current user actions. It is refreshed at each update, and any finished changes are immediately moved to a newly generated predicate canvas as the interactive canvas is cleared for the next interaction.

- **Predicate canvases.** Predicate canvases for each drawn predicate. Using one canvas for one predicate makes it easier to delete selected predicate or change the opacity of drawn predicates e.g. for predicates with action set to *none*.

  Each predicate canvas has its unique ID referencing its predicate. In case of change/deletion of the given predicate, canvas with its ID is found and changed/deleted.

  We suppose that the number of drawn predicates will not be high enough to slow down the web page.

- **Results canvas.** Results canvas is meant for plotting detection bounding boxes or trajectories.

- **Crop canvas.** Crop canvas is used for displaying detection crops.

- **Export canvas.** Auxiliary canvas that is created for merging all canvas in order to export reference frame with predicates and drawn results. It is deleted after the export is finished.

### Event handlers

Due to the fact that all predicates defined within Videolytics behave differently, we need to listen to the changes made in the selection of mentioned predicates and behave accordingly. Due to that, the type of event handler for user interaction with the canvas has to be updated dynamically as well. Upon selecting a different predicate in the predicates menu, the listeners are updated, and an appropriate set of drawing listeners is selected.

We use two types of event handlers for every predicate:

- **On mouse down handler.** Handler that handles the *mousedown* event. Handling this event includes starting drawing the selected predicate shape on the interactive canvas if the drawing process has not yet begun or end drawing and store drawn shape otherwise.
- **On mouse move handler.** Handler that handles the *mousemove* event. Handling includes update interactive canvas based on the user mouse moves.

### Storing predicates

The analytics module supports four types of predicates:

- **Line.**
- **Polygon.**
- **Circle.**
- **Directed line.**

After drawing, the predicate must be saved for further interactions with it (e.g. action change) and for passing data to generate a query. Every predicate is stored as a JavaScript Object with properties specific to each type of predicate. Some examples are shown in Figure 6.1.

```
{
  "type": "line",
  "id": 1,
  "action": "contains",
  "coordinates": {
    "points": [
      { "X": 0, "Y": 0 },
      { "X": 1000, "Y": 1000 }
    ]
  }
}
```

```
{
  "type": "circle",
  "id": 2,
  "action": "contains",
  "coordinates": {
    "point": { "X": 0, "Y": 0 },
    "radius": 100
  }
}
```

Figure 6.1: Examples of structures for line and circle predicates.

### 6.2.2 Communication with the database

Again, we must note that the actual generation of queries and communication with the database is the work of another student. We will focus more on the design of communication with the database.

#### Internal state

To generate a query, it is necessary to pass all relevant information to the generator. Because the generator assumes JSON file as an input, all important data are kept in a JavaScript object which can be easily converted to JSON.

Properties of the internal state include:

- **Camera ID.** ID of the currently chosen camera. Because we work in the offline mode, in our case this is the ID of the currently selected video.
- **Mode of the analytics manager.** We support detections and trajectories mode, based on the user input.
- **Trajectory model.** Trajectory model that is currently selected. Works only in trajectories mode.
- **Selected arguments and selected classes.** Lists of checked arguments and classes.
- **Time interval.** Two dates from the interval element (from-to) in the ISO standard format.
- **Count.** Boolean whether only the final count of results or every result should be sent back.
- **Predicates.** List of predicate objects.
- **Logical operator for combining predicates.** Can be *and* or *or*.

When needed, the internal state object is converted to JSON and passed to the generator to generate a query. Generated query is then executed by the executor. Both generator and executor are the work of another student and will not be discussed.

#### Database response

We assume that the received response is a JSON file. Once the response from the executor is received, it is parsed to a JavaScript Object and stored. The parsed object is then used for creating an HTML table that lists results. Due to the fact that the number of results tends to be really high, only the first 500 results are shown. Otherwise, the web page becomes laggy and slow.

### Further communication with the database

It may be necessary to communicate with the database again when working with the results, especially when drawing results. For drawing detections, it is necessary to know the boundaries of bounding boxes. Moreover, we need to have the actual RGB data of the detected object for drawing detection crops. The same is true for trajectories - in order to draw a trajectory accurately, we need to know its individual points.

There are two options how to ask for additional information:

- **Ask for information about all the results.** This can be done by using almost the same query we have already used - with predicates. All we need to do is query different arguments that are essential for drawing.
- **Ask for information about some results.** In some cases we are only interested in information about certain results. Therefore it is possible to query information using IDs of the results.

The usage of these two approaches depends on the situation. Generally, for querying information about a small number of results, the IDs approach is used. Otherwise, we query all by using predicates. This also implies that the IDs must be returned in every response.

Additional information is queried only when needed. Before drawing a result, it is always checked whether the necessary information is already cached in memory. If not, it is queried from the database.

A schematic description of communication with the server and the database can be found in Figure 6.2.

### Storing results

The choice of data storage structure is crucial for the speed of the visualization results. Initially, results were stored in an array. However, querying additional data on only some results and associating the response with given results took too long.

Due to that, we had to switch to a different way of storing data which is more effective. We opted for a JavaScript Object representation with detection/trajectory IDs as keys and the detection/trajectory JavaScript Objects as values.

This representation has the main advantage that access to the detection/trajectory data with a given ID is possible at a constant time. This allowed us to significantly speed up the storage of new data and thus speed up the visualization of results.

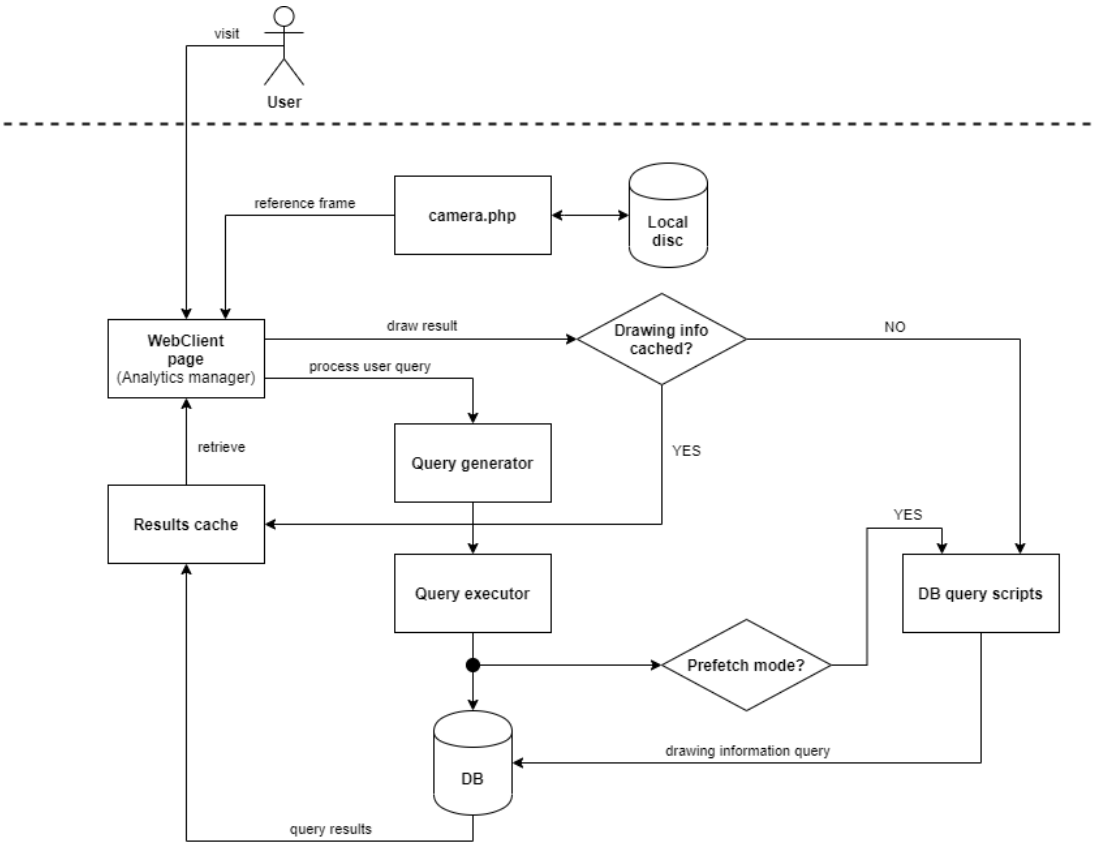


Figure 6.2: A schema of communication with the server and the database.

### 6.2.3 Drawing results

The presentation part of our graphical interface deals with the processing of results and their drawing on the canvases. In the following paragraphs, we will describe its implementation.

#### Drawing bounding boxes

The easiest and the most basic way to plot detections is to draw bounding boxes. For that, we need to know their dimensions and positions. In our case, four corner points (left, right, top, bottom) of the bounding boxes are stored in the database and need to be retrieved. To draw bounding boxes, we must query this information based on whether all bboxes should be drawn or not. More information about queries can be found in 6.2.2. Detection bounding boxes are then drawn on the results canvas as rectangles. Bbox detection drawing uses already defined methods from the *Detection manager*.

#### Drawing crops

Drawing actual crops of the detected object is a more complex task. We need not only the dimensions and positions of the bboxes but also the crop data itself. Crops are stored in the database as hexadecimal bytes in the RGB representation. To draw crops, we must query the dimensions and the crop data from the database.

The next step is to transform the data to a form that can be used with the HTML canvas element. For that, we need an array of RGBA bytes. Therefore the string has to be sliced and the alpha component has to be added and set to FF as a fully opaque color. Then we can put the crop data on the crop canvas.

The last thing that has to be solved is the dimension of the canvas. In HTML canvas, it is important to note that there is a crucial difference between dimension (width, height), and their style counterparts. Whereas width and height attributes in canvas determine its resolution, width and height specified in style determine only its size on the screen.

In WebClient, all canvases are styled to the same dimensions for obvious reasons. However, because of different ways of interacting with different sets of canvases, some canvases may require different resolutions.

For historical reasons, all original canvases used within the WebClient had a fixed size of 1280x720. These dimensions have been uniformly used for the video player, canvases visualizing reference frame image, as well as others.

These dimensions, however, are an issue when trying to draw crops within the canvas. Crops are made directly by cutting out a specific part of the video at specific times, and therefore their position and dimensions are directly dependent on the dimensions of the video itself. This video is played within a video player, which only uses style width and height, and is resized to fit the player. Canvas, on the other hand, has to conform to this size, otherwise inconsistencies happen when trying to put an image on canvas with a different size, which is to be expected.

Therefore, there are two possible ways to fix this issue:

- **Set the attribute size of canvas which draws crops to the same size.** This would result in a canvas with a possibly different size than the original video. Crops would then have to be adjusted by translating and stretching or shrinking them internally.
- **Use canvases with different attribute size.** Use attributes that correspond to the size of the original video. This way we can avoid changing the crops in any way. They can be simply converted and inserted without any additional steps.

We chose the second approach for multiple reasons. First, the second option requires less code on our side. The algorithm for resizing canvas to fit the style dimensions is already implemented in the native code of JavaScript. Additionally, the first option is a lossy and fairly complex conversion. In case of resizing the crops to the canvas with smaller dimensions, the crops can become blurry, and unless we save both original and resized crop in memory, any change would require running the complex procedure again (or in case of getting the larger version of the crop from the smaller one, completely impossible).

### Drawing trajectories

Each trajectory can be represented as a set of points that form a curve. As well as for detections, all data about trajectories are stored in the database and are queried on demand.

To draw a trajectory we use already existing methods from the WebClient *Trajectory manager*.

Drawing of trajectories is done by initiating a timeout loop. In each iteration, all trajectories are filtered based on their starting and final frame ID for the specific model. All trajectories starting after or finishing before the current frame are discarded, and the ones left are drawn.

In cases when the current frame ID is irrelevant, such as during analytics when the stream is not running, the trajectories are drawn by disregarding the time and instead are drawn fully for the whole duration of the video. Based on fps, each trajectory is then drawn by a sequence of curves, where points are interpolated and smoothed based on their recent history and future.

### Drawn results

The module keeps track of currently drawn results by updating an array of detections/trajectories IDs.

In case some result should be deleted, its ID is deleted from this array, and result canvases are cleared. Then the results whose IDs are contained in this array are redrawn.

### Animation

Creating the illusion of movement is possible by drawing and deleting detections at small intervals. Firstly, all data needed for drawing an animation must be queried.

In the case of animating detections, detections with the same frame ID are drawn at the same time.

However, the question is how trajectories should be animated. There are two options:

- **Animate each trajectory individually.** By sequentially animating individual trajectories, we effectively increase the time necessary for rendering, as the same time frame has to be animated for each obtained trajectory.

- **Animate trajectories by frame IDs.** Animating trajectories by frame IDs first results in blinking images, which look unnatural.

Because of that, we have to first group the crops by frame IDs and animate the group as a whole. In such a way, we obtain natural and well-formed animation for trajectories.

#### Heatmap

In order to render a heatmap based on drawn predicates, we have to process a large amount of data as effectively as possible. These data can vary in size, which also has to be taken into account.

During development, we tried multiple methods of drawing heatmaps. One such method was by drawing semi-transparent ellipses at the positions of individual detections. The ellipses had a radial gradient, which resulted in smooth shading of the final heatmap. However, this method had bad performance due to repeatedly drawing shapes on canvases with complex structures.

Due to that, we selected another approach. Instead of drawing a heatmap directly, we first generate an image internally, each pixel having a value equal to the number of detections intersecting it. This internal image is then normalized, formatted, and finally passed in the canvas as an ImageData object, which results in only a single update of the canvas internal data. This method performed better, reducing the processing time significantly. The visual comparison between these two approaches is shown in Figure 6.3 and Figure 6.4.

### 6.2.4 Exporting results

#### Exporting results data

Our module supports exporting data as JSON and CSV files. Because the query results from the server are in JSON format, no big conversion is required when exporting data as JSON. On the other hand, the data must be converted to an appropriate format when exporting as CSV.

However, because we want to keep the arguments entered by the user, we must export the data without additional information for drawing. This information must first be filtered out before the data can be exported.

Along with the results, a metadata JSON file containing the internal state of the application is exported as well.

#### Exporting reference frame

The export of the reference frame is done by creating an auxiliary canvas. Next, all predicate canvases, the reference image canvas, as well as canvases holding result crops or bounding boxes, are merged, and the resulting image is exported.

### 6.2.5 Importing results

As well as for export, JSON and CSV files with results data are supported for import. Metadata can be imported as a JSON file.

In the process of parsing imported files, it is verified whether the contents of the files meet the basic criteria for result files. Next, the internal state is restored from the metadata file (if this file was provided) and the results are listed in the results table.

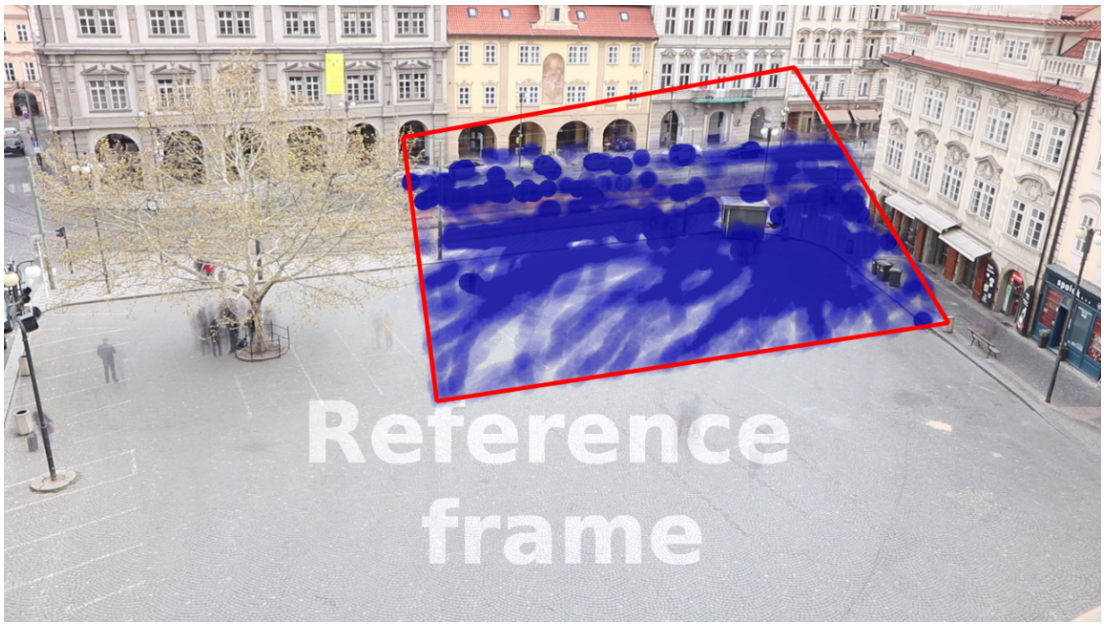


Figure 6.3: A visual example of the initial implementation of the heatmap. Apart from being slow, it did not use color normalization - some areas are too dark.

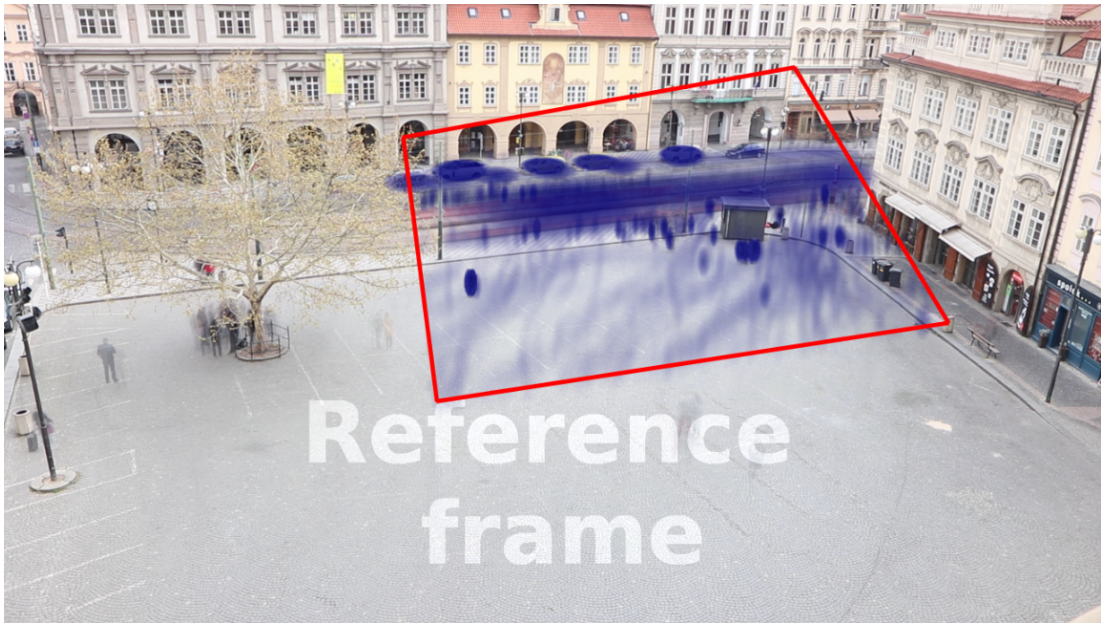


Figure 6.4: A visual example of the new and better implementation of the heatmap. Colors are normalized and the heatmap is more pleasant to look at.

# Conclusion

Our work was focused on the analytical module for the Videolytics system. We mainly dealt with the design and creation of the graphical user interface. The implementation of this interface provides the user with an efficient and user-friendly tool for querying, processing, reviewing, and exporting requested data as well as their import back to the application.

- In Chapter 1 we have reviewed multiple projects from different backgrounds that are dealing with video analysis.
- We have described the framework that the Videolytics system is based on in Chapter 2. We also provided a brief overview of approaches towards object detection and the choice of detector for Videolytics. We have provided a description of existing modules. We ended the chapter with a short description of possible practical use cases.
- In Chapters 3 and 4 we have proposed and analyzed the graphic user interface for the analytical module from the user's point of view. We explained its operation including entering queries and working with the results.
- Chapter 5 was reserved for the presentation of practical examples that can be analyzed by our module. We showed a variety of queries and explained what can be done with the results, including their post-processing by external applications.
- Lastly, in Chapter 6 we have discussed the implementation of the user interface from the programmer's point of view. We have described some obstacles we have encountered during the implementation and the solutions we used to overcome them.

## Future work

The potential of Videlytics is great and offers plenty of room for further development. Another module dedicated to detecting crowds is expected to be added in the near future. Regarding the analytical module future work and improvements, we would suggest:

- **More options for combining predicates.** Currently, it is possible to select a logical operator to combine predicates, and this operator will be used to combine all predicates. It would make sense to extend this functionality with the ability to define exactly when to use which operator.
- **Optimizing detection crops size.** The largest bottleneck of the analytical module is the data download time from the server. This time could be reduced especially when downloading crops by compressing the data to a lower quality instead of using the resolution from the original video.

- **Heatmap that distinguishes object classes.** The current implementation of the heatmap does not take into account the classes of detected objects. Their color differentiation could be an interesting way to improve it and provide the user with an even better visual view of the analyzed situation.
- **Better import file validation.** Because the validation of imported files is only very basic so far, it is necessary to make a better validator that better detects invalid files.
- **Event detection.** Using trajectories, it would be possible to create a higher abstraction of the analytical module that would be able to detect certain actions based on trajectory analysis.

# Bibliography


[1] Christopher R. Wren, Ali Azarbayejani, Trevor Darrell, and A.P. Pentland. Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19:780 – 785, 08 1997.

[2] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[3] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *IEEE transactions on pattern analysis and machine intelligence*, 36(8):1532–1545, 2014.

[4] Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. A survey on trajectory clustering analysis. *arXiv preprint arXiv:1802.06971*, 2018.

[5] J Arraiza, N Aginako, S Kioumourtzis, G Leventakis, G Stavropoulos, D Tzovaras, N Zotos, A Sideris, E Charalambous, and N Koutras. Fighting volume crime: an intelligent, scalable, and low cost approach. *Journal of Polish Safety and Reliability Association*, 6, 2015.

[6] Suzanne Little, Iveel Jargalsaikhan, Kathy Clawson, Hao Li, Marcos Nieto, Cem Direkoglu, Noel E O'Connor, Alan F Smeaton, Aitor Rodriguez, Pedro Sanchez, et al. Savasa project@ trecvid 2012: Interactive surveillance event detection. 2012.

[7] David Schreiber, Martin Boyer, Peter Gemeiner, and Andreas Opitz. Generic object detection and tracking for accelerating video analysis within victoria. In *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, pages 1–6, 2019.

[8] M. Boyer and S. Veigl. Privacy preserving video surveillance infrastructure with particular regard to modular video analytics. In *6th International Conference on Imaging for Crime Prevention and Detection (ICDP-15)*, pages 1–5, 2015.

[9] Tomáš Skopal, Dominika Ďurišková, Petr Pechman, Marek Dobranský, and Vladislav Khachaturian. Videolytics: System for data analytics of video streams, 2021. `http://videolytics.ms.mff.cuni.cz/stream.html`.

[10] Marek Dobranský and Tomáš Skopal. On fusion of learned and designed features for video data analytics. In *MultiMedia Modeling - 27th International Conference, MMM 2021, Prague, Czech Republic, June 22-24, 2021, Proceedings, Part II*, volume 12573 of *Lecture Notes in Computer Science*, pages 268–280. Springer, 2021.

[11] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[13] W. Liu, Dragomir Anguelov, D. Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and A. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[14] Marek Dobranský. Object detection for video surveillance using the SSD approach. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2019.

[15] Vojtěch Antošík. Module for real-time object detection in video stream. Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2020.

[16] Dominik Smrž. Re-identification of objects in video stream using data analytics. Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2021.

[17] Vladislav Khachaturian. Web interface for real-time video analytics system. Master's thesis, Czech Technical University, Faculty of Information Technology, Prague, 2020.

# List of Figures

# List of Abbreviations

- IP camera - Internet Protocol camera
- GUI - Graphical User Interface
- UI - User Interface
- bbox - bounding box
- YOLO detector - You Only Look Once detector
- SSD detector - Single Shot Detector
- ID - Identification Number
- JSON - JavaScript Object Notation
- CSV - Comma-Separated Values
- HTML - HyperText Markup Language
- RGB - Red-Green-Blue color model
- RGBA - Red-Green-Blue-Alpha color model
- fps - frames per second