VideoTerror demonstrator

Event-based Video Analytic Tool (EVIDANT)

Technical Report - FIT - VG20102015006 - 2015 – 02

Ing. Vítězslav Beran, Ph.D.
Ing. Michal Kapinus
Ing. Lukáš Klicnar
Ing. Zdeněk Materna
Ing. Michal Hradiš,Ph.D.
Ing. Vojtěch Fröml
Ing. Tomáš Volf
Ing. Štěpán Mráček
Doc. Ing. Jaroslav Zendulka, CSc.
prof. Dr. Ing. Pavel Zemčík

Faculty of Information Technology, Brno University of Technology

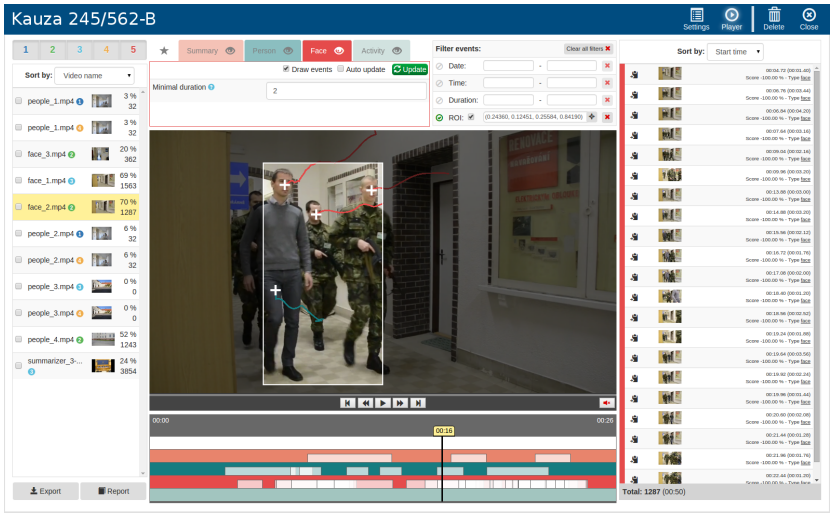
August 30, 2015

Abstract

Event-based Video Analytic Tool (EVIDANT) demonstrates the video-processing methods developed in VideoTerror project and enables the user to effectively browse interesting events automatically detected in video sequences. The functional video-processing modules integrated in EVIDANT are: Video-Summarizer, Activity-Detector, Face-Tracker, People-Tracker, Video-Comparator and Scene-Classifier. The system workload is divided into two parts: video-processing and event-browsing. After the user prepares video-dataset for further analysis and configures functional modules, the video-processing phase executes time-consuming computationally expensive methods of selected modules. Depending on the amount and length of videos in the dataset, video-processing phase may take longer time (from hours to days). When the video-dataset is processed, the event-browser provides the user with effective graphical user interface to browse, filter, select, comment and export the detected interesting events. The integration of developed functional modules is based on VTAPI and VTServer implementing data-storage and task processing control mechanisms. The EVIDANT is implemented as the client-server web application.

Contents

1 Introduction



The VideoTerror project was focused on development of image and video processing methods and tools. Several methods have been developed and implemented in the project. (e. g. video summarization or face tracking) together with a framework VTAPI (VideoTerror Application Programming Interface) for data and processing modules management. A data storage, which is accessed via VTAPI, contains raw image and video data, metadata related to and/or extracted from this data by processing modules and can also contain some other analytical results obtained by processing modules. Such a way, VTAPI supports and simplifies not only simple image and video applications and processing modules development, but also development of more complex image and video processing systems with many processing modules and a sophisticated GUI allowing effective work of the end users.

No matter of the fact that the main objective of the project was development of methods and, in some cases, also simple applications demonstrating the methods' functionality, we have decided to implement a more advanced demonstrator integrating some of the methods developed in the project in order to show their potential and the potential of VTAPI better, especially if they are used for analytical purposes. As a result, we have developed a demonstration tool called EVIDANT (Event-based Video Analytic Tool), which is described in this technical report.

During its development we employed consultations with detectives of the Police of the Czech Republic. Based on their requirements, some interesting situations occurring in image and video data were specified and appropriate methods developed in the project have been adapted and implemented as processing modules into the tool.

The EVIDANT tool aims to enable the user to select images and videos and a module for their processing. After the data was processed, the user can effectively browse the events of interest in a these images and videos, make notes to selected events or select events for export. The tool also allows preparing a summary video material or a text report for sharing or printing.

This technical report is structured as follows. Chapter 2 describes the design process applied in EVIDANT development. The general view of the demonstrator concept is presented in Chapter 3. Modules included in EVIDANT are briefly described in Chapter 4. Chapter 5 contains information related to the graphical user interface of EVIDANT, including workflow and good practises presentation. components and their usage.

2 Design process

The goal of the EVIDANT demonstrator is to apply the scientific results of the VideoTerror project to efficient tool for the end-users. We used Double Diamond Design[1] methodology to design and develop the EVIDANT application concept and its graphical user interface.

2.1 Double Diamond Design

The model presents four main stages across two adjacent diamonds. As illustrated in the Double Diamond model's first diamond (see Figure 1), the problematisation and understanding of a problem are equally important. Each of the four stages is characterised by either convergent or divergent thinking. These stages are:

- *Discover* –identify, research and understand the initial problem.
- *Define* – limit and define a clear problem to be solved.
- *Develop* – focus on and develop a solution.
- *Deliver* – test and evaluate, ready the concept for production and launch.

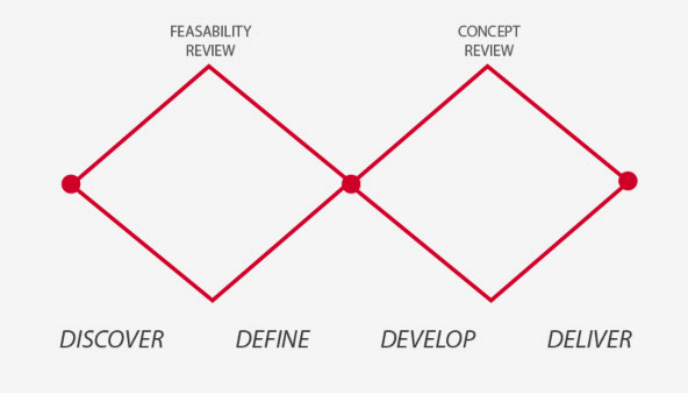


Figure 1: Double Diamond Design methodology.

2.2 Discover phase

Usually the starting point for most projects is a preliminary idea or inspiration. The discover phase is characterized by divergent thinking as a team opens a solution space and investigate a broad range of ideas and opportunities. During

[1] http://www.designcouncil.org.uk

the presentations of developed video-processing technologies to potential users, many qualitative interviews a discussions in focused groups, dozens of ideas have been gathered. Further, more ideas has been summed up based on the research of existing tools and solutions, and expectations from state-of-the-art technology knowledge.

The usual and most required task, when analyzing larger set of long video recordings (e.g. from surveillance or monitoring systems), is to detect video parts where some *interesting event* occurs. The problem is that the information value of particular video part is usually application-specific and is hardly possible to define interesting events generally.

We organized personal interviews with detectives from the Police of the Czech Republic and defined the domain of the demonstrator. The definition of interesting events is based on assumption that recording come from surveillance or monitoring systems of public spaces (example of presented ideas are shown on Figure 2). Besides this scenario, two additional use-cases and requirements have been discussed. When a large set of videos of unknown domain needs to be analyzed, the important ability of the system is to automatically recognize, what scene types occurs in the dataset. The second specific task is to compare the query video to the video dataset; i.e. determine whether any part already exists in one of the recordings in video dataset.

Figure 2: Examples of developed video-processing technologies.

Based on introduced requirements, we defined list of interesting events, that the system should be able to detect for further analysis. Interesting events are

video parts, where some of the following object appears or situation occurs:

- face, person, people,
- any type of activity or temporal changes,
- locally distinctive changes or movements,
- specific scenario type,
- similar (or dissimilar) video content to given input video.

Besides the interesting events specification, the interviews revealed the requirements for efficient interaction with the application leading to effective browsing, navigation and manipulation with automatically detected results. The application should provide the user with following functionality:

- navigate in the video dataset and replay selected video,
- browse through detected interesting events,
- tune the quality and quantity of detected interesting events,
- filter the detected events by various parameters (date and time range, duration, spatial location in video frame etc.),
- select, edit and add comments to particular interesting events,
- export selected interesting events in the form of text report or video material,
- partly anonymize the exported video parts.

Last but not least, the useful application should provide additional functions to efficiently manage the analytical work:

- organize work into private projects,
- organize videos into groups according to various criteria, e.g. location of recording device,
- store additional information to project, groups and videos, e.g. date, time and location of the recordings, notes etc.
- configure detection modules for each group separately, and
- keeps track of user settings and activities.

Gathered requirements formed the definition of frequent and important tasks, refined the adaptation of the image and video processing methods to functional modules, specified the extension of VTAPI and VTServer services, lead the design of GUI layouts and mock-ups and helped to prepare and verify the UI testing and evaluation.

2.3 Define phase

This stage involves the evaluation and selection of ideas. Discover stage results are analysed, developed and detailed, while ideas for solutions are prototyped and pitched. We focused on making sense of all the possibilities identified in previous phase, resulted with a clear list of requirements and user processes.

The EVIDANT is designed to reflect the contradictory user requirements; to browse and manage the video-processing results in **interactive and efficient** way and to **process large amount of video materials**. Depending on the image or video processing method complexity and also on the video sequence length, the time to process a single video-sequence may vary from minutes to hours.

Time-efficient video event detection is achieved by splitting the process into two phases. Computationally expensive **video processing** methods are executed in advance to less expensive and faster **event detection** phase. Designed concept allows efficiently repeat the event detection phase with various parameters and analyze different results without re-execution of video-processing phase. The work-flow of the concept is in Figure 3.

During the Define phase, we pointed out following main requirements that had the most influence on user interface design of the project management. User should be able to quickly and easily:

- upload multiple videos, add some detailed information about videos, delete or add new videos (even if some videos are already processed),
- configure functional modules differently for several groups of videos,
- minimize data that must be recomputed when something in project is changed,
- not have to care about video processing progress, just receive a notification e-mail when it is done.

The second part of the application is the interface for detected event analysis itself, which serves for browsing events and making a final report from them. The user interface was designed with preference to:

- browse events, be able to instantly fine-tune modules parameters,
- filter events and sort events, to be able to quickly find important ones,
- select events for final report, edit them, add some detailed information,
- generate report and video composition with events of interest and added description.

Figure 3 shows two main phases of the entire process that determines the design of functional blocks implementing the detection methods. Each image or video data processing method integrated into the EVIDANT system is also divided into two parts: pre-processing the image or video data, and detection of specified image or video events (objects, scene classes, object trajectories etc.).

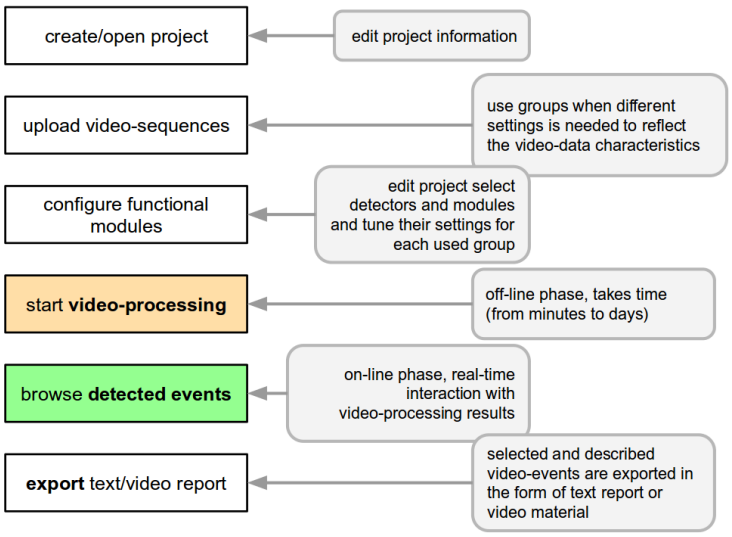


Figure 3: Work-flow of the typical application usage.

2.4 Develop phase

During the Develop stage, one or more of the concepts is developed further. Design methods deployed during this stage include brainstorming, visualization, prototyping, testing and scenario development. The methods resemble those from the Define stage, but focus more on realization.

Based on the outputs from Define phase, analysis of the user requirements, the following functional blocks has been defined, both for pre-processing and detection phases:

- Activity - activity detector based on temporal changes (Sec. 4.2),
- Summary - compact video-parts with changing image content and constrained by required maximal length of final video (Sec. 4.1),
- Face - face detection and tracking (Sec. 4.3),
- People - people detection and tracking (Sec. 4.4),
- Matcher - block comparing the image content of query video with recordings in video dataset (Sec. 4.5), and
- VideoType - classification of the scenes by visual content (Sec. 4.6).

The technical description of designed modules, its interfaces and implemented methods are presented in details in Section 4. The integration and user interaction with those modules are described in Section 3.3.

As the application name suggests, the video content analysis itself is based on detected events. **The events** are the fundamental elements for the user to work with. The developed modules are therefore designed and modified to provide results of their processing as video events. The event is generally defined as the particular part of the video (in spatial and temporal domain) and may contain some augmenting information (class type, score etc.).

During the Develop phase, we designed and prepared blueprints and mock-ups of the fundamental GUI components (see examples on Figure 4 and 5. The main frames, layouts and components have been also implemented in preliminary stage (see Sec. 3.3 for details) and connected to VTServer.

Figure 4: The mock-up of the project management screen.

The EVIDANT demonstrator development introduced couple of new demands to previosuly developed VTAPI and VTServer solutions, so the interfaces, functions and services have been adapted and extended to provide newly required functionality (see Sec. 3.2 for the system overview and APIs).

2.5 Deliver phase

During the Deliver phase, the process revolves around the final concept, final testing, production and launch. The product or service developed to solve a specific problem during the Discovery stage has reached completion. Important activities at this stage are: final testing, approval, launch and evaluation.

The developed concept has been presented to end-users. The experiment has been divided into two parts. First, the users (3 detectives from the Police of the Czech Republic) were provided with the tool and think-loud protocol was used

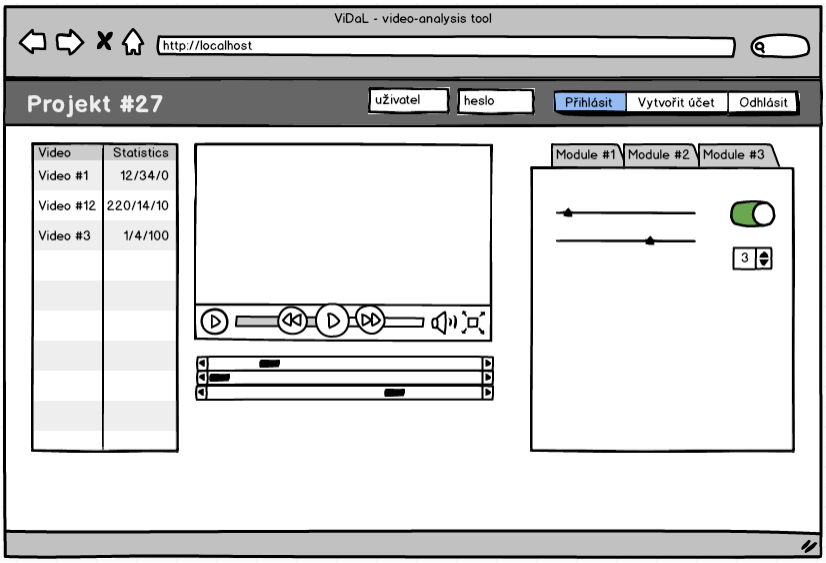


Figure 5: The mock-up of the browser of the detected events.

to capture the difficulties, misunderstandings or problems that user encounters during the free tool usage. Second part was based on predefined tasks including most of the fundamental functionality of the tool. Again the think-loud protocol was used to gather the user feedback. Finally, the discussion in the focus group took place to tune the requirements for finalization.

The final solution, the presented EVIDANT application (Section 5), is the result of incorporated users' feedback and comments.

3 Demonstrator concept

3.1 System architecture

The EVIDANT is an application based on VTServer and VTAPI solution (see Section 3.2 for more details) and designed as **client-server application** using web technologies. The client implements GUI to manage user projects and video datasets, configure and control the functional blocks' execution, analyze detected video events and prepare graphical or textual reports. The client is thin front-end of the web server application that implements data model for user-specific data, database interface and application-dependent services. The web-server video-processing services are connected to VTServer and provide the EVIDANT processing functionality. Figure 6 shows system layers.

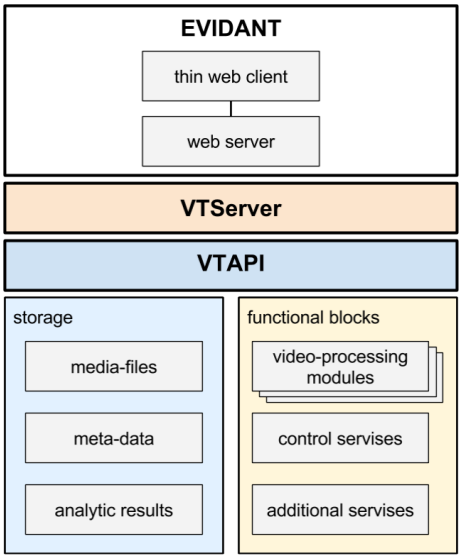


Figure 6: System architecture based VTServer and VTAPI layers.

VTServer is designed as a RPC server utilizing VTAPI abilities and adding further controlling mechanisms and functionality. Its interface is defined using Google Protocol Buffers and messages are passed through ZMQ[2]. Implementation is based on RPCZ library combining previously mentioned technologies, having C++ and Python API. Overview of available methods is grouped into subsets of the API servicing different stages of the event-detection process.

Brief overview of VTServer funcionality follows. Detailed technical specification for all supported RPC methods may be found on project results server[3]

[2]distributed messaging; http://zeromq.org/
[3]http://vidte.fit.vutbr.cz/vtapi.html

- **Dataset API** services projects management phase of the work-flow. Its methods allow user to create or manage datasets for specific projects. Datasets encapsulate all project-related information - video metadata, defined processing tasks, processing units, computed or otherwise processed data and various metrics to query the event detection process and results.
- **Videos API** allows a user to load or delete video data to previously created dataset.
- **Processing tasks API** provides methods for definition and querying the processing tasks. Definition consists of selecting pre-installed computation method and specifying values of its input parameters. Prerequisite processing task may also be specified to utilize its previously computed outputs as inputs for the new task, thus allowing process-chaining (Video-processing - Event-detection). During processing task creation, VTServer automatically allocates necessary system resources for output data and checks if all required inputs are available, notifying user of possible errors. Created tasks may be queried for their progress or successfully completed computing operations on certain videos.
- **Processes API** allows launching and control of system process instances performing assigned processing tasks. Each process instance is assigned previously defined processing task and a set of video data for which it is responsible to complete the processing. Process parallelization is supported, multiple process instances responsible for same computation will correctly prevent themselves from performing redundant work.
- **Events API** methods handle retrieving computed events - outputs of Event-detection tasks. For faster analysis of events relevancy, it provides built-in statistical and filtering functionality.

3.2 VTServer and VTAPI

VTAPI is a general application programming interface for video application development. It can be used both for development of modules (computational algorithms of processing methods), like previously mentioned modules for activity detection, face tracking etc., or some other newly implemented ones, and also for development of a complex background system, which launches the implemented methods (here the VTServer). It is based on the open-source PostgreSQL database system and provides developers with a specific functionality enabling them to define and implement complex computational tasks without worrying about certain implementation details (e.g. where and how to store data).

Figure 7 shows major functionality provided by VTAPI and VTServer. They support:

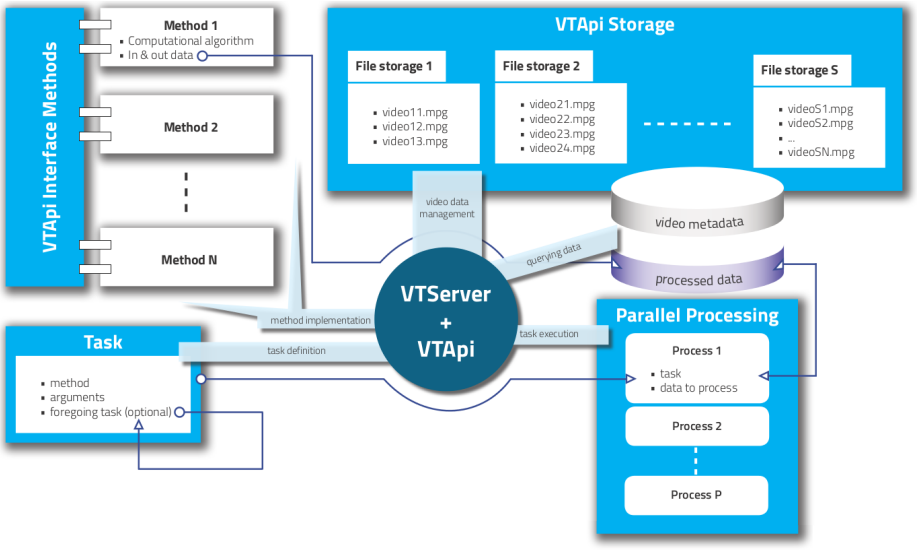


Figure 7: Overview of VTApi and VTServer usage.

- Management of video data and their metadata in a VTAPI storage (some of metadata is automatically maintained by the VTAPI, for example detection and storing framerate data of video).
- Implementation of video processing methods, the definition of their parameters, features/properties extracted from video.
- Definition of tasks for the execution of a given method by specifying the values of its parameters and optionally another task the output of which is to be used as input for execution. This allows task chaining.
- Task execution - each process processes the specified task together with assigned data, the processes can run in parallel.
- Querying output data produced by tasks.

The most common scenario of the VTServer end-user client application typically includes the videos insertion/selection, the task definition, tasks execution and querying steps. Other scenarios may include repeated execution of methods with modified parameters, re-processing reduced set of videos etc.

Based on VTServer API, the work-flow example may be implemented as the following procedure:

```
# 1. Add new dataset (project)
addDataset = call('addDataset', {'name': test_dataset})
```

```
# 2. Add video to dataset
addVideo = call('addVideo', {
  'dataset_id': addDataset['dataset_id'],
  'filepath': test_video_path.mpg,
  'start_time': {'seconds': 1439734399, 'nanos': 0}})

# 3. Define Video-processing task
addTask1 = call('addTaskVideoProcessing', {
  'dataset_id': addDataset['dataset_id'],
  'module':'people',
  'params':[
    {'type':'TP_FLOAT', 'name':'input_frame_scale' ,
     'value_float':0.5},
    {'type':'TP_FLOAT', 'name':'threshold' ,
     'value_float':0.5}]})

# 4. Run Video-processing task
runProcess1 = call('runProcess', {
  'dataset_id': addDataset['dataset_id'],
  'video_ids': [addVideo['video_id']],
  'task_id':addTask1['task_id']})

# 5. Get process state (this should be called
     repeatedly in loop)
getProcessInfo1 = call('getProcessInfo', {
  'dataset_id': addDataset['dataset_id'],
  'process_ids': [runProcess1['process_id']]})

# 6. Define Event-detection task
addTask2 =call('addTaskEventDetection', {
  'dataset_id': addDataset['dataset_id'],
  'module':module,
  'prereq_task_id':addTask1['task_id'],
  'params':{'type':'TP_FLOAT', 'name':'min_duration' ,
  'value_float':2.0}})

# 7. Run Event-detection task
runProcess2 = call('runProcess', {
  'dataset_id': addDataset['dataset_id'],
  'video_ids': [addVideo['video_id']],
  'task_id':addTask2['task_id']})

# 8. Get process state (this should be called
     repeatedly in loop)
getProcessInfo2 = call('getProcessInfo', {
  'dataset_id': addDataset['dataset_id'],
```

```
  'process_ids': [runProcess2['process_id']]})

# 9. Load detected events
getEventList = call('getEventList', {
  'dataset_id':addDataset['dataset_id'],
  'task_id':addTask2['task_id'],
  'video_ids':[addVideo['video_id']]})
```

3.3 Web-based client-server application

This application provides a user-friendly graphical interface able to control settings and processing of different modules, dealing with detected events and creating of various reports from uploaded videos and detected events.

The application is based on Model-View-Controller design pattern and uses a lot of open-source projects. The server is written in Python, using web framework Django[4]. The PostgreSQL[5] has been used as a database back-end, but the application could be easily adapted to others database systems like sqlite, MySQL and others. Gunicorn[6] (Python WSGI HTTP Server for UNIX) had been selected as a http server for our demo setup and the Nginx[7] proxy server is used for serving static files like JavaScript files, images, videos and so on. This web server is acting like a layer between thin web client (which is described below) and the VTServer (which is described above).

Application like this needs to store a lot of different data. It uses VTAPI to store multimedia data like videos and to control processing of modules. Projects and video metadata are stored in separated database independently on VTAPI. Figure 8 shows selected parts of the entire E-R diagram with the most important entities. Necessary information about the video recordings is stored; e.g. name, identifier to VTServer, date and time of video acquisition etc. Application settings for particular configuration of project, modules and video-groups are also stored. Further, the detected events in VTAPI does not contain information important for the user, so the application data model also includes metadata for the GUI and the user like events selection, start and stop refinements, user notes etc.

Web server provides various functions of these categories for the thin client:

- **Project management:** Functions for creating, deleting and managing project. These functions uses Dataset API of VTServer for creating and deleting datasets.
- **Videos management:** It allows the thin client to work with videos (uploading, deleting, meta-data settings etc.). Functions from this category

[4] https://www.djangoproject.com/
[5] http://www.postgresql.org/
[6] http://gunicorn.org/
[7] http://nginx.org/

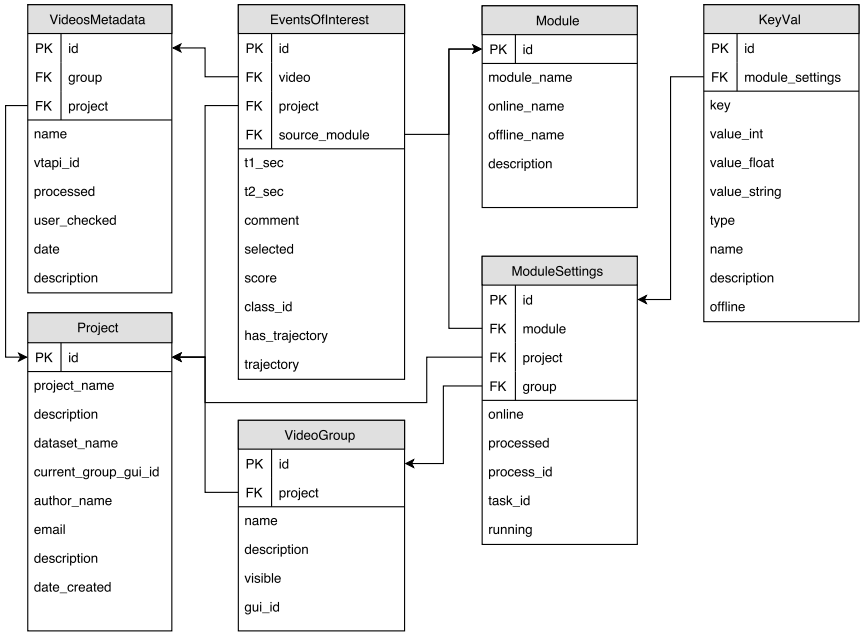


Figure 8: Partial ER diagram of web-based client-server application

uses Videos API of VTServer to register video to VTApi database and possibly to delete video from dataset and storage.

- **Video groups management:** This category provides function for managing video groups. These groups are completely separated from VTApi and they are handled only in EVIDANT.
- **Modules processing:** These functions allows client to control processing of modules and getting info about this processing (like progress etc.). They uses Processing tasks API and Process API of VTSever.
- **Events management:** It handles loading events from VTApi to client, creating interest events for export and so on.
- **Others:** In this category are other functions needed by web client.

EVIDANT web client is implemented using modern HTML5 and CSS3-based technologies. Twitter Bootstrap[8] library is used for most of the standard GUI elements, also jQuery[9] and several jQuery-based libraries (for date and time

[8] http://getbootstrap.com/
[9] https://jquery.com/

pickers, dialogs, form validation, etc.) are involved. Live data updates and communication with server are handled by asynchronous requests (AJAX technology) with JSON data structure. To achieve a most interactive experience, web client also involves data caching.

4 Video-processing modules

Based on the designed concept of the video analytic tool and the user requirements, several developed methods have been selected and adapted to functional video-processing modules. Usually, each functional block contains two modules: one for video-processing and one for event detection. The event detection modules so contain as one of their mandatory inputs the one of the outputs of their video-processing sibling. The presented demonstration solution does not allow linkage of different modules.

4.1 Video-Summarizer

Video summarization is a process of creating a short video summary from a long video sequence. Several properties are typically demanded, the summary should be much shorter, and also contain every important parts or even contain only the most important ones. Application-specific requirements follows: duration of every part, continuity, preservation of time order, preference of certain types of objects, or even more complex attributes.

The Video-Summarizer module is based on presented research [5] also reported in details in project technical report [6]. Video processing is very straightforward, it consist of extracting descriptors from keyframes, which are distributed with constant, pre-configured period over the whole video sequence. The descriptors are based on histogram of simple frame differences: Two consecutive frames are subtracted and resulting values are quantized into several bins, which approximates the degree of video activity at given time.

Table 1: Video-Summarizer processing module.

type	video-processing module
binary	summarizer_offline
params	int keyframe_freq 25 (1,MAX_INT)
inputs	string video_name?? nebo ID??
outputs	float[34] summfeatures
notes	

- *keyframe_freq* Set the keyframes sampling frequency.

Event detection process is based on differences between subsequent keyframes. Using descriptors introduced in video processing part, frame differences are computed, which results into activity estimation metric for the whole video sequence. It is normalized, Gaussian smoothed and local minima are found. Summarization is then performed by picking fixed count of strongest local minima from not yet included parts of video and exporting segments centered around this point, also with fixed length. These segments directly equal events, only overlapping or very close ones are merged together.

Table 2: Video-Summarizer event-detection module.

type	event-detection module
binary	summarizer_online
params	float seg_length 1.2 (1,MAX_INT) int seg_count 10,(1,MAX_INT)
inputs	float[34] summfeatures
outputs	list of events
notes	

- *seg_length* Set the extracted segments length.
- *seg_count* Set the (maximal) count of extracted segments.

4.2 Activity-Detector

The Activity-Detector is able to find spatially-constrained regions in video, where high importance is assumed. Since no general importance can be defined, it is approximated with a degree of activity – spatiotemporal change of pixel values of video frames. It may appear slightly similar to the Video-Summarizer, which is also focused on finding video parts with high activity/importance, but the Activity-Detector is designed to be sensitive to *local activity* (instead of global video summarization). It is useful, if only part of the image is important, for example when some place needs to be monitored.

Video processing is done by utilizing a OpenCV implementation of Gaussian mixture model background subtraction (MOG2) algorithm [13]. It is very computationally efficientt and performs also shadow detection. This results into a frame mask for activity regions, which is then subject of a morphological closure and contour-extraction algorithm. Its output is a collection of detected activity regions, each characterized by mask and bounding box. Finally, tracking them along multiple frame is done by a matching pairs of two overlapping activity regions with highest F-measure. These are directly converted into events by sampling their trajectory with a constant step between detected regions.

Table 3: Activity-Detector processing module.

type	video-processing module
binary	actdet_offline
params	int keyframe_freq 25 (1,MAX_INT) double min_width 0.0 (0,1) double min_height 0.0 (0,1) double max_width 1.0 (0,1) double max_height 1.0 (0,1)
inputs	video
outputs	vtevent event_off
notes	

- *keyframe_freq* Set the sampling frequency of the detected activity regions trajectories.
- *min_(width/height)* Set the relative minimal width or height of the detected activity regions.
- *max_(width/height)* Set the relative maximal width or height of the detected activity regions.

Since the video processing produces events directly, following event detection task is very simple – it involves only an another stage of event filtering. Meaning of min., max. width and size parameters remains the same, but they should be used more as a part of the analysis rather than activity noise suppression.

Table 4: Activity-Detector event-detection module.

type	event-detection module
binary	actdet_online
params	double min_width 0.0 (0,1) double min_height 0.0 (0,1) double max_width 1.0 (0,1) double max_height 1.0 (0,1)
inputs	vtevent event_off
outputs	list of events
notes	

4.3 Face-Tracker

The face detection and tracker module is able to localize and track faces in the input video. The output of this module are trajectories of detected faces that can be further processed, e.g. with blur function in order to prevent identification of subjects.

The module consists of two parts – video processing and event detection. The video processing portion is responsible for detection of faces using tree of stage classifiers based on Haar-like features [8, 9]. In order to deal with frames when false non-detects occur (i.e. the face is not detected), Lukas-Kanade method for optical flow estimation is used [1].

- *min_size_(x/y)* Set the minimal size of the detected face region.
- *max_size_(x/y)* Set the maximal size of the detected face region.
- *input_frame_scale* Set the image scale prior to the face detection.

The event detection module serves as a simple filter that allows a user to define the minimal length of trajectories.

Table 5: Face-Tracker processing module.

type	video-processing module
binary	face_offline
params	int min_size_x (2, 16) int min_size_y (2, 16) int max_size_x (16, 96) int max_size_y (16, 96) double input_frame_scale (0.25, 1)
inputs	video
outputs	trajectories of faces
notes	

Table 6: Face-Tracker event-detection module.

type	event-detection module
binary	face_online
params	double minimal_duration (0.0, DBL_MAX)
inputs	trajectories of faces
outputs	filtered trajectories of faces
notes	

- *minimal_duration* Set the minimal duration (in seconds) of the output trajectories.

The example of the face detection module functionality is in Figure 9. First two images in figure show arbitrary frames from the video with detected faces labeled. The last black and white image shows trajectories of the faces movement.

4.4 People-Tracker

The main task of the people tracker module is to detect and follow pedestrians in the input video. The detection algorithm is based on the Histogram of Oriented Gradients (HOG) and Support Vector Machines (SVM) binary classifier [2, 10]. The video-processing part of the module is similar to the face-tracker – it detects pedestrians and follows their movement using Lukas-Kanade optical flow estimation. The second portion of the module (event-detector) serves as a filter that allows user to select minimal duration of the output trajectories. The detailed information about the video-processing part is in Table 7 and the information related to the subsequent event-detection is in Table 8.

- *threshold* Set the detection threshold.
- *input_frame_scale* Set the image scale prior to the people detection.

Figure 9: Face detection module. Two example frames from the input video and the resulting trajectories.

Table 7: People-Tracker processing module.

type	video-processing module
binary	people_offline
params	double threshold (0, 1) double input_frame_scale (0.25, 1)
inputs	video
outputs	trajectories of pedestrians
notes	

Table 8: Face-Tracker event-detection module.

type	event-detection module
binary	people_online
params	double minimal_duration (0.0, DBL_MAX)
inputs	trajectories of pedestrians
outputs	filtered trajectories
notes	

- *minimal_duration* Set the minimal duration (in seconds) of the output trajectories.

The example of people detection module functionality is in Figure 10. First three images in figure show arbitrary frames from the video with detected people labeled. The last black and white image shows trajectories of the pedestrians movement.

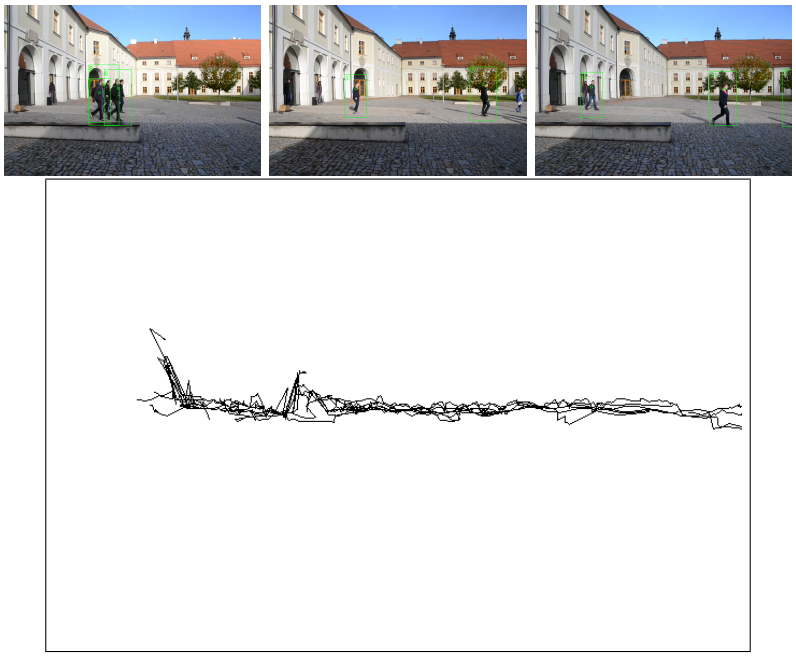


Figure 10: People detection module. Three example frames from the input video and the resulting trajectories.

4.5 Video-Comparator

The main task of the video comparison module is to detect and validate the differences between two visually *almost* identical video sequences. In some cases, even when two video sequences are declared as the identical, small differences might appear and manual detection and validation of such video parts may become extremely time consuming and unbearable. The example of video-pair disruption with dissimilarity types is showed in Figure 4.5.

The Video-Comparator is based on adapted published approach [5] also presented in details in project technical report [4]. Video processing is done by extracting global image features for all keyframes. These features are color-histogram based, they are computed from frame subdivided into multiple overlapping regions and designed for efficient frames comparison. Keyframes are equally distributed with constant interval, but also with addition of boundary frames (e.g. where the video is cut) obtained by temporal analysis. This is done by continuously computing the differences between several adjacent frames and

thresholding. Frame interval between two boundaries is called a segment, which is also output of this module.

Table 9: Video-Comparator processing module.

type	video-processing module
binary	matcher_offline
params	int keyframe_freq 25 (1,MAX_INT)
inputs	video
outputs	int segment_id uchar[1024] descriptor
notes	

- *keyframe_freq* Set the equally distributed keyframes frequence (maximal distance between two consecutive keyframes in a same segment).

The event-detection module needs two video sequences – reference and query – and the procedure starts with computing a so called similarity matrix. Its rows represent keyframes from a reference sequence VR, columns equals to keyframes from a query sequence, and its elements are obtained as a metric of similarity between the two corresponding frames. Detection of differences/similarities is done utilizing the Needleman–Wunsch algorithm for aligning two sequences, which can also be understood as finding an optimal path on a similarity matrix. This path is then segmented and the resulting parts are classified into several types of video disruptions.

Table 10: Video-Comparator event-detection module.

type	event-detection module
binary	matcher_online
params	string ref_seq float sensitivity 0.5 (0,1)
inputs	keyframes (segments) and its descriptors
outputs	list of events
notes	

- *ref_seq* Set the video sequence for base of the comparison.
- *sensitivity* Set the comparison sensitivity, adding value increases amount of detected changes, but some of them may be false alarms.

4.6 Video-Type classifier

This module extracts high-level semantic information from videos and uses it to provide efficient search and navigation. The semantic information which can be extracted includes what types of objects are present in the video (cars, animals, buildings, ...), what type of scene is depicted (pub, mountains, street, stadium, ...), and possibly other similar content descriptions. After the semantic information is extracted from videos, the content can be filtered in real time by the desired content type.

The semantic information is extracted using convolutional neural networks which are the state-of-the-art in many video recognition tasks including object detection [3], scene classification [12], and facial recognition [11]. We applied the technology developed in this project to the classification and localization task of ImageNet Large Scale Recognition Challenge 2014[10] and presented it at ECCV 2014 conference[11].

The module consists of an off-line general engine which is able to efficiently compute convolutional neural networks on the video content. These networks take video frames as input and they output probabilities of the relevant content types. The off-line engine can work with large range of networks many of which are publicly available e.g. at Model Zoo[12]. The relevant publicly available networks can detect objects types and classify scene types. Additional networks can be trained for custom content types on demand.

Table 11: Video-Type processing module.

type	video-processing module
binary	videotype_offline
params	int keyframe_freq 25 (1,MAX_INT)
inputs	video
outputs	float[N] score
notes	

- *keyframe_freq* Set the equally distributed keyframes frequence (maximal distance between two consecutive keyframes in a same segment).
- N is the actual number of trained classes.

[10] http://www.image-net.org/challenges/LSVRC/2014/results
[11] http://www.image-net.org/challenges/LSVRC/2014/eccv2014
[12] https://github.com/BVLC/caffe/wiki/Model-Zoo

The event-detection module uses scores and selected classes by user for each keyframe, executed the flowing window analysis and detects then such video parts where some of the detected scene classes are significantly stable and high-scored. Each such video-part with stable scene classification and high score is detected as result event.

Table 12: Video-Type event-detection module.

type	event-detection module
binary	videotype_online
params	double minimal_duration (0.0, DBL_MAX) int[M] selected_class_id
inputs	float[N] score
outputs	list of events
notes	

- *minimal_duration* Set the minimal duration (in seconds) of the output event.
- *selected_class_id* List of classes selected by the user to be used by event detection.

5 Event-based Video Analytic Tool

This chapter presents the EVIDANT graphical user interface: the application important parts, its main interaction components, user frequent tasks and processes. Besides the key UI components developed for the needs of the EVIDANT, the workflow and good-practices are presented.

5.1 Project management

Every project consists of three main components that can be (or have to be) set by the user: Data for analysis (video sequences), processing modules with their configuration and additional project data (description, author name, e-mail for notifications, etc.). After creating an empty project, user usually wants to add data for processing, since the purpose of this application is to analyze them. Video sequences are uploaded and normalized at server and since it is not instant operation, the progress is visualized by progress bar. To be able to use some functions (for example filtering events by real-world date and time), user can enter video capture date and time. Also, some additional data can be added, such as detailed description, etc. Basic layout of project management is shown in Fig. 11, it consists of several controls blocks:

- *List of videos* allows user to selected a single video group and upload its videos. Since the videos must be preprocessed at the server side first, progress is signalized by a progress bar. Of course, they can be deleted. By clicking on a video item, it is selected and can be edited with a controls in the current video controls block (described later).
- *Current group controls* are used for adding or editing details of current groups of videos (name and description) and primarily, for starting video processing of current group. When started, the process button turns into a stop button, which allows cancelling the current processing.
- *Current video controls* are used for adding or editing details of selected video (name, capture date and description).
- *Settings of video processing modules* displays all modules installed in EVIDANT, several (or all) of them can be checked and applied. Of course, various module parameters can be set before hitting the start button. After that, progress is signalized for every module independently and a green tick appears when the processing is done.
- *Main toolbar* – common for the entire user interface – is used for changing modes (settings, player) and manipulation with project (edit details, close, delete).

During user testing, following requirement arose: to set different parameters of video processing modules for different video sets (for example, when having videos from several locations or viewing distances). It can be easily achieved

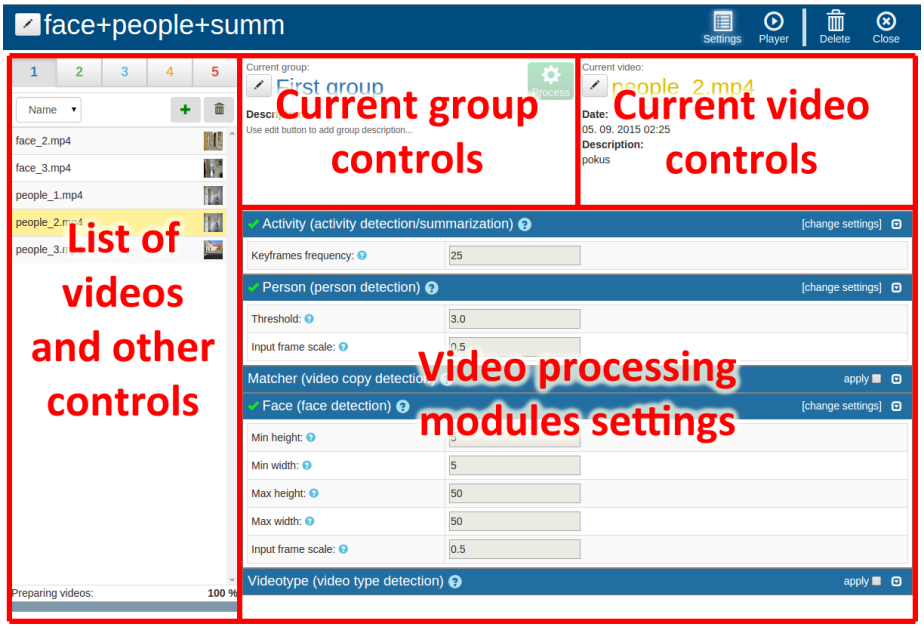


Figure 11: Layout of project management.

with the video groups functionality – videos can be uploaded into several groups and the actual module settings are independent for each one. For a better orientation, groups can be named and have aditional data assigned. After data preparation at the server side is done, video processing can be started. Depending on current video analysis task, user can select one or more processing modules and set their parameters for every group of videos independently. Also the video processing is started for every group separately, which enables user e.g. to browse events from one group of videos, while the other groups are still being processed and not ready yet for browsing events. Example of possible video processing states is shown in Fig. 12. Since video processing can possibly take a large amount of time, a notification e-mail address can be associated with the project.

5.2 Event browser

Event browser is the main part of EVIDANT. It is used for the actual video content analysis – mainly visualizing events and performing various operations with them. The user interface consists of several parts arranged according to workflow described in Sec. 2.3. User can select video for browsing using the video-selector component on the left side, which also offers various sorting options. When working only with a limited set of videos, it is possible to apply a filter to display one or several video groups. Another useful functionality is

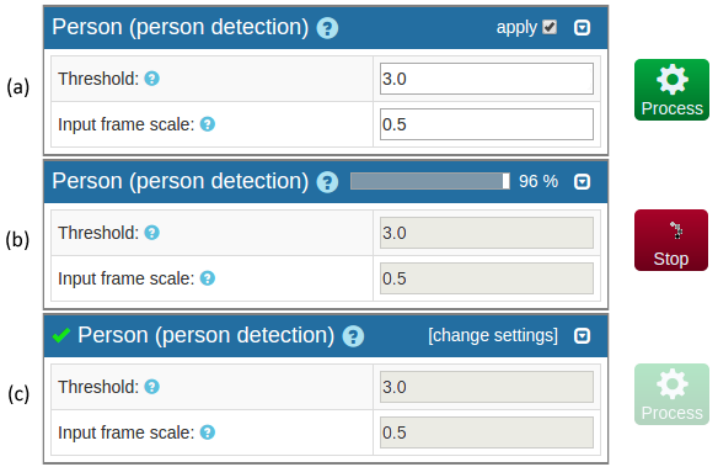


Figure 12: Video processing modules: (a) Module can be selected and its parameters changed, (b) Progress of running process is shown, it can also be stopped, (c) Video processing is done, changing parameters is possible only after unlocking module with [change settings].

marking video as "already seen", immediately telling the user which videos are already analyzed and which of them have to be inspected yet. This can be helpful when work has to be interrupted and continued later. Basic layout of event browser is shown in Fig. 13:

- *List of videos* contains videos of the current project. Only videos from selected groups are shown, they can also be sorted by several criteria (name, group, already seen flag, events count, events coverage). Besides video name, group and thumbnail, every item contains a checkbox, which sets the already seen flag, signalizing that user has completed its analysis. Also basic statistics are shown, indicating events count and coverage for current module and video sequence. By clicking on a video item, it is selected, video is loaded in the video player and its events are displayed.

- *Modules settings* is used for selecting current module and for setting its parameters. It consists of tabs for all modules, a single one is selected by clicking on it, which means that events for this module are displayed. Current module parameters can be changed, recomputing events is done by clicking at the update button. Also, auto update can be checked, new parameters are then automatically sent to the server, immediately after they were changed. If the "draw events" checkbox is checked, events are also visualized as a video overlay.

- *Filter controls* offers several events filtering criteria. Events are filtered and updated immediately after changing some of the filter parameters.

User interface indicates, which filter is active, and also provides a controls for reseting them to default value. Filtering by region of interest (ROI) is done easily by drawing a rectangle mask on the video overlay.

- *Video player* plays current video and provides basic controls for starting/stopping video playback, stepping by a minor amount of time and skipping events. Also, events with spatial data are visualized with their trajectories and bounding boxes.

- *Timelines* displays events of every module enabled in current project for current video. First timeline shows events of interest, the other ones are for visualization of events detected by particular modules. For better orientation, modules are color-coded. One event can be selected as current by clicking on it, which means it is highlighted through the entire user interface and also selected in the events list.

- *List of events* contains all events for current module and current video. It shows basic informations (position, duration, type, score, thumbnail, etc.) and provides controls for copying events to events of interest collection and manipulation with them. The list can be also sorted by several criteria. At the bottom, simple statistics are displayed – count and total length of all and selected events.

- *Exporting and reporting controls* are used for generating a text report and a video summary of selected events of interest. The report button opens a new browser window with document reporting selected events. Generating a export video is not instant operation – by clicking on the particular button, the generation process is started and its progress is shown. When it is done, a notification is displayed and the exported video may be downloaded.

The video analysis consists of browsing events detected by selected event-detection module in current video sequence. Unlike the video processing described in Sec. 2.3, event detection is instant, which – also with simple statistics (event count and coverage for every video) – helps to quickly decide how to fine-tune parameters. It is important to note, that the previous events (obtained with previous module parameters) are lost and replaced with the new ones, when parameters are changed and results of some module are recomputed. By selecting a video and an event-detection module, events for this combination are visualized in three main components of the EVIDANT user interface (see Fig. 14):

- Video player with overlay visualization of events with spatial data (for example tracking of a person)

- Event timeline

- Event list

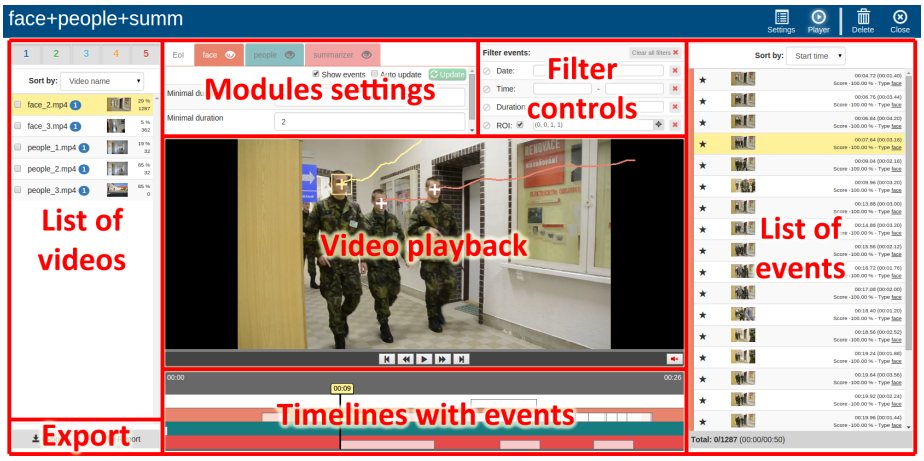


Figure 13: Layout of events browser.

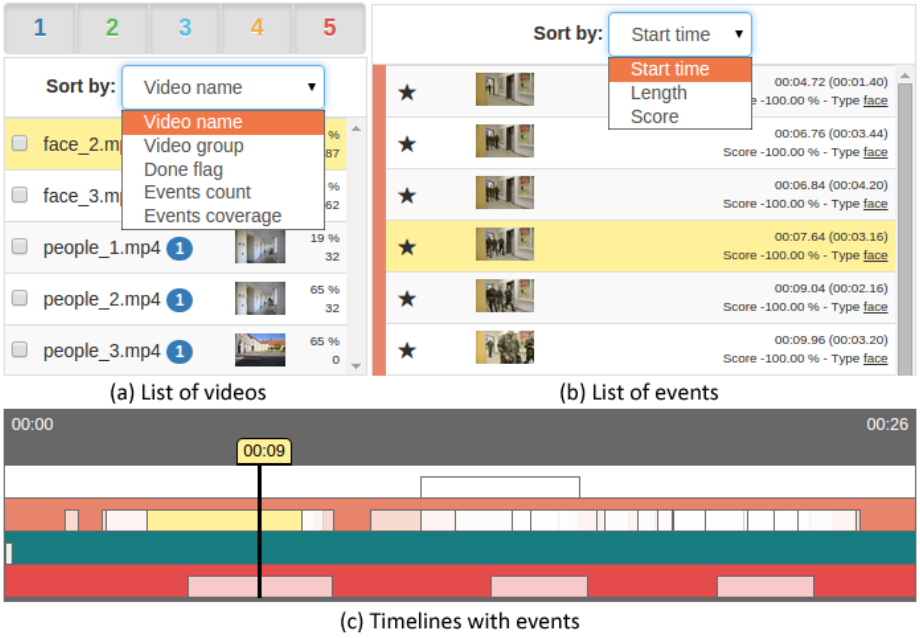


Figure 14: Several components of event browser interface.

Video player is the center of the browser interface, it offers basic functionality as video playing, seeking to desired time, stepping to next frame, etc. If user wants to view events one by one, a "skip to next or previous event" function can

be used. Events from all modules are displayed on several timelines, which can also be used for navigating in current video sequence. More detailed description of events from currently selected module is shown in the event list, for example: start and end position, duration, type, detection score, etc. It can be also sorted by various criteria. For better orientation, when a user wants to focus on a single event, it can be marked as current by clicking on this event at timeline (or event list described later). It is then highlighted in the whole user interface and also the video sequence can be instantly seeked to this event beginning.

5.3 Filtering

When searching for a specific events, a filtering interface is available. Displayed events and statistics (count and coverage) are updated instantly when some filter parameter is changed. Multiple filtering options can be applied (user interface is shown in Fig. 15):

- Date and time range – can be used to point out the important part of the whole video, only events from selected time range will be shown.
- Daytime period – if video time range extends to multiple days, only activity e.g. at 7:00-9:00 AM every day may be important. It can be reflected using this filter.
- Duration – for example, to select only "strong" events, minimal (or maximal) event length can be constrained.
- Region of interest – user can highlight some important area on the image (e. g. space around a door, some guarded item, etc.), only events passing this region will be retrieved.

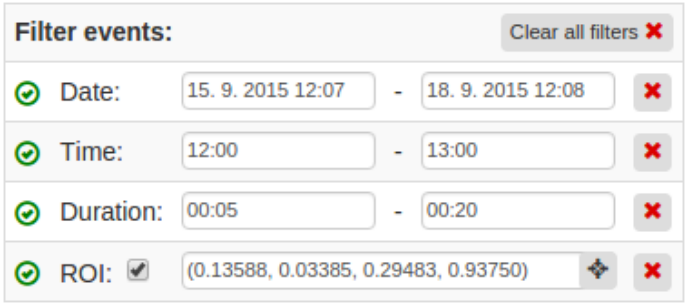


Figure 15: Various event filtering options.

5.4 Events of Interest

Many events can be detected in analyzed videos, but only some of them may be important for a current task. These are called "events of interest" (EoI) and their importance consists in being a source for final report and exported video summary. They are always based on "general" events produced by event-detection modules, created by duplicating them. User can mark some event as important, which results in creating a copy of this event, that is afterwards treated independently from the source module. Also some extended functionality can be applied (see Fig. 16):

- A commentary can be added (it is also displayed in final report).
- Start and end position in time can be adjusted.
- Event can be selected for export/report.
- Event can be deleted.

Because EoI are no longer attached to their source modules, they are preserved even when event detection is reapplied with different parameters and the base events are lost. Events of interest are stable and they can be deleted only at user request. Also, creating several slightly different reports may be demanded, which is easily possible by another property of EoI – they must be checked to be included in the report. Multiple reports are then quickly made by checking different events for each one of them.

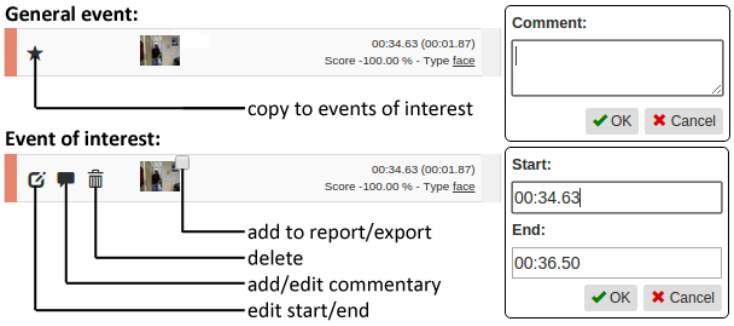


Figure 16: Events of interest controls: Any general event can be copied to EoI, additional controls are then available. Dialogs for adding commentary and editing EoI are shown on the right.

5.5 Export and report

The text report is generated as a printer-friendly HTML page. It contains name of the project, information about its author, project and report creation date.

There is a list of modules used for detection of selected events. For each module, there is its description and list of videos with events. The event is specified by its start and end time and optionally by a user-entered comment.

To generate a video-summary, the user selects one video from the video list and then presses export button. The video-summary will include all EoI for the selected video. For each event the summary contains 10 second information slide (see Fig. 17) and respective part of the video which length is variable. The information slide contains project, group and video name and description, source module, event timing and optional user-entered comment.

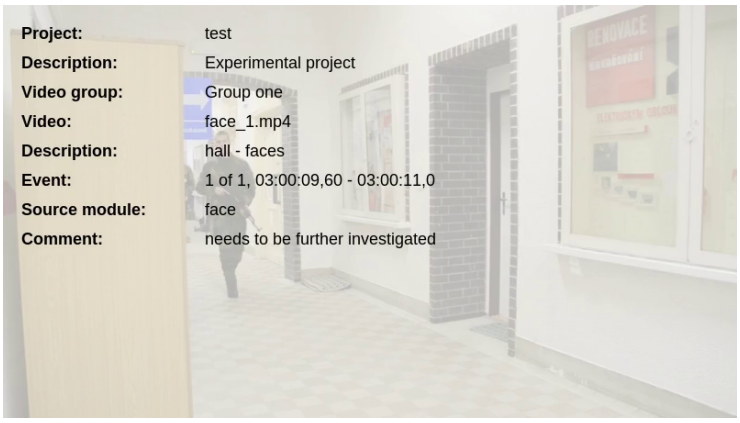


Figure 17: Information slide example for a video summary containing one EOI.

6 Conclusion

The main objective of the EVIDANT demonstrator is to show the usage of some scientific results of the VideoTerror project in a tool tailored to real-world conditions of some departments of police. The specific category of tasks we focused in the tool was the processing of events of interest in a set of images or videos. The processing modules, which are available in EVIDANT, include video summarizer, activity detector, face tracker, people tracker, video comparator and video type classifier.

The development of the demonstrator also proved the significant benefit of VTAPI framework in both processing modules and more complex image and video processing applications and systems implementation.

References

[1] Bouguet, J.-Y. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Technical report, Intel Corporation, 2001.

[2] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2005.* (2005), vol. 1, IEEE, pp. 886–893.

[3] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition* (2014).

[4] Klicnar, L., and Beran, V. Dissimilarity detection of two video sequences. Technical Report VG20102015006-2012-04, Faculty of Information Technology, Brno University of Technology, 2012. This is an expanded version of [7].

[5] Klicnar, L., and Beran, V. Robust motion segmentation for on-line application. In *Proceedings of WSCG'12* (2012), 20-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, University of West Bohemia in Pilsen, pp. 1–6.

[6] Klicnar, L., and Beran, V. Video sequence summarization and synopsis. Technical Report VG20102015006-2014-03, Faculty of Information Technology, Brno University of Technology, 2014.

[7] Klicnar, L., Beran, V., and Zemčík, P. Dissimilarity detection of two video sequences. In *Proceedings of SCCG 2013* (2013), vol. 2013 of *Conference Materials and Posters*, Comenius University in Bratislava, pp. 28–31.

[8] Lienhart, R., and Maydt, J. An extended set of haar-like features for rapid object detection. In *International Conference on Image Processing* (2002), vol. 1, IEEE, pp. 900–903.

[9] Mráček, S. Object detection. Technical Report VG20102015006-2012-02, Faculty of Information Technology, Brno University of Technology, 2012.

[10] Mráček, S. Motion and object detection. Technical Report VG20102015006-2013-04, Faculty of Information Technology, Brno University of Technology, 2013.

[11] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), p. 8.

[12] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. Learning Deep Features for Scene Recognition using Places Database. *Advances in Neural Information Processing Systems 27* (2014), 487–495.

[13] Zivkovic, Z. Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04)* (2004), vol. 2, IEEE, pp. 28–31.