VTApi: an Efficient Framework for Computer Vision Data Mining and Management

Petr Chmelar*, Martin Pesek*, Tomas Volf*, Jaroslav Zendulka*†
†*IT4Innovations Centre of Excellence*
**Faculty of Information Technology, Brno University of Technology*
Brno, Czech Republic
{*chmelarp, ipesek, ivolf, zendulka*}*@fit.vutbr.cz*

***Abstract*—VTApi is an open source application programming interface designed to fulfill the needs of specific distributed computer vision data and metadata management systems and to unify and accelerate their development. It is oriented towards processing and efficient management of image and video data and related metadata for their retrieval, analysis and mining with the special emphasis on their spatio-temporal nature in real-world conditions. VTApi is a free extensible framework based on progressive and scalable open source software as OpenCV for high-performance computer vision and data mining, PostgreSQL for efficient data management, indexing and retrieval extended by similarity search and integrated with geography/spatio-temporal data manipulation.**

***Keywords*-VTApi, computer vision, data management, similarity search, clustering, API, methodology, spatio-temporal.**

I. Introduction

Ever expanding multimedia content necessitates the research of new technologies for content understanding and the development of a wide variety of academic, commerce and government applications [1]. The main objective of the VideoTerror (the Ministry of the Interior) project is to create a prototype of a system warehousing image and video accomplished with computer vision and data mining for preventing and protecting against illegal activities and natural or industrial disasters affecting citizens, organizations or infrastructure. The basic requirements include image and video feature extraction, storage and indexing to enable (content-based) retrieval, summarization and data mining in the meaning of object detection and activity recognition in an interactive and iterative process.

In addition to the technology, we also target usual aspects of the research – to unify and accelerate it by choosing an appropriate design methodology and architectural framework for the composition of domain and application specific tools focusing on open source software. In particular, we propose a solution that will enable the development and adaptation of a complex computer vision application at a reduced cost in terms of time and money. We target this goal by (re)using and integrating tool chains of (CV) methods and (multimedia) data and metadata in an arbitrary combination as simple and versatile as possible.

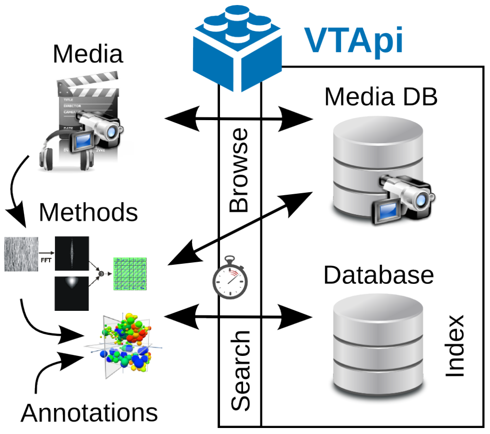

Figure 1: The illustration of VTApi and the concept of methods' chaining.

The VT methodology is based on the fact, that most methods of the same purpose have similar types of inputs and outputs, so there may be chains of them. Moreover, the input of a process (a running instance of a method) can be seen as another process's output (e.g., annotation, feature extraction, classification) including media data creation as illustrated in Figure 1. The VT project is not limited to specific methods or data - they are created by users of (VideoTerror API) VTApi.

In this paper, we present the most general part of the system and methodology – VTApi[1], open source C++ and Python code. At the moment, VTApi is technologically based on a (remote) file-system media storage (with multimedia scraping capability) and PostgreSQL database for metadata management extended by our vector-based simililarity search (distance) metrics, originaly developed for efficient local (invariant) features search (pgDistance). We have integrated GEOS and PostGIS to be able of multidimensional indexing of the geography/spatio-temporal nature of real-world multimedia data acquired by (phones' and surveillance) cameras and appearing objects (trajectories). OpenCV is used as the primary vision framework. In the future, we plan to integrate other technologies and databases (SQLite at the moment) and a NoSQL storage.

[1]http://gitorious.org/vtapi, http://vidte.fit.vutbr.cz/ in Czech

II. State of the Art

In the past decade multimedia technology has become ubiquitous. There is an instantly growing tendency of multimedia data produced by many applications in today's world. It requires to organize and manage this data and to provide support for its processing. First, image processing and data management have been a great challenge for researchers. So far, OpenCV supports only (XML/YAML) file storages, which are flexible, but not really efficient. Content-based image retrieval (CBIR) emerged as an important area in computer vision and information retrieval. Later, the video database management systems were supported by the SQL/MM standard. Its Part 5 Still Image provides structured user-defined types both for still images and their features that allow to store images into a database, retrieve them, modify them and to locate them by applying various "visual" predicates [2]. These data types are implemented in several commercial database products, e.g., in Oracle Multimedia and IBM DB2 Image Extender. There are also some extensions to open source database products to facilitate CBIR system development. For example, PostgreSQL-IE [3] extends the architecture of PostgreSQL.

MPEG-7 standard (Multimedia Content Description Interface [4]) published in 2002 has brought a standard model of multimedia content. However, most of XML-enabled and native XML databases ignore management of time intervals, vectors and matrix data types, which MPEG-7 defines as the extent to the XML. The MPEG-7 model has been adopted or supported by several multimedia database management systems, for example, BilVideo-7 [5]. It supports multimodal queries, video indexing and retrieval and spatio-temporal queries. MPEG-7 Multimedia Database System (MPEG-7 MMDB) [6] is another example. It is based on extensibility services of Oracle 10g. It maps MPEG-7 schema types to database types and introduces new indexing and querying of similarity. Cortina [7] is another CBIR system based on MySQL. Besides image search, retrieval, classification and duplicate detection, it offers the face detection, image annotation and relevance feedback. Another example of a CBIR system is our TRECVid Search 2009 Demo (http://minerva3.fit.vutbr.cz:8080/).

III. Concepts and Specifications

The VTApi framework is based on the following fundamental concepts that grow from both computer vision and data management:

- *Dataset* is a named set of (multimedia) data along with metadata (descriptive data). Datasets can be organized hierarchically, i.e., one may be based on several others. Each dataset contains sequences.
- *Sequence* is a named ordered set of frames. There are two types of sequences - *Video* and *Images*. The ordering of frames in video is time-based. There can be intervals defined for a sequence.
- *Interval* is any subsequence of *Video* or *Images* whose elements share the same metadata (and thus may define their order). Naturally, it can be a video shot or any sequence of frames containing the monitored object in the video or scene. Metadata of an interval are created by a process.
- *Process* (task or operation) is a named run of a method. *Method* defines the custom algorithm and the structure of metadata consumed and produced by its processes. Such a way, processes represent all activities of a VTApi-based application related to video and image data processing. Implementation of a specific method may or may not be included in VTAPI, it is created by developers, who can share the code using VTCli.
- *Tag* is a term representing an ontology class (in hierarchy). Tags are assigned to the multimedia data as descriptor or annotation of a scene, object or action.
- *Selection* is a set of logically related metadata specified by developers as means of doing operations on metadata effectively and making it possible to chain processes. This concept is related to the effective implementation and access to metadata in the database. A common example of a *Selection* is *Interval*.
- *Key-Value* is the basic way of metadata organization in VTApi. It is a generic data structure (associative array) that allows to store data as <key, value> pairs, so changes in data definition do not imply changes of the VTApi code.

In VTApi, all these concepts are mapped to classes as it is shown in Figure 2. Our approach is not based directly on MPEG-7 XML descriptors and description schemes, because they do not provide the flexibility for efficient streaming and database storage and they are tree structured. Thus, we focus more on the BiM (Binary Format for MPEG-7 [4]). We generally support all structures of descriptors including operations as the first order temporal interpolation, spatial transformation and coordinate mapping, including their indexing using GiST and GIN [8]. The same states about MPEG-A (Multimedia application format) Part 10: Surveillance application format.

A. Data Model

The simplified class diagram of VTApi is illustrated in Figure 2. It follows the concepts given in the previous section and operations that logically belong to. Most classes inherit from the class *KeyValues* that provides the basic operations needed to manage key-value pairs, on which the VTApi model is based. The *KeyValues* class is crucial to ensure the functionality and generality of the framework by the main function *next*(), which includes not only navigation over data structures, but also executes database queries, commits changes made by setter methods and commits new data added by adder methods. It uses the lazy approach –

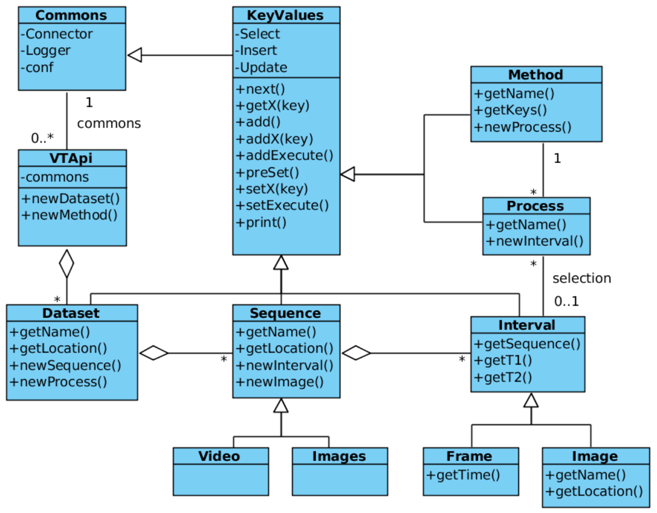

Figure 2: The simplified class diagram of VTApi.

accessing objects only when they are needed by using caches and batches where possible.

Classes derived from *KeyValues* contain only functionality related to the consistency of data and to make some operations easier for VTApi users and factory methods. For instance, *getLocation*() returns the physical data location (e.g., a dataset or a directory with pictures). The method *newSequence*() of the *Dataset* class object is an example of a factory method. It creates a new object of the class *Sequence*, with all necessary parameters. So, then it is possible to access all the current dataset's sequences identified by *getName*() by calling the *next*() method. This is illustrated in the sample code in Section IV-A.

VTApi is strongly typed. The following description uses notation of X referring to any data type implemented as integers, floating points, strings, 4D geometry points, lines and polygons and their structures, vectors, arrays and (OpenCV) matrices. For instance, $getX(k)$ or $setX(k,v)$ operates key k and its value v of type X.

The entry point to the application is the *VTApi* class based on a configuration file and command-line arguments. This class is used to implement the additional command line interface (VTCli[1]), which can give you an insight into the managed data and metadata.

There are several other important classes, some of them nested in the class KeyValues (shown as its attributes) in Figure 2:

- *Commons* class provides a very basic functions such as loading configuration file and command line parameters. It provides a connection to the database (PostgreSQL), a data storage (remote file system) and provides some centralized services, as error processing and logging.
- *Select* class is used to construct queries that as a part of the first call of the function *next*() retrieves the required data from the database. There are special functions to simplify the construction of queries available. Other functions simplify the work with selections, keys and their values to filter queries, use functions and indexes.
- *Insert* class provides insertion of data by the function $addX(k,v)$ into the database. There are two possible ways of inserting – immediate (*addExecute*()) or batch (implicitly) by calling *next*().
- *Update* class allows modification of the current element using the typed family of functions $setX(k,v)$. The approach is similar to the one described in the *Insert* class.

A *Selection* can be either a relational table or a storage in a non-SQL database (using modified *KeyValues* class) for storing data as image features, trajectories or tags. The database schema is available at the project page[1].

IV. Use Cases

We have chosen three simple use-cases demonstrating the use of VTApi for common tasks based on TRECVid evaluations and an object trajectory extraction and transformation experiment. The first use-case is a simple CBIR and the second presents trajectory clustering. It is followed by a performance experiment.

A. Content-Based Image Retrieval

The OpenCV library provides powerful feature extraction and classification techniques. However, it doesn't have capabilities to store the data to be efficiently searched and processed further. This is especially useful for tools like Google Image Search or developed within TRECVid [1], where the retrieval is based on multiple types of (low to high-level, local and global) features related to any object together with annotations (tags).

Thus, we have implemented various similarity-based distance functions. The pgDistance extension (included in the VTApi code) performs the similarity queries measuring distances of feature vectors in PostgreSQL database, e.g., Cosine or Euclidean distance. So that we can employ the feature-based similarity search supported by efficient indexing techniques (SP-)GiST and GIN [8] using Heap and Bitmap indexes, Quad-Tree, KD-Tree, R-Tree, Inverted Document Index and other general indexing structures, supporting containment and nearest neighbor search on vectors (using `@>` and `<->` operators). However, the example in Figure 3 is quite simple. Assume you have a large dataset of images called "search" already populated in the database, and we want to perform MPEG-7 Color layout descriptor based CBIR.

B. Object tracking, trajectory querying and analysis

In the surveillance video, it is important to be able to track moving objects and to extract their visual and spatio-

```
// VTApi entry point, using Dataset "search"
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("search");
dataset->next();
// code of the ColorLayout Method, using Selection "image"
Image* image = new Image(&dataset, "image");
while (image->next()) {
    int[32] color = colorLayout(image->getDataLocation());
    image->setIntA("color", color, 32);
}
// retrieve Image(s) according to their similarity to "Q.jpg"
Image* nearest = new Image(&dataset);
nearest->select->from("image", "distance_square_int4(color, "
    + toString(colorLayout("Q.jpg")) + ")");
nearest->select->orderby("distance_square_int4");
nearest->next(); // is the most similar image
```

Figure 3: A simple CBIR code example

temporal features. Such extracted metadata then should be cleaned, stored and indexed to be able to query and analyze.

Object tracking is a complex task, especially in crowded scenes. We created a tracker based on OpenCV blobtrack demo that we extended with feature extraction as in [9]. The outputs of such methods include spatio-temporal locations in the form of trajectories, blobs and other features of moving objects.

The features might be used and searched for similarity by VTApi. A trajectory query may relate either to relationships between moving objects or a specific spatio-temporal region. Such an analysis can be performed both on VTApi clients and server, because we have adopted the OpenGIS GEOS library, that has been adopted by PostGIS. In order to perform these operations efficiently, VTApi adds a binary access to geometry types and n-dimensional cubes that are used as spatio-temporal minimum bounding boxes of (moving) objects.

Data mining and machine learning techniques can be performed on moving objects metadata. Such an analysis may involve trajectory clustering, classification, object recognition, outliers detection and so on. The following example shows a clustering of trajectories using VTApi and an OpenCV implementation of Expectation-maximization (EM) algorithm, which estimates parameters of a Gaussian mixture model (GMM) [10]. First, feature vectors representing trajectories are read from the database and training samples for the EM algorithm are prepared (see Figure 4). Suppose that trajectories are stored in selection "tracks" in this example. Second, GMM is trained by the EM algorithm and appropriate cluster labels are stored in the database.

We performed the trajectory clustering on a set of trajectories extracted from the second dataset of videos forming the i-LIDS dataset (http://www.homeoffice.gov.uk/science-research/hosdb/i-lids/) used for NIST evaluations; it comes from five cameras at the LGW airport. An example of visualization of some obtained results is shown in Figure 5. There is a result of clustering trajectories from the first camera

```
Mat samples; // cv::Mat of training feature vectors
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("train");
dataset->next();
Sequence* sequence = dataset->newSequence();
while (sequence->next()) { // for each video
  Interval* track = new Interval(*sequence, "tracks");
  while (track->next()) { // for each trajectory
    Mat sample; // cv::Mat for feature vector of trajectory
    float feature = track->getFloat("feature");
    // ... read features and fill feature vector
    samples.push_back(sample);
  }
}

CvEM model, labels; // GMM-EM model and cluster labels
CvEMParams params; // EM parameters
// ... set EM parameters including number of clusters
model.train(samples, Mat(), params, labels);
// ... choose dataset and sequences according to code above
while (track->next()) {
  Mat sample;
  // ... read features and fill feature vector
  int cluster = (int)model.predict(sample); //get cluster label
  track->setInt("cluster", cluster); // store cluster label
}
```

Figure 4: Sample code reading trajectories and preparing training samples (first block), training GMM and storing cluster labels (second block).

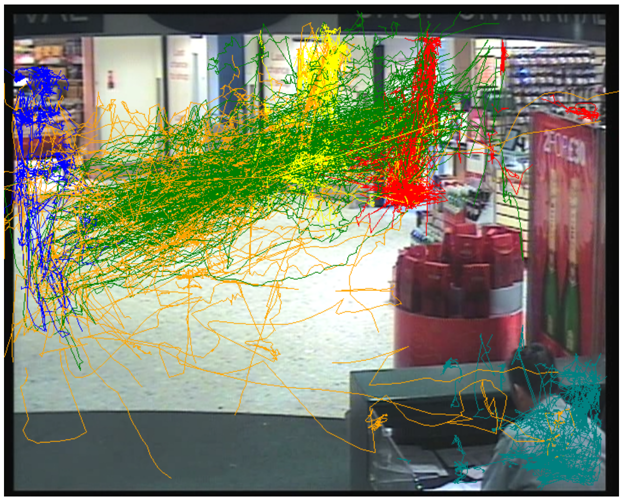

Figure 5: Examples of trajectory clustering results obtained by EM algorithm on trajectories from the first camera.

using the EM algorithm mentioned above. Different colors of trajectories refer to different clusters. We have prepared also an outliers analysis within the VideoTerror project.

C. *Real-time tracking and trajectory indexing*

Because processing trajectories often relates to the real-time, we have performed experiments focused on how much time and resources are needed for the trajectory management. In the experiment (see Table I), we transformed tracked trajectories, we stored them in the database as

Table I: A simple performance test of insertion and querying trajectory data.

	insert/update (7269)	select (7269)	find similar (4)	total network data
(a) local	220 ± 1 s	43 ± 1 s	0.1 s	0
(b) remote	220 ± 5 s	74 ± 1 s	0.2 s	334 MB

vectors capable of the first order temporal interpolation even stored as discrete points, and we index them using 3D bounding-cube and GiST (bitmap index), so that they can be retrieved very efficiently. The similarity query was performed by the containment operator (`@>`) returning 4 trajectories contained in a spatio-temporal bounding box (selected randomly).

At this simple demonstration we dealt with 7269 trajectories, tracked of about 4 hours of video (49:17 minutes by 5 cameras in parallel) in a very crowdy airport traffic of the i-LIDS dataset. According to the table, we show that this system can eventually run in real-time both on: (a) 13” notebook (dual 1.4 GHz ULV processor, 2 GB RAM, 128 GB SSD) including both the trajectory processing part and the local database and (b) the same machine connected using Ethernet network to a remote VTApi database server, where the network delivery time must be taken into consideration in favor of the server hardware.

V. Conclusion

In the paper, we present an innovative open source computer vision data and metadata management framework we offer to the public. The main advantages of the proposed API is the reduction of effort and time to produce high-quality intelligent vision and mining applications by unified and reusable methods and multimedia data and metadata sets on all levels. Above that, we offer novel data and methods interfaces and methodology to be used by researchers and developers of both academic and commercial sectors to collaborate and chain their efforts.

We have selected, integrated and extended a set of progressive and robust open source tools to be efficient for multimedia data and related metadata storage, indexing, retrieval and analysis. The system uses the best from (post)relational databases, it offers alternative storages and data structures we need to manage (e.g. vectors or matrices) to make the data access more efficient, especially for rapidly changing geography/spatio-temporal data of a very complex nature in the binary form that can be now processed both on VTApi clients and in the database.

Also, we plan to extend VTApi with the MPEG-7 Library to provide a standardized framework for methods’ performance evaluation, followed by the KALDI audio and speech processing framework and some others to enable further multimedia analysis and mining.

Acknowledgment

This work was partially supported by the research plan MSM0021630528, the specific research grant FIT-S-11-2, the VG20102015006 grant of the Ministry of the Interior of the Czech Republic and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] A. F. Smeaton, P. Over, and W. Kraaij, “Evaluation campaigns and trecvid,” in *MIR ’06: Proc. of the 8th ACM Int. Workshop on Multimedia Information Retrieval*. New York, NY, USA: ACM Press, 2006, pp. 321–330.

[2] J. Melton and A. Eisenberg, “SQL multimedia and application packages (SQL/MM),” *SIGMOD Rec.*, vol. 30, no. 4, pp. 97–102, Dec. 2001.

[3] D. Guliato et al., “POSTGRESQL-IE: An image-handling extension for postgreSQL,” *Journal of Digital Imaging*, vol. 22, pp. 149–165, 2009.

[4] H. Kosch, *Distributed Multimedia Database Technologies: Supported MPEG-7 and by MPEG-21*. Boca Raton: CRC Press, 2004.

[5] M. Bastan et al., “BilVideo-7: An MPEG-7- compatible video indexing and retrieval system,” *IEEE Multimedia*, vol. 17, pp. 62–73, 2010.

[6] M. Döller and H. Kosch, “The MPEG-7 multimedia database system (MPEG-7 MMDB),” *J. Syst. Softw.*, vol. 81, no. 9, pp. 1559–1580, Sep. 2008.

[7] E. D. Gelasca et al., “CORTINA: Searching a 10 million + images database,” Tech. Rep., Sep 2007. [Online]. Available: http://vision.ece.ucsb.edu/publications/elisa_VLDB_2007.pdf

[8] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, “Generalized search trees for database systems,” in *VLDB’95, Zurich, Switzerland*. Morgan Kaufmann, 1995, pp. 562–573.

[9] P. Chmelar, A. Lanik, and J. Mlich, “SUNAR: Surveillance network augmented by retrieval,” in *Advanced Concepts for Intelligent Vision Systems*, ser. LNCS. Springer Berlin/Heidelberg, 2010, vol. 6475, pp. 155–166.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Singapore: Springer, 2006.