

# VTApi v. 1.5

Technická zpráva - FIT - VG20102015006 - 2012 - 03

Petr Chmelař, Vojtěch Fröml, Tomáš Volf  
Martin Pešek, Jaroslav Zendulka



Fakulta informačních technologií, Vysoké učení technické v Brně

20. prosince 2012

## Abstrakt

VTapi, zkráceně Video Teror API (Application Programming Interface) a doplňující metodika je rozhraní pro zpracování a efektivní správu, indexování, vyhledávání a analýzu obrazových, video a přidružených metadat, které bylo vytvořeno za účelem sjednocení a urychlení vývoje inteligentních aplikací počítačového vidění.

VTapi je v současné verzi založeno na databázi PostgreSQL obsahující rozšíření PostGIS (Geografický Informační Systém), GEOS (Geometry Engine, Open Source) a především OpenCV (Open Source Computer Vision Library).

## Abstract

VTapi is a short name of Video Teror Application Programming Interface and methodology, a database interface used for processing and efficient management, indexing, retrieval and analysis of image and video data and related metadata, which is being developed to unify and accelerate the intelligent vision applications development.

VTapi is based on PostgreSQL database including PostGIS (Geographic Information System) and GEOS (Geometry Engine, Open Source) extensions, and especially OpenCV (Open Source Computer Vision Library).

# Obsah

1	Úvod.....	1
2	Základní pojmy .....	3
3	Příklady použití .....	4
4	Popis řešení .....	7
4.1	Diagram tříd.....	7
4.2	Databázový model.....	7
4.3	Přístup k obrazovým datům.....	9
4.3.1	Třída Dataset.....	9
4.3.2	Sequence .....	12
4.3.3	Interval .....	14
4.4	Správa a dotazování dat.....	16
4.4.1	Operace next() .....	17
4.4.2	Sada operací getX .....	17
4.4.3	Třída Select .....	19
4.4.4	Třída Insert .....	20
4.4.5	Třída Update.....	20
4.5	Konfigurace .....	21
5	Implementace případových situací .....	23
5.1	Vyhledávání a klasifikace statických obrázků .....	23
5.2	Shlukování trajektorií a detekce odlehlých trajektorií .....	24
5.3	Sledování objektů v reálném čase.....	26
6	Technická dokumentace .....	27
6.1	VTcli.....	27
6.1.1	Ukázky použití aplikace VTcli.....	29
6.2	Poznámky k instalaci.....	30
7	Závěr.....	31
8	Příloha Sady dat.....	32
9	Literatura .....	33

# 1 Úvod

Hlavním cílem projektu VideoTeror je definovat, zkoumat a vytvořit funkční vzorek systému pro zpracování záznamů obrazového a video charakteru za účelem boje proti terorismu.

Tento systém bude sestávat z "datového skladu", do kterého se budou umisťovat záznamy obrazů a videosekvencí, a databáze, v níž budou uloženy výsledky analýzy dat na různých úrovních. Dotazy na výsledky analýzy tak budou prováděny ve formě databázových dotazů. Algoritmy analýzy budou získávat údaje z původních dat v kombinaci s dotazováním výsledků v databázi. Výsledky budou ukládány zpět do databáze. Tímto postupem se vytvoří analytické prostředí, které umožní kombinaci řady přístupů a stane se flexibilním prostředím pro pokročilou analýzu dat. Funkce, které by mohly být v systému implementovány, jsou například:

- ⤴ Skladování anotovaných videozáznamů, například typu "anotovaná videosekvence" s identifikací typů scény a výskytem objektů ve scéně. Samotné pořizování takových dat není předmětem projektu a spíše se předpokládá získání dat od zadavatele.
- ⤴ Extrakce příznaků z obrazů a videosekvencí, například založených na detekci klíčových bodů a obdobných (SIFT, SURF, MSER apod.), integrálních a lokálních vlastností obrazů (histogramy jasů, barev, gradientů) a lokálních příznaků (LBP, LRF, Haarovy apod.)
- ⤴ Strojové učení například pro parametrizované trénování metodou SVM pro analytické úlohy včetně implementace některých vybraných analytických úloh, jako jsou například detekce typu scény na základě anotovaných příkladů, sumarizace videosekvencí, detekce objektů (obličeje osob, dopravní značky apod.)
- ⤴ Aplikace dalších metod získávání znalostí z dat, například pro detekci nežádoucích událostí, perzistentní sledování podezřelých osob v prostoru pokrytém několika kamerami apod. Vybrané biometrické metody pro identifikaci osob a objektů (například pro zkoumání vztahů mezi mírami objektů, metriky dynamiky pohybu, případně identifikaci osob).

Na základě tohoto cíle bylo vytvořeno aplikační rozhraní VTapi, což je soubor tříd (knihovna) pro zpracování záznamů videa a obrazové informace – kategorizace, vyhledávání a porovnávání, cílem je sjednotit a urychlit vývoj aplikací v oblasti počítačového vidění (v rámci projektu VideoTeror na FIT VUT v Brně). Je implementováno v programovacím jazyce C++ (dostupná také v jazyce Python). Využívá nástroje pro zpracování obrazové informace (OpenCV, <http://opencv.willowgarage.com/wiki/>) a post-relační databázi PostgreSQL (<http://www.postgresql.org/>) pro uchování metadat k obrazovým datům.

VTapi bude doplněno i metodikou použití, která bude zahrnovat registraci a správu metod vytvářejících metadata a to k jednotlivému použití či řetězení

v komplexním analytickém procesu. VTApi je tedy primárně určeno pro zjednodušení a podporu vývoje komplexních analýz, vyhledávání či dolování z video a obrazových dat. Umožňuje programátorům aplikací analyzujících video a obrazová data soustředit se pouze na vývoj samotných analytických metod.

Tato technická zpráva navazuje na zprávu projektu z roku 2011 [1], kde byla prezentována verze 1.0 VTApi. Předkládaná zpráva obsahuje aktualizovaný a rozšířený popis jak funkčnosti dostupné již ve verzi 1.0, tak zejména nové rysy rozhraní ve verzi 1.5. Lze ji proto považovat za základní dokumentaci k VTApi, ke které jsou doplňující dokumentací generovaná dokumentace a zdrojové kódy dostupné na <https://gitorious.org/vtapi/>.

## 2 Základní pojmy

V rámci projektu VideoTeror, jsme se shodli na následujících základních konceptech souvisejících se správou video a obrazových dat a metadat k nim se vztahujícím<sup>1</sup>:

- ↗ **Datová sada** (Dataset) je množina (multimediálních) dat spolu s metadaty (popisem dat). Datové sady jsou logicky vzájemně disjunktní, ale některá může být založena na několika dalších.
- ↗ **Sekvence** (Sequence) je množina snímků (videa) nebo obrázků, je základní jednotkou datové sady. V sekvenci je předpoklad časové uspořádanosti snímků.
- ↗ **Interval** je podmnožina sekvence (množina snímků) zvolená tak, aby bylo možné jim přiřadit shodné informace (metadata). Příkladem je sledovaný objekt ve videu nebo jeho scéna. Metadata mohou být obecně libovolná, ale jsou vždy vytvářena procesem, který je *instancí* nějaké *metody*.
- ↗ **Metoda** (Method) definuje strukturu metadat.
- ↗ **Proces** (Process) jako konkrétní instance nějaké metody vyváří metadata se strukturou definovanou metodou, která jsou extrahována ze zdrojových video či obrazových dat nebo vytvořena „ručně“. Proces tedy reprezentuje základní jednotku poskytované funkčnosti vytvářeného systému.

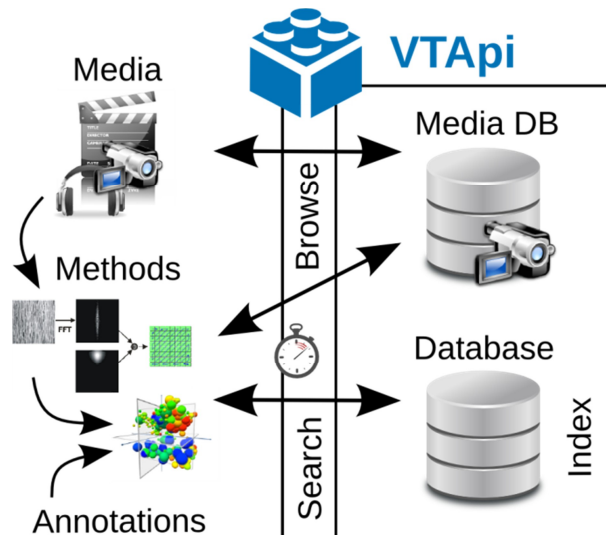
Následují doplňující pojmy:

- ↗ **Obrázek** (Image) a **rámec** videa (Frame), případně jejich kolekce (Images a Video), v jejich běžné interpretaci, reprezentují instance intervalů, respektive sekvencí.
- ↗ **Tag** je indexační termín. Tyto jsou (v hierarchii) přiřazené multimediálním datům pro jejich popis (anotaci).
- ↗ **Selekce** (Selection) je podmnožina logicky souvisejících metadat vhodně zvolená tak, aby operace (dotazy) nad nimi byly efektivní a umožňovaly přirozené řetězení procesů (vstupem jednoho je výstup druhého nebo obrazová data). Speciální příklady selekce jsou intervaly nebo tagy.
- ↗ **Klíč-hodnota** (Key-Value) je základní mechanismus organizace metadat ve VTAPI. Je to obecná datová struktura umožňující uchování dat ve dvojici <klíč, hodnota> tak, aby při změně definice dat nebylo nutné měnit kód VTAPI.

Základní koncepce vyvíjeného systému a role VTAPI v něm je znázorněna na obrázku *Obrázek 2.1*.

---

<sup>1</sup> Uvedeným konceptům odpovídají v rozhraní VTAPI softwarové třídy označené anglickými názvy. Ty jsou, liší-li se od názvů v češtině, uvedeny v závorce.



Obrázek 2.1 Základní koncepce vyvíjeného systému

### 3 Příklady použití

Pro účely zpřesnění požadavků na použití navrhovaného systému byly definovány čtyři modelové situace (případové studie).

#### Situace 1: Sledování objektů více kamerami

V objektu je několik kamer, každá snímá z jiného místa/úhlu. Probíhá záznam videa z kamer. Obecné dotazy na databázi poté jsou:

- ⤴ Zobraz video sekvence v určitých časových rozpětích z určitých kamer (7:00 - 11:30; 12:00 - 18:00).
- ⤴ Zobraz zrychlené/zpomalené (násobek) záznamy v daných časových okamžicích z výběru kamer (z datasetu) (7:00 - 11:30; 12:00 - 18:00).
- ⤴ Možnosti synchronizace videa v rámci datasetu (shift videa do minulosti/budoucnosti), příkladem mohou být stereokamery, rozladěné hodiny na kamerách apod.

Dotazy/situace týkající se trackingu:

- ⤴ Časově závislé (umístění, velikost) ROI trackovaného objektu; anotace uložená v DB z jednoho videa; Dotaz: ukaž mi ROI sledovaného objektu v ostatních videích v datasetu (dotaz tvořící nové anotace pomocí algoritmu).
- ⤴ Situace: kamera + termokamera na jednom místě.
- ⤴ Situace s termokamerou. Dotaz: zobraz úseky videí (abstrakt frames) z datasetu, kde pohybující se objekty měly vyšší teplotu než 30°C (detekce obličejů).
- ⤴ Může pomoci synchronizaci kamer/trackovaných objektů, výpočtu vzdálenosti objektu (obecně algoritmům).
- ⤴ Určení vzdálenosti sledovaného objektu – stereokamery.

## Situace 2: Klasifikace segmentů videa

Video záznamy uložené v databázi jsou rozděleny na segmenty, které lze reprezentovat jedním nebo více klíčovými snímky (inspirováno úlohou TRECVID SIN). Druhou alternativou je, že se pracuje pouze s klíčovými snímky, tedy obrázky, ale může se jednat o libovolné obrázky v databázi (např. projekt ITS - Image Tag Suggestion). Cílem je extrahovat informace z obrázků tak, aby je bylo možné klasifikovat do předem definovaných kategorií (v terminologii VTApi: *přidělit jim tagy*):

- ⤴ Možnost ukládat obrázky/klíčové snímky do databáze (úložiště)
- ⤴ Možnost načtení obrázků z úložiště a jejich vyhledávání podle tagů přiřazených (anotace) nebo automaticky zjištěných (klasifikace). Uchovávat a dotazovat tyto tagy.
- ⤴ Podobnostní dotaz na obrázky
- ⤴ Možnost přidat proces (2 typy metod – anotace a klasifikace).
- ⤴ Přidat sekvenci (soubor obrázků).
- ⤴ Přidat intervaly (obrázky).
- ⤴ Možnost vložit výsledky procesu.
- ⤴ Update výsledků procesu, na dotaz: ke kterým obrázkům není spočítaný proces.

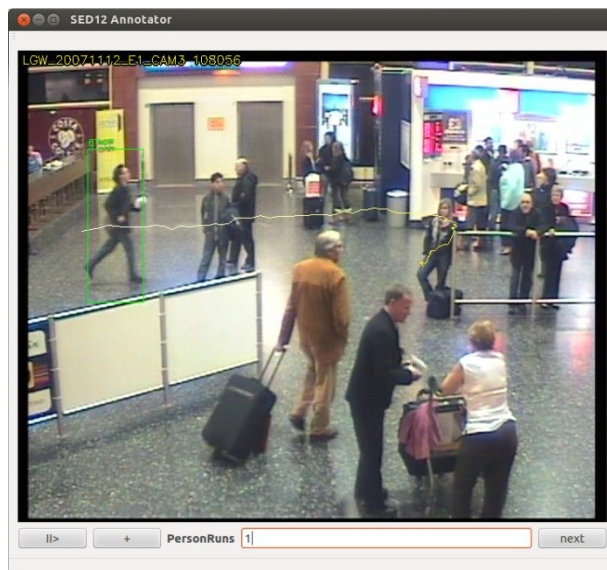
## Situace 3: Klasifikace objektů v dohledovém videu

Video záznamy dohledového videa jsou zpracovány a metadata uložena v databázi, systém by měl nabízet funkčnost obdobnou Situaci 1. Navíc by měl poskytovat podporu pro anotování, fázi učení, testování a aplikaci vytvořených klasifikačních modelů. Obrázek *Obrázek 3.1* ukazuje možné je jednoduché uživatelské rozhraní s funkcností aktivního učení.

Cíle této situace jsou shodné s cíli evaluace TRECVID Interactive SED (surveillance event detection). Jejich úlohou je podpořit interaktivní technologii pro detekci vizuálních událostí ve velké kolekci videa. Tyto události zahrnují například:

- ⤴ PersonRuns – člověk běží.
- ⤴ CellToEar – telefonování nebo používání telefonu.
- ⤴ ObjectPut – odložení nebo předání objektu (ale ne dítěte).
- ⤴ PeopleMeet – setkání několika osob.
- ⤴ PeopleSplitUp – rozdělení osob.
- ⤴ Embrace – objetí.
- ⤴ Pointing – ukazování.
- ⤴ ElevatorNoEntry – výtah byl přivolán, ale nebyl použit.
- ⤴ OpposingFlow – postup proti pohybu osob.
- ⤴ TakePicture – fotografování.





Obrázek 3.1 Ilustrace interaktivního uživatelského rozhraní anotátoru událostí.

#### **Situace 4:** Sledování objektů v dohledovém videu a analýza jejich trajektorií

V prostředí dohledového videa je důležité sledovat pohybující se objekty a extrahovat jejich vizuální a časoprostorové rysy. Taková metadata by pak pro účely dotazování a analýzy měla být vyčištěna, uložena a indexována.

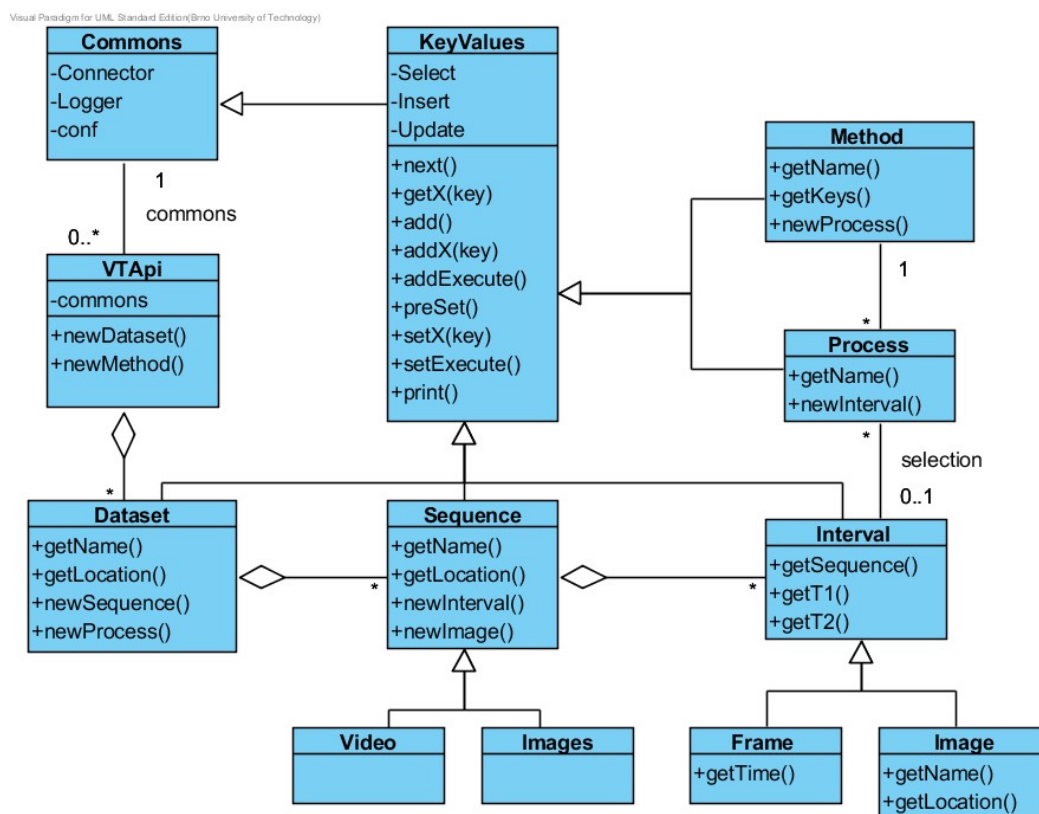
Sledování objektů je složitá úloha, zejména pokud je prováděno v zaplněných scénách. Pro sledování objektů mohou být využity různé nástroje či metody. Výstup takové metody může zahrnovat trajektorie v podobě sekvencí časoprostorových pozic, tvary objektů a případně další rysy pohybujících se objektů. Všechny tyto rysy mohou být dále agregovány, sumarizovány, analyzovány a obohaceny o případné další rysy jako jsou anotace, tagy nebo příslušnost ke třídám.

Pro analýzu dat o pohybujících se objektech mohou být použity různé metody dolování z dat a strojového učení. Taková analýza může zahrnovat shlukování trajektorií, klasifikaci, rozpoznávání objektů, detekci odlehlých trajektorií a další.

## 4 Popis řešení

### 4.1 Diagram tříd

Obrázek 4.1 představuje zjednodušený (jen některé třídy a některé vlastnosti) diagram tříd VTapi s třídami podstatnými z hlediska uživatele. Tyto třídy odpovídají konceptům uvedeným v kapitole 2 *Základní pojmy*.



Obrázek 4.1 Zjednodušený diagram tříd VTapi

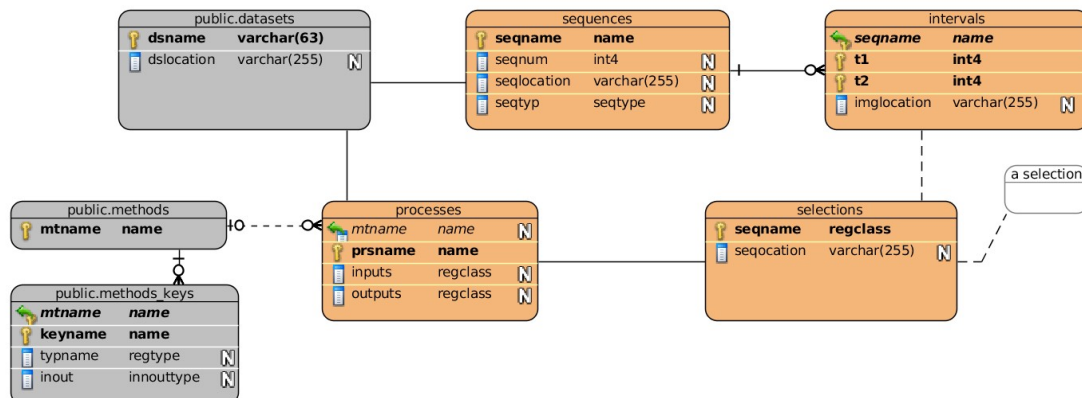
Všechny tyto třídy dědí převážnou většinu své funkčnosti z obecné třídy **KeyValues** (klíč-hodnota sloupce v tabulce), která v sobě obsahuje funkce pro manipulaci s databázovými objekty prostřednictvím vnořených tříd **Select**, **Insert** a **Update**. Specifikaci a funkčnosti těchto tříd se věnují kapitoly 4.3 *Přístup k obrazovým datům* a 4.4 *Správa a dotazování dat*.

Třída **VTapi** slouží jako vstupní třída API. Její hlavní funkcí je umožnění konfigurace připojení k databázi a dalších vstupních parametrů prostřednictvím vnořené třídy **commons**. Konfiguraci se věnuje kapitola 4.4 *Správa a dotazování dat*.

### 4.2 Databázový model

Jak bylo vidět na obrázku *Obrázek 2.1*, jsou metadata ukládána v databázi. Odpovídající zjednodušený logický datový model je uveden na obrázku *Obrázek 4.2*. Šedé objekty mají globální charakter, oranžové jsou specifické pro

každou datovou sadu (nachází se v odpovídajícím databázovém schématu – viz dále).



Obrázek 4.2 Zjednodušený logický datový model

Porovnáním s diagramem tříd z obrázku *Obrázek 4.1* lze jednoduše odvodit, že:

- třídy **Dataset**, **Sequence**, **Interval**, **Method** a **Process** přímo odpovídají entitním množinám datového modelu, resp. tabulkám databáze. Zapouzdřují přístup k perzistentním objektům uloženým v nich a prováděním databázových dotazů umožňují jejich dotazování, vkládání, mazání a aktualizaci.
- třídy **Video** (video) a **Images** (sada obrázků), resp. **Frame** (snímek videa) a **Image** (obrázek) jako specializace tříd **Sequence**, resp. **Interval** jsou mapovány na entity množin **sequences**, resp. **intervals** datového modelu.

Entitní množiny datového modelu lze rozdělit do následujících kategorií:

- **Reprezentace obrazových dat**
  - **public.datasets**: obsahuje informace o různých datových sadách. Tyto sady jsou typicky určeny pro různé projekty. Každá sada má své databázové objekty umístěny ve vlastním databázovém schématu, schéma *public* je základní.
  - **sequences**: informace o videích či složkách obrázků
  - **intervals**: informace o jednotlivých obrázcích či částech videa
- **Reprezentace algoritmů**
  - **public.methods**: registrované metody. Jsou společné pro všechny datové sady. Strukturu vstupních a výstupních dat metody určuje kolekce objektů **public.methods\_keys**.
  - **processes**: procesy spuštěné nad jednou datovou sadou.

- **Optimalizace**

- **selections:** výběry dat (selekce) vztahujících se k dané datové sadě. Jedná se o podmnožinu všech obrazových dat (intervalů). Jednu takovouto selekci představuje objekt *a selection*.

V následujících podkapitolách budou popsány mechanismy užití a implementace uvedených tříd ve VTapi.

### 4.3 Přístup k obrazovým datům

Aplikace na vyhledávání či získávání znalostí z obrazových dat typicky požaduje v souladu s koncepcí z obrázku *Obrázek 2.1* úložiště pro dva typy dat:

- *Vstupní multimediální data:* zpracovávané **obrázky** (či jejich sady) a **videa**.
  - Ukládány uživatelem do **diskových souborů**, logicky uspořádaným do adresářů
- *Vstupní / výstupní popisná data:* související metadata např. ve formě **anotací** či **klasifikací**.
  - Ukládána v **databázi**

VTapi zapouzdřuje přístup a manipulaci s oběma uvedenými typy dat. Poskytuje metody pro reprezentaci uložených obrazových dat (místo uložení, logická hierarchie, popis multimediálního typu a další specifické informace). Tato funkčnost je reprezentována třídami **Dataset**, **Sequence** a **Interval**, které dědí společnou funkčnost ze třídy **KeyValues**.

Následující sekce podrobněji popisují, jaký je vztah těchto tříd k organizaci zdrojových dat v souborovém systému, jak jsou objekty těchto tříd reprezentovány v databázi a jaké operace definují samotné softwarové třídy. Pro ilustraci jsou zde použity jednoduché příklady přístupu v C++. Použití VTapi pro implementaci některých případových studií z kapitoly 3 *Příklady použití* obsahuje kapitola 5 *Implementace případových situací*.

#### 4.3.1 Třída Dataset

Množina dat související s jednou datovou sadou<sup>2</sup> a současně jedním projektem. Jak už bylo uvedeno, jsou datové sady obvykle navzájem disjunktní, nicméně jeden může být modelován na základě předchozího.

##### Zdrojová data

Na úrovni diskových souborů se jedná o adresář obsahující soubory zpracovávaných videí a/nebo adresáře s obrázky. Tento adresář je uložen

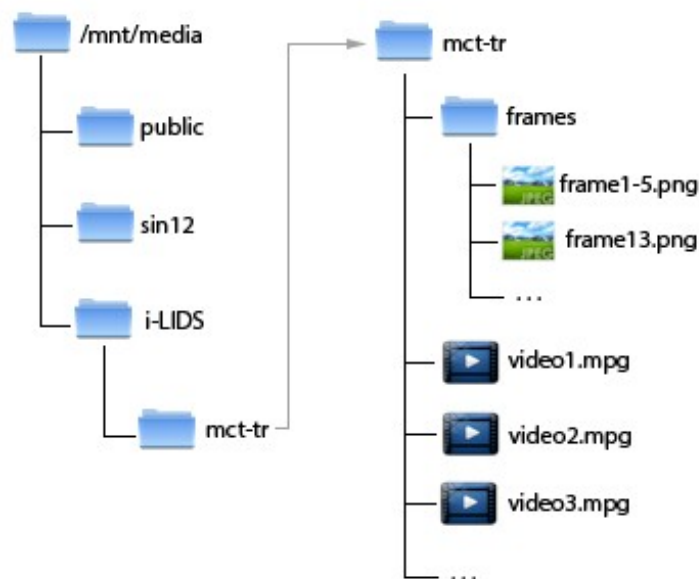
---

<sup>2</sup> V dalším textu je používáno pro datovou sadu na některých místech používáno i označení *dataset*.

v základním umístění (*location*), které je konfigurovatelné před použitím VTApi (více v kapitole 4.5 Konfigurace).

Obrázek 4.3 ilustruje příklad reprezentace tří datových sad na disku. Struktura adresářů je následující:

- Základním umístěním je adresář */mnt/media*.
- Datové sady *public* a *sin12* zde nejsou zobrazeny podrobně.
- Datová sada *i-LIDS* má komplikovanější cestu k datům, ta jsou uložena v jeho podadresáři *mct-tr*. Obsahuje jednu složku s obrázky *frames* a množství zpracovávaných videí



Obrázek 4.3 Ukázka adresářové struktury datových sad

### Databázový objekt

Seznam všech dostupných datových sad je uložen v tabulce **public.datasets**. Může vypadat následovně:

	dsname [PK] name	dslocation character varying	userid name	groupid name	created timestamp with timestamp	changed timestamp	notes text
1	public	public/	vfroml	vidte	2012-02-03		test dataset
2	sin12	sin12/	vfroml	vidte	2012-09-23		
3	sunar	i-LIDS/mct_tr/	vfroml	vidte	2012-02-03		TrecVID

Obrázek 4.4 Ukázka seznamu datových sad v tabulce *public.datasets*

Tabulka popisuje **databázi**, která obsahuje pro každou datovou sadu:

- název schématu, kde jsou umístěné databázové objekty reprezentující daný dataset (hodnota ve sloupci *dsname*)

- název adresáře, kde jsou umístěna zdrojová data sady (hodnota ve sloupci *dslocation*). Zdrojová data jsou pak uložena v adresáři „*location/dslocation*“ (k *location* více v kapitole 4.5 Konfigurace)
- další doplňující informace (vlastník, skupina, datum vytvoření, poznámky)

Každé **schéma** odpovídající datové sadě typicky **obsahuje** následující objekty:

- základní tabulky *sequences, intervals, methods a processes* (viz datový model na obrázku *Obrázek 4.2*)
- vlastní tabulky/sloupce tabulek **výstupních dat**, specifické pro vyvíjenou aplikaci
- uživatelem definované **datové typy**, lze rozdělit do dvou kategorií:
  - **povinné**: zejména výčtové typy, nutné v každé datové sadě
  - **nepovinné**: zejména geometrické datové typy, příp. typy pro OpenCV, specifické pro konkrétní datovou sadu/projekt
- další specifické databázové objekty (sekvence, trigger, pohledy atd.)

Základní schéma databáze (základní tabulky – viz výše) odpovídající jednomu datasetu se v současné verzi VTapi z bezpečnostních důvodů **vytváří manuálně** administrátorem databáze.

### Softwarová třída

Objekt reprezentující existující datový soubor lze v aplikaci vytvořit k další manipulaci jednoduše tovární metodou třídy VTapi:

```
// vstupní třída API
// nastavení konfiguračním souborem "./vtapi.conf" (viz kap. 3.6)
VTapi* vtapi = new VTapi("./vtapi.conf");
// Vytvoření objektu reprezentujícího dataset sunar
Dataset* ds_sunar = vtapi->newDataset("sunar");

// ... práce

// Destrukce objektů
delete (ds_sunar);
delete (vtapi);
```

Třída *Dataset* dědí metody a atributy třídy *KeyValues* (viz kapitola 4.4 *Správa a dotazování dat*), poskytuje následující specifické operace:

*Tabulka 4.1* Specifické operace třídy *Dataset*

string getName ( )	
<ul style="list-style-type: none"> <li>• získá název datové sady</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ string: název datové sady</li> </ul> </li> </ul>

string getLocation ( )	
<ul style="list-style-type: none"> <li>• získá umístění datové sady</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ string: umístění datové sady</li> </ul> </li> </ul>
Sequence* newSequence ( const string& name )	
<ul style="list-style-type: none"> <li>• vytvoří objekt pro reprezentaci sekvencí (video, složky obrázků) pro aktuální datová sada</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ name: nepovinný, lze jím vybrat pouze jednu sekvenci datové sady</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ Sequence*: ukazatel na objekt reprezentující sekvence</li> </ul> </li> </ul>
Video* newVideo ( const string& name )	
<ul style="list-style-type: none"> <li>• vytvoří objekt pouze pro reprezentaci videa pro aktuální datová sada</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ name: nepovinný, lze jím vybrat pouze jedno video datové sady</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ Video*: ukazatel na objekt reprezentující video (video)</li> </ul> </li> </ul>

### 4.3.2 Sequence

Základní jednotka datové sady, reprezentuje jednu část dat, která je vnitřně časově uspořádána.

#### Zdrojová data

Na úrovni diskových souborů se jedná o samostatná videa či adresáře s obrázky. Tato data jsou uložena v umístění datové sady, do kterého přísluší. V obrázku *Obrázek 4.4* (výše) existují v datové sadě *sunar* čtyři sekvence: *frames* (složka obrázků), *video1.mpg*, *video2.mpg* a *video3.mpg* (video).

#### Databázový objekt

Seznam všech sekvencí příslušné datové sady je uložen v tabulce **[jméno\_datasetu].sequences**. Záznam o sekvenci se do ní uloží pomocí operace *Sequence->add()* (viz tabulka *Tabulka 4.2*). Tabulka *sequences* může vypadat následovně:

	seqname [PK] name	seqnum integer	seqlocation character varying	seqtyp seqtype	userid name	groupid name	created timestamp with timestamp	changed timestamp	notes text
1	frames	1001	frames/	images	vfroml	vidte	2012-02-03		
2	video1	1002	video1.mpg	video	vfroml	vidte	2012-02-03		
3	video2	1003	video2.mpg	video	vfroml	vidte	2012-02-03		
4	video3	1004	video3.mpg	video	vfroml	vidte	2012-02-03		

*Obrázek 4.5* Ukázka seznamu sekvencí v tabulce *sunar.sequences*

Je vidět, že obsahuje podobná metadata jako tabulka *public.datasets*, tentokrát vztahující se k sekvencím.

#### Softwarová třída

Objekt reprezentující všechny sekvence datové sady nebo jenom jednu konkrétní sekvenci lze v aplikaci vytvořit tovární metodou třídy *Dataset*:

```

// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující všechny sekvence datasetu sunar
Sequence* all_sunar = ds_sunar->newSequence();
// Objekt reprezentující pouze sekvenci video2
Sequence* video2_sunar = ds_sunar->newSequence("video2");

// .. práce

// Destrukce objektů
delete (all_sunar); delete (video2_sunar);
delete (ds_sunar);

```

V některých situacích může být vhodné pracovat pouze s jednou vybranou sekvencí, nikoli se všemi dostupnými. Z ukázky je zřejmé, že tohoto omezení lze dosáhnout omezením názvu sekvence ve funkci *newSequence*.

Jako alternativu ke třídě *Sequence* lze použít intuitivnější třídu **Video**, která poskytuje stejnou funkčnost a navíc zajišťuje kontrolu konzistence a správný formát videa. Je tedy závislá na knihovně OpenCV.

```

// Objekt reprezentující video2
Video* video2_sunar = ds_sunar->newVideo("video2");

```

Třída *Sequence* (a rovněž *Video*) opět dědí většinu svých metod a atributů ze třídy *KeyValues* (viz kapitola 4.4 *Správa a dotazování dat*). Poskytuje tyto vlastní funkce:

Tabulka 4.2 Specifické operace třídy *Sequence*

<b>string getName ( )</b>	
<ul style="list-style-type: none"> <li>• získá název sekvence</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ string: název sekvence</li> </ul> </li> </ul>
<b>string getLocation ( )</b>	
<ul style="list-style-type: none"> <li>• získá umístění sekvence</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ string: umístění sekvence</li> </ul> </li> </ul>
<b>Interval* newInterval( const string&amp; name )</b>	
<ul style="list-style-type: none"> <li>• vytvoří objekt pro reprezentaci intervalů (obrázků, snímků videa) pro aktuální sekvenci</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ name: nepovinný, lze jím vybrat pouze jeden interval sekvence</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ Interval*: ukazatel na objekt reprezentující interval (intervaly)</li> </ul> </li> </ul>
<b>Image* newImage ( const string&amp; name )</b>	
<ul style="list-style-type: none"> <li>• vytvoří objekt pouze pro reprezentaci obrázku pro aktuální sekvenci</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ name: nepovinný, lze jím vybrat pouze jeden obrázek sekvence</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ Image*: ukazatel na objekt reprezentující obrázek (obrázky)</li> </ul> </li> </ul>



bool add ( string name, string location, string type )	
<ul style="list-style-type: none"> <li>• přidá do databázové tabulky <i>[jméno_datasetu].sequences</i> záznam o nové sekvenci</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ name: název sekvence</li> <li>○ location: umístění sekvence (název videa či složky obrázků)</li> <li>○ type: jeden z typů sekvence (<i>images</i> – obrázky, <i>video</i>)</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ bool: ukazatel úspěchu</li> </ul> </li> </ul>

Následující příklad ilustruje přidání sekvence (existující video) do datové sady *sunar*:

```
// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující všechny sekvence datasetu sunar
Sequence* all_sunar = ds_sunar->newSequence();
// Přidání nového videa do datasetu pod názvem video8
all_sunar->add("video8", "video8.mpg", "video");
// Provedení databázového dotazu
all_sunar->addExecute();
```

### 4.3.3 Interval

Podmnožina sekvence zvolená tak, aby jí bylo možné přiřadit jednotný příznak či metadata (např. výskyt nějakého objektu v obraze). Jedná se tedy o jeden či více obrázků (snímků videa), u kterých se předpokládá časová spojitost.

#### Zdrojová data

Na úrovni diskových souborů se jedná o jednotlivé obrázky ve složce sekvence. Časové intervaly v konkrétním videu jsou vyjádřeny pouze metadaty v databázi.

#### Databázový objekt

Seznam všech intervalů příslušné sekvence je uložen v tabulce **[jméno\_datasetu].intervals**. Záznam o intervalu se do ní uloží pomocí funkce *Interval->add()* (viz tabulka *Tabulka 4.3*). Tabulka může vypadat následovně:

	seqname [PK] character	t1 [PK] integer	t2 [PK] integer	imglocation character varying	tags integer[]	svm real[]	userid name	created timestamp	notes text
1	frames	1	1	frame1-5.png	{2,3,5,8}		vfroml	2011-10-	
2	frames	2	2	frame13.png	{0,2,3,9}		vfroml	2011-10-	
3	video1	1	3		{1,9,10}		vfroml	2011-12-	
4	video1	7	12		{3,12,8,9,6}		vfroml	2013-01-	
5	video3	5	69		{0,1,7,8}		vfroml	2013-01-	

**Obr. 3.7:** Ukázka seznamu intervalů v tabulce *sunar.intervals*

Sloupce *t1* a *t2* vymezují rozsah intervalu – snímků videa dle jejich pořadí, u obrázků vymezují pořadí ve složce (obě čísla jsou tedy identická).

Sloupce *tags* a *svm* slouží jako ukázka uživatelských vstupně/výstupních dat anotací.

Sloupce *userid* (uživatel) *created* (časové razítko) a *notes* (poznámky) mají pouze informační charakter, jsou volitelné.

Objekt reprezentující všechny intervaly dané sekvence nebo konkrétní interval lze vytvořit pomocí tovární metody třídy *Sequence* následovně:

```
// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující pouze sekvenci video2
Sequence* video3_sunar = ds_sunar->newSequence("video3");
// Objekt reprezentující všechny intervaly videa video3
Interval* intall_video3 = video3_sunar->newInterval();
// Objekt reprezentující pouze intervaly videa video3 <5,69>
Interval* intl_video3 = video3_sunar->newInterval(5, 69);

// .. práce

// Destrukce objektů
delete (intall_video3); delete (intl_video3);
delete (video3_sunar); delete (ds_sunar);
```

V některých situacích může být opět vhodné pracovat pouze s jedním vybraným intervalem, nikoli se všemi dostupnými. Podobně jako u třídy *Sequence* toho lze dosáhnout omezením rozsahu intervalu či názvu obrázku ve funkci *newInterval*.

Jako alternativu k třídě *Interval* pro manipulaci s obrázky lze použít třídu **Image**:

```
// Objekt reprezentující pouze složku obrázků frames
Sequence* frames_sunar = ds_sunar->newSequence("frames");
// Objekt reprezentující obrázek frame13.png
Image* frame13_sunar = frames_sunar->newImage("frame13.png");
```

Třída **Interval** (*Image*) opět dědí většinu svých metod a atributů ze třídy *KeyValues* (viz kapitola 4.4 *Správa a dotazování dat*). Poskytuje tyto vlastní funkce:

Tabulka 4.3 Specifické funkce třídy *Interval*

string getSequenceName ( )	
<ul style="list-style-type: none"> <li>získá název sekvence, do které tento interval přísluší</li> </ul>	<ul style="list-style-type: none"> <li><b>parametry</b> <ul style="list-style-type: none"> <li>žádné</li> </ul> </li> <li><b>návratová hodnota</b> <ul style="list-style-type: none"> <li>string: název sekvence</li> </ul> </li> </ul>
Sequence* getSequence ( )	
<ul style="list-style-type: none"> <li>získá objekt reprezentující sekvenci, do které tento interval přísluší</li> </ul>	<ul style="list-style-type: none"> <li><b>parametry</b> <ul style="list-style-type: none"> <li>žádné</li> </ul> </li> <li><b>návratová hodnota</b> <ul style="list-style-type: none"> <li>Sequence*: ukazatel na objekt sekvence</li> </ul> </li> </ul>

int getStartTime ( )	
<ul style="list-style-type: none"> <li>• získá počáteční čas intervalu</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ int: počáteční čas intervalu</li> </ul> </li> </ul>
int getEndTime ( )	
<ul style="list-style-type: none"> <li>• získá koncový čas intervalu</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ žádné</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ int: koncový čas intervalu</li> </ul> </li> </ul>
bool add (const string& sequence, const int t1, const int t2, const string& location)	
<ul style="list-style-type: none"> <li>• přidá do databázové tabulky <i>[jméno_datasetu].intervals</i> záznam o novém intervalu</li> </ul>	<ul style="list-style-type: none"> <li>• <b>parametry</b> <ul style="list-style-type: none"> <li>○ sequence: název sekvence, do které bude nový interval příslušet</li> <li>○ t1: počáteční čas intervalu</li> <li>○ t2: koncový čas intervalu</li> <li>○ location: umístění obrázku</li> </ul> </li> <li>• <b>návratová hodnota</b> <ul style="list-style-type: none"> <li>○ bool: ukazatel úspěchu</li> </ul> </li> </ul>

Následující příklad ilustruje přidání intervalu do videa *video3* v datové sadě *sunar*:

```
// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující pouze sekvenci video3
Sequence* video3_sunar = ds_sunar->newSequence("video3");
// Objekt reprezentující všechny intervaly sekvence video3
Interval* intall_video3 = video3_sunar->newInterval();
// Přidání nového intervalu do videa video3
intall_video3->add("video3", 5, 8);
// Provedení databázového dotazu
intall_video3->addExecute();
```

#### 4.4 Správa a dotazování dat (třída KeyValues)

Všechny výše popsané třídy reprezentující datové objekty dědí atributy a metody třídy **KeyValues**. Tato třída je pojmenována podle základního principu přístupu klíč-hodnota k uloženým popisným metadatům obrazových dat. Jedná se tedy o nejpodstatnější část celého konceptu VTapi.

Třída **KeyValues** zajišťuje následující funkčnost:

- **procházení obrazových dat:** jak bylo uvedeno v kapitole 4.3 *Přístup k obrazovým datům*, objekty třídy *Dataset*, *Sequence* a *Interval* mohou reprezentovat jeden objekt, ale i seznam objektů (např. konkrétní datový soubor nebo všechny datové soubory). Jejich procházení zajišťuje nejdůležitější a zřejmě nejpoužívanější operace **next()** (viz kapitola 4.4.1 *Operace next()*).

- **získávání uložených metadat**, pomocí sady tzv. **operací getX** (viz kapitola 4.4.2 *Sada operací getX*)
- **provádění databázových dotazů** a příkazů pomocí vnořených tříd **Select**, **Insert** a **Update** (viz kapitoly 4.4.3 až 4.4.5).
- **uživatelské nastavení**: připojení k databázi, logování a další parametry (viz kapitola 4.5 *Konfigurace*). Toto zajišťuje vnořená třída **Commons**, která je koncovému uživateli skryta.
- **tisk** výsledků dotazů (operace *print()* a *printAll()*)

#### 4.4.1 Operace next()

Všechny instance tříd reprezentující obrazová data v sobě obsahují seznam příslušných objektů, které mohou potenciálně reprezentovat. Tedy např. objekt třídy *Dataset* obsahuje seznam všech datových sad (pokud jsme jej neomezili pouze na jeden specifický dataset, jak bylo popsáno výše). Podobně objekty *Sequence*, *Interval* apod.

Operace *next()* třídy *KeyValues* umožňuje procházení tohoto seznamu posouváním vnitřního ukazatele na právě aktivní prvek. Každým jejím zavoláním nad objektem bude tedy tento objekt reprezentovat následující datovou sadu, sekvenci či interval. Toto umožňuje efektivní procházení velkého množství obrazových dat a vyhledávání v jejich příslušných metadatech.

Jednoduchý příklad užití ilustruje následující část kódu, ve kterém se vypíše všechny sekvence datové sady *sunar*.

```
// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující všechny sekvence datasetu sunar
Sequence* seq_sunar = ds_sunar->newSequence();
// Cyklus procházení sekvencí
while (seq_sunar->next()) {
    string name = seq_sunar->getName();
    cout << name;
}
```

Objekt *seq\_sunar* bude postupně reprezentovat všechny dostupné sekvence. Vrátili-li operace *next()* hodnotu NULL, objekt se nastaví opět do původního stavu. Analogicky lze procházet datové soubory, intervaly, videa, snímky videa, obrázky a procesy.

Další důležitou vlastností operace *next()* je zajištění konzistence reprezentace dat při jejich procházení. Toto je zajištěno **automatickým prováděním** nepotvrzených příkazů **add** a **update** před zavoláním operace.

#### 4.4.2 Sada operací getX

Operace *next()* umožňuje navigaci v databázi dat a popisných metadat. K získání požadovaných hodnot z databáze slouží sada operací **getX**, kde *X* je

příslušný **datový typ**. Jako parametr operace *getX* se zadává jeho název. VTapi implementuje následující operace třídy *KeyValues*:

Tabulka 4.4 Přehled podporovaných datových typů

<b>getValue</b>	Získání hodnoty <b>jakéhokoliv</b> typu převedeného na textovou reprezentaci		
<b>Číselné typy</b>			
<b>getInt</b>	32bit číselná hodnota <i>int</i>	<b>getFloat</b>	64bit reálná hodnota <i>float</i>
<b>getIntA</b>	pole hodnot <i>int</i>	<b>getFloatA</b>	pole hodnot <i>float</i>
<b>getIntV</b>	vektor hodnot <i>int</i>	<b>getFloatV</b>	vektor hodnot <i>float</i>
<b>getIntVV</b>	libovolně rozměrný vektor hodnot <i>int</i>	<b>getFloatVV</b>	libovolně rozměrný vektor hodnot <i>float</i>
<b>getInt8</b>	64bit číselná hodnota <i>long</i>	<b>getFloat8</b>	64bit reálná hodnota <i>double</i>
<b>Textové typy</b>			
<b>getChar</b>	znak	<b>getString</b>	textový řetězec
<b>getCharA</b>	pole znaků		
<b>Geometrické typy (PostGIS)</b>			
<b>getPoint</b>	2D bod	<b>getPolygon</b>	mnohoúhelník
<b>getPointV</b>	pole 2D bodů	<b>getPath</b>	cesta
<b>getLineSegment</b>	část lomené čáry	<b>getCube</b>	n-dimenzionální kostka
<b>getBox</b>	Ohraničující kvádr	<b>getGeometry</b>	univerzální geometrie (GEOS)
<b>getCircle</b>	kružnice	<b>getLineString</b>	lomená čára (GEOS)
<b>OpenCV typy</b>			
<b>getCvMat</b>	OpenCV matice <i>cvMat</i>	<b>getCvMatND</b>	OpenCV matice <i>cvMatND</i>
<b>Ostatní typy</b>			
<b>getTimestamp</b>	časové razítko	<b>getIntOid</b>	databázový identifikátor <i>OID</i>

Následující příklad ilustruje použití operace *getX* (konkrétně *getFloatV*).

```

// Objekt reprezentující dataset sunar
Dataset* ds_sunar = vtapi->new Dataset("sunar");
// Objekt reprezentující video3 z datasetu sunar
Sequence* video3_sunar = ds_sunar->newSequence("video3");
// Objekt reprezentující intervaly v sekvenci video3
Interval* int_video3 = video3_sunar->newInterval();
// Cyklus procházení intervalů
while (int_video3->next()) {
    // Získání metadat ze sloupce tags
    vector<float> tags = int_video3->getFloatV("tags");
    if (tags.empty()) {
        // interval není otagován
        // .. práce
    }
    tags.clear();
}

```

### 4.4.3 Třída Select

Každý objekt třídy *KeyValues* obsahuje jako atribut instanci třídy **Select**. Slouží ke konstrukci databázových dotazů, které se primárně volají po zavolání funkce *next()*. Lze je ale také konstruovat a provádět pomocí následujících operací třídy *Select*:

Tabulka 4.5 Specifické operace třídy *Select*

<b>bool from ( const string&amp; table, const string&amp; column )</b>	
<ul style="list-style-type: none"> <li>určuje tabulku, nad kterou se dotaz bude provádět (FROM) a příslušný sloupec</li> </ul>	<ul style="list-style-type: none"> <li><b>parametry</b> <ul style="list-style-type: none"> <li><i>table</i>: tabulka</li> <li><i>column</i>: sloupec tabulky</li> </ul> </li> <li><b>návratová hodnota</b> <ul style="list-style-type: none"> <li>bool: vždy true</li> </ul> </li> </ul>
<b>bool whereString ( const string&amp; key, const string&amp; value, const string&amp; oper, const string&amp; table )</b>	
<ul style="list-style-type: none"> <li>konstruuje WHERE sekci dotazu</li> </ul>	<ul style="list-style-type: none"> <li><b>parametry</b> <ul style="list-style-type: none"> <li><i>key</i>: sloupec tabulky</li> <li><i>value</i>: hodnota pole</li> <li><i>oper</i>: operátor (implicitně =)</li> <li><i>table</i>: tabulka obsahující sloupec <i>key</i> (implicitně '')</li> </ul> </li> <li><b>návratová hodnota</b> <ul style="list-style-type: none"> <li>bool: vždy true</li> </ul> </li> </ul>
<b>bool execute ( )</b>	
<ul style="list-style-type: none"> <li>provede dotaz (commit)</li> </ul>	<ul style="list-style-type: none"> <li><b>parametry</b> <ul style="list-style-type: none"> <li>žádné</li> </ul> </li> <li><b>návratová hodnota</b> <ul style="list-style-type: none"> <li>bool: ukazatel úspěchu</li> </ul> </li> </ul>

Třídu **KeyValues** lze instanciovat i samostatně k obecné manipulaci s databázovou tabulkou. Následující příklad ukazuje konstrukci vnořené třídy *Select*. Dotaz vrací názvy sekvencí (*seqname*) z tabulky *tags*, které obsahují pole anotací [1,3].

```
// Objekt datasetu
Dataset* dataset = vtapi->newDataset();
// Obecný objekt KeyValues
KeyValues* tags = new KeyValues (*dataset);
// Vytvoření instance třídy Select
tags->select = new Select(*tags);
// SELECT  seqname
// FROM    tags
// WHERE   annotations @> ARRAY[1,3]
tags->select->from("tags", "seqname");
tags->select->whereString("annotations", "ARRAY[1,3]", "@>");
// Procházení výsledků
while (tags->next()) {
    // .. prace
}
```

#### 4.4.4 Třída Insert

Třída **Insert** slouží ke konstrukci databázových příkazů vkládání do tabulky. Typicky není nutné ji používat, tuto funkčnost plně zaobalují operace *add()* tříd *Sequence* a *Interval* (viz. kapitola 4.3 *Přístup k obrazovým datům*). Nicméně lze ji (s určitou dávkou opatrnosti – kontrola překlepů, vkládání do správné tabulky apod.) využít i samostatně pro konstrukci vlastních příkazů za pomoci sady operací **keyX**, kde **X** je datový typ vkládané hodnoty.

Tabulka 4.6 Přehled sady operací *keyX* třídy *Insert*

<b>keyInt</b>	32bit číselná hodnota <i>int</i>	<b>keyFloat</b>	64bit reálná hodnota <i>float</i>
<b>keyIntA</b>	pole hodnot <i>int</i>	<b>keyFloatA</b>	pole hodnot <i>float</i>
<b>keyString</b>	textový řetězec	<b>keySeqtype</b>	výčtový typ typů sekvencí
<b>keyStringA</b>	pole textových řetězců	<b>keyInouttype</b>	výčtový typ typů vstup/výstup
		<b>keyPermissions</b>	výčtový typ přístupových práv

Všechny funkce *keyX* požadují jako první parametr název sloupce (klíč), do kterého hodnotu vkládat. Druhým parametrem je samotná hodnota. K potvrzení příkazu slouží funkce *execute()*.

Záznam se vkládá do tabulky určené objektem, ze které se třída *Insert* instanciuje (v následujícím příkladě je tímto objektem třída sekvencí *seq\_sunar*).

Následující příklad ukazuje vložení informace o sekvenci *video5* (videoubor *video5.mpg*) do datasetu *sunar*. Není specifikována vlastní tabulka, implicitně se tedy použije základní tabulka *sequences*.

```
// Objekt datasetu
Dataset* dataset = vtapi->newDataset();
// Objekt reprezentující všechny sekvence datasetu sunar
Sequence* seq_sunar = ds_sunar->newSequence();
// Vytvoření instance třídy Insert
seq_sunar->insert = new Insert(*seq_sunar);
// Specifikace příkazu Insert
seq_sunar->insert->keyString("seqname", "video5");
seq_sunar->insert->keyString("seqlocation", "video5.mpg");
seq_sunar->insert->keySeqtype("seqtyp", "video");
// Potvrzení dotazu
seq_sunar->insert->execute();
```

#### 4.4.5 Třída Update

Třída **Update** slouží ke konstrukci databázových příkazů aktualizace záznamů tabulky. Pro její použití zaobaluje sadu operací **setX**, kde **X** je datový typ aktualizované hodnoty.

Tabulka 4.7 Přehled sady operací *setX*

<b>setInt</b>	32bit číselná hodnota <i>int</i>	<b>setFloat</b>	64bit reálná hodnota <i>float</i>
<b>setIntA</b>	pole hodnot <i>int</i>	<b>setFloatA</b>	pole hodnot <i>float</i>

<b>keyString</b>	textový řetězec		
------------------	-----------------	--	--

Všechny operace *setX* požadují jako první parametr název sloupce (klíč), do kterého hodnotu vkládat. Druhým parametrem je samotná hodnota. K potvrzení příkazu slouží funkce *execute()*.

Upravovaný záznam (tabulka, řádek) je určen objektem, ze kterého se třída Update instanciuje (v následujícím příkladě se jedná o řádek se záznamem o sekvenci *video5*).

Předpokládejme, že máme naplněné pole tagů (*float\* tags*) pro *video5* v datasetu *sunar*. Následující příkaz ukazuje jejich aktualizaci.

```
// Objekt datasetu
Dataset* dataset = vtapi->newDataset();
// Objekt reprezentující sekvenci video5
Sequence* video5_sunar = ds_sunar->newSequence("video5");
//Přesuneme se na konkrétní záznam
video5_sunar->next();
// Vytvoření instance třídy Update
video5_sunar->update = new Update(*video5_sunar);
// Aktualizace pole tagů sekvence video5
video5_sunar->setFloatA("tags", my_tags);
// Potvrzení dotazu
video5_sunar->update->execute();
```

## 4.5 Konfigurace

VTApi podporuje nastavení připojení k datovému úložišti a dalších možností, a to prostřednictvím **konfiguračního souboru** a možných **parametrů příkazové řádky** (lze kombinovat, příkazová řádka má v případě konfliktu přednost). Tato konfigurace se předává jako parametr konstruktoru třídy *VTApi* a následně se kopíruje do všech objektů reprezentujících data prostřednictvím skryté vnořené třídy **Commons**.

Spuštění VTApi pouze s konfiguračním souborem:

```
VTApi* vtapi = new VTApi("/path/to/vtapi.conf");
// .. práce
delete(vtapi);
```

Spuštění VTApi s parametry a konfiguračním souborem:

```
int main(int argc, char** argv) {
    VTApi* vtapi = new VTApi(argc, argv);
    // .. práce
    delete(vtapi);
}
```

V tabulce *Tabulka 4.8* je uveden seznam konfigurovatelných parametrů, včetně zkratk pro použití v příkazové řádce.



Tabulka 4.8 Přehled konfigurovatelných parametrů

Parametr	Zkratka	Implicitní hodnota	Popis
config	-	./vtapi.conf	Umístění konfiguračního souboru
log	-	./vtapi.log	Umístění souboru logu
user	u	-	Jméno uživatele
password	p	-	Heslo uživatele
location	l	-	Základní umístění souborů obrazových dat
connection	c	-	Specifikace připojení ve formátu: <i>host=... port=... dbname=... user=... password=....</i>
format	f	standard	Formát výstupu tisku ( <i>standard / csv / html / binary / sparse</i> )
input	i	-	Specifikace vstupního streamu
output	o	-	Specifikace výstupního streamu
queryLimit	-	-	Omezení maximálního počtu řádků ve výsledku jednoho dotazu
arrayLimit	-	-	Omezení maximálního počtu zobrazených prvků tisknutého pole
<b>Specifikace kontextu dat</b>			
where	W	-	SQL řetězec definující klauzuli WHERE všech dotazů SELECT
dataset	D	-	Užití pouze určené datové sady
sequence	S	-	Užití pouze určené sekvence
interval	I	-	Užití pouze určeného intervalu
method	M	-	Užití pouze určené metody
process	P	-	Užití pouze určeného procesu
selection	E	-	Užití pouze určené selekce

Syntaxe konfiguračního souboru je jednoduchá – *parametr=hodnota*, záznamy odděleny řádkováním. Komentáře lze uvodit znakem #. Ukázka jednoduchého konfiguračního souboru:

```
# VTApi configuration file

user="vfroml"
password="*****"

location="/mnt/vidte/"
connection="host=vidte.cz port=1111 dbname=dbvidte user=vfroml password=*****"

verbose

log="./logs/vtapi.log"
format="csv"

querylimit=10000
arraylimit=10

dataset="sunar"
```

## 5 Implementace případových situací

Typické použití knihovny VTApi pro účely zpracování obrazových dat zahrnuje následující kroky:

- Nahrání informací o obrazových datech do úložiště (databáze). K tomuto účelu lze použít automatizovaný skript, který uloží např. celý adresář s videi.
- Alokace místa pro výstupy algoritmů (tabulky, sloupce v databázi).
- Registrace analytických algoritmů, tedy především formát jejich vstupů a výstupů. V současné verzi se provádí manuálně.
- Implementace samotného analytického programu – procházení obrázků/videí dle uložených metadat a dosavadních výsledků analýzy; spuštění registrovaných algoritmů, ukládajících extrahovaná data.
- Dotazování na výsledky procesů.

V této kapitole je popsáno použití VTApi pro dílčí úlohy dvou případových studií z kapitoly 3 *Příklady použití*. Jde o úlohy z evaluací TRECVID (situace 2) – kapitola 5.1 *Vyhledávání a klasifikace statických obrázků* a shlukování trajektorií (situace 4) – kapitola 5.2 *Shlukování trajektorií a detekce odlehlých trajektorií*. V kapitole 5.3 *Sledování objektů v reálném čase* je popsán experiment zaměřený na časovou náročnost některých operací s trajektoriemi.

### 5.1 Vyhledávání a klasifikace statických obrázků

OpenCV nabízí široké možnosti extrakce rysů a klasifikačních technik, ale neobsahuje prostředky pro efektivní ukládání dat pro jejich vyhledávání a další zpracování. Příkladem je vyhledávání obrázků Google nebo TRECVID, kde je vyhledávání založeno na různých typech (nízko až vysoko-úrovňových, lokálních i globálních) rysů vztažených k nějakému objektu společně s jejich anotacemi.

Proto byly v rámci VTApi implementovány různé podobnostní metriky. Rozšíření pgDistance dostupné ve VTApi provádí podobnostní dotazy (CBIR) pomocí vzdálenosti jednotlivých vektorů rysů v databázi PostgreSQL, například cosinovou nebo Eukleidovskou vzdáleností. Toto je možné podpořit efektivními indexačními technikami, jako jsou GIN (invertovaný index) GiST spolu s filtračními metodami jako jsou bitmapové nebo haldové indexy, R-strom, KD-strom nebo Quad-strom pro obsažnost (operátor @>) nebo vyhledání prvků v nejbližší okolí (operátor <->).

Následující příklad předpokládá dataset obrázků nazvaný „search“ v databázi. Pokud chceme vytvořit dotaz založený na podobnosti (distance\_square\_int4()) deskriptorů rozložení barvy (colorLayout) k obrázku "Q.jpg", pak může vypadat jako v následujícím příkladu.

```

// VTApi entry point, using Dataset "search"
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("search");
// code of the ColorLayout Method, using Selection "image"
Image* image = new Image(&dataset, "image");
while (image->next()) {
    vector color = colorLayout(image->getDataLocation());
    image->setIntA("color", color);
}
// 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

```

## 5.2 Shlukování trajektorií a detekce odlehlých trajektorií

Pro nalezení shluků trajektorií pohybujících se objektů jsme využili OpenCV implementaci algoritmu Expectation-maximization (EM), jehož cílem je odhadnout parametry modelu GMM (Gaussian Mixture Model). Trajektorie byly z dohledového videa extrahovány pomocí OpenCV *blobtrack* dema rozšířeného o extrakci dalších rysů. Extrahované trajektorie byly následně agregovány do podoby vektoru rysů.

Před vlastním shlukováním je nutné načíst z databáze vektory rysů, které reprezentují jednotlivé trajektorie. Z načtených vektorů rysů jsou pak připraveny trénovací vzorky pro EM algoritmus. S využitím VTApi je tento proces možné provést například následujícím způsobem (uvažujeme, že trajektorie jsou uloženy v selekci „tracks“):

```

Mat samples; // cv::Mat pro trénovací vzorky
VTApi* vtapi = new VTApi(argc, argv);
Dataset* dataset = vtapi->newDataset("train"); //
trénovací sada
dataset->next();
Sequence* sequence = dataset->newSequence();
while (sequence->next()) { // pro každé video
    Interval* track = new Interval(*sequence, "tracks");
    while (track->next()) { // pro každou trajektorii
        Mat sample; // cv::Mat pro vektor rysů trajektorie
        float feature = track->getFloat("feature");
        // ... načtení rysů a naplnění vektoru rysů
        samples.push_back(sample);
    }
}

```

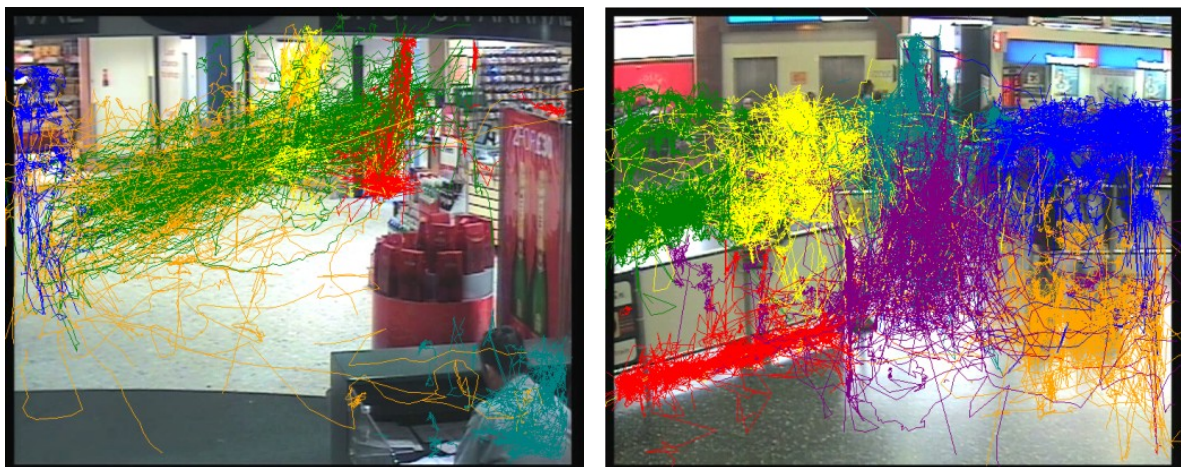
Připravené trénovací vzorky jsou následně předloženy EM algoritmu, který odhadne parametry GMM modelu. Naučený model je následně využit pro

přiřazení jednotlivých trajektorií k příslušným shlukům. Vlastní shlukování a uložení výsledků do databáze s využitím VTApi ukazuje následující příklad:

```
CvEM model;           // GMM-EM model
CvEMParams params;   // parametry pro EM
// ... nastavení EM parametrů včetně počtu shluků
model.train(samples, Mat(), params, NULL);
// ... zvolení datové sady a sekvencí (viz předchozí
příklad)
while (track->next()) {
    Mat sample;
    // ... načtení rysů a naplnění vektoru rysů
    int cluster = (int) model.predict(sample); // zjištění
čísla shluku
    track->setInt("cluster", cluster);         // uložení
čísla shluku
}
```

Pro ilustraci bylo provedeno shlukování trajektorií extrahovaných z druhého datasetu videí z datové sady i-LIDS, která pochází z pěti kamer v prostředí LGW letiště. Na obrázku *Obrázek 5.1* je příklad vizualizace některých získaných výsledků. Různé barvy trajektorií zde odpovídají různým shlukům. Vykreslení trajektorie do obrázku může být realizováno například následujícím způsobem:

```
// ... zvolení datové sady a sekvencí (viz předchozí
příklad)
while (track->next()) {
    int cluster = track->getInt("cluster"); // zjištění
čísla shluku
    // načtení GEOS geometrie reprezentující trajektorii:
    GEOSGeometry* ls = track->getLineString("locations");
    // vyjádření geometrie jako sekvence prostorových
souřadnic:
    const GEOSCoordSequence* cs = GEOSGeom_getCoordSeq(ls);
    // ... vykreslení trajektorie do obrázku (barva podle
čísla shluku)
}
```



*Obrázek 5.1* Příklad výsledků shlukování trajektorií z první kamery pomocí algoritmu EM (vlevo) a příklad shlukování trajektorií ze třetí kamery pomocí algoritmu k-means (vpravo).

GMM model naučený během shlukování lze využít také pro detekci odlehlých trajektorií, tedy takových, které mohou odpovídat neobvyklému nebo podezřelému chování pohybujících se objektů. Jako odlehlá trajektorie je pak označena trajektorie, která nevyhovuje naučenému modelu. Odlehlost trajektorie se může projevit buď tím, že se trajektorie nachází v oblasti s nízkou pravděpodobností vzhledem k modelu, nebo lze trajektorii přiřadit do více shluků zároveň, ale do žádného s dostatečnou pravděpodobností.

### 5.3 Sledování objektů v reálném čase

Zpracování trajektorií obvykle probíhá v reálném čase, proto byly provedeny experimenty zaměřené na zjištění časové náročnosti a dalších zdrojů při správě trajektorií z OpenCV blobtracku.

Na úrovni databáze vznikly funkce *box3d2cube* a *cube2box3d*, které slouží pro vzájemnou konverzi datových typů pro časoprostorová data, a to konkrétně funkce pro vzájemnou konverzi mezi datovým typem *box3d* (z databázového rozšíření PostGIS) a typem *cube* (z databáze PostgreSQL). Současně jsou k funkci *box3d2cube* napsány pomocné databázové trigger, které v případě vložení nebo změny geometrie uloží minimální ohraničující kvádr (Minimum Bounding Box, MBB) reprezentovaný ve formě kostky (*cube*).

Příklad využití: Pro každý datový typ pro časoprostorová data (tedy datový typ pro trojrozměrná data) z rozšíření PostGIS, viz datové typy PostGIS podporované ve VTAPI (kapitola 4.4.2 *Sada operací getX*), například *LineString* pro trajektorie, je možné získat pomocí funkce *box3d* jeho MBB a ten se pak pomocí funkce *box3d2cube* převede na kostku. Samotná kostka je pak indexována pomocí indexu GIST. Index nad těmito daty se vytváří manuálně.

Podobnostní dotazy byly provedeny pomocí operátoru '@>' (containment), který vrátil 4 trajektorie obsažené v dotazovaném ohraničujícím kvádru (vybrán náhodně).

*Tabulka 5.1* Výsledky výkonostního testu vkládání a dotazování trajektorií.

	insert/update	vyber vše	obsažené (4)	síťová data
lokální DB	220 ± 1 s	43 ± 1 s	0.1 s	0B
vzdálená	220 ± 5 s	74 ± 1 s	0.2 s	334 MB

V tomto případě bylo použito 7269 trajektorií, sledovaných během 4 hodin ve videu pomocí 5 kamer (paralelně) v prostoru plném lidí na letišti (z datasetu AVSS / i-LIDS). Z tabulky 1 plyne, že v obou sestaveních na 13" notebooku (1.4 GHz ULV CPU, 2 GB RAM, 128 GB SSD) s databází lokálně nebo připojenou přes síť je systém schopný běžet v reálném čase.

## 6 Technická dokumentace

Technická dokumentace je přiložena v příloze společně s diagramem tříd v podobě generované ze zdrojových kódů (C++), které je společně s dokumentací možné získat na stránkách <https://gitorious.org/vtapi>. Zde se dále nachází Wiki, která popisuje základní předpoklady pro funkci (použité knihovny), dále návod na kompilaci ze zdrojových kódů (pro operační systémy POSIX/Linux a Windows) různých částí API:

- ⤴ Vlastní projekt VTApi (C++, adredáře include, src).
- ⤴ Rozhraní zpřístupňující API z příkazové řádky (VTCLI).
- ⤴ Programová dokumentace včetně plnohodnotného diagramu tříd a databáze dle případu použití „Situace 2“ (TRECVID SIN, ITS).
- ⤴ Rozhraní jazyka Python za podpory knihovny Boost (VTAPyI, adresář pyi).
- ⤴ VTApi může být závislé na několika dalších knihovnách – OpenCV (doporučeno), PostGIS, jak je popsáno v sekci instalace (kapitola 6.2).

Technická dokumentace je dále dostupná z <http://vidte.fit.vutbr.cz/vtapi.html>.

### 6.1 VTCLI

Nástroj **VTCLI** je konzolová aplikace implementující jednoduché rozhraní pro spouštění nejpoužívanějších funkcí VTApi. Byla vyvinuta specificky pro účely automatizace některých objemově náročných úkonů (např. nahrání dat do databáze) a testování funkčnosti VTApi.

Konfigurace aplikace se provádí pomocí parametrů příkazové řádky (viz předchozí sekce) a příslušného konfiguračního souboru.

Nástroj může pracovat ve dvou různých módech:

- **Interaktivní**
  - Aplikace bude čekat na vstupy uživatele a vypisovat na standardní výstup
- **Jednorázový**
  - Aplikace se spustí jednorázově, zadá-li uživatel jako parametry příkazové řádky za konfigurační parametry také **samotný příkaz** (jejich přehled viz níže). Tento příkaz se pak jednorázově provede a aplikace se ukončí.

Tabulka 6.1 Přehled příkazů aplikace VTCLI

Příkaz	Následuje	Popis
query	SQL řetězec dotazu	Provedení vlastního databázového dotazu
select	dataset   sequence   interval   process [ARGS]	Provedení dotazu
insert	sequence   interval   process [ARGS]	Provedení příkazu vložení
test	-	Provedení testovací funkce
help	-	Zobrazení nápovědy
exit	-	Ukončení nástroje

Tab. 3.9: Přehled příkazů nástroje VTCLI

Formát parametrů [ARGS] je *vlastnost=hodnota*, příp. *vlastnost=hodnota1,hodnota2,...*

Tabulka 6.2 Parametry příkazů *select* a *insert* aplikace VTCLI

Parametry select			
<b>dataset</b>			
<b>name</b>	název datasetu	<b>location</b>	umístění datových souborů datasetu
<b>sequence</b>			
<b>name</b>	název sekvence	<b>location</b>	umístění datových souborů sekvence
<b>num</b>	unikátní číslo sekvence	<b>type</b>	typ sekvence ( <i>images / video</i> )
<b>interval</b>			
<b>seqname</b>	název sekvence, do které tento interval patří	<b>t1</b>	počáteční čas intervalu
<b>location</b>	umístění obrázku	<b>t2</b>	koncový čas intervalu
<b>method</b>			
<b>name</b>	název metody		
<b>process</b>			
<b>name</b>	název procesu	<b>method</b>	název metody, jejíž instancí je tento proces
<b>inputs</b>	datový typ vstupů (tabulka)	<b>outputs</b>	datový typ výstupů (tabulka)
Parametry insert			
<b>sequence</b>			
<b>name</b>	název sekvence (povinné)	<b>location</b>	umístění datových souborů sekvence
<b>num</b>	unikátní číslo sekvence	<b>type</b>	typ sekvence ( <i>images / video</i> )
<b>interval</b>			
<b>seqname</b>	název sekvence, do které tento interval patří	<b>t1</b>	počáteční čas intervalu (povinné)
<b>location</b>	umístění obrázku	<b>t2</b>	koncový čas intervalu (povinné)
<b>process</b>			
<b>name</b>	název procesu (povinné)	<b>method</b>	název metody, jejíž instancí je tento proces
<b>inputs</b>	datový typ vstupů (tabulka)	<b>outputs</b>	datový typ výstupů (tabulka)

## 6.1.1 Ukázky použití aplikace VTcli

### Interaktivní mód

Spuštění s parametrem umístění konfiguračního souboru:

```
./vtcli --config="./vtapi.conf"
```

Dotaz na dostupné datasety:

```
> select dataset
dsname|dslocation      |userid|groupid|created          |changed  |notes
name  |varchar              |name  |name   |timestamp        |timestamp|text
-----+-----+-----+-----+-----+-----+-----
sin12 |sin12/               |vfroml|vidte  |2012-08-23 04:41:22|          |
sunar  |i-LIDS/mct_tr/      |vfroml|vidte  |2012-01-03 20:41:41|          |TrecVID
public|public/              |vfroml|vidte  |2012-01-03 20:41:41|          |test dataset
(3 rows)
```

Dotaz na sekvence datasetu:

```
> select sequence
seqname|seqnum|seqlocation|seqtyp |userid|groupid|created          |changed  |notes
name   |int4  |varchar    |seqtype|name  |name   |timestamp        |timestamp|text
-----+-----+-----+-----+-----+-----+-----+-----
video1 |1002  |video1.mpg|video  |vfroml|vidte  |2012-01-03 20:41:41|          |
video2 |1003  |video2.mpg|video  |vfroml|vidte  |2012-01-03 20:41:41|          |
video3 |1004  |video3.mpg|video  |vfroml|vidte  |2012-01-03 20:41:41|          |
frames |1001  |frames/    |images|vfroml|vidte  |2012-01-03 20:41:41|          |
(4 rows)
```

Dotaz na intervaly sekvence *video1*:

```
> select interval seqname=video1
seqname|t1  |t2  |imglocation|tags      |svm  |userid|created          |notes
varchar|int4|int4|varchar     |_int4    |_float4|name  |timestamp        |text
-----+-----+-----+-----+-----+-----+-----+-----
video1 |1   |3   |            |1,9,10   |      |vfroml|2011-11-08 20:01:55|
video1 |7   |12  |            |3,12,8,9,6|    |vfroml|2013-00-09 21:51:25|
video1 |2   |6   |            |2,5      |      |vfroml|2013-00-10 18:04:06|
(3 rows)
```

Vložení nového intervalu do sekvence *video1*:

```
> insert interval seqname=video1 t1=10 t2=16 tags=2,6,3
seqname|t1  |t2  |imglocation|tags      |svm  |userid|created          |notes
varchar|int4|int4|varchar     |_int4    |_float4|name  |timestamp        |text
-----+-----+-----+-----+-----+-----+-----+-----
video1 |1   |3   |            |1,9,10   |      |vfroml|2011-11-08 20:01:55|
video1 |7   |12  |            |3,12,8,9,6|    |vfroml|2013-00-09 21:51:25|
video1 |2   |6   |            |2,5      |      |vfroml|2013-00-10 18:04:06|
video1 |10  |16  |            |2,6,3    |      |vfroml|2013-00-10 18:30:23|
(4 rows)
```

### Jednorázový mód

Spuštění sady příkazů vložení intervalů do datasetu *sunar*:

```
$ ./vtcli --dataset="sunar" insert interval seqname=video2 t1=1 t2=6 tags=1,5,6,9
$ ./vtcli --dataset="sunar" insert interval seqname=video2 t1=3 t2=9 tags=2,6,9,11,12
$ ./vtcli --dataset="sunar" insert interval seqname=video3 t1=2 t2=3 tags=2,6,9,11,12
```



## 6.2 Poznámky k instalaci

Instalace vyžaduje následující předpoklady:

- ♣ Windows, Linux, 32 nebo 64bit
- ♣ PostgreSQL 9.1 – libpq
- ♣ [libpqtypes](#) 1.5
- ♣ OpenCV 2.4 (doporučeno)
- ♣ GCC 4.5.2+ (4.6 and 4.7) pro kompilaci
- ♣ Postgis 2.0 (nově doporučeno), PROJ4 4.8 (reprojection), GEOS 3.3.3 (geometry), GDAL 1.9 (raster).

Získání aktuálních zdrojových kódů VTapi (v. 1.5)

```
> git clone git://gitorious.org/vtapi/vtapi.git
```

Pro verzi v1 (2011)

```
> git clone -b v1 git://gitorious.org/vtapi/vtapi.git
> git pull origin v1
```

Pro uživatele systému Windows (x86) je k dispozici balíček na adrese [http://vidte.fit.vutbr.cz/dist/VTapi\\_Windows.zip](http://vidte.fit.vutbr.cz/dist/VTapi_Windows.zip), který slouží pro podporu instalace VTapi. Čtěte README.txt, prosíme.

Sestavení (doporučeno NetBeans nebo použijte make.sh)

```
> cd vtapi/src && make
```

Sestavení CLI

```
> cd vtapi & make
```

Sestavení programové dokumentace

```
> cd doc && doxygen
```

Online k nalezení na <http://vidte.fit.vutbr.cz/doc/>.

V závislosti na konfiguraci daného počítače může být třeba změnit nebo zrušit závislosti v souboru `include/vtapi_config.h`. Pro inicializaci databáze použijte skripty v adresáři `postgres` `createdb.sql` a `createschema.sql`. Pro spuštění bude nutné řádně nastavit hodnoty v souboru **vtapi.conf**.

## 7 Závěr

V rámci testování proběhly 2 typy testů – funkční a výkonnostní. Prvním typem funkčního testu (verifikačním) je test vestavěný a spustitelný z VTCLI (\$ vtapi test). Druhým typem funkčního testu (validační) je demonstrativní použití VTapi pro klasifikaci a vyhledávání obrazových dat, jak je uvedeno v případě použití „Situace 2“. V rámci tohoto použití byl proveden výkonnostní test na serveru pořízeném v rámci projektu VideoTeror (s testem na síti FIT) s následujícími výsledky:

- ⤴ Množství dotazů je c.a. 1000 za sekundu (pomocí operace newInterva()).
- ⤴ Množství vkládaných/upravovaných dat je c.a. 1000 za sekundu (při celkové velikosti 4MB).

VTapi je využito v systému SUNAR-ED. Ten sestává ze tří částí – modulu počítačového vidění (SUNAR-SED), serverová část aplikace (SUNAR-VRM) a uživatelského rozhraní (SED12-Annotator). Z hlediska správy anotovaných dat VTapi poskytuje podporu v podobě ukládání dvojic klíč-hodnota a používáním selekcí, které umožňují efektivní přístup k prvkům kolekce takových dvojic.

Ve verzi 2.0, která bude vyvíjena v roce 2013, plánujeme doplnění a zjednodušení funkčnosti metod a procesů (nová metodologie), implementaci databázové nezávislosti, která vytvoří podmínky pro použití i jiná datových úložišť, než stávající PostgreSQL (např. lokální databázi SQLite), a realizaci případných dalších požadavků a námětů, které vzejdou z používání VRapi v. 1.5.

Výchozí bod pro získání dalších informací a programové dokumentace je na adrese <http://vidte.fit.vutbr.cz/vtapi.html>. Tato zpráva včetně jejích příloh zde bude pravidelně aktualizována.

## 8 Příloha Sady dat

V rámci projektu VideoTeror testujeme VTapi na následujících datových sadách (Dataset):

- ♣ TRECvid SIN (ITS) na základě dat „Internet Archive videos with Creative Commons licenses“ (50GB, 200 hodin) v MPEG-4/H.264 s trváním mezi 10s s 3,5min (IACC.1, <http://www.archive.org/details/movies>) spolu s anotovanými 346 koncepty (herec, letadlo, pláž, ...).

Více viz <http://www-nlpir.nist.gov/projects/tv2011/tv2011.html#sin>.

- ♣ TRECvid SED / AVSS Challenge na základě dat „HOSDB’s i-LIDS Multiple Camera Tracking Training Corpus“ (128GB, 44 hodin, 5 kamer) z letiště London Gatwick v MPEG-2 nebo AVI (i-LIDS MCTTR, <http://scienceandresearch.homeoffice.gov.uk/hosdb/cctv-imaging-technology/video-based-detection-systems/i-lids>) spolu s anotovanými trajektoriemi některých osob (v XML) a popisem některých událostí (sjednoceno v 2012).

Více viz <http://www-nlpir.nist.gov/projects/tv2011/tv2011.html#sed> a <http://www.itl.nist.gov/iad/mig/tests/avss/2010/>.

- ♣ Volně přístupný dataset University of Washington Image Database (Washington), (300MB, 1200 obrázků) zařazených do 21 kategorií (domorodci, fotbal, yellowstone, ...).

Více viz <http://www.cs.washington.edu/research/imagedatabase/>.

- ♣ ImageNet je obrázková databáze organizovaná podle hierarchie [WordNet](#) (podstatná jména), kde v každém uzlu hierarchie jsou stovky až tisíce obrázků.

Více viz <http://www.image-net.org/explore>.

## 9 Literatura

- [1] Chmelař, P., Fröml, V., Volf, T., Zendulka, J.: VTApi v. 1.0 Technická zpráva FIT - VG20102015006 - 2011 – 01. FIT VUT v Brně. 2011, 16 s.