FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Aplikace pro zpracování videa za účelem měření rychlosti objektů
Student:	Bc. Daniel Šup
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Oblastí práce je aplikace zpracovávající video (např. záznam z kamery) za účelem detekce objektů (např. vozidel) a měření jejich rychlosti.
Výstupem práce je nástroj v podobě knihovny a služby pro zpracování videa, jeho anotaci a prezentaci výstupů.
Seznamte se s problematikou detekce objektů ve videu zahrnující zejména měření jejich rychlosti, proveďte průzkum existujících algoritmů a jejich dostupných implementací.
Proveďte rešerši formátů pro anotace multimediálního obsahu za účelem přidání metadat o objektech jako je rychlost.
Navrhněte, implementujte a otestujte v jazyce Python knihovnu obsahující minimálně:
- vybrané přístupy pro detekci objektů a měření rychlosti,
- generování anotace videa s výstupy algoritmů.

Navrhněte, implementujte a otestujte službu (REST) poskytující funkcionality knihovny.
Navrhněte, implementujte a otestujte webovou aplikaci využívající službu a prezentující výstupy.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Aplikace pro zpracování videa za účelem měření rychlosti objektů

Bc. Daniel Šup

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jaroslav Kuchař Ph.D.

26. května 2020

Poděkování

Děkuji svému vedoucímu práce Ing. Jaroslavu Kuchařovi Ph. D. za jeho vedení práce a poskytování cenných rád. Dále bych chtěl poděkovat své rodině za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 26. května 2020 . . . . . .

Odkaz na tuto práci

Šup, Daniel. *Aplikace pro zpracování videa za účelem měření rychlosti objektů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Práce se zaměřuje na návrh knihovny, která detekuje objekty ve videozáznamu, změří jejich rychlost a vygeneruje anotace, REST API a webové aplikace pro zpracovávání videí a měření rychlosti objektů ve videozáznamu. Při řešení se nejdříve analyzují existující metody detekce objektů, algoritmy pro měření rychlosti detekovaných objektů a způsoby anotace videa. Následně se vyberou vhodné způsoby detekce objektů a vhodné algoritmy pro měření rychlosti a způsoby anotace videa. Poté se knihovna navrhne a implementuje v programovacím jazyce Python. Dále se navrhne a implementuje REST API, které vhodným způsobem využívá funkce knihovny a zajišťuje detekci pohybujících se objektů ve videozáznamu, měření jejich rychlosti a generování anotací. Dále se navrhne a implementuje webová aplikace, která využívá REST API. V práci je implementována knihovna, která obsahuje funkce pro detekci pohybujících se objektů, měření jejich rychlosti a generování anotací. Dále je implementováno REST API, které využívá funkce knihovny a webová aplikace, která využívá implementované REST API. Webová aplikace umožňuje nahrát video, spustit pro dané video proces, který video zpracuje, a zobrazit video s vypočtenými rychlostmi objektů zpracované daným procesem. Knihovnu i REST API je možné využít jako základ pro aplikace, které analyzují objekty na základě jejich pohybu.

Klíčová slova měření rychlosti objektů, objekty ve videozáznamu, detekce objektů, anotace, knihovna, REST API, webová aplikace, Python

Abstract

The work focuses on the design of the library that detects objects in a video record, measures their speed and generates annotations, REST API and web applications for video processing and measuring the speed of objects in a video record. First, existing object detection methods, algorithms for measuring the speed of detected objects and video annotation methods are analyzed during the solution. Subsequently, suitable methods of object detection and suitable algorithms for measuring speed and methods of video annotation are selected. Then the library is designed and implemented in Python programming language. Furthermore, the REST API, which appropriately uses the functions of the library and ensures the detection of moving objects in the video, measuring their speed and generating annotations, is designed and implemented. Then the web application, which uses the REST API, is designed and implemented. In the work, the library that contains functions for detecting moving objects, measuring their speed and generating annotations, is implemented. Furthermore, the REST API, which uses the functions of the library, and the web application, which uses the implemented REST API, are implemented. The web application allows to upload a video, run a process for the video, which processes the video, and display the video with the calculated speeds processed by the process. The library and the REST API can be used as a basis of applications that do analysis of the objects based on their movement.

Keywords speed measurement of objects, objects in video recording, object detection, annotations, library, REST API, web application, Python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Měření rychlosti a analýza pohybu vozidel i jiných objektů ve videozáznamu je pro lidi, kteří monitorují pohyb vozidel, či jiných objektů, jako například policisté, či ostraha v obchodě nezbytností. Kamery, které monitorují pohyb objektů, se nachází všude a je potřeba z nich získat informace o pohybujících se objektech. Lidé, kteří monitorují pohyb objektů, potřebují systém dostupný z webu, který umožňuje detekovat pohybující se objekty, zaznamenávat jejich polohy a měřit jejich rychlost. Potřebují systém, který dá informace o pohybu jednotlivých objektů a případně zhodnotí, zda není pohyb nějakým způsobem podezřelý. Policisté potřebují například zjistit, zda vozidla nepřekračují maximální povolenou rychlost. Ostraha v obchodě potřebuje zjistit, zda se někdo nepohybuje podezřele pomalu, či naopak rychle.

Téma jsem si zvolil, protože sice existují desktopové aplikace, které měří pohyb vozidel, ale neexistují aplikace, které monitorují pohyb chodců a vypočítávají jejich rychlost. Rád bych těmto lidem umožnil monitorovat pohyb objektů přes webové rozhraní. Dále bych rád dal základ pro jejich aplikaci lidem, kteří chtějí vytvořit systém pro detekci podezřelého chování chodců, či zákazníků v obchodě.

Práce je zaměřena na návrh využití existujících anotačních nástrojů a návrh knihovny, která umožňuje zpracovat videozáznam, detekovat pohybující se objekty, vypočítávat jejich rychlost a generovat anotace. Dále je práce zaměřena na návrh REST API, které využívá funkce knihovny, a návrh webové aplikace, která volá funkce z REST API. V teoretické části nejdříve popisuji možné způsoby anotace videa a využití těchto způsobů. Poté jsou popsány způsoby detekce objektů a algoritmy pro měření jejich rychlosti. V této části jsou také popsány existující implementace popsaných algoritmů pro měření rychlosti objektů ve videozáznamu.

V praktické části se zaměřuji na návrh knihovny a popisuji její architekturu, použitý algoritmus pro měření rychlosti a použitý způsob využití anotačních nástrojů. Dále se zaměřuji na návrh REST API a popisuji způsob uložení dat,

se kterými REST API pracuje. V praktické části se také zaměřuji na návrh a implementaci webové aplikace, využívající navržené REST API.

Kapitola 1

Cíl práce

Hlavním cílem práce je navrhnout a implementovat knihovnu, která obsahuje funkce pro zpracování videa, detekci objektů, měření rychlosti objektů a generování anotací, REST API, které využívá funkce knihovny a zajišťuje zpracování videí i generování anotací a webovou aplikaci, která využívá REST API.

Teoretická část se zaměřuje na analýzu způsobů anotace a způsobů využití anotačních nástrojů, které přidávají do videa informace o pozici, čase a rychlosti objektu. Teoretická část je dále věnována analýze způsobů detekce objektů a algoritmů pro měření rychlosti objektů ve videozáznamu. Dále se zaměřuji na existující implementace algoritmů pro měření rychlosti objektů. Na základě těchto poznatků je možne navrhnout a implementovat knihovnu, REST API i webovou aplikaci.

Cílem praktické části je navrhnout a implementovat knihovnu, která umožňuje detekovat pohybující se objekty ve videozáznamu, změřit jejich rychlost a vygenerovat anotace s informacemi o jejich pohybu. Při implementaci bude knihovna vytvořena dle návrhu. Dalším cílem praktické části je navrhnout a implementovat REST API, které využívá funkce knihovny a webovou aplikaci, která toto REST API využívá. REST API i webová aplikace budou implementovány dle návrhu.

Kapitola 2

Analýza

V první části této kapitoly analyzuji způsoby, jakými je možné anotovat video. Do videa je možné přidat čas, souřadnice a velikost objektu na snímku a libovolné další informace. Některé z nich umožňují přímo vložit i další informace kromě času, souřadnic a velikosti, u jiných je potřeba další informace jako je naměřená rychlost objektu dodat.

Dále tu analyzuji způsoby, jak je možné detekovat objekty ve videozáznamu a měřit jejich rychlosti. Detekovat objekty je možné mnoha různými způsoby, ale podrobněji jsou popsány detekce pomocí Haarových XML souborů, ořezávání pozadí a detekce objektů s využitím knihovny ImageAI v Pythonu. Kromě způsobů detekce jsou také popsány algoritmy měření rychlosti již detekovaných objektů jako například automatická kalibrace kamery, měření rychlosti pomocí zadaných úseček, nebo lineární transformace snímků na základě daného čtverce na ploše s objekty, která se ve snímku zobrazí jako lichoběžník.

Dále jsou tu popsány implementace analyzovaných algoritmů pro detekci objektů a měření jejich rychlosti. Většina implementací nejdříve detekuje objekty a poté měří jejich rychlosti. První dvě implementace měří rychlost na základě úseček na snímků a dané reálné vzdálenosti tak, že zaznamenají, kdy první objekt navštívil první úsečku a poté zaznamenají, kdy objekt navštívil druhou úsečku a na základě zjištěných časů vypočtou rychlost. Třetí implementace zjišťuje tři úběžníky a ohniskovou vzdálenost na základě směru pohybu vozidel.

2.1 Anotace

Tato kapitola pojednává o způsobech anotace a jejich možnostech využití. V první sekci jsou popsány a srovnány jednotlivé anotační nástroje. V druhé části je popsáno několik způsobů využití popsaných anotačních nástrojů.

2.1.1 Způsoby anotace

Existuje několik způsobů, jak je možné přidávat metadata o objektech do videa. Do videa je možné přidat, kdy se objeví daný objekt ve videu, na jaké pozici se nachází. O objektu je možné také přidat další metadata jako například jeho rychlost. Mezi způsoby anotace videa patří například Media fragments URI, WebVTT, MPEG-21, MPEG-7, či SMIL. Media fragments URI a MPEG-21 jsou založené na URI, WebVTT, MPEG-7 a SMIL nejsou založené na URI.

2.1.1.1 Media fragments URI

Media fragments URI (Uniform Resource Identifier), specifikované v [1], je jeden ze způsobů fragmentace videa a patří mezi běžně využívané. Umožňuje vymezit časový úsek, či prostorový segment, který nás zajímá. Dále umožňuje vybrat si stopu (audio, či video), která nás zajímá. V neposlední řadě také umožňuje přidat další informace o daném fragmentu do id. Id může obsahovat například rychlost vozidla v daném čase a daném prostoru. Id označuje nějaký pojmenovaný časový úsek videa, na kterém nějaké vozidlo jede danou rychlostí.

Časový úsek videa je možné vymezit buď pomocí fragmentu přímo v URI adrese, nebo v URL adrese videa v elementu source uvnitř elementu video na HTML stránce. Vymezit zajímavý časový úsek videa je možné pomocí přidání fragmentu t. Fragment t může umožnit vymezení začátku, či konce zajímavého úseku videa, či vymezit, od jakého času do jakého času se zajímavý úsek ve videu nachází. Chceme-li identifikovat časový úsek videa na adrese `http://server.cz/mojevideo.mp4` od 12.5 sekundy do 14.4 sekundy, je možné to zajistit přidáním fragmentu do URI adresy: `http://server.cz/mojevideo.mp4#t=12.5,14.4`. Media Fragments URI také umožňuje vymezit určitý časový úsek videa pomocí identifikátoru, který může obsahovat například identifikátor měření rychlosti pohybujícího se objektu.

Po zadání URI adresy s fragmentem t, či id, či zaslání HTTP požadavku pro výběr identifikovaného časového úseku pomocí fragmentu id, či neidentifikovaného časového úseku pomocí fragmentu t se vrátí část videa se stavovým kódem 206 - Partial Content, který označuje, že se vrátila část videa, která přibližně odpovídá danému fragmentu. Část videa, která se vrátí, neodpovídá úplně přesně vymezenému časovému fragmentu a začátek i konec vráceného úseku se oproti zadaným časům mohou lišit o několik set milsekund.

Media Fragments také umožňuje vybrat prostorový segment videa pomocí fragmentu xywh. Je možné vymezit obdélníkový segment pomocí souřadnic x, y a jeho šířky a výšky. Souřadnice x, y a výšku a šířku obdélníkového segmentu je možné udávat v pixelech, nebo v procentech z velikosti videa. Pokud chceme prostorový segment na souřadnicích 20 px, 20 px se šířkou 50 px a výškou 70 px, je možné zadat následující URI adresu: `http://server.cz/mojevideo.mp4#wywh=20,20,50,70`. Chceme-li prostorový segment se souřadnicemi 20 %, 50 %, šířkou 25 % výškou 10 %, je nutné do hodnoty xywh fragmentu přidat prefix

"percent:" a zadáme následující URI adresu: `http://server.cz/mojevideo.mp4#xywh=percent:20,50,25,10`. Prostorový fragment můžeme vybrat tím, že zakryjeme pixely mimo vybraný prostor nějakým CSS (Cascading Stylesheets) přístupem. Zobrazení prostorového segmentu řeší User Agent, ale rozměry se rozlišují na všech úrovních protokolu.

K Media Fragments URI je možné přidat RDF (Resource Description Framework) metadata pomocí ontologie Ontology for Media Resources [2]. Ontologie umožňuje popsat dané video a jeho anotace. K anotacím je možné s využitím slovníku Ontology for Media Resources a slovníku OAC (Open Annotation Collaborative) přidávat i další objekty s metadaty pomocí predikátu ma:hasKeywords podobně jako v práci [3]. Objekt může obsahovat například informace o rychlosti pohybujícího se objektu, či jeho pozicích v rámci časového úseku.

Chceme-li k Media Fragments URI přidat další metadata, je potřeba nejdříve popsat video s pohybujícími se objekty jako objekt typu MediaResource a typu Target ze slovníku OAC, který obsahuje mnoho fragmentů. V datech se využívá prefix ma, který se využívá místo URI adresy `https://www.w3.org/ns/ma-ont`. Fragment se k videu přidá pom cí predikátu ma:hasFragment. Jednotlivé fragmenty jsou popsány objektem typu MediaFragment a typu Target. K fragmentu se přidá URI s dalšími informacemi jako například rychlost pohybujícího se objektu, či pozice, kam se přemístí, pomocí predikátu ma:hasKeyword.

Výhodou tohoto způsobu fragmentace videa je, že podporuje více formátů videa a není zaměřený na nějakou omezenou množinu formátu videa. Další jeho výhodou je, že je možné k němu přidat data ve formátu RDF s libovolnými dalšími metadaty. Lze například ke každé Media Fragments URI s vymezením času přidat rychlost, identifikátor pohybujícího se objektu a jeho pozice na snímcích.

Jeho nevýhodou je, že není možné přímo v URI definovat nějaký pohybující se obdélník, který reprezentuje pohybující se objekt v daném časovém úseku. Z tohoto důvodu je tedy nutné definovat přesuny objektů v RDF datech a minimálně pro snímky, které ohraničují nějaký časový úsek, definovat URI s fragmentem xywh.

2.1.1.2 WebVTT

WebVTT (Web Video Text Tracks) je formát souboru specifikovaný v [6], který se používá pro přidání metadat, či titulek k videu. Soubor vždy začíná řetězcem "WEBVTT", který signalizuje, že se jedná o WEBVTT titulky, či metadata. Tento formát umožňuje identifikovat a definovat libovolné časové úseky a ke každému časovému úseku je možné přidat text, který může obsahovat titulky, JSON, či jiný zápis s vlastními metadaty. Metadata ve formátu WebVTT se importují k videu pomocí elementu track uvnitř HTML5 (Hypertext Mar-

kup Language) elementu video s videem, jehož atribut kind má hodnotu "metadata" a atribut type má hodnotu "text/vtt".

WebVTT umožňuje přidávat libovolné časové úseky, které mohou být identifikovány. Identifikované časové úseky se nazývají cues. S těmito úseky a jejich textem je možné pracovat pomocí JavaScriptu. JavaScript umožňuje získat počáteční a koncový čas identifikovaného úseku, jeho text (popisek), popisek jako HTML, který je vrácen jako DocumentElement.

WebVTT je určen pro přidání časově sladěných metadat, jako je například rychlost vozidla a jeho poloha v čase. Rychlost objektu i polohu objektu v čase je možné přidat do popisku, který může být ve formátu JSON. Z popisku je možné pomocí JavaScriptu získat libovolná metadata.

Výhodou tohoto anotačního nástroje je, že je snadné ho přidat k videu pomocí elementu track a je snadné s ním pracovat pomocí JavaScriptu. V JavaScriptu stačí získat element video, z něj získat stopu (track) a ze stopy získat seznam identifikovaných časových úseků s metadaty (cues).

Nevýhodou tohoto anotačního nástroje je, že neobsahuje žádné rozšíření pro získání souřadnic ve videu. Souřadnice tedy musí být součástí popisku. Další metadata kromě identifikátoru vozidla a času také musí být součástí popisku. Je tedy nutné vytvořit popisek v nějakém formátu jako například JSON, či HTML a získat z něj ta metadata. Přidání metadat o poloze a rychlosti videa je možné pomocí JSON payload, ze které tyto údaje vyextrahujeme.

2.1.1.3 SMIL

SMIL (Synchronized multimedia Language), specifikovaný v [8], je jazyk založený na XML, který se používá pro různé účely včetně definování časových a prostorových fragmentů videa. Jedná se tedy o způsob fragmentace videa, který narozdíl od MPEG-21, či Media Fragments není založený na URI. Umožňuje vybrat určitý časový úsek videa i určitou prostorovou část videa. Od verze 3.0 je možné přidat metadata ve formátu RDF.

SMIL umožňuje vytvářet časové úseky videa s identifikátorem, který může obsahovat například rychlost vozidla i jeho identifikátor. Zajímavý časový úsek videa s identifikátorem umožňuje přehrát video od času, který nás zajímá. Mějme link na SMIL kód, který je umístěn na adrese `https://example.org/videoSmilPresentation`. Máme-li časový úsek videa od 12. do 14. sekundy s identifikátorem car1, je možné k tomuto úseku videa přistoupit například takto: `https://example.org/videoSmilPresentation#car1` a začít přehrávat video od 12. sekundy. Identifikované časové úseky zde umožňují rychlou indexaci a jednoduché skákání na čas videa, který nás zajímá.

SMIL dále umožňuje v rámci identifikovaného časového úseku přidat prostorové fragmenty, kde se může nacházet například nějaké vozidlo. Prostorový fragment může být přidán pomocí elementu area uvnitř elementu video. SMIL umožňuje definovat stejně jako Media Fragments pouze obdélníkový fragment.

Nevýhodou tohoto způsobu anontace videa je, že pro integraci SMIL do HTML5 je potřeba použít knihovnu timesheets.js [9], [10]. Na druhou stranu implementace s touto knihovnou je jednoduchá a stačí jen importovat tuto knihovnu do HTML5 a uvnitř HTML5 nastavit elementu ze SMIL základní jmenný prostor xmlns na SMIL.

2.1.1.4 MPEG-21

MPEG-21 (MPEG - Moving Picture Experts Group), popsaný v [11], je standard pro přidání metadat do videa ve formátu mpeg, nebo mp4, který je založený podobně jako Media Fragments na přidání metadat do videa pomocí fragmentu. Na rozdíl od media fragments umožňuje pracovat pouze s formáty mpeg a mp4.

Umožňuje vybrat libovolný čas, či region z videa. Pomocí něj je možné přidat informace o tom, kde a kdy se dané auto nachází. Oproti Media Fragments umožňuje přidat do videa pohyblivý region, což umožní efektivně přidat informaci o tom, jak se dané auto v daném časovém rozpětí pohybuje.

Časový, či prostorový segment je možné přidat pomocí fragmentu mp(). Fragment mp() umožňuje například vybrat úsek videa od 10. do 12. sekundy takto: `http://server.cz/myvideo.mp4#mp(~time('npt','10','12'))`. Fragment mp() dále umožňuje vybrat pohybující se region, který umožňuje zachytit jedoucí auto v nějakém úseku videa. Pomocí fragmentu ffp() je možné předat videu identifikátor objektu, který nás zajímá. Součástí identifikátoru mohou být i další informace, jako je rychlost vozidla. Další informace jako je rychlost vozidla musí být vyextrahovány z identifikátoru.

Nevýhodou tohoto způsobu anotace videa je, že stejně jako v Media fragments musí být rychlost objektu součástí identifikátoru. Pokud se ve videozáznamu objeví objekt, který jede rychlostí 84 km/h, hodnota fragmentu id bude například id. Je tedy nutné extrahovat rychlost vozidla z identifikátoru, což není úplně vhodné.

2.1.1.5 MPEG-7

„MPEG-7 je standard pro popis multimediálního obsahu“ [12]. Jedná se také o způsob anotace videa, popsaný v [13], který je založen na XML, nikoliv na fragmentech v URL. MPEG-7 umožňuje definovat časové, prostorové i časově prostorové segmenty a každý segment může obsahovat anotaci. MPEG-7 obsahuje anotaci pomocí libovolného textu, strukturovanou anotaci, anotaci klíčových slov, či vestavěný sémantický deskriptor. MPEG-7 umožňuje přistupovat k fragmentům .mp7 (formát pro uložení MPEG-7 popisu videa) souboru pomocí identifikátoru, který se vloží do URI. User Agent potřebuje zpracovat tento element, aby se mohl odkázat na daný fragment videa.

Kořenovým elementem MPEG7 je mpeg7, elementy, do kterých se vloží element video reprezentující video, jsou “Description” a “Multimedia content”.

Uvnitř elementu video se nachází informace o tom, na jaké URL adrese se video nachází a jak je dlouhé. V elementu video může být i dekompozice videa na jednotlivé časové, prostorové, či časově prostorové fragmenty. Informace o videu mohou být umístěny i v elementu VideoSegment.

MPEG-7 umožňuje dekompozici videa, či jeho segmentu na časové úseky pomocí elementu TemporalDecomposition. Uvnitř elementu TemporalDecomposition je možné mít elementy VideoSegment s identifikátorem, které budou reprezentovat jednotlivé časové úseky. Časové úseky je možné vymezit přesně na snímky a je možné vybrat například 60. - 65. snímek videa.

MPEG-7 dále umožňuje dekompozici videa, či jeho segmentu na prostorové segmenty pomocí elementu SpatialDecomposition. Prostorový segment je možné definovat pomocí elementu VideoSegment. Prostorovým segmentem zde může být libovolný mnohoúhelník.

MPEG-7 dále umožňuje dekompozici videa, či jeho segmentu na časové prostorové segmenty (tj. pohyblivé regiony) pomocí elementu SpatialTemporalDecomposition. Časově prostorový segment je možné definovat pomocí elementu MovingRegion. MovingRegion v sobě obsahuje element SpatioTemporalLocator, který slouží pro definování času, v němž se objekt nachází na dané pozici, a element SpatioTemporalMask, který definuje prostor.

Do segmentu videa je možné vložit jeho anotaci pomocí libovolného textu, strukturovanou anotaci, anotaci klíčových slov, či vestavěný sémantický deskriptor. Pro zaznamenání rychlosti vozidla ve videozáznamu lze použít libovolný způsob anotace, ale je vhodné použít buď strukturovanou anotaci, anotaci pomocí klíčových slov, či vestavěný sémantický deskriptor.

Výhodou tohoto způsobu anotace je, že je možné rozdělit video na jednotlivé segmenty podle času, či prostoru. Je tedy snadné přidat ke každému vozidlu v daném čase informaci o jeho rychlosti. Pro sledování rychlosti vozidel v čase a daném prostoru je vhodné použít buď kombinaci časové a prostorové dekompozice, nebo dekompozici na časově prostorové segmenty.

2.1.2 Využití anotací

V této kapitole jsou popsány různé návrhy využití anotačních nástrojů, které umožňují získat informace o objektu jako je čas, kdy se objevuje ve videozáznamu, jeho umístění a velikost na snímku a jeho rychlost. Jeden z přístupů využívá čistě Media Fragments, další přístup kombinuje Media Fragments a WebVTT a třetí přístup využívá časově prostorovou dekompozici v MPEG-7.

2.1.2.1 Media fragments

Jedním z přístupů, jak je možné přidat čas, souřadnice x,y a rychlost objektu do videa, je využití Media Fragments s přidáním metadat do RDF. Souřadnice objektu x, y, jeho výška a šířka na začátku časového úseku, kdy objekt jede

danou rychlostí, jsou hodnotou fragmentu xywh. Časový úsek, po který se objekt pohybuje danou rychlostí, je hodnotou fragmentu t. Souřadnice objektu v dalších snímcích a rychlost objektu je přidána k Media Fragments URI pomocí RDF.

Mějme například automobil, který jede od 9. do 10. sekundy rychlostí 80 km/h. V 9. sekundě se nachází na souřadnicích 50, 50 a má rozměry 32 x 32 a v 10. sekundě je na pozici 500, 500 a má rozměry 100 x 90. Automobil má identifikátor 1. Media Fragments URI, pomocí které je možné získat pohyb automobilu, je `http://www.mujserver.com/video.mp4#t=9,10&xywh=50,50,32,32`.

Následující návrh i popis se zakládá na práci [3]. Video je popsáno v RDF pomocí objektu typu MediaResource ze slovníku Ontology for Media Resources a typu Target ze slovníku OAC. Existují zde trojice s URI videa a predikáty hasFragment, které k danému videu přiřazují určitou URI pro Media Fragments. Media Fragments URI je popsána pomocí objektu typu MediaFragment a typu Target. S danou Media Fragments URI existuje trojice, která přiřazuje dané Media Fragments URI objekt s metadaty jako je například rychlost objektu a Media Fragments URI s pozicemi objektu na snímcích v daném časovém úseku.

Metadata jsou k dané Media Fragments URI přiřazena pomocí objektu typu Keyword, či jiného definovaného vlastního typu. V ukázce se využívá typ Keyword, který může obsahovat trojice s rychlostí objektu a trojice s Media Fragments URI. Rychlost objektu v daném časovém úseku je v ukázce popsána pomocí predikátu speed. Pozice objektu na snímcích v rámci časového úseku jsou definovány pomocí Media Fragments jako objekt URI a predikátu mediaFragmentsURI. Dále se pomocí trojice s predikátem identificator ukládá k dané Media Fragments URI s fragmenty t a xywh identifikátor pohybujícího se objektu.

Listing 2.1: Ukázka reprezentace metadat v RDF přidaných k Media Fragments URI s časovým úsekem, souřadnicemi a velikostí obdélníku, ve kterém se nachází pohybující se objekt na začátku časového úseku

```
@prefix keywords: <http://www.mujserver.com/keyword/>
@prefix ma: <http://www.w3.org/ns/ma-ont#>.
@prefix oac:<http://www.openannotation.org/ns/>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.

<http://www.mujserver.com/api/video1> a~ma:MediaResource;
        a oac:Target;
        rdfs:label "Video 1",

        ma:hasFragment <http://www.myserver.com/video1.
            mp4#t=9,10&xywh=50,50,32,32>;
        ma:locator <http://www.myserver.com/video1.mp4>.

<http://www.mujserver.com/video1.mp4#t=9,10&xywh
    =50,50,32,32> a~ma:MediaFragment;
```

```
            a oac:Target;
            ma:description "Media fragment t=9,10&xywh
                =50,50,32,32 with added metadata"@en;
            ma:hasKeyWord keywords:keyword1.

keywords:keyword1 a~keywords:Keyword;
            keywords:identificator "1";
            keywords:speed "80 km/h";
            keywords:mediaFragmentsURI <http://www.mujserver.
                com/video1.mp4#t=9&xywh=50,50,32,32>;
            keywords:mediaFragmentsURI <http://www.mujserver.
                com/video1.mp4#t=10&xywh=500,500,100,90>.
```

Výhodou tohoto způsobu využití anotačních nástrojů je, že pomocí RDF lze k danému časovému a prostorovému fragmentu videa přidat další metadata jako je například rychlost objektu a pozice na snímcích v rámci časového úseku a že metadata mohou být reprezentována libovolným způsobem. Pozice objektu na snímcích mohou být uloženy pomocí trojic s Media Fragments URI jako objekt, či trojic, kde objektem je jiný objekt obsahující pozici a velikost daného objektu a číslo snímku, či čas.

Nevýhodou tohoto způsobu je, že není možné v URI zaznamenat přesun objektu ani pozici objektu na začátku a na konci časového úseku. Do fragmentu xywh je možné uložit jen pozici a velikost objektu na začátku časového úseku, či pozici a velikost objektu na konci časového úseku. Informace o pohybu objektu musí být přidány do metadat pomocí RDF trojic.

2.1.2.2 Media fragments a WebVTT

Další možností, jak přidat metadata o čase, pozici a rychlosti objektů, je kombinovat Media Fragments URI a WebVTT. Media Fragments URI se využívá pro předání identifikátoru časového úseku, který také identifikuje měření rychlosti objektu. WebVTT přidává ke každému časovému úseku další metadata jako například pozice objektu či jeho rychlost.

V souboru ve formátu vtt (WebVTT titulky, či metadata) se nachází definice časových úseků, ve kterém se daný objekt pohybuje danou rychlostí. Jednotlivé časové úseky reprezentují měření rychlosti objektu. Každý časový úsek je identifikován svým číslem, které odpovídá identifikátoru měření rychlosti. V textových metadatech se nachází informace o rychlosti objektu a pozici objektu na začátku a na konci časového úseku a identifikátor pohybujícího se objektu. Informace o pozicích a velikostech objektu na snímcích je možné reprezentovat jako řetězce s Media Fragments URI, či jako vnořenou strukturu s pozicí, rozměry a číslem snímku, nebo časem v sekundách.

Mějme například objekt s identifikátorem 6, který se nachází ve videu od 9. do 10. sekundy. V 9. sekundě se objekt nachází na souřadnicích 542, 435 a má šířku 125 a výšku 212. Objekt se do 10. sekundy přemístí na pozici 200, 850 a bude mít šířku 250 px a výšku 414 px. Media Fragments URI pro daný časový úsek s identifikátorem 11 je `http://www.mujserver.com/video1.mp4#id=11`.

Soubor ve formátu vtt s metadaty bude obsahovat níže uvedenou definici časového úseku s metadaty.

Listing 2.2: Ukázka identifikovaného fragmentu WebVTT metadat s vymezením časového rozmezí a metadaty včetně rychlosti objektu

```
11
00:09.000 -> 00:10.000
{
        "objectId": "6",
        "speed": "84 km/h",
        "mediaFragmentsURIs": ["http://www.mujserver.com/video1.
            mp4#t=9&xywh=542,435,125,212", "http://www.mujserver.
            com/video1.mp4#t=10&xywh=200,850,250,414"]
}
```

Výhodou tohoto způsobu využití anotačních nástrojů je, že je možné k dané Media Fragments URI snadno najít časový úsek s metadaty. Pro získání metadat k časovým úsekům není třeba se připojovat do databáze ani do TripleStoru, protože WebVTT soubor může být uložen lokálně na stroji. WebVTT soubor může být snadno načten a časové úseky mohou být uloženy například do pole objektů.

Další výhodou je, že je možné reprezentovat metadata libovolným způsobem. Pozice a velikosti objektů na snímcích je možné uložit jako řetězce s Media Fragments URI, či objekty. Pozice a velikosti objektů je možné uložit do pole a snadno tak získat seznam pozic a velikostí objektu na snímcích.

2.1.2.3 SpatialTemporalDecomposition v MPEG7

Jedním z možných způsobů, jak anotovat objekty v nějakém čase, na nějaké pozici, které se pohybují danou rychlostí, je časově-prostorová dekompozice v MPEG7. Časové rozmezí je vymezené v elementu MediaTime, pozice, kde se objekt nachází, je vymezená pomocí elementu Subregion a vnořeného elementu Poly. Rychlost, jakou se daný objekt pohybuje, bude obsažena v elementu TextAnnotation a uvnitř elementu FreeTextAnnotation. Rychlost je možné anotovat i dalšími způsoby jako je například KeywordAnnotation, či StructuredAnnotation a vnitřní element Speed.

Každý objekt v daném čase bude reprezentován pomocí elementu MovingRegion, či elementu StillRegion, který bude mít identifikátor s označením objektu a jeho rychlosti. Tento element bude obsahovat element SpatioTemporalLocator, který bude v sobě obsahovat element MediaTime s definicí časového rozpětí, ve kterém se bude daný objekt pohybovat danou rychlostí a bude na daném místě. Dále bude v sobě obsahovat element s anotací rychlosti objektu. Element MovingRegion může v sobě obsahovat element KeywordAnnotation, uvnitř kterého bude element Speed s rychlostí objektu. V neposlední řadě bude potomkem tohoto elementu i element Mask, který v sobě obsahuje element SubRegion s definicí pozice objektu uvnitř elemetu Poly.

Chceme-li anotovat objekt ve videu se snímkovou frekvencí 50 snímků za sekundu v čase 10 sekund, který se nachází na souřadnicích 542, 435, bude mít šířku 125 a výšku 212 a jede rychlostí 84 km/h, bude element MovingRegion vypadat například takto:

Listing 2.3: Ukázka časové prostorové dekompozice v MPEG-7

```
<VideoSegment>
  <SpatialTemporalDecomposition>
    <MovingRegion id="car1s84">
      <StructuredAnnotation>
        <Speed>84</Speed>
      </StructuredAnnotation>
      <SpatioTemporalLocator>
        <MediaTime>
          <MediaTimePoint>T00:00:10:0F25000</MediaTimePoint>
          <MediaDuration>PT10S500N25000F</MediaDuration>
        </MediaTime>
      </SpatioTemporalLocator>
      <SpatioTemporalMask>
        <Mask xsi:type="SpatialMaskType">
          <SubRegion>
            <Poly>
              <Coords>542 435, 667 435, 667 647, 542 647</Coords>
            </Poly>
          <SubRegion>
        </Mask>
      </SpatioTemporalMask>
    </MovingRegion>
  </SpatialTemporalDecomposition>
</VideoSegment>
```

Výhodou tohoto přístupu je, že je možné snadno získat informace o tom, v jakém čase se daný objekt na daném snímku nachází, na jakých souřadnicích se daný objekt nachází a jakou rychlostí se pohybuje. Rychlost objektu je snadné získat z elementu Speed, který se nachází v elementu StructuredAnnotation.

Další výhodou tohoto přístupu je, že lze pomocí elementu SubRegion definovat další polygon, který bude obsahovat souřadnice bodů, na kterých se nachází objekt. Pokud chceme zaznamenat pozici objektu na začátku časového snímku a pozici objektu na konci časového snímku, stačí definovat v elementu Mask uvnitř elementu SpatioTemporalMask definovat dva elementy SubRegion. První element SubRegion obsahuje polygon, který reprezentuje, kde se objekt nachází ve videu na začátku daného časového úseku. Druhý element SubRegion obsahuje polygon, který obsahuje souřadnice bodů polygonu, reprezentujícího daný objekt na konci daného časového úseku.

2.1.2.4 Shrnutí způsobů využití anotací

V tabulce jsou vyhodnoceny způsoby využití anotací jako je MediaFragments, kombinace MediaFragments a WebVTT a využití MPEG7. Způsoby využití

anotací jsou srovnávány z hlediska obtížnosti implementace a získávání metadat o čase, souřadnicích a velikosti na snímku a rychlosti. Využití Media Fragments má výhodu v tom, že se nemusí složitě získávat časový úsek, souřadnice objektu ani identifikátor měření rychlosti. Na druhou stranu nelze přímo získávat další metadata jako je rychlost objektu. Rychlost objektu a další metadata musí být buď v RDF, nebo ve VTT souboru s metadaty. Media Fragments také neumožňuje pohyblivé regiony (tj. vymezit časový úsek, počáteční souřadnice a velikost a koncové souřadnice a velikost) bez použití RDF. MPEG-7 umožňuje snadno přidávat pohyblivé regiony pomocí elementů Poly a přidávat další metadata například v elementu StructuredAnnotation. Na druhou stranu je nutné parsovat MPEG-7 v XML a implementovat výběr určitého časového fragmentu videa.

Tabulka 2.1: Srovnání využití MediaFragments, kombinace MediaFragments a WebVTT a využití MPEG7 z hlediska využitelnosti a obtížnosti implementace a získání metadat

Kritérium/Způsob anotace	Media Fragments	Media Fragments + WebVTT	MPEG 7
Obtížnost implementace	Získání metadata pomocí RDF, dotazování na danou URI do TripleStoru pomocí SPARQL dotazu	Jednoduché zpracování pomocí JavaScriptu, získání dat z WebVTT pomocí JavaScriptu	nutnost implementovat výběr časového úseku, dále je nutné načíst XML soubor a získat z něj potřebná metadata - jednodušší díky externím knihovnám
Získání rychlosti objektu	vyhledání v RDF datech k dané Media Fragments URI	přímo z textových metadat WebVTT	přímo z anotací (např. element StructuredAnnotation)
Výběr časového úseku videa	přímo fragmentem t	přímo fragmentem t	pomocí fragmentu id s tím, že XML soubor musí být na straně UA (User agent) parsován
Výběr prostorového segmentu videa	přímo fragmentem xywh	přímo fragmentem xywh	z elementu Poly po parsování XML souboru
pohyblivé regiony	je možné přidat do RDF k dané URI	možné přidat do WebVTT textu k časovému úseku - každý časový úsek může obsahovat JSON - získání JSONu z textových metadat	přímo v elementu Poly

2.2 Algoritmy pro měření rychlosti a detekce objektů

Existuje několik algoritmů pro měření rychlosti objektu ve videozáznamů. Rychlost objektu lze měřit například pomocí sledování, kdy daný objekt navštívil daný bod, pomocí zadaného obdélníku a lineární transformace, pomocí parametrů kamery jako je úhel, umístění a ohnisková vzdálenost, či pomocí zadaných bodů a matice kamery. Některé z těchto algoritmů jsou sice rychle, ale zanedbávají radiální zkreslení kamery.

Samotné algoritmy pro měření rychlosti ale potřebují mít detekované objekty. Existuje několik způsobů detekce objektů jako například odečítání pozadí, detekce pomocí XML souboru, či s knihovnou ImageAI v Pythonu. Další způsoby detekce objektů jako je SIFT (Scale Invariant Feature Transform) a SURF (Speeded up robust features) nejsou zde podrobně popsány.

2.2.1 Detekce objektů

K měření rychlosti je zapotřebí detekovat objekty, abychom věděli, kde se nacházejí. Detekovat objekty lze několika různými způsoby jako je odečítání pozadí, detekce pomocí haarového XML souboru, či pomocí knihovny ImageAI v Pythonu. Existují ale i další způsoby detekce objektů jako je například SIFT a SURF. V této kapitole jsou podrobněji rozepsáno odečítání pozadí, detekce pomocí Haar Cascade XML souboru a detekce objektů s využitím knihovny ImageAI v Pythonu.

2.2.1.1 Odečítání pozadí

Objekty lze detekovat například pomocí odečítání pozadí. Tento způsob, použitý například v [15], spočívá v tom, že se vypočte rozdíl mezi aktuálním snímkem a prvním snímkem, či libovolným z předchozích snímků. Je-li aktuální snímek odlišný od předchozího snímku, znamená to, že na snímcích se nachází pohybující objekty, které jsou detekovány.

Máme-li vypočtený rozdíl mezi aktuálním snímkem a zadaným předchozím snímkem, či prvním snímkem videa, je možné spatřit rozdíly a na základě nich detekovat objekty. Zpravidla se to implementuje tak, že se uloží reprezentace rozdílu mezi aktuálním snímkem, kde pozadí má černou barvu a nalezené rozdíly mají šedou, či bílou barvu. Tento rozdíl je reprezentován jako objekt typu snímek, nebo jako pole. Pokud ve snímku, reprezentujícím rozdíly, najdeme plochu s mnoha bílými pixely, máme již detekovaný pohybující se objekt.

Výhodou tohoto způsobu detekce objektů je, že lze použít univerzálně na všechny typy objektů jako jsou auta, chodci, motorky. Pomocí odečtu pozadí je možné detekovat více typů objektů najednou a je možné sledovat pohyb aut i motocyklů.

Nevýhodou tohoto způsobu detekce objektů je, že těžko rozpozná objekty, které mají podobnou barvu jako povrch. Tento způsob nepřesně detekuje například vozidla, které jsou tmavě stříbrné, protože mají podobnou barvu jako vozovka.

2.2.1.2 Haar Cascade XML soubory

Detekovat objekty lze pomocí Haar Cascade XML (Extensible Markup Language) souborů, které obsahují definici typických rysů daného druhu objektů jako například vozidla, či chodci. Tento přístup, popsaný v [16], [17] a [18], je založen na strojovém učení a používá trénovací množinu pozitivních a negativních obrázků. Tento způsob je použitelný zejména v případě, zajímá-li nás jeden druh pohybujících se objektů. XML soubor obsahuje definice typických rysů daného druhu objektů, nebo definici pozitivních a negativních obrázků.

Tato metoda potřebuje velké množství pozitivních a negativních obrázků. Pozitivním obrázkem se rozumí obrázek, který obsahuje objekt, který chceme detekovat. Chceme-li detekovat automobily, pozitivní obrázek je takový obrázek, na kterém se nachází nějaký automobil. Negativní obrázek je takový obrázek, na kterém se nenachází objekt, který chceme detekovat. Pro předchozí příklad je negativním obrázkem například obrázek, na kterém se nachází jen osoba.

Haar Cascade XML soubor lze vygenerovat pomocí příkazu *opencv_traincascade* s parametry jako je umístění pozitivních obrázků a umístění negativních obrázků. Tento příkaz také obsahuje počet epoch při trénování.

Výhodou tohoto způsobu detekce objektů je, že rozpozná daný druh objektu bez ohledu na jeho barvu. V případě detekce automobilů lze tímto způsobem detekovat i automobily, které mají podobnou barvu jako vozovka.

Nevýhodou tohoto způsobu detekce objektů je, že přesnost závisí na velikosti trénovací množiny objektů. Pro velmi přesnou detekci je zapotřebí, aby trénovací množina obsahovala více než desetitisíce pozitivních a negativních obrázků a počet epoch byl v řádu desítek. Vygenerovat XML soubor, který umožňuje přesnou detekci objektů, bude trvat velmi dlouho.

2.2.1.3 Image AI v Pythonu

ImageAI je knihovna v Pythonu, která umožňuje detekovat objekty ve videu, či v obrázku. Tato knihovna, popsaná v [19], detekuje objekty a k nim přiřazuje, co to pravděpodobně je a s jakou pravděpodobností se jedná o daný druh objektu.

Tato knihovna pracuje s modelem, který lze načíst ze souboru ve formátu h5. Pro detekci objektů ve videozáznamu se používá typ modelu YOLO verze 3. Pro detekci objektů ve videozáznamu se používá soubor yolo.h5.

Výhodou použití knihovny ImageAI je, že je možné detekovat více druhů objektů naráz. Je možné například detekovat automobily i motocykly najednou.

S využitím této knihovny je možné sledovat pohyb automobilů i motocyklů a zároveň vědět, zda se jedná o automobil, či motocykl.

2.2.2 Algoritmy měření rychlosti

Existuje několik algoritmů pro měření rychlosti objektu ve videozáznamů. Rychlost objektu lze měřit například pomocí sledování, kdy daný objekt navštívil daný bod, pomocí zadaného obdélníku a lineární transformace, pomocí parametrů kamery jako je úhel, umístění a ohnisková vzdálenost, či pomocí zadaných bodů a matice kamery. Některé z těchto algoritmů jsou sice rychlé, ale zanedbávají radiální zkreslení kamery.

2.2.2.1 Měření rychlosti pomocí zaznamenání návštěvy úseček

Dalším způsobem měření rychlosti je, že zjistíme počet snímků mezi okamžikem, kdy objekt navštíví první úsečku, a okamžikem, kdy objekt navštíví druhou úsečku. [21] Vstupními parametry jsou souřadnice koncových bodů obou úseček a reálná vzdálenost mezi nimi. Nejdříve si pro každý objekt zapamatujeme pořadí snímku, kdy navštívil první úsečku. Poté si zapamatujeme pořadí snímku, kdy objekt navštívil druhou úsečku. Následně se spočítá rozdíl mezi oběma zaznamenanými snímky.

Pro každý snímek je potřeba nejdříve najít kontury pohybujících se objektů. Po nalezení kontur pohybujících se objektů, či obdélníků, do nichž je možné vepsat příslušné kontury, pro každou konturu, či obdélník zjistíme, zda se první, či druhá úsečka nachází uvnitř kontury, či obdélníku.

Pokud se uvnitř kontury, či obdélníku nachází první úsečka, zaznamenáme pořadí snímku, kdy objekt navštívil první úsečku. Pro objekty, které navštívili první úsečku, je možné zjistit, zda navštívili druhou úsečku. V případě, že se uvnitř kontury takového objektu, či obdélníku, do něhož je možné vepsat příslušnou konturu, nachází druhá úsečka, zapamatujeme si pořadí aktuálního snímku. Od čísla, které udává toto pořadí, odečteme číslo, které odpovídá pořadí snímku, kdy daný objekt navštívil první úsečku. Rychlost objektu v metrech za sekundu se dá vypočítat tak, že vydělíme vzdálenost mezi danými úsečkami reálným časem, po který se objekt pohyboval mezi první a druhou úsečkou. Reálný čas, který uplynul mezi okamžikem, kdy objekt navštívil první úsečku, a okamžikem, kdy objekt navštívil druhou úsečku, se vypočte tak, že se rozdíl pořadí snímků vydělí snímkovou frekvencí videa.

Tento algoritmus dává dobré výsledky, pokud jsou obě úsečky dostatečně vzdálené od sebe a objekty se pohybují v přímém směru. Pokud se jedná o víceproudou rovnou dálnici a úsečky jsou vzdálené více než 60 metrů, algoritmus dává dobré výsledky. Nepřesnosti jsou v takových případech nižší než 3 %. V případě městských víceproudých pozemních komunikací stačí, aby úsečky byly vzdálené od sebe okolo 35 metrů.

Nevýhodou tohoto algoritmu je, že dává nepřesné výsledky například pro vozidla, která předjíždí jiné vozidlo a zpět se zařadí. Pokud je měřený úsek dlouhý, nepřesnost způsobená porušením předpokladu rovného pohybu vozidel není značná.

2.2.2.2 Měření rychlosti pomocí obdélníku na 3D ploše

Další podobný postup měření rychlosti, který je popsán v [22], spočívá v tom, že bude zadán obdélník na 3D ploše, který se na snímcích zobrazí jako lichoběžník. Budou zadány body lichoběžníků a jeho reálná šířka a výška v metrech. Tento lichoběžník se pomocí lineárně transformuje na obdélník a zbytek snímku se stejným způsobem přetransformuje.

Na základě transformovaného obdélníku se vypočte, kolik pixelů za sekundu odpovídá jednomu kilometru za hodinu. Ve snímcích se detekují jednotlivá vozidla, která se také budou transformovat. Z transformovaných snímků se vypočte, kolik pixelů za sekundu auto urazilo a jednoduše se dopočte jeho reálná rychlost v kilometrech za hodinu.

Tento způsob dává dobré výsledky pro kamery, které mají velmi malé radiální zkreslení. Nepřesnosti ve výpočtu vzdálenosti jsou přímo úměrné radiánímu zkreslení kamery a závisí na tvaru čočky. Tyto nepřesnosti jsou způsobeny tím, že transformace je lineární a nepředpokládá se, že kamera bude mít nějaké větší radiální zkreslení.

2.2.2.3 Měření rychlosti pomocí zjednodušeného modelu kamery

Dalším ze způsobů, jak měřit rychlost objektu ve videu je, že spočteme jeho relativní souřadnice v čase, kdy byl objekt detekován a o jeden, nebo více snímků později. Vstupem bude úhel kamery vůči vozovce, či jiné ploše, umístění kamery nad plochou, po které se objekty pohybují, velikost čočky a ohnisková vzdálenost. Ze získaných souřadnic vypočítáme vzdálenost, na základě které se vypočítá rychlost vozidla.

Jedním ze způsobů, jak vypočítat souřadnice objektu vůči kameře je, že zvolíme jeho libovolný bod a spočteme jeho souřadnice. Jedním z vhodných bodů může být například spodní bod uprostřed objektu. O tomto bodu budeme předpokládat, že se nachází na ploše v dané výšce pod centrem kamery, které bude počátkem souřadného systému. Souřadný systém je definován tak, že na osa Z je kolmá vůči dané ploše, na které jsou objekty, osa Y vede od centra kamery přes střed snímače, či nad střed snímku a osa X vede nalevo od centra kamery. Pokud je úhel naklonění kamery nulový, osa Y vede od centra kamery přes střed snímače. V případě kladného úhlu naklonění vede osa Y od centra kamery přímo nad střed snímače. Body s nulovou souřadnicí X a kladnou souřadnicí Y mohou být vidět na prostřední čáře snímku, která vede zdola nahoru.

Prvním předpokladem je, že plocha pod kamerou, po které se pohybují objekty, je rovná. Vzhledem k tomu, že sklon většiny měřených úseků je malý, není tento předpoklad problém. Dalším předpokladem je, že principální bod se nachází ve středu obrazu, není-li explicitně zadán. Principální bod „je takový bod, ve kterém optická osa kamery protíná obrazovou rovinu.“ [24]. Umístění principálního bodu do středu obrazovky není problémem, pokud se nachází blízko středu obrazovky.

Nechť kamera má souřadnice [0, 0, 0], H je výška nad plochou, ve které se nachází kamera, α je úhel naklonění kamery vůči ploše, na které se mohou pohybovat objekty, β je úhel otočení kamery okolo osy z, f je ohnisková vzdálenost, W_k je šířka čočky, H_k je výška čočky. Ohnisková vzdálenost i rozměry čočky jsou udávány zpravidla v milimetrech. [x,y] jsou souřadnice bodu na obrazovce, w je šířka snímku a h je výška snímku. Souřadnice body a rozměry snímku jsou udávány v pixelech.

Hledáme reálné souřadnice objektu [X, Y, Z] v milimetrech, který se nachází na zadaných souřadnicích na obrazovce. Dalšími parametry mohou být například souřadnice principálního bodu $[p_x, p_y]$, pokud se liší od středu snímku. Předpokládejme, že objekty se pohybují po rovné ploše. Díky tomuto předpokladu lze snadno odvodit, že souřadnice Z bude zápornou hodnotou výšky H nad plochou.

$$Z = -H \tag{2.1}$$

Dále je potřeba spočítat rozdíl souřadnic x a y a souřadnic principálního bodu. Pokud nejsou souřadnice principálního bodu explicitně zadány, považuje se za principální bod střed obrazovky. Pokud nejsou souřadnice principálního bodu zadány, platí:

$$p_x = \frac{w}{2} \tag{2.2}$$

$$p_y = \frac{h}{2} \tag{2.3}$$

V každém případě odečteme souřadnici x od souřadnice p_x principálního bodu a rozdíl uložíme do proměnné Dx. Proměnná Dy bude obsahovat rozdíl souřadnice y a souřadnice p_y principálního bodu. Tyto rozdíly se následně převedou z pixelů na milimetry pomocí zadaných rozměrů čočky. Se znalostí, že h pixelů odpovídá W_k milimetrům na šířku a h pixelů odpovídá H_k pixelů na výšku, snadno dopočteme reálné umístění R_x a R_y vůči principálnímu bodu na snímací ploše.

$$Dx = x - p_x \tag{2.4}$$

$$Dy = y - p_y \tag{2.5}$$

$$R_x = \frac{Dx * w}{W_k} \tag{2.6}$$

$$R_y = \frac{Dy * h}{H_k} \tag{2.7}$$

Přímka, která vede od centra kamery přes získaný bod na obrazovce se získaným umístěním R_x, R_y, svírá se snímací plochou úhel α'. Je-li zadaný bod nad principálním bodem, úhel α' je záporný. Je možné ho dopočítat pomocí souřadnice R_y a ohniskové vzdálenosti.

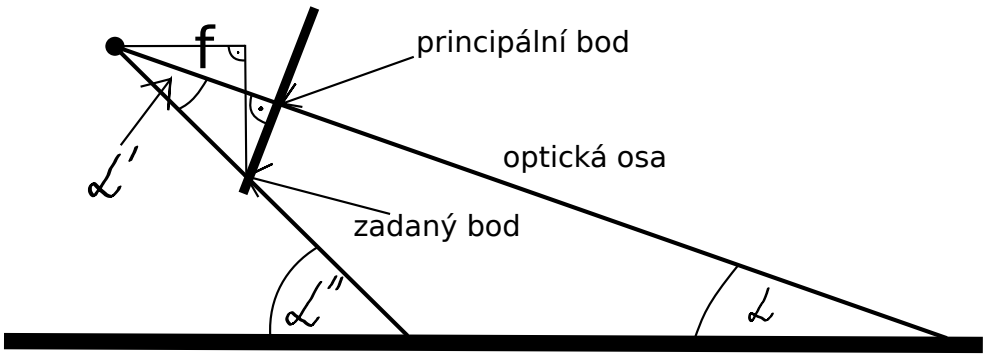
$$tan(\alpha') = \frac{R_y}{f} \tag{2.8}$$

Víme, že optická osa, která je kolmá na snímací plochu a prochází principálním bodem, svírá s plochou, na které se pohybují objekty, úhel α. Poté získáme úhel α'', který svírá přímka, která vede centrem kamery a daným bodem na snímací ploše se získaným umístěním vůči principálnímu bodu, s vozovkou.

$$\alpha'' = \alpha - \alpha' \tag{2.9}$$

Přímka, která vede přes centrum kamery a zadaný bod na obrazovce, svírá s plochou získaný úhel α''. Obrázek je převeden na 2D plochu, kde se ignoruje souřadnice x, a znázorňuje, jak je nakloněná kamera na vozovku, kudy prochází optická osa a kudy prochází přímka, která vede od centra kamery přes daný bod na snímací ploše a vede dále skrz plochu, po které se pohybují objekty, jejichž rychlost měříme.

Na obrázku Je možné spatřit, že úsečka, která vede od centra kamery do zadaného bodu na snímací ploše, je po ignorování souřadnice x přeponou trojúhelníka, jehož jedna odvěsna má délku, odpovídající ohniskové vzdálenosti f, a druhá odvěsna má délku, která odpovídá reálnému rozdílu souřadnice y zadaného bodu a souřadnice y principálního bodu. Pomocí dalšího trojúhelníku se stejnou přeponou a dvěma na sebe kolmými odvěsnami, z nichž jedna vede od zadaného bodu na snímací ploše a druhá vede od centra kamery, lze dopočítat souřadnice z' a y'. Souřadnice y' udává reálnou souřadnici y zadaného bodu na snímací ploše. Souřadnice z' odpovídá inverzní hodnotě souřadnice z zadaného bodu na snímací ploše vůči sčítání. Souřadnici x Je možné snadno dopočítat jako rozdíl souřadnice x zadaného bodu a souřadnice x principálního bodu.



Obrázek 2.1: Nákres centra kamery, snímací plochy, optické osy a přímky, která prochází zadaným bodem

$$z' = \sqrt{(f^2 + R_y^2)} * \sin(\alpha'') \tag{2.10}$$

$$y' = \sqrt{(f^2 + R_y^2)} * \cos(\alpha'') \tag{2.11}$$

$$x' = R_x \tag{2.12}$$

Následně ze získané souřadnice z' dopočteme koeficient coef, který udává poměr výšky, ve které se nachází centrum kamery nad vozovkou, a spočtené hodnoty souřadnice z'. Chceme najít, v jakých souřadnicích x a y od centra kamery se nachází bod na ploše, který je zobrazen v daném bodě. Je-li úhel β nulový, stačí získané souřadnice x' a y' vynásobit koeficientem coef. V ostatních případech si zavedeme proměnné x" a y", které vyjadřují souřadnice bodu na snímací ploše po otočení o daný úhel β. V nákresu je znázorněna snímací plocha zezhora a otočení snímací plochy o úhel β, který je roven -45 stupňům. Souřadnice x" a y" je možné vypočítat na základě souřadnic x' a y' a úhlu β.

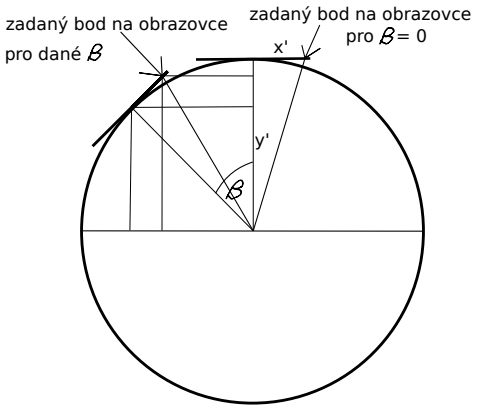
$$coef = \frac{H}{z'} \tag{2.13}$$

$$x'' = \cos(\beta) * x' + \sin(\beta) * y' \tag{2.14}$$

$$y'' = \cos(\beta) * y' - \sin(\beta) * x' \tag{2.15}$$

$$X = x'' * coef \tag{2.16}$$

$$Y = y'' * coef \tag{2.17}$$



Obrázek 2.2: Nákres snímací plochy zezhora a jednotkové kružnice, pomocí které je znázorněno, jak se mění souřadnice x' a y' v závislosti na úhlu β

Tyto výpočty jsou platné jen pro body na snímací ploše, které mají souřadnici z nižší, než má centrum kamery. Pokud má zadaný bod na snímací ploše stejnou, či vyšší souřadnici z, jako centrum kamery, souřadnice X a Y nelze spočítat. Pro stejnou souřadnici z bodu na snímací ploše by vycházely souřadnice X a Y nekonečné.

Tento způsob je možné rozšířit ještě o parametry radiálního zkreslení kamery. Na vstupu předchozího výpočtu zjištění reálných souřadnic tedy nebudou souřadnice x, y na snímku, ale souřadnice x' a y', vzniklé korekcí radiálního zkreslení kamery. Dalším vstupem budou i souřadnice x_c a y_c centra zkreslení. V neposlední řadě budou vstupem i koeficienty radiálního zkreslení k_1, k_2 atd. Pokud nechceme zanedbat tangenciální zkreslení, budou vstupem i koeficienty p_1, p_2 atd.

Dále ještě existuje tangentové zkreslení kamery, které působí kolmo na radiální zkreslení. Přesnost určení polohy i měření rychlosti tedy bude záviset na počtu koeficientů radiálního zkreslení a počtu koeficientů tangentciálního zkreslení. Obvykle se nepoužívá více než 6 koeficientů radiálního zkreslení a více než 3 koeficienty tangenciálního zkreslení.

Radiální zkreslení kamery se obvykle modeluje polynomiálním modelem zkreslení. Tento často využívaný model, popsaný například v [25], je známý jako Brown-Conrady model. Výpočet souřadnic x' a y', které vzniknou korekcí radiálního zkreslení, popisují následující vztahy.

$$t_x = [p_1(r^2 + 2(x - x_c)^2) + 2p_2(x - x_c)(y - y_c)](1 + p_3 r^2 + p_4 r^4 + ...) \quad (2.18)$$

$$x' = x + (x - x_c)(k_1 r^2 + k_2 r^4 ...) + t_x \quad (2.19)$$

$$t_y = [2p_1(x - x_c)(y - y_c) + p_2(r^2 + 2(y - y_c)^2)](1 + p_3r^2 + p_4r^4 + ...) \quad (2.20)$$

$$y' = y + (y - y_c)(k_1r^2 + k_2r^4...) + t_y \quad (2.21)$$

$$r = \sqrt{((x - x_c)^2 + (y - y_c)^2)} \quad (2.22)$$

Získané souřadnice x', y' jsou vhodným vstupem pro výpočet reálných souřadnic objektu výše popsaným způsobem. Po výpočtu souřadnic objektu na daném snímku, či aktuálním snímku stačí vypočítat vzdálenost mezi aktuálně získaným bodem se souřadnicemi X, Y, Z a předtím získaným bodem se souřadnicemi X_{prev}, Y_{prev}, Z_{prev}. Vzhledem k tomu, že oba body mají stejnou souřadnici z, je možné vypočítat jejich vzdálenost d pomocí vztahu:

$$d = \sqrt{((X - X_{prev})^2 + (y - Y_{prev})^2)} \quad (2.23)$$

Předpokládejme, že vzdálenost je udávána v milimetrech a poloha objektu se zaznamenává každý n-tý snímek. K výpočtu rychlosti je potřeba znát ještě snímkovou frekvenci videa, kterou je však možné získat pomocí volání metody na objektu, reprezentujícím daný videozáznam. Rychlost objektu v kilometrech za hodinu je možné snadno dopočítat pomocí vztahu:

$$speed = \frac{d * 3.6 * fps/frames}{1000 * n} \quad (2.24)$$

Tento způsob měření rychlosti dává dobré výsledky, pokud se zadá dostatečné množství koeficientů radiálního zkreslení. Obvykle stačí zadat 3 - 6 koeficientů radiálního zkreslení a „tangenciální zkreslení lze zanedbat“ [26]. Koeficienty radiálního zkreslení a centrum zkreslení kamery by měly být volitelnými parametry, protože některé kamery mohou mít velmi malé radiální zkreslení. Většinou ale nelze radiální zkreslení zcela zanedbat.

Výhodou tohoto způsobu měření rychlosti je, že je možné mu předat libovolné množství parametrů. Přesnost měření rychlosti tedy závisí jen na zadaných parametrech, nikoliv na tvaru čočky, či jiném faktoru. Pokud se zadá dostatečné množství parametrů včetně koeficientů radiálního zkreslení a souřadnic centra radiálního zkreslení, algoritmus je poměrně přesný.

Nevýhodou tohoto způsobu měření rychlosti je, že je potřeba předat větší množství parametrů kamery jako je ohnisková vzdálenost, úhel naklonění kamery vůči ploše, velikost snímače, či principální bod. Pokud jsou parametry kamery známy, není problém tento algoritmus použít. Pro spuštění algoritmu je nutné buď znát parametry kamery, nebo si je zjistit a následně je ručně zadat.

2.2.2.4 Automatická kalibrace kamery na základě sledování pohybu vozidel

Další metoda, jak měřit rychlost vozidel ve videu, spočívá v tom, že nalezneme tři úběžníky, na základě třetího úběžníku vypočteme vnější i vnitřní parametry kamery. Tento přístup, popsaný v pracích [27] a [28], je plně automatický a nevyžaduje žádný vstup od uživatele jako například některé z parametrů kamery, šířka jízdního pruhu, průměrná rychlost vozidla, či průměrné rozměry vozidla. Úběžníky jsou získány na základě informací o směru pohybu vozidel a předpokladu, že se vozidla pohybují rovně a po rovné ploše. Na základě získaných úběžníků je spočtena ohnisková vzdálenost i další parametry kamery.

Jedním z předpokladů je, že plocha země je víceméně rovná a neobsahuje žádné strmé stoupání, či klesání. Dalším z předpokladů je, že vozidla se pohybují jedním základním směrem. Na druhou stranu je tento přístup tolerantní k různým odlišnostem jako je občasné přejíždění do vedlejšího pruhu. V neposlední řadě se předpokládá, že principální bod kamery se nachází ve středu snímku, či velmi blízko něj.

Popis výpočtu úběžníků a ohniskové vzdálenosti, což je podstatná část této kapitoly, vychází ze zdroje [28]. Při výpočtu úběžníků se využívá kosočtvercový prostor, který vznikne namapováním z 2D prostoru do prostoru paralelních souřadnic. Prostor paralelních souřadnic vznikne tak, že místo osy y bude osa u_p a definují se dvě osy v_p a -v_p, které jsou rovnoběžné s osou u_p a jsou od ní stejně vzdálené. Toto mapování probíhá ve 2 krocích. Prvním krokem je vytvoření paralelních souřadnic, kde se místo osy y používá osa x_p. V určité vzdálenosti od osy x_p se nachází osa y_p. V dalším kroku se namapuje původní osa x_p na osu u_p a v určité vzdálenosti od osy u_p se nachází osa v_p.

Tato metoda mapuje nekonečný 2D prostor rozdělený na čtyři kvadranty do konečného prostoru, označovaného jako kosočtvercový prostor. Libovoný bod [x, y] je po první transformaci převeden na přímku a ve druhém kroku je zpět převeden na bod. 2D bod se souřadnicemi [x, y] je v prostoru paralelních souřadnic přímkou, která protíná osy v bodech x a y. V druhém kroku se tato přímka převede zpět na bod.

Tato transformace z nekonečného prostoru do prostoru paralelních souřadnic vymezených osami u_p, v_p a -v_p může vzniknout dvěmi transformacemi. Každá z těchto transformací převádí předchozí prostor do prostoru s paralelními souřadnicemi buď se dvěma osami, které vedou stejným směrem, nebo se dvěma osami, které vedou navzájem opačným směrem. Transformaci do prostoru paralelních souřadnic s osami, které vedou stejným směrem, označme jako S. Transformaci do prostoru paralelních souřadnic, kde osy vedou opačným směrem, označme jako T.

Tímto způsobem vzniknou čtyři různá mapování z nekonečného prostoru do prostoru paralelních souřadnic s osami u_p, v_p či -v_p, a to $S \circ S$, $S \circ T$, $T \circ S$ a $T \circ T$. První kvadrant nekonečného prostoru se převede do kosočtvercového prostoru na třetí kvadrant kosočtvercového prostoru pomocí transformace

$T \circ T$. Druhý kvadrant se převede na druhý kvadrant kosočtvercového prostoru pomocí transformace $S \circ T$, třetí kvadrant se převede na čtvrtý kvadrant kosočtvercového prostoru pomocí transformace a čtvrtý kvadrant se převede na převede na první kvadrant kosočtvercového prostoru pomocí transformace $S \circ S$.

Tímto způsobem vznikne z přímky v nekonečném 2D prostoru křivka v kosočtvercovém prostoru, která prochází odpovídajícími kvadranty. Pokud přímka prochází prvním, druhým a čtvrtým kvadrantem v nekonečném prostoru, vznikne z ní pomocí této transformace křivka, která prochází prvními třemi kvadranty v kosočtvercovém prostoru.

Má-li přímka homogenní souřadnice (a, b, c), vznikne z ní pomocí transformace do kosočtvercového prostoru křivka, která je definována čtyřmi body. Přímka s homogenními souřadnicemi (a, b, c) má tvar $ax + by = c = 0$. Tato křivka prochází osou u_p i osou, která je kolmá na osu u_p. Nejdříve si zadefinujeme pomocné proměnné α, β a γ, které jsou nenulovou hodnotou funkce sgn(x).

$$\alpha = sgn(ab), \beta = sgn(bc), \gamma = sgn(ac) \tag{2.25}$$

$$(a,b,c) -> [\frac{\alpha * a}{c + \gamma * a}, \frac{-\alpha * c}{c + \gamma * a}], [\frac{b}{c + \beta * b}, 0], [0, \frac{b}{a + \alpha * b}], [\frac{-\alpha * a}{c + \gamma * a}, \frac{\alpha * c}{c + \gamma * a}] \tag{2.26}$$

Křivka se 4 různými body z přímky vznikme, pokud parametry a, b i c jsou nenulové. Pokud přímka prochází dvěma kvadranty (jedná se například o osu x, osu y, nebo přímku, která prochází počátkem), jeden segment se zredukuje na bod.

Pomocí této transformace je možné převést bod ve 2D rovině na bod v kosočtvercovém prostoru. Vstupní bod má homogenní souřadnice $[p, q, 1]$, kde p je souřadnice x a q je souřadnice y v nekonečném 2D prostoru. Třetí homogenní souřadnice udává váhu daného bodu. Chceme-li převést homogenní souřadnice na souřadnice v kartézském souřadném systému, stačí vzít první dvě souřadnice a vydělit je třetím číslem, udávajícím váhu daného bodu. Tento bod se převede na bod v kosočtvercovém prostoru podle níže uvedeného vztahu.

$$[p, q, 1] -> [q, sgn(p)p + sgn(q)q - 1, p] \tag{2.27}$$

Kosočtvercový prostor se používá pro nalezení úběžníku. Úběžník je nalezen tak, že se najde bod, kterým prochází nejvíce přímek.

Úběžník ve směru pohybu vozidel je považován za první úběžník. Tento bod se detekuje tak, že se pomocí KLT (Kanade-Lucas-Tomaski) trackeru detekují zájmové body v předchozím snímku. Na základě detekovaných bodů z předchozího snímku se vytvoří krátké fragmenty trajektorií vozidel. Tyto

fragmenty trajektorií jsou prodlouženy na nekonečné přímky a zkoumá se, kudy všechny ty nekonečné přímky prochází. Bod, kterým tyto prodloužené fragmenty trajektorií prochází, je prvním úběžníkem.

Prodloužené fragmenty trajektorií vozidel jsou převedeny z 2D prostoru do kosočtvercového prostoru pomocí výše uvedených transformací. Úběžník se vybírá hlasováním těchto fragmentu v kosočtvercovém prostoru. Nejčastěji hlasovaný bod je prvním úběžníkem.

Celý tento proces není citlivý na různé odchylky od základního směru pohybu. Pomocí Houghovy transformace se převedou všechny přímky, vzniklé prodloužením fragmentů trajektorií a následným převedením do kosočtvercového prostoru, na body. Takto vznikne akumulační prostor, ze kterého se hlasováním vybírají kandidáti na první úběžník. Nejčastěji zvolený kandidát je prvním úběžníkem.

Houghova transformace zde umožňuje ignorovat různé odchylky od obvyklého pohybu vozidel jako je například změna pruhu. Pokud občas nějaký objekt změní pruh, nejčastěji zvolený bod (tj. první úběžník) zůstane stejný. Pro poměrně přesné nalezení úběžníku stačí sledovat pohyb 3 objektů. Pokud sledujeme pohyb většího množství objektů, první úběžník zůstane stejný.

Druhý úběžník odpovídá směru, který je kolmý na základní směr pohybu objektů. Druhý úběžník je bodem, kterým prochází všechny prodloužené hrany pohybujících se objektů. Tento bod se detekuje pomocí prodloužení detekovaných hran vozidel a následného hlasování v kosočtvercovém prostoru. Pro detekci hran pohybujících se objektů se používá model pozadí.

Model pozadí si pro každý pixel příchozího snímku o velikosti w*h ukládá konfidenční skóre výskytu orientované hrany. Dále si reprezentujeme obrázek pomocí pole H o velikost w*h*8, kde číslo 8 znamená počet přihrádek pro každý pixel, kam se ukládají gradienty. Do této reprezentace si pro každý pixel uložíme diskretizovanou reprezentaci gradientů, získanou z horizontálních a vertikálních rozdílových filtrů. Nejdříve si zavedeme model pozadí $B_1 = H_1$. Tento model je následně průběžně upraven pomocí vztahu:

$$B_t = \alpha * B_t - 1 + (1 - \alpha) * H_t \tag{2.28}$$

kde α je koeficient blízký 1.

Test pozadí se provádí pro každý pixel příchozího snímku, jehož velikost gradientu je vyšší, než daný práh (threshold) τ_1. Významné hrany jsou takové hrany, které po otestování pozadí mají rozdíl mezi velikostí a hodnotou gradientů v dané přihrádce větší než práh τ_2. Pokud hrana má podobný směr, jako směr pohybu vozidla, je z dalšího zpracování vyloučena. Hrany, které projdou testem, se následně hlasují v kosočtvercovém prostoru. Nejčastěji zvolený bod je druhým úběžníkem.

Třetí úběžník odpovídá směru, kolmému na plochu, po které se pohybují objekty. Vzhledem k tomu, že hran pro detekci třetího úběžníku není velké

množství, se třetí úběžník spočítá na základě prvních dvou úběžníků. Kromě toho se na základě prvních dvou úběžníků U, V a principálního bodu P (dle předpokladu střed snímku) dá vypočítat ohnisková vzdálenost pomocí vztahu:

$$f = \sqrt{-(U - P) \cdot (V\ - P)} \tag{2.29}$$

Pokud již známe ohniskovou vzdálenost f, je možné spočítat světové souřadnice třetího úběžníku, označený jako W. Vytvoříme se Tyto souřadnice označme jako W'. Nechť U' a V' jsou světové souřadnice prvních dvou úběžníků U a V, P' jsou světové souřadnice principálního bodu P. Homogenní souřadnice třetího úběžníku lze spočítat následujícím vztahem:

$$W' = (u'_x, u'_y, f) - (pp_x, pp_y, 0) \times (v'_x, v'_y, f) - (pp_x, pp_y, 0) \tag{2.30}$$

Dalším volitelným krokem je výpočet radiálního zkreslení kamery a upravených souřadnic po korekci radiálního zkreslení. Vzhledem k tomu, že mnoho kamer má nezanedbatelné radiální zkreslení, se radiální zkreslení počítá a souřadnice na snímku prochází korekcí. Radiální zkreslení je modelováno pomocí polynomů, ve kterých se vysktují sudé mocniny vzdálenosti r daného bodu se souřadnicemi x, y od centra zkreslení se souřadnicemi x_c, y_c. Upravené souřadnice x', y', které vzniknou korekcí radiálního zkreslení, se dají vypočítat pomocí následujících vztahů:

$$x' = x + (x - x_c) * (k_1 * r^2 + k_2 * r^4 ...) \tag{2.31}$$

$$y' = y + (y - y_c) * (k_1 * r^2 + k_2 * r^4 ...) \tag{2.32}$$

$$r = \sqrt{((x - x_c)^2 + (y - y_c)^2)} \tag{2.33}$$

Koeficienty radiálního zkreslení nejsou známy, ale lze je vypočítat tak, že proložíme trajektorii objektu přímkou a vypočítáme vzdálenosti reálné trajektorie od vzniklé přímky. Reálnou trajektorii reprezentujeme jako množinu bodů, které jsou počátkem, či koncem krátkého fragmentu trajektorie objektu. Každá reálná trajektorie je reprezentována sekvencí bodů $\tau = a_1, a_2, ... \in T$. Tato množina bodů se převede pomocí výše uvedených vzorců pro výpočet souřadnic pro korekci radiálního zkreslení a získáme tak množinu bodů $\tilde{\tau_k}$. Optimální parametry radiálního zkreslení K_m jsou nalezeny minimalizací součtu vzdáleností bodů reálné trajektorie od bodů na vzniklé přímce l_{τ_k}, kterou se prokládá trajektorie objektu.

$$K_m = arg \min_K \sum_{\tilde{\tau} \in T} \sum_{\tilde{a} \in \tilde{\tau}} (l_{\tilde{\tau_k}} \cdot \tilde{a})^2 \tag{2.34}$$

Po získání úběžníků se dají spočítat reálné souřadnice objektu ze souřadnic x, y na snímku pomocí matice vnitřních parametrů kamery K, rotační matice R a vektoru posunutí T. Projekční matice P transformuje reálné souřadnice na souřadnice na obrazovce a platí následující vztah:

$$\lambda_p[x_p, y_p, 1]' = P[x_w, y_w, z_w]' \tag{2.35}$$

Projekční matice může být rozložena na součin tří matic, a to matici vnitřních parametrů kamery K, rotační matici R a vektor posunu T. Projekční matice P transformuje bod s reálnými souřadnicemi $[x_w, y_w, z_w, 1]'$ na bod na obrazovce se souřadnicemi $[x_p, y_p, 1]'$, a tedy platí následující vztah:

$$\lambda_p[x_p, y_p, 1]' = P[x_w, y_w, z_w, 1]' \tag{2.36}$$

Vzhledem k tomu, že matici P je možné rozložit na součin matic K a R a vektoru T, platí:

$$\lambda_p[x_p, y_p, 1]' = K[RT][x_w, y_w, z_w, 1]' \tag{2.37}$$

Další část kapitoly, popisující výpočet rotační matice R a vektoru posunu t, pochází ze zdroje [29]. Jediným odvozeným paramtetrem, který se používá v matici K, je ohnisková vzdálenost f. Vzhledem k předpokladu, že pixely jsou čtvercové, je ohnisková vzdálenost ve směru osy x f_x i ve směru osy f_y stejná a rovná se hodnotě f. Kromě toho se také předpokládá nulová hodnota zkreslení s. Matice K tedy obsahuje ohniskovou vzdálenost f a souřadnice principálního bodu x_c, y_c.

$$K = \begin{pmatrix} f & 0 & x_c \\ 0 & f & y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{2.38}$$

Rotační matici R je možné snadno dopočítat pomocí matice, která obsahuje souřadnice tří úběžníků se souřadnicemi x_{p_1}, y_{p_1}, x_{p_2}, y_{p_2}, x_{p_3}, y_{p_3}. Tato matice se souřadnicemi tří úběžníků je rovna součinu rotační matice R a matici vnitřních parametrů K. Spočtenou matici R lze dosadit do níže uvedeného vztahu a pro nalezení vektoru T stačí vyřešit soustavu rovnic.

$$\begin{pmatrix} x_{p_1} & x_{p_2} & x_{p_3} \\ y_{p_1} & y_{p_2} & y_{p_3} \\ 1 & 1 & 1 \end{pmatrix} = K[RT] \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.39}$$

Vektor posunu T může být vypočítán také na základě informací o výšce kamery nad plochou. Může být také vypočten na základě známé vzdálenosti dvou bodů na snímku, či znalosti souřadnic bodu na snímku, který má reálné souřadnice 0, 0, 0.

Rychlost vozidla je možné vypočítat na základě vybraných bodů v aktuálním snímku a vybraných bodů objektu v předchozím snímku. Pomocí výše uvedených postupů se vypočítají reálné souřadnice bodů X a Y v aktuálním snímku. Poté vypočítáme jejich vzdálenost od bodů na předchozím snímku, jejichž reálné souřadnice byly spočítány, dle vzorce 2.23 a rychlost na základě vzdálenosti d dle vzorce 2.24.

Výhodou tohoto přístupu je, že nejsou potřeba na vstupu žádné hodnoty od uživatele. Parametry kamery je možné snadno dopočítat. Z parametrů kamery je možné pomocí projekční matice P vypočítat souřadnice objektu. Následně se dá rychlost vypočítat na základě získané vzdálenosti dle vzorce 2.24.

Nevýhodou tohoto přístupu je, že je zaměřený pouze na sledování pohybu vozidel po silnici. Pro rozšíření tohoto přístupu je potřeba definovat průměrnou šířku objektu, který se pohybuje po rovině. Pokud chceme například měřit rychlost běžcům, je potřeba zadefinovat, jak široký může být běžec. Tento algoritmus také není určení pro videozáznamy s chodci, kteří se mohou pohybovat různým směrem.

Další nevýhodou tohoto přístupu je, že potřebuje nějaké informace o tom, jakým směrem se objekty pohybují, předtím, než začne měřit jejich rychlost. V případě předání vstupního proudu dat tomuto algoritmu je potřeba nejdříve chvíli sledovat pohyb vozidel před měřením jejich rychlosti. Tento přístup je hůře použitelný pro krátké záznamy, ve kterých se pohybuje jen několik objektů. Pro dlouhé videozáznamy a proudy dat z dopravních kamer tato nevýhoda nepředstavuje závažný problém.

2.2.2.5 Další existující algoritmy pro měření rychlosti objektů ve videu

Existují i další algoritmy pro měření rychlosti objektů ve videozáznamu. Některé přístupy využívají strojové učení a jsou založeny na neuronových sítích. V této kapitole je také stručně popsán přístup, který vypočítává reálnou vzdálenost tak, že vynásobí získanou vzdálenost v pixelech konstantou. Násobení vzdálenosti v pixelech konstantou je statické škálování. Další přístup, který je zde stručně popsán, detekuje poznávací značky vozidel a se znalostí rozměrů poznávací značky vypočítá rychlost. Poslední popsaný přístup je založen na algoritmu, popsaném v kapitole 2.2.2.1, ale místo vzdálenosti mezi dvěma úsečkami se používá vzdálenost na snímku.

Existuje ještě několik dalších algoritmů pro měření rychlosti objektů ve videu, které predikují rychlost objektů strojovým učením a využívají neuronové sítě. Predikce rychlosti pomocí strojového učení vyžaduje nějakou trénovací sadu. Jeden z těchto přístupů využívá i3D model, který pro odhad rychlosti

využívá trénovací množinu s informacemi o čase a reálných 3D souřadnicích. Přesnost tohoto algoritmu závisí na trénovací množině a její velikosti. Je zapotřebí velká trénovací množina, aby tento algoritmus byl přesný.

Další přístup počítá rychlost vozidla tak, že vypočítá vzdálenost v pixelech zájmového bodu na aktuálním snímku od bodu, který byl zájmovým bodem v předchozím snímku a vynásobí tuto vzdálenost konstantou a získá tak vzdálenost v metrech. Následně se rychlost vypočte pomocí vzorce 2.24. Tento algoritmus dává poměrně dobré výsledky jen pro určité videozáznamy a pro jiné videozáznamy může měřit rychlost velmi nepřesně. Je použitelný jen v případě, že konstanta bude vstupním parametrem.

V neposlední řadě také existuje algoritmus, který detekuje číslice a písmena na SPZ vozidel a následně na základě známých rozměrů vypočítá rychlost. Tento přístup je podrobněji popsán v práci [32]. Tento přístup nejdříve detekuje pohyb vozidla. Poté detekuje poznávací značku vozidla, předpovídá pohyb pomocí SIFTů a sleduje její zájmové body pomocí KLT trackeru. Po získání zájmových bodů a trajektorií pohybu se na základě známých rozměrů jako například velikost poznávací značky vypočte rychlost vozidla. Algoritmus dává pro záznamy z pouličních kamer s automobily dobré výsledky a odchylka nebývá vyšší než 3 km/h. Jeho přesnost je dána i díky přesně definovaným rozměrům poznávací značky. Na druhou stranu tento algoritmus není dobře použitelný pro jiné druhy objektů jako například chodci, či běžci. Pro měření rychlosti jiných objektů je potřeba si rozmyslet, jaké rysy se známými rozměry lze sledovat. U běžců je tak možné sledovat například výšku a šířku, která se ale může výrazněji lišit.

Další přístup počítá rychlost tak, že vypočítá rychlosti za několik krátkých časových úseků a výslednou rychlost vypočítá jako průměr dílčích rychlostí. Nejdříve se nějakým způsobem vypočítá počet pixelů na metr. Počet pixelů na metr se dá vypočítat například na základě dvou úseček a jejich dané vzdálenosti. Následně se vypočte, o kolik pixelů se liší pozice středového bodu pohybujícího se objektu na aktuálním snímků oproti pozici o několik snímků dříve. Vypočtená vzdálenost v pixelech se převede na metry. Vzdálenost v metrech se vydělí délkou časového úseku v sekundách a získáme tak dílčí rychlost. Jednotlivé dílčí rychlosti se zprůměrují a získáme tak výslednou rychlost. Tento algoritmus je dobře použitelný pro záznamy, na kterých jsou vidět pohybující se objekty zboku a pohybují se zprava doleva, či opačným směrem. Pro záznamy z kamery, která je nakloněná na vozovku pod nějakým úhlem, není tento algoritmus vhodný.

2.2.2.6 Shrnutí algoritmů pro měření rychlosti objektů ve videu

V tabulce níže jsou srovnány algoritmy pro měření rychlosti, popsané v předchozích kapitolách. Zaznamenávání návštěvy bodů, či zaznamenávání návštěvy úseček jsou z hlediska přesnosti i rychlosti zpracování snímků vhodnými přístupy, ale mají nevýhodu v tom, že je potřeba mít změřenou vzdálenost

Tabulka 2.2: Srovnání existujících algoritmů z hlediska potřebných vstupů, přesnosti výsledků a efektivity, či rychlosti zpracování snímků

Algoritmus Kritérium	Potřebné vstupy	Přesnost výsledků	Efektivita (rychlost zpracování, či potřebné kroky)
Zaznamenávání návštěvy úseček	koncové body úseček a jejich reálná vzdálenost	chyba nejvýše okolo 1,5 % (stejně jako předchozí způsob)	pro každý snímek - detekce objektů a pro každý obdélník, do kterého lze vepsat konturu objektu, se testuje, zda obsahuje první, či druhou úsečku
Měření rychlosti pomocí obdélníku na 3D ploše	souřadnice bodů lichoběžníku na snímku a reálné velikosti stran	závisí na radiálním zkreslení kamery, udává se RMSE (Root mean square error) okolo 8,6, detekována přibližně 81,5 % vozidel	lineární transformace každého snímku a detekce objektů v každém snímku
Měření rychlosti pomocí výpočtu reálných souřadnic	parametry kamery	záleží na zadaných parametrech a koeficientech radiálního zkreslení	detekce objektů a výpočet reálných souřadnic pro každý snímek
Automatická kalibrace kamery	nejsou potřeba vstupy od uživatele, je potřeba z pohybu vozidel vypočítat úběžníky a ohniskovou vzdálenost	chyba v průměru mezi 1 a 1,5 %, maximální chyba 6 %	okolo 5 snímků za sekundu

mezi zadanými imaginárními body, či úsečkami. Automatická kalibrace kamery dává poměrně přesné výsledky, ale snímky se zpracovávají poměrně dlouho a je potřeba pozorovat několik objektů k nalezení úběžníků. U dalších algoritmů závisí přesnost na vstupech a radiálním zkreslení kamery, se kterým nepočítají.

2.2.3 Implementace algoritmů pro měření rychlosti

V této kapitole jsou popsány implementace algoritmů pro měření rychlosti objektů ve videozáznamu. První dvě implementace měří rychlost objektu tak, že nejdříve zaznamenají, na jakém snímku objekt navštívil první úsečku na snímku a poté zaznamenají, na jakém snímku objekt navštívil druhou úsečku. Vzhledem k tomu, že reálná vzdálenost úseček v metrech je daná, stačí vydělit vzdálenost úseček časem pohybu objektu mezi dvěma úsečkami v sekundách. Třetí implementace automaticky kalibruje kameru a získává tři úběžníky a ohniskovou vzdálenost.

2.2.3.1 Implementace měření rychlosti zaznamenáváním průjezdu úsečkou - VehicleSpeedTracker

Algoritmus, který umožňuje měřit rychlost tak, že měří čas mezi dvěma úsečkami, které jsou od sebe v určité reálné vzdálenosti, je implementován v jazyce C++. Tento algoritmus je popsaný v kapitole 2.2.2.1. Jedná se o implementaci v [35], která požaduje buď napojení na kameru, která zaznamenává vozidla, jedoucí na dvouproudé vozovce dvěma opačnými směry, nebo složku s videozáznamy z takové kamery. Tato implementace zpracovává proud snímků z kamery, či videozáznam. Implementace je podrobněji popsána v C.1.

Všechny zdrojové kódy jsou napsány v jazyce C++. Zdrojové kódy jsou k dispozici v adresáři src/Wintel64. Ke spuštění aplikace není třeba předávat žádný parametr v příkazové řádce. Parametry jako například cesta k videím ke zpracování, či IP kameře, či vzdálenost mezi úsečkami se načítají z konfiguračního souboru VST.cfg ve adresáři bin.

Implementace detekuje jedoucí vozidla a zaznamenává, které vozidlo projíždí jednou ze dvou daných úseček na snímku. Pokud vozidlo projelo první úsečkou na snímku, zaznamená se snímek průjezdu pro dané vozidlo. Pokud dané vozidlo projede i druhou úsečkou, zaznamená se snímek průjezdu druhou úsečkou a následně se spočítá počet snímků mezi průjezdem první úsečkou a průjezdem druhou úsečkou. Na základě této informace a dané vzdálenosti mezi úsečkami se snadno vypočte rychlost vozidla.

Zde jsou definovány dva základní směry, kterými se může pohybovat vozidlo ve videozáznamu. Vozidlo se může pohybovat buď zleva doprava, nebo zprava doleva. Pokud se vozidlo pohybuje zprava doleva, zaznamená se nejdříve průjezd úsečkou napravo a o několik snímků později se zaznamená průjezd úsečkou nalevo. V případě, že se vozidlo pohybuje opačným směrem, zaznamená se nejdříve průjezd úsečkou nalevo a o několik snímků později se zaznamená průjezd úsečkou napravo. Vždy se ale spočítá rozdíl počtu snímků mezi průjezdem první úsečkou a průjezdem druhou úsečkou.

Zde se detekují objekty tak, že se najde popředí, z kterého se detekují pohybující se objekty. Pohyb detekovaných objektů je reprezentován třídou

`VehicleDynamics`, která v sobě obsahuje pole snímků. Tyto objekty v sobě obsahují pole se všemi snímky, na kterých se dané pohybující se vozidlo objevilo.

Výhodou této implementace je, že parametry jako například vzdálenost úseček od sebe není pevně daná v kódu a je možné ji nastavit pomocí konfigurace. Nevýhodou této implementace je její paměťová náročnost pro záznamy s větším množstvím vozidel na delším úseku, ve kterém se měří rychlost vozidel.

2.2.3.2 Implementace měření rychlosti zaznamenáváním průjezdu úsečkou - Speed detector

Speed Detector je implementace algoritmu, popsaného v kapitole 2.2.2.1, v jazyce Java. Jedná se o implementaci v [36], která se skládá z aplikace v Javě a webové aplikace. Samotné zpracování videozáznamů a výpočet rychlosti umožňuje aplikace v Javě, které obsahuje GUI (Graphical User Interface). Implementace je podrobněji popsána v příloze C.2.

Hlavní třídou, která obsahuje metodu main pro spuštění GUI (Graphical User Interface), je třída Application. V metodě main se vytvoří grafické uživatelské rozhraní pro přehrání videa, zjištění rychlosti vozidel a jejich zobrazení a následně se spustí voláním metody init(). Třída GUI zajišťuje zobrazení okénka, ve kterém je možné si přehrát vstupní video, detekování vozidel pomocí spočítání rozdílu aktuálního snímku oproti předchozímu snímku i výpočet rychlosti vozidel. Rychlost vozidel se vypočítává tak, že se definují dvě úsečky v pevně dané reálné vzdálenosti od sebe a zjistí se počet snímků mezi průjezdem první úsečkou a průjezdem druhou úsečkou.

V kódu je napevno dána vzdálenost mezi dvěma úsečkami, která je nastavena na 6 metrů. Koncové body obou úseček vybere uživatel kliknutím na video v příslušném místě. Objekty jsou zde detekovány tak, že se ze snímků vybere popředí a v popředí se najdou kontury pohybujících se objektů. Tato implementace také počítá osobní automobily, dodávky a nákladní vozy a počítá průměrnou rychlost pro každou z uvedených kategorií vozidel (osobní automobily, dodávky a nákladní vozy).

Kromě samotného projektu v Javě obsahuje tato implementace i webovou aplikaci, napsanou v jazyce PHP, která zobrazuje seznam vozidel, která jela vyšší rychlostí než 50 km/h. Rychlostní limit 50 km/h je napevno dán v kódu. Aplikace má sloužit pro posílání pokut řidičům, kteří překročili maximální povolenou rychlost, ale zatím ještě není funkčí odesílání pokut. Stránka *loadHighSpeed.php* slouží k načtení vozidel z databáze, která překročila maximální povolenou rychlost. V tomto souboru jsou také napevno definovány přihlašovací údaje k databázi.

Implementace obsahuje dva adresáře, z nichž jeden obsahuje webovou aplikaci, která obsahuje skripty napsané v jazyce PHP a skripty napsané v JavaScriptu a kaskádové styly v CSS. Druhý adresář obsahuje projekt se zdrojovými kódy v Javě SE (Standard Edition), které se nachází v adresáři src. Webová aplikace i aplikace v Javě se připojují k MySQL databázi, kam

se ukládají informace o rychle jedoucích vozidlech. Videozáznam se nahrává přímo přes GUI (Graphical User Interface), koncové body úseček volí uživatel přímo v uživatelském rozhraní klikáním na snímek a další parametry jako je například reálná vzdálenost mezi úsečkami, jsou napevno definované ve zdrojovém kódu.

Nevýhodou této implementace je, že vzdálenost mezi dvěma úsečkami je napevno daná v kódu. Vzhledem k tomu, že tato vzdálenost je poměrně malá, je nutné pro použití videozáznamů z dálnic a okresních silnic mimo obec, je potřeba upravit kód. Bez úpravy kódu není tato implementace použitelná pro záznamy z dálnic, či okresních silnic mimo obec a rychlost vozidla může být spočítána velmi nepřesně.

2.2.3.3 Implementace automatické kalibrace kamery

Automatická kalibrace kamery a následné měření rychlosti, popsané v kapitole 2.2.2.4, jsou implementovány v MatLabu. Tato implementace se nachází v ZIP archivu v [37] a je založena na [38] a [39]. Implementace je podrobněji popsána v příloze C.3.

Třída `DiamondSpace` v adresáři @DiamondSpace obsahuje wrapper C++ funkcí ze souboru, který se nachází v podadresáři private. Funkce ze zdrojového kódu v C++ se volají v MEX (MATLAB executable) wrapperu, který je potom použit v MatLabu. Pro spuštění potřebuje Computer Vision System Toolbox, Image Processing Toolbox, NNet toobox a Piotr's Toolbox verze 3.40 a vyšší.

Všechny zdrojové kódy mimo zdrojový kód v adresáři @DiamondSpace/private, který je napsaný v C++ a slouží k převodu ze 2D prostoru do kosočtvercového prostoru, jsou napsány v MatLabu. Ve složce @DiamondSpace/private se nachází wrapper, který volá funkce z C++ souboru. Zdrojový kód, který se přímo volá, se nachází v hlavním adresáři a další zdrojové kódy se nachází ve všech podadresářích hlavního adresáře. Implementace se spouští pomocí volání funkce calibRoad ze spouštěcího souboru calibRoad.m. Nejsou povinné žádné parametry a je možné předat parametry FocalRange, HorizonRange, StartAt, FinishAt, FrameStep, ROI a pp.

Jednou z implementovaných tříd je třída `RoadCalibration`, která zajišťuje výpočet úběžníků a detekci zájmových bodů. Implementace dále obsahuje třídu, která reprezentuje kosočtvercový prostor. V další složce se nachází třída, která se používá při výpočtu gradientů, používaných při výpočtu druhého úběžníku. V balíčku geometry se nachází soubory s implementovanými pokročilejšími matematickými funkcemi jako například normalizace homogenních souřadnic bodů.

Kapitola 3

Návrh

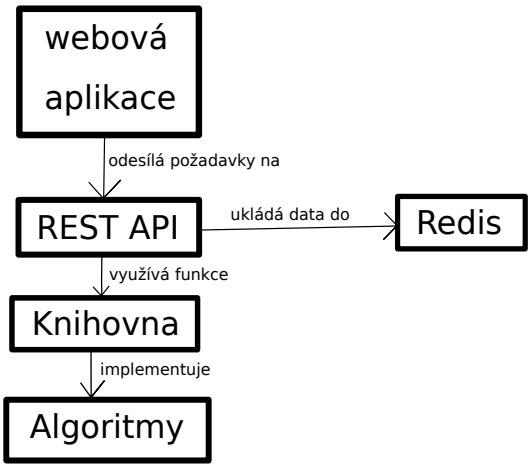
V této kapitole je podrobně popsán návrh aplikace a její architektura. Jednou z komponent aplikace je knihovna s funkcemi pro zpracování videozáznamu, získání videozáznamu, metadat, či anotací. Další komponentou je REST (Representational state transfer) API (Application programming interface), které využívá funkce knihovny. Dále je implementována webová aplikace, která využívá REST API.

Základním jádrem celé aplikace je knihovna v Pythonu, která obsahuje funkce pro různé účely jako je zpracování videa, výpočet rychlosti objektů, či generování anotací. V knihovně jsou implementovány algoritmy pro měření rychlosti objektů ve videu. Dále bude implementováno REST API, které využívá funkce knihovny. REST API využívá key-value databázi Redis (viz. 3.2.4.3) pro mapování cesty k videu, anotacím, či metadatům na identifikátor. Prezentační vrstvou je webová aplikace, která využívá REST API, volající funkce knihovny.

3.1 Návrh knihovny v Pythonu

V této kapitole je popsán návrh knihovny a je zde zdůvodněna volba algoritmu pro měření rychlosti objektů a způsobu přidání anotací. Zde je popsán algoritmus pro měření rychlosti, který pro každý pohybující se objekt počítá jeho reálné souřadnice vůči kameře na aktuálním snímku a následně vypočte vzdálenost aktuálního bodu od předchozího bodu. Na základě získané vzdálenosti vypočte rychlost. Dále se zde popisuje, že pro přidání anotací se využívají MediaFragments a WebVTT. MediaFragments se využívají pro přidávání informací o čase a prostoru a WebVTT slouží pro přidání dalších metadat jako je rychlost objektu ve videu.

Knihovna v Pythonu obsahuje funkce pro zpracování videa, výpočet rychlosti objektů a generování anotací. Obsahuje hlavní třídu VideoProcessor, která slouží pro zpracování videa, výpočet rychlosti objektů a generování anotací. Dále obsahuje třídu Point, která reprezentuje bod na snímku a vypočítává



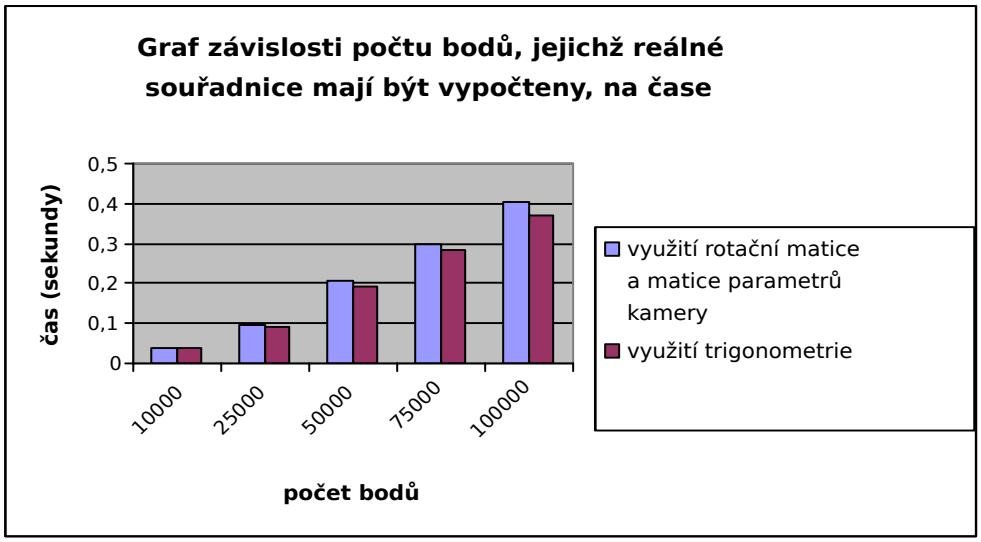
Obrázek 3.1: Architektura aplikace pro měření rychlosti objektů ve videu

jeho souřadnice v trojrozměrném souřadném systému, kde centrum kamery leží v počátku. V neposlední řadě obsahuje třídu ObjectWithSpeed, která obsahuje informace o tom, v jakém časovém úseku se daný objekt pohyboval danou rychlostí. Kromě toho slouží pro získání řetězce, který má být zapsán do souboru s metadaty ve formátu VTT. Obsahuje ještě třídu Rect, která se využívá pro vyznačení objektu ve videozáznamu na základě informace o poloze z Media Fragments URI.

Pro generování anotací se používají MediaFragments a WebVTT, jejichž způsob využití je popsán v kapitole 2.1.2.2. Pro měření rychlosti objektů se používá algoritmus popsaný v kapitole 2.2.2.3, který pro výpočet rychlosti vypočítává reálné souřadnice vůči kameře ze souřadnic bodu na snímku.

Pozici objektu vůči kameře lze na základě získaných souřadnic spočítat různými způsoby. Často se reálné souřadnice počítají pomocí matice vnitřních parametrů kamery, rotační matice kamery a vektoru posunu. Kameru je možné rotovat okolo libovolné osy a rotace kolem os je možné kombinovat. Tento způsob výpočtu reálných souřadnic je časově a implementačně trochu náročnější a bere v úvahu rotace okolo libovolné osy i jejich kombinace. Pokud chceme kombinovat rotace okolo všech tří os x, y a z, je vhodné počítat reálné souřadnice tímto způsobem.

Pro snížení časové náročnosti výpočtu souřadnic je obvykle dostačující uvažovat jen rotaci kolem osy x - naklonění kamery na plochu a rotaci kolem osy z. Existují tedy i další způsoby, jak lze získat reálné souřadnice na základě

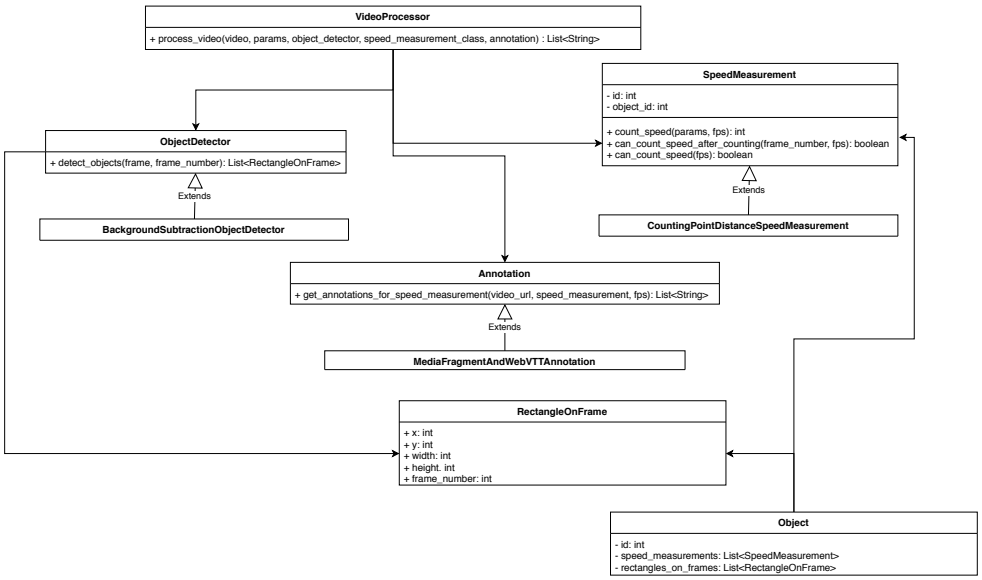


Obrázek 3.2: Graf závislosti počtu bodů, jejichž reálné souřadnice se mají vypočítat, na čase pro způsob výpočtu s využitím trigonometrie a způsob výpočtu pomocí matice kamery a rotační matice

souřadnic pixelu na snímku.

V grafu je naznačeno, jak závisí počet bodů, jejichž reálné souřadnice mají být vypočteny z jejich souřadnic na snímků, na čase. Výsledné časy pro oba způsoby výpočtu a počty bodů od 10000 do 100000 reálných souřadnic byly získány s využitím funkce `time()`. Doby běhu pro oba způsoby výpočtu souřadnice byly naměřeny na notebooku, popsaném v kapitole 5. Z grafu je možné vysledovat, že způsob výpočtu souřadnic s využitím trigonometrie bez matic je zhruba o desetinu rychlejší než způsob výpočtu s využitím matice kamery a rotační matice. Z tohoto důvodu se pro výpočet souřadnic nepoužívají rotační matice a matice kamery a souřadnice se vypočítávají pomocí trigonometrických vzorců popsaných v kapitole 2.2.2.3.

Hlavním důvodem, proč byl vybrán tento algoritmus je, že se při výpočtu rychlosti objektu provádí poměrně nízký počet výpočetních operací a není nutné si pamatovat, jakým směrem se objekty pohybují. Algoritmus umožňuje měřit rychlost objektů bez ohledu na směr jejich pohybu. Pro výpočet rychlosti objektu si stačí pamatovat, na jaké pozici byl objekt před 0,5 sekundou, či jiným časovým intervalem, a pomocí parametrů spočítat, kde se nachází daný bod na snímací ploše.



Obrázek 3.3: Diagram tříd pro knihovnu s navrženými třídami a základními metodami

3.1.1 Návrh tříd a metod

V této kapitole je popsán návrh tříd a jejich metod pro knihovnu, která zpracovává videozáznam, měří rychlosti objektů ve videu a generuje anotace. Třídy a jejich metody jsou znázorněny na diagramu.

V diagramu je znázorněna třída `VideoProcessor`, která obsahuje metodu `process_video`. Metoda `process_video` zpracovává video, detekuje objekty daným způsobem, měří rychlost detekovaných objektů daným algoritmem a generuje anotace daným způsobem. Metoda po každém změření rychlosti vrací seznam již vygenerovaných anotací. Způsob detekce objektů je dán instancí libovolného potomka třídy `ObjectDetector`, algoritmus měření rychlosti je dán třídou, která je potomkem třídy `SpeedMeasurement` a způsob generování anotací určuje daná instance libovolného potomka třídy `Annotation`.

Předkem všech tříd, které detekují objekty určitým způsobem, je třída `ObjectDetector`. Tato třída představuje nějaký způsob detekce objektů v daném snímku. Obsahuje metodu `detect_objects`, která detekuje objekty nějakým algoritmem a vrátí seznam obdélníků, do kterých je vepsána kontura nějakého pohybujícího se objektu na daném snímku. Jejími potomky jsou třídy, které detekují objekty určitým způsobem jako například ořezáváním pozadí a získáváním kontur z popředí.

V diagramu je dále znázorněna třída `SpeedMeasurement`, která je předkem všech tříd, reprezentujících nějaký algoritmus měření rychlosti. Obsahuje v sobě identifikátor měření rychlosti a identifikátor objektu, jehož rychlost je změřena.

Obsahuje metodu `count_speed`, která vypočítává rychlost objektu. Metody `can_count_speed_after_counting` a `can_count_speed` slouží ke zjištění, zda rychlost daného objektů může být změřena a zda je pro výpočet rychlost objektu uloženo dostatek informací. Jejími potomky jsou třídy, které vypočítávají rychlost objektu určitým algoritmem (např. algoritmem popsaným v kapitole 2.2.2.3) na základě uložených informací.

V neposlední řadě je v diagramu naznačena třída `Annotation`, která je předkem tříd, generujících anotace určitým způsobem. Třída obsahuje metodu `get_annotations_for_speed_measurement`, která generuje anotace a vrací seznam anotací pro dané měření rychlosti. Jejími potomky jsou třídy, které generují anotace určitým způsobem jako například využití Media Fragments a WebVTT.

V diagramu je dále znázorněna třída `RectangleOnFrame`, která reprezentuje obdélník na snímku, do kterého je možné vepsat konturu nějakého pohybujícího se objektu. Tato třída se používá zejména pro uložení pozice a velikosti pohybujících se objektů na snímcích. Instance této třídy se také používají při výpočtu rychlosti objektu.

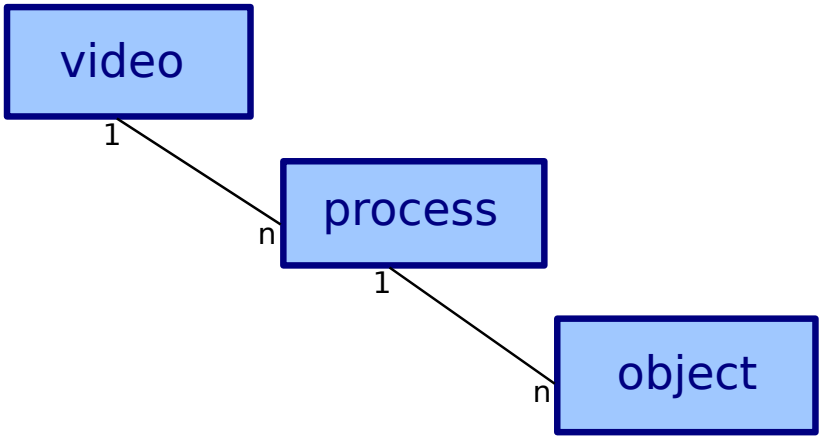
Diagram obsahuje i třídu `Object`, která představuje nějaký pohybující se objekt. Obsahuje identifikátor, seznam provedených měření jeho rychlosti a seznam obdélníků na snímku s pozicemi a velikostmi v pixelech. Tato třída se využívá pro měření rychlosti objektů i generování anotací.

3.2 Návrh REST služby

REST API provádí CRUD (Create, Read, Update, Delete) operace s videozáznamy a procesy, které zpracovávají videozáznam a měří rychlost objektů, a získává informace o objektech ve videozáznamu. Existují zde tři různé zdroje, a to videa, procesy zpracování a objekty, detekované v rámci procesu zpracování. K videozáznamům je možné přistoupit pomocí URI s cestou `/video`, k procesům zpracování videozáznamu s identifikátorem 15 lze přistoupit pomocí URI s cestou `/video/15/processes` a objekty v rámci procesu s identifikátorem 2 lze získat pomocí URI cestou `/video/15/processes/2/objects`.

3.2.1 Identifikace zdrojů

REST API provádí operace nad třemi zdroji, a to video, proces a objekt. Zdroj video reprezentuje videozáznam, ve kterém se mohou objevit pohybující se objekty jako například automobily, či běžci. Zdroj proces reprezentuje proces zpracování daného videa, v rámci něhož jsou detekovány objekty a změřeny jejich rychlosti a vygenerovány anotace. Zdroj objekt představuje detekovaný objekt ve videu v rámci daného procesu. Video může obsahovat n procesů, ale jeden proces se vždy týká jednoho videa. V rámci jednoho procesu se vytvoří n objektů a jeden objekt se vytvoří vždy v rámci jednoho procesu zpracování. V datovém modelu jsou naznačeny zdroje, vazby mezi nimi a jejich kardinality.



Obrázek 3.4: Datový model se zdroji pro REST API

Každý zdroj má svou vlastní URI adresu, která se řídí datovým modelem na obrázku. Níže je uveden seznam zdrojů a ke každému zdroji je uvedeno, co reprezentuje a jakou má URI adresu.

video
: videozáznam, na kterém se objevují pohybující se objekty jako například automobily, či běžci, obsahuje v sobě procesy jeho zpracování
`/video`

proces
: proces zpracování videa s daným identifikátorem, v rámci kterého se detekují objekty, měří se jejich rychlosti a generují se anotace na základě pozic detekovaných objektů v čase a jejich naměřených rychlostí, obsahuje objekty, které byly během něj detekovány
`/video/{video-id}/process`

objekt
: pohybující se objekt, který je detekován v rámci procesu zpracování videa s daným identifikátorem, obsahuje anotace s pozicemi v čase a naměřenými rychlostmi
`/video/{video-id}/process/{process-id}/object`

3.2.2 Reprezentace zdrojů

V této kapitole je popsána reprezentace videa, procesu zpracování videa i detekovaného objektu v REST API. Reprezentace videa obsahuje cestu k videu a metadata jako je délka, rozměry snímků, snímková frekvence, MIME (Multipurpose Internet Mail Extensions) typ a typ souboru a odkazy na procesy zpracování daného videa. Reprezentace procesu zpracování videa obsahuje

odkazy objekty, které byly detekovány, a parametry, se kterými byl daný proces spuštěn. Reprezentace objektu obsahuje seznam URI pro Media Fragments, ve kterých jsou obsaženy informace o časech, kdy se objekt objevil ve videu, a souřadnice a velikost na snímku. Dále obsahuje pole webVTT fragmentů s vymezením časového úseku a naměřenou rychlostí pro každé měření rychlosti. Každý WebVTT fragment v sobě obsahuje identifikátor, identifikátor objektu, čas od a do a naměřenou rychlost.

Video je reprezentováno pomocí objektu, který obsahuje cestu k nahranému videu, pole odkazů na procesy jeho zpracování a objekt s metadaty. Odkazy na procesy zpracování videa jsou reprezentovány řetězci, které obsahují absolutní URI. Cesta k videu je reprezentována řetězcem, který obsahuje absolutní URI. Objekt s metadaty obsahuje vlastnosti reprezentující délku videa, snímkovou frekvenci, výšku a šířku snímků v pixelech, MIME typ a typ souboru. Délka videa je reprezentována řetězcem ve formátu minuty:sekundy, či formátu minuty:sekundy.setiny, MIME typ a typ souboru jsou řetězce a snímková frekvence, počet snímků a rozměry snímků jsou reprezentovány celými čísly.

Mějme video, které se nahrálo na server `https://www.mujserver.cz` do složky `files/videos` pod názvem `video1.mp4`. K videu je tedy možné se dostat pomocí URI `https://www.mujserver.cz/files/videos/video1.mp4`. Toto video má identifikátor 2. Jedná se o video ve formátu MP4 a jeho MIME typ je tedy `video/mp4`. Pro zpracování videa byly spuštěny procesy s identifikátory 5 a 12. Video je dlouhé 28,5 sekundy, jeho snímková frekvence je 30 snímků za vteřinu a rozměry snímků jsou 1280 x 720. Níže je uvedena ukázka reprezentace tohoto videa ve formátu JSON.

Listing 3.1: Ukázka reprezentace videa ve formátu JSON

```
{
        "id": 1,
        "path": "https://www.mujserver.cz/videos/video1.
            mp4",
        "metadata": {
                "mimeType": "video/mp4",
                "type": MP4,
                "duration": "00:28.50",
                "frameRate": 30,
                "frameWidth": 1280,
                "frameHeight": 720
        },
        "processes": [
                "https://www.mujserver.cz/video/2/process
                    /5",
                "https://www.mujserver.cz/video/2/process
                    /12"
        ]
}
```

Proces je reprezentován pomocí objektu, který obsahuje mapu parametrů, se kterými byl spuštěn, a pole odkazů na objekty, které byly v rámci něj dete-

kovány. Parametry jsou zde reprezentovány jako mapa, kde název parametru je klíč a jeho hodnota je hodnota. Každý odkaz na objekt je reprezentován řetězcem s absolutní URI, na které se nachází daný objekt. Reprezentace procesu obsahuje ještě pole s Media Fragments URI pro každé měření rychlosti nějakého pohybujícího se objektu.

Mějme proces s identifikátorem 5, který zpracovával video s identifikátorem 2. Byl spuštěn s parametry f = 10, h = 5000, len_width = 12,6, len_height = 8,4, alpha = 20, width = 1280, height = 720, p_x = 640 a p_y = 360. Parametr f značí ohniskovou vzdálenost, v parametrech len_width a len_height jsou rozměry snímače, parametr alpha značí úhel naklonění kamery nad plochou (rotace kolem osy X), v parametrech width a height jsou rozměry snímků v pixelech a p_x a p_y jsou souřadnice principálního bodu. Během procesu byly detekovány objekty s identifikátory 25, 26 a 27 a byla provedena měření rychlosti s identifikátory 36, 37, 38 a 39. Níže je uvedena ukázka reprezentace tohoto procesu zpracování videa ve formátu JSON.

Listing 3.2: Ukázka reprezentace procesu, v rámci kterého je zpracováno video, ve formátu JSON

```
{
        "id": 5,
        "params": {
                "f": 10,
                "h": 5000,
                "len_width": 12.6,
                "len_height": 8.4,
                "alpha": 20,
                "beta": 5,
                "width": 1280,
                "height": 720,
                "p_x": 640,
                "p_y": 360
        },
        "objects": [
                "https://www.mujserver.cz/video/2/process
                    /5/object/25",
                "https://www.mujserver.cz/video/2/process
                    /5/object/26",
                "https://www.mujserver.cz/video/2/process
                    /5/object/27"
        ],
        "mediaFragmentsURIs": [
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=36",
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=37",
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=38",
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=39"
        ]
```

```
}
```

Pohybující se objekt, který je detekován v rámci procesu zpracování videa, je reprezentován objektem, který obsahuje pole řetězců pole řetězců s Media Fragments URI. Každá Media Fragments URI je reprezentována řetězcem s URI videa a fragmentem id s identifikátorem měření rychlosti.

Mějme objekt, jehož rychlost byla změřena dvakrát. Identifikátory měření rychlosti objektu jsou 22 a 23. Níže se nachází reprezentace tohoto objektu ve formátu JSON.

Listing 3.3: Ukázka reprezentace objektu, který byl detekován v rámci procesu zpracování videa a byla mu dvakrát změřena rychlost

```
{
        "id": 25,
        "mediaFragmentsURIs": [
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=22",
                "https://www.mujserver.cz/videos/video1.
                    mp4#id=23",
        ]
}
```

3.2.3 Operace, odpovědi a stavové kódy

V této kapitole jsou popsány operace nad zdroji, odpovědi a stavové kódy. Ke každé operaci je popsáno, co dělá. U každé operace je uveden seznam možných stavových kódu a ke každému stavovému kódu je uvedeno, kdy je vrácen. U některých operací je také uveden příklad těla možné odpovědi.

GET `/video` - získá seznam všech nahraných videí

Metoda GET

Stavové kódy

- **200 - OK** - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a v hlavičce Accept je podporovaný MIME typ.
- **401 - Unauthorized** - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic
- **403 - Forbidden** - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje
- **406 - Not Acceptable** - pokud je vyžadován hlavičkou Accept jiný formát odpovědi než JSON

Tělo odpovědi

Listing 3.4: Ukázka odpovědi na požadavek pro získání seznamu videí

```
[{
  "id": "1",
  "path": "http://www.mujserver.com/media/video1.mp4",
  "metadata": {
    "type": "MP4",
    "mimeType": "video/mp4",
    "duration": "00:01:30",
    "frameRate": "30",
    "width": "1280",
    "height": "960"},
  "URL": "http://www.mujserver.com/api/video/1"
},
{
  "id": "1",
  "path": "http://www.mujserver.com/media/video2.webm
     ",
  "metadata": {
    "type": "WEBM",
    "mimeType": "video/webm",
    "duration": "00:18:44",
    "frameRate": "50",
    "width": "1280",
    "height": "960"},
  "URL": "http://www.mujserver.com/api/video/2"
}]
```

GET `/video/{video-id}` - Získá detail videa s identifikátorem video-id

Metoda GET

Stavové kódy

200 - OK - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a video s daným identifikátorem existuje

401 - Unauthorized - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje

406 - Not Acceptable - pokud je vyžadován hlavičkou Accept jiný formát odpovědi než JSON

Tělo odpovědi

Listing 3.5: Ukázka odpovědi na požadavek pro získání detailu videa

```
{
      "id": "1",
  "path": "http://www.mujserver.com/media/video1.mp4",
  "metadata": {
    "type": "MP4",
    "mimeType": "video/mp4",
    "duration": "00:01:30",
    "frameRate": "30",
    "width": "1280",
    "height": "960"},
  "processes: ["1", "2"]
}
```

POST `/video` - nahraje video na server, součástí těla HTTP požadavku musí být video v atributu video

Metoda POST

Stavové kódy

- **200 - OK** - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a tělo požadavku obsahuje video
- **400 - Bad Request** - požadavek obsahuje hlavičku Authorization se správnými přihlašovacími údaji, ale neobsahuje v těle video (obsahuje například textový soubor, či soubor není předán)
- **401 - Unauthorized** - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic
- **403 - Forbidden** - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje
- **404 - Not Found** - pokud video s identifikátorem video-id neexistuje

DELETE `/video/{video-id}` - smaže video s identifikátorem video-id a všechny procesy, které dané video zpracovaly

Metoda DELETE

Stavové kódy

- **200 - OK** - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a video s daným identifikátorem existuje
- **401 - Unauthorized** - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje

GET `/video/{video-id}/process` - získá seznam procesů pro video s daným identifikátorem video-id

Metoda GET

Stavové kódy

200 - OK - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64, v hlavičce Accept je podporovaný MIME typ a video s daným identifikátorem existuje

401 - Unauthorized - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje

406 - Not Acceptable - pokud je vyžadován hlavičkou Accept jiný formát odpovědi než JSON

Tělo odpovědi

Listing 3.6: Ukázka odpovědi na požadavek pro získání seznamu procesů

```
[{
  "id": "1",
  "status": "Done",
  "URL": "http://www.mujserver.com/video/1/process/1"
},
{
  "id": "2",
  "status": "Running",
  "URL": "http://www.mujserver.com/video/1/process/2"
}]
```

GET `/video/{video-id}/process/{process-id}` - získá detail procesu s daným identifikátorem

Metoda GET

Stavové kódy

200 - OK - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64, hlavička Accept obsahuje podporovaný MIME typ a video a proces s danými identifikátory existují

401 - Unauthorized - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje

406 - Not Acceptable - pokud je vyžadován hlavičkou Accept jiný formát odpovědi než JSON

Tělo odpovědi

Listing 3.7: Ukázka odpovědi na požadavek pro získání detailu procesu

```
{
  "id": "1",
  "parameters": {
      "h": "14000",
      "f": "10",
      "alpha": "40",
      "beta": "5",
      "len_width": "12",
      "len_height": "9",
      "width": "1280",
      "height": "960",
      "p_x": "640",
      "p_y": "480"
  },
  "webVTTFile": "http://www.mujserver.com/annotations/
     metadata1.vtt",
  "objects": ["1", "5", "25", "26", "27"],
  "status": "Done",
  "mediaFragmentURIs":[
    "http://www.mujserver.com/video/video1.mp4#id=1",
    "http://www.mujserver.com/video/video1.mp4#id=2",
    "http://www.mujserver.com/video/video1.mp4#id=3",
    "http://www.mujserver.com/video/video1.mp4#id=36",
    "http://www.mujserver.com/video/video1.mp4#id=37",
    "http://www.mujserver.com/video/video1.mp4#id=38",
    "http://www.mujserver.com/video/video1.mp4#id=39",
    "http://www.mujserver.com/video/video1.mp4#id=40",
  ]
}
```

POST `/video/{video-id}/process` - spustí proces zpracování videa s danými parametry, požadované parametry jsou h, f, alpha, len_width a len_height,

width, height, p_x a p_y, volitelné parametry jsou beta, k, p, x_c a y_c. Parametr h udává výšku kamery v milimetrech nad plochou, f udává ohniskovou vzdálenost, alpha udává úhel naklonění kamery na plochu ve stupních, len_width a len_height jsou rozměry snímače v milimetrech, width a height jsou rozměry snímku v pixelech, p_x a p_y jsou souřadnice principálního bodu v pixelech, beta je úhel rotace kamery kolem osy z, k je pole koeficientů radiálního zkreslení, p je pole koeficientů tangenciálního zkreslení a x_c a y_c jsou souřadnice centra zkreslení kamery v pixelech.

Metoda POST

Stavové kódy

- **200 - OK** - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64, video s daným identifikátorem existuje a jsou předány všechny požadované parametry a případně volitelné parametry
- **400 - Bad Request** - pokud v těle požadavku nejsou předány všechny povinné parametry, či jsou předány parametry, které nejsou povinné ani volitelné
- **401 - Unauthorized** - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic
- **403 - Forbidden** - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje
- **404 - Not Found** - pokud video s identifikátorem video-id neexistuje

Příklad těla validního požadavku

Listing 3.8: Příklad těla požadavku pro spuštění procesu

```
{
  "h": "14000",
  "f": "10",
  "alpha": "40",
  "beta": "5",
  "len_width": "12.8",
  "len_height": "7.2",
  "width": "1280",
  "height": "720",
  "p_x": "640",
  "p_y": "360"
}
```

DELETE `/video/{video-id}/process/{process-id}` - smaže proces s daným identifikátorem process-id se všemi detekovanými objekty a souborem s WebVTT metadaty, který byl v rámci procesu vytvořen

Metoda DELETE

Stavové kódy

200 - OK - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a video a proces s danými identifikátory existují

401 - Unauthorized - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje, nebo proces s identifikátorem process-id neexistuje, či zpracoval jiné video

GET `/video/{video-id}/process/{process-id}/object` - získá seznam objektů, detekovaných v rámci procesu s identifikátorem process-id

Metoda GET

Stavové kódy

200 - OK - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64, video a proces s danými identifikátory existují a hlavička Accept obsahuje podporovaný MIME typ

401 - Unauthorized - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic

403 - Forbidden - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje

404 - Not Found - pokud video s identifikátorem video-id neexistuje, nebo proces s identifikátorem process-id neexistuje, či zpracoval jiné video

406 - Not Acceptable - pokud je vyžadován hlavičkou Accept jiný formát odpovědi než JSON

Tělo odpovědi

Listing 3.9: Ukázka odpovědi na požadavek pro získání seznamu objektů detekovaných v rámci daného procesu

```
[
        "http://www.mujserver.com/video/1/process/1/
            object/1",
        "http://www.mujserver.com/video/1/process/1/
            object/2",
```

```
            "http://www.mujserver.com/video/1/process/1/
                object/25",
            "http://www.mujserver.com/video/1/process/1/
                object/26",
            "http://www.mujserver.com/video/1/process/1/
                object/27"
    ]
```

GET `/video/{video-id}/process/{process-id}/object/{object-id}`
získá detail objektu s identifikátorem object-id, který byl detekován v rámci procesu s identifikátorem object-id

Metoda GET

Stavové kódy

- **200 - OK** - pokud je předána hlavička Authorization se správnými přihlašovacími údaji kódovanými v Base64 a video, proces a objekt s danými identifikátory existují
- **401 - Unauthorized** - pokud není předána hlavička Authorization, nebo v Autorizační hlavičce není metoda Basic
- **403 - Forbidden** - pokud v hlavičce Authorization jsou nesprávné přihlašovací údaje
- **404 - Not Found** - pokud video s identifikátorem video-id, process s identifikátorem process-id, nebo objekt s identifikátorem object-id neexistuje

Tělo odpovědi

Listing 3.10: Ukázka odpovědi na požadavek pro získání detailu objektu

```
{
      "id": "26",
      "mediaFragmentsURIs": [
        "http://www.mujserver.com/videos/video1.mp4#id
            =37",
        "http://www.mujserver.com/videos/video1.mp4#id
            =38"
      ]
}
```

3.2.4 Technologie pro ukládání dat

V této kapitole popisuji technologie, které umožňují ukládat data o videích, procesech a detekovaných objektech. U každé technologie je popsáno, o co se jedná, jak se získávají a zapisují data, jak je možné ji využít pro uložení videí, procesů , detekovaných objektů a vygenerovaných anotací a její výhody a nevýhody.

3.2.4.1 Soubory

Jedním ze způsobů, jak je možné ukládat data, jsou soubory ve formátu CSV (Comma Separated Values), či jiném formátu. Data mohou být uložena v jednom, či více souborech v libovolném počtu adresářů. Do souboru lze uložit buď data o jedné entitě, data o všech entitách určitého typu, nebo všech entitách v systému. Data v souboru mohou být v libovolném existujícím formátu jako je CSV, JSON či XML, nebo v nějakém vlastním formátu.

Data lze zapsat do souboru i načíst ze souboru z kódu bez použití speciálních knihoven, či nástrojů. Pro zápis do souboru je potřeba nejdříve daný soubor otevřít s atributem, který dovoluje čtení, či psaní, či obojí. Po otevření souboru je možné zapsat do souboru, či z něj přečíst obsah. V neposlední řadě je možné pomocí atributu daného při otevření souboru vytvořit soubor, do kterého chceme zapsat, pokud neexistuje. Pokud chceme umístit soubor do neexistující složky, je potřeba před otevřením souboru vytvořit složku pomocí vestavěné knihovny. Ukládáme-li každou entitu zvlášť do souboru a při mazání entity chceme smazat i soubor, smažeme soubor s daty pomocí vestavěné knihovny. V Pythonu umožňuje vytváření nových složek, mazání existujících složek a mazání souborů vestavěná knihovna os.

Výhodou tohoto způsobu ukládání dat je, že není potřeba instalovat žádný nástroj a nejsou potřeba žádné přihlašovací údaje. Další výhodou souborů je, že není potřeba se nikam připojovat a potřebný obsah může být načten okamžitě a bez použití knihoven, které se musí instalovat.

Jednou z hlavních nevýhod souborů je, že pro získání seznamu entit navázaných na jinou entitu jako například seznam procesů daného videa je nutné projít celý adresář se soubory s entitami, nebo přečíst určitou řádku v souboru, který se po přidání procesu musí aktualizovat. Další nevýhodou souborů je, že je nutné sám řešit problém s konzistencí, když několik uživatelů zapisuje v jeden okamžik. Toto soubory neumožňují a je nutné zápis několika uživatelů najednou ošetřit například pomocí zámků.

3.2.4.2 MySQL

Dalším technologií pro ukládání informací je relační databáze MySQL. Relační databáze jako MySQL jsou založeny na tabulkách, do kterých se ukládají entity jednoho typu. Každá entita jako například video je reprezentována jedním řádkem. Tabulka se skládá také ze sloupců, které reprezentují určité vlastnosti entity jako například cesta k videu. Sloupce mohou obsahovat jen hodnoty předem daného typu jako například celé číslo, reálné číslo, či řetězec. „Každý řádek v tabulce má svůj unikátní identifikátor, který obvykle odpovídá číslu řádku“ [40].

Pro zápis dat do databáze MySQL a jejich čtení z ní je potřeba nejdříve mít nainstalovaný MySQL driver. Po instalaci driveru je potřeba nastavit přihlašovací údaje do databáze. Čtení dat i zápis dat je možné provést pomocí

jazyka SQL (Structured Query Language), který je strojově i lidsky čitelný. SQL umožňuje filtrovat entity dle libovolného sloupce, spojovat tabulky, seskupovat záznamy dle sloupce, či vybrat maximální hodnotu nějakého sloupce.

Výhodou ukládání dat s využitím databáze MySQL je, že lze rychle a efektivně najít video, proces, či objekt s daným identifikátorem. Další výhodou MySQL je, že lze pomocí SQL dotazu vypočítat identifikátor nově vložené entity.

Jednou z hlavních nevýhod MySQL je, že není možné do nějakého sloupce uložit pole ani hash, který může obsahovat předem neznámý počet parametrů. V případě implementace dalších algoritmů není možné parametry ukládat přímo do tabulky s procesy a je nutné uložit parametry do tabulky s cizím klíčem na proces a získávat parametry filtrováním dle identifikátoru procesu.

3.2.4.3 Redis

Další technologií pro ukládání dat je key-value NoSQL (Not only SQL) databáze Redis (REmote DIctionary Server). Key-value databáze znamená, že lze data uložit pod nějakým klíčem, upravit hodnotu uloženou pod daným klíčem, získat hodnotu dle klíče či smazat klíč. Klíč v Redisu je reprezentován řetězcem. „Podporuje datové typy a struktury jako řetězce, hashe, seznamy, množiny a seřazené množiny s rozsahovými dotazy.“[41] Redis je možné provozovat buď na jednom lokálním stroji, nebo ho nakonfigurovat tak, že data budou rozložena na více strojů, či se použije architektura Master-slave.

Pro využívání Redisu je potřeba mít buď připojení k nějakému vzdálenému serveru, nebo ho mít nainstalovaný a spuštěný na svém stroji. V Pythonu existuje knihovna pro provádění libovolných operací nad daty v Redisu. Pro používání Redisu není potřeba znát žádné přihlašovací údaje a Redis na vzdáleném serveru je vhodnější zabezpečit pomocí síťového tunelu, či SSL/TLS. Redis obsahuje příkazy GET a HGETALL pro získání hodnoty pod daným klíčem, příkaz SET a HSET pro nastavení hodnoty pod daným klíčem. Dále obsahuje příkazy SADD pro přidání prvku do množiny pod daným klíčem, SMEMBERS pro získání prvků množiny, SREM pro smazání prvku z množiny, LPUSH a RPUSH pro přidání prvku do seznamu, LREM pro smazání prvku se seznamu a LRANGE pro získání prvků mezi danými indexy (0 a -1 pro získání celého seznamu).

Jednou z hlavních výhod ukládání dat do Redisu je, že je možné pod libovolným klíčem uložit seznam, množinu či hash a jednoduše je získat. Pokud chceme získat seznam entit, které mají vazbu na danou entitu, lze jednoduše získat seznam jejich identifikátorů pomocí rozsahového dotazu.

Jeho další výhodou oproti jiným databázím je, že do seznamu lze uložit až 4 miliardy prvků. Je možné tak uložit například seznam identifikátorů, či spuštěných procesů a jednoduše získat identifikátor nově vloženého videa, či procesu ještě před uložením.

Další výhodou Redisu je, že je možné ho nakonfigurovat dle toho, kolik dat bude uloženo. Pokud se předpokládá, že nebude uloženo velké množství dat, Redis zprovozníme na jednom lokálním stroji. V ostatních případech ho můžeme nakonfigurovat tak, že se data distribuují do více strojů. V neposlední řadě má Redis oproti některým databázím jako například MySQL výhodu v tom, že není třeba znát žádné přihlašovací údaje.

V neposlední řadě je jeho výhodou, že neobsahuje žádný dotazovací jazyk a data lze získat pomocí jednoduchého příkazu. Z tohoto důvodu je oproti ostatním technologiím snadný na naučení i na práci s daty.

3.2.4.4 MongoDB

Jednou z dalších technologií pro ukládání dat je distrubovaná dokumentová databáze MongoDB. MongoDB ukládá data jako dokumenty ve formátu BSON (Binary JSON) do kolekce. Dokument může v sobě obsahovat vlastnosti s číselnými hodnotami, řetězci, poli i složitějšími objekty. MongoDB umožňuje základní CRUD (Create, Read, Update, Delete) operace s dokumenty, přidávání a odebírání prvků v polích, odkazovat na další entity uvnitř dokumentu.

Pro používání MongoDB je potřeba mít buď připojení k vzdálenému serveru, nebo ho mít nainstalované na svém stroji. Po instalaci MongoDB je potřeba ještě nainstalovat knihovnu pro provádění operací v MongoDB. V Pythonu existují například knihovny Djongo a Mongoengine. MongoDB obsahuje příkazy pro CRUD operace s dokumenty v rámci určité kolekce. Uvnitř příkazů pro CRUD operace se mohou vyskytovat funkce pro vložení prvku do pole a smazání prvku z pole.

Jednou z výhod MongoDB je, že je flexibilní a struktura dat je téměř neomezená. Do jednoho dokumentu je možné uložit data, která obsahují pole i složitější objekty. Pro uložení vazeb na danou entitu je možné použít pole, jehož prvky lze získat přímo.

Jeho nevýhodou oproti Redisu je, že vzhledem k omezení velikosti dokumentu na 16 MB není možné ukládat seznam identifikátoru všech spuštěných procesů, pokud se předpokládá, že aplikace bude hojně využívaná. Identifikátor nově vložené entity je snadné získat až po jejím uložení.

3.2.4.5 Neo4j

Další technologií pro ukládání dat je grafová databáze Neo4j. Každá entita je reprezentovaná uzlem a vztah mezi entitami je reprezentován orientovanou hranou. Každý uzel může mít v sobě obsažená další data, které obsahují identifikátor, řetězec, bod (objekt třídy Point) či informaci o datumu a čase. Neo4j využívá k operacím nad uzly dotazovací jazyk Cypher.

Pro používání Neo4j je potřeba mít buď přístup na vzdálené server, nebo ho mít nainstalované. Po jeho instalaci je potřeba nainstalovat knihovnu v daném programovacím jazyce. V Pythonu existuje knihovna neo4j-driver pro spouštění

dotazů v jazyce Cypher. Cypher umožňuje přidávat a mazat uzly i vztahy mezi nimi, nastavovat hodnoty pro dané uzly i vyhledat uzly na základě daných vztahů.

Nevýhodou Neo4j je, že vlastností uzlu nemůže být hash ani pole. Pokud chceme v Neo4j k uzlu uložit hash, je potřeba ho reprezentovat jako další uzel a vytvořit vztah mezi původním uzlem a nově vytvořeným uzlem. Pro složitější objekty zde vznikne velké množství vztahů.

Další jeho nevýhodu je, že pro získání uzlu dle identifikátoru je nutné prohledat všechny uzly. Získání entity s daným identifikátorem tedy trvá déle než v jiných databázích. Neo4j není vhodné používat v aplikacích, kde se často vyhledává entita dle identifikátoru.

3.2.4.6 Další technologie

Existuje ještě mnoho dalších technologií pro ukládání dat, které ukládají data různým způsobem. Technologiemi pro ukládání dat mohou být i XML databáze, či wide-column databáze jako například Cassandra. Pro uložení dat je možné použít i jiné key-value databáze, dokumentové databázeMezi další key-value databáze patří i Riak, či Dynamo. Pro uložení dat je možné použít i další dokumentové databáze jako například CouchDB.

Dalším způsobem uložení dat, který je podobný grafovým databázím, je RDF (Resource Description Framework). V RDF je každá entita identifikována IRI (Internationalized Resource Identifier) a každé tvrzení o entitě je reprezentováno trojicí subjekt - predikát - objekt. Pro přístup k RDF datům a provádění operací nad nimi je potřeba mít přístup do Triplestoru.

3.2.4.7 Vybraná technologie

Pro ukládání dat je zvolena key-value databáze Redis. Redis je vhodný zejména díky možnosti ukládat hashe a pole. Pole umožňují velmi snadno získat identifikátory entit, které souvisí s danou entitou, či identifikátory všech entit. Identifikátor nově vložené entity lze snadno dopočítat na základě identifikátoru poslední uložené entity.

Dalším důvodem, proč byl Redis vybrán, je, že je možné ho mít instalovaný lokálně, nebo v clusteru. Redis se tedy hodí jak pro menší aplikace, tak pro větší aplikace. Pro aplikace, které nevyužívají opravdu velké množství dat, je možné mít Redis instalovaný lokálně bez využití clusteru.

V neposlední řadě Redis obsahuje několik jednoduchých běžně používaných příkazů a je tak poměrně snadný na naučení. Pro běžnou práci s daty v Redisu se stačí naučit příkazy GET, SET, DEL, HSET, HGETALL, LPUSH, RPUSH, LREM, LRANGE, SADD, SREM a SMEMBERS. Data se získají pomocí jednoduchého příkazu jako například GET, HGETALL, LRANGE, či SMEMBERS a není nutné se učit nějaký dotazovací jazyk.

3.2.5 Návrh uložení anotací a dalších informací

V této kapitole je popsán návrh uložení informací o videích, procesech a objektech. Všechny informace kromě metadat ve formátu VTT jsou uloženy do key-value databáze Redis popsané v kapitole 3.2.4.3. Tato kapitola se zaměřuje na to, jak jsou informace uloženy a jaké struktury a datové typy se používají.

Do Redisu se ukládá seznam již vložených videozáznamů. Seznam již vložených videozáznamů obsahuje identifikátory nahraných videí. Ke každému videozáznamu se do Redisu uloží cesta k souboru jako řetězec, metadata jako hash a seznam procesů jako množiny s identifikátory procesů. Informace o videích jsou uloženy pod klíčem, který obsahuje identifikátor videa.

Dále se do seznamu procesů ukládá identifikátor procesu. Ke každému procesu, zpracovávající videozáznam, se do Redisu uloží parametry, se kterými byl spuštěn, jako hash, seznam všech detekovaných objektů jako množina s identifikátory objektů a cesta k WebVTT souboru s anotacemi jako řetězec. Dále se k procesu uloží seznam všech Media Fragments URI jako seznam řetězců. Informace o procesu jsou uloženy pod klíčem s identifikátorem videa a identifikátorem procesu.

Ke každému detekovanému objektu se uloží seznam URI pro Media Fragments. Každá URI pro Media Fragments je uložena do seznamu jako řetězec. Seznam Media Fragments URI je uložen pod klíčem, který obsahuje identifikátor videa, identifikátor procesu a identifikátor objektu.

Do souboru ve formátu VTT jsou ukládány anotace, které obsahují časový úsek a další metadata ve formátu JSON. Metadata pro každý časový úsek obsahují identifikátor objektu, jeho rychlost v daném časovém úseku a seznam Media Fragments URI s pozicemi objektu na snímcích v rámci daného časového úseku.

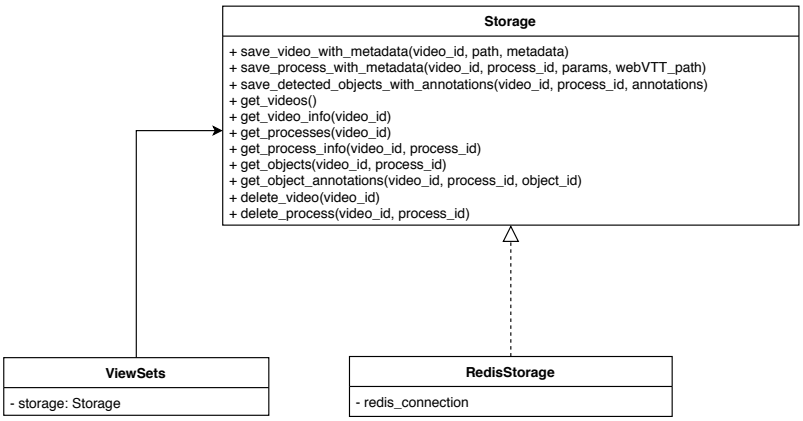
3.2.6 Návrh tříd a metod

V této kapitole popisuji návrh tříd a jejich metod, zajišťujících funkčnost REST API. Návrh je zaměřen zejména na ukládání informací o videích, procesech a detekovaných objektech do nějakého úložiště. Třída reprezentující úložiště je následně využitá v metodách a třídách, které zajišťují zpracování HTTP požadavků. Třída reprezentující úložiště má potomky, které reprezentují konkrétní způsoby uložení.

V diagramu je znázorněna třída `Storage`, která reprezentuje nějaké úložiště, její potomek `RedisStorage`, který slouží pro ukládání dat s využitím Redisu. Třída

`Storage`

obsahuje metody pro uložení a získávání informací o videích, procesech a detekovaných objektech. Tato třída má potomky, které reprezentují konkrétní



Obrázek 3.5: Diagram tříd pro REST API s navrženými třídami a základními metodami

způsoby uložení dat jako například Redis, MySQL, soubory, či MongoDB. V diagramu je dále znázorněno, že třídy, obsahující metody pro zpracování HTTP požadavků využívají nějaké úložiště (tj. nějakého potomka třídy Storage).

V diagramu je znázorněna třída `Storage`, která představuje úložiště pro uložení a čtení dat o videích, procesech zpracování videa i detekovaných pohybujících se objektech. Třída `Storage` je předkem všech tříd, které ukládají a čtou informace o videích, procesech a objektech z určitého úložiště jako například MySQL, Redis, MongoDB, či CSV soubor. Třída `Storage` i její potomci obsahují metody:

save_video_with_metadata
: Metoda ukládá informace o videu a jeho metadata (délka, snímková frekvence, typ a MIME typ, šířka a výška snímků v pixelech)

save_process_with_info
: Metoda ukládá informace o procesu zpracování videa jako například parametry při spuštění a cesta k WebVTT souboru s metadaty.

save_detected_objects_with_annotations
: Metoda ukládá detekované objekty v rámci daného procesu

get_videos
Metoda získá z úložiště seznam všech nahraných videozáznamů na serveru.

get_video_metadata
Metoda získá z úložiště metadata daného videozáznamu.

get_processes
Metoda získá z úložiště seznam všech procesů, které zpracovaly dané video.

get_process_info
Metoda získá z úložiště informace o procesu (parametry při spuštění, seznam detekovaných objektů)

get_objects
Metoda získá z úložiště seznam všech detekovaných objektů v rámci daného procesu.

get_object_annotations
Metoda získá z úložiště anotace informace o daném objektu - jeho anotace s informacemi o časech, pozicích a vypočítaných rychlostech

delete_video
Metoda smaže daný videozáznam a zajistí smazání všech procesů, které ho zpracovaly, včetně detekovaných objektů.

delete_process
Metoda smaže daný proces a všechny uložené informace o něm, všechny detekované objekty v rámci něj a jejich anotace.

Potomkem třídy `Storage` je třída `RedisStorage`, která využívá pro ukládání a získávání dat a jejich mazání Redis. V diagramu je znázorněno, že třída `RedisStorage` dědí všechny metody rodičovské třídy `Storage`. V diagramu je znázorněna i její třídní proměnná `redis_connection`, ve které je uloženo spojení k Redisu a slouží k provádění základních příkazů v Redisu jako získání hodnoty pro daný klíč, nastavení hodnoty, získání elementů seznamu s daným klíčem, či přidání prvku do seznamu s daným klíčem.

V diagramu je dále naznačeno třídou `ViewSets`, že třídy, obsahující metody pro zpracování HTTP požadavků, potřebují instanci třídy `Storage` pro uložení dat při zpracování HTTP požadavků. Vazba mezi třídou `ViewSets` a `Storage` značí, že třídy, zajišťující obsluhu HTTP požadavků využívají instancí třídy `Storage`, do které je přiřazena instance libovolného potomka třídy `Storage`. Instance třídy `Storage` je využita při zpracování HTTP požadavku pro uložení získaných dat, či získání dat z úložiště.

3.3 Návrh webové aplikace

V této kapitole popisuji návrh webové aplikace z hlediska použitých technologií, architektury a uživatelského rozhraní. Zde je popsáno, jak a k čemu jsou použity webové technologie a o kolikastránkovou aplikaci se jedná. Je zde také popsán návrh uživatelského rozhraní a formulářů. Tato aplikace slouží pro demonstrační účely a slouží k vizualizaci funkcionality knihovny pro měření rychlosti objektů ve videu a generování anotací.

3.3.1 Architektura a použité technologie

V této kapitole je popsána architektura webové aplikace a způsob využití webových technologií. Je zde také popsáno, o kolikastránkovou aplikaci se jedná, jak se načítají seznamy a jak se zpracovávají formuláře. Tato kapitola také obsahuje informace o tom, jak je generováno HTML (Hypertext Markup Language).

Webová aplikace se skládá ze 4 stránek, na kterých jsou zobrazována data získaná z REST API a stránku pro přihlášení uživatele. Jedná se tedy o hybridní aplikaci složenou ze 4 dílčích single page aplikací. Na některých stránkách jsou také formuláře pro vložení nového videa, či spuštění nového procesu a tlačítka pro smazání videa, či procesu. Pro volání akcí z REST API po odeslání formuláře se využívá AJAX (Asynchronous JavaScript and HTML), který asynchronně odešle požadavek pro vložení videa, či spuštění procesu. AJAX se využívá také pro načítání seznamů videí, či procesů, které zpracovaly dané video. AJAXové požadavky se volají a zpracovávají pomocí JavaScriptové knihovny JQuery. AJAXové požadavky pro získání seznamů, či detailu procesu jsou zavolány ihned po načtení stránky. Některé HTML elementy jsou generovány pomocí knihovny JQuery, která umožňuje manipulovat s DOM (Document Object Model) a přidávat tak HTML elementy na stránku. Další HTML elementy jsou vygenerovány renderováním proměnných a formulářů předaných v controlleru (v Djangu view).

Webová aplikace obsahuje hlavní stránku s formulářem pro přidání videa a seznamem videí. U každého videa je jeho náhled a odkazy na seznam procesů a na formulář pro přidání nového procesu a tlačítko pro smazání videa. Seznam videí je načten pomocí AJAXového požadavku pro získání seznamu videí z REST API. Jednotlivé HTML elementy s videem, odkazy a tlačítkem pro smazání jsou přidány pomocí JavaScriptové knihovny JQuery poté, co přijde odpověď se seznamem videí v těle. Nahrání videa a uložení informací o něm po odeslání formuláře pro nahrání videa je zajištěno pomocí AJAXového požadavku pro nahrání videa na REST API. Tlačítka pro smazání videí mají nastavenou funkci v JavaScriptu, která se zavolá po kliknutí na tlačítko, zpracuje AJAXový požadavek na REST API pro smazání videa včetně jeho procesů a pomocí JQuery odstraní element s daným videem, odkazy a tlačítkem pro smazání. Token v aktuální session je předán do HTML šablony v controlleru

(v Djangu view) a je vyrenderován v šabloně uvnitř JavaScriptu s AJAXovými požadavky.

Webová aplikace dále obsahuje stránku s formulářem pro spuštění procesu pro dané video. HTML pro formulář je vygenerováno tak, že se předaný formulář z controlleru vyrenderuje pomocí šablonovacího systému. Zpracování formuláře po jeho odeslání je zajištěno pomocí JQuery, které odešle AJAXový požadavek na REST API pro spuštění procesu.

V neposlední řadě webová aplikace obsahuje stránku se seznamem procesů pro dané video. Seznam procesů se načítá pomocí AJAXového požadavku na REST API pro získání procesů daného videa. Identifikátor videa je předán z controlleru do šablony a v JavaScriptu je vyrenderována proměnná s identifikátorem videa. Ke každému procesu je zobrazen stav (zda je proces již dokončený, zda běží, nebo zda ještě nezačal), odkaz na jeho detail a tlačítko pro jeho smazání. Tlačítko pro smazání procesu má nastavenou JavaScriptovou funkci, která se zavolá po kliknutí na něj, odešle AJAXový požadavek na REST API pro smazání videa a pomocí JQuery vymaže HTML element s procesem.

Webová aplikace obsahuje i stránku s detailem procesu, kde se nachází video, parametry, se kterými byl proces spuštěn. Pomocí elementu track se načtou k videu vygenerované anotace ve formátu WebVTT. Detail procesu je načten pomocí AJAXového požadavku na REST API pro získání detailu procesu.

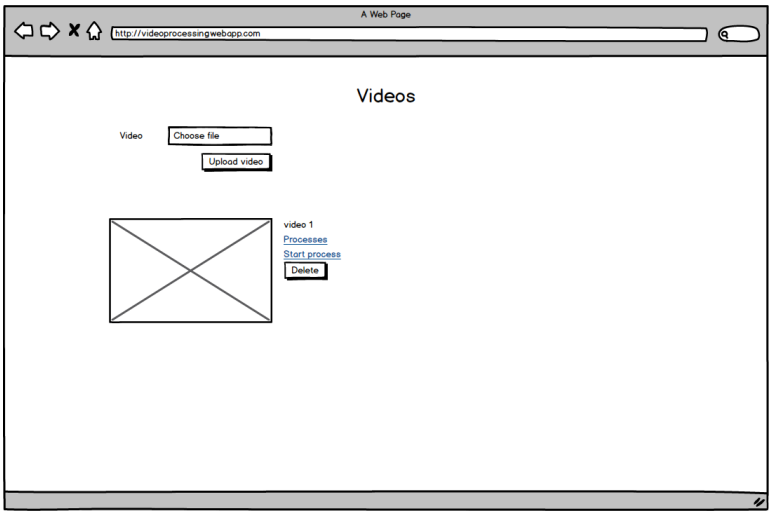
3.3.2 Návrh uživatelského rozhraní

V této kapitole je popsán návrh webové aplikace z hlediska uživatelského rozhraní. Je zde popsáno, z jakých stránek se webová aplikace skládá, kam vedou odkazy a jak jsou rozmístěny jednotlivé prvky na stránce. Jednotlivé stránky a rozmístění prvků jsou načrtnuty ve wireframech.

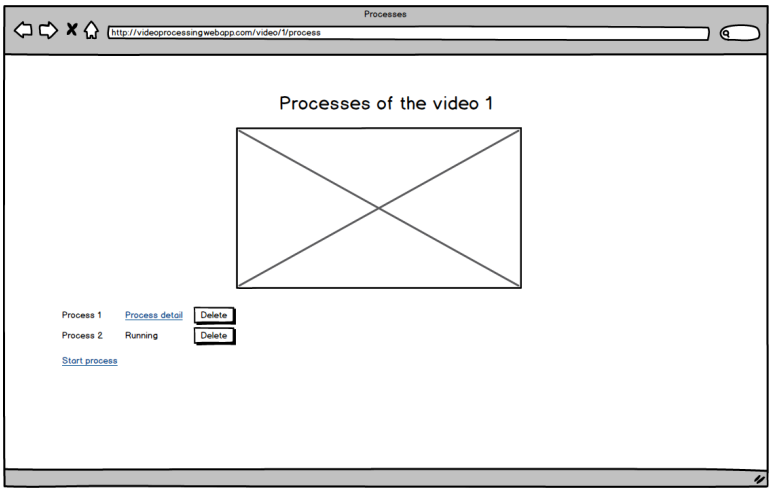
První stránkou a zároveň hlavní stránkou je stránká s formulářem pro přidání videa a seznamem videí. Na stránce se seznamem videí je nahoře umístěn formulář pro přidání videa. Pod formulářem pro nahrání videa se nachází seznam videí. Pro každé video je zobrazen jeho náhled. Vedle náhledu videa se nachází jeho identifikace, odkaz na stránku se seznamem procesů, které dané video zpracovaly, odkaz na stránku s formulářem pro spuštění nového procesu a tlačítko pro smazání videa.

V dalším wireframu je načrtnuta stránka se seznamem procesů, které zpracovaly video s daným identifikátorem. Nad seznamem procesů se nachází náhled videa s daným identifikátorem. Pod náhledem se nachází samotný seznam procesů, které zpracovaly dané video. Pro každý proces se zobrazuje jeho identifikace, stav, či odkaz na detail, pokud proces doběhl a tlačítko pro jeho smazání. Pod seznamem procesů se nachází odkaz na formulář pro přidání nového procesu zpracovávajícího dané video.

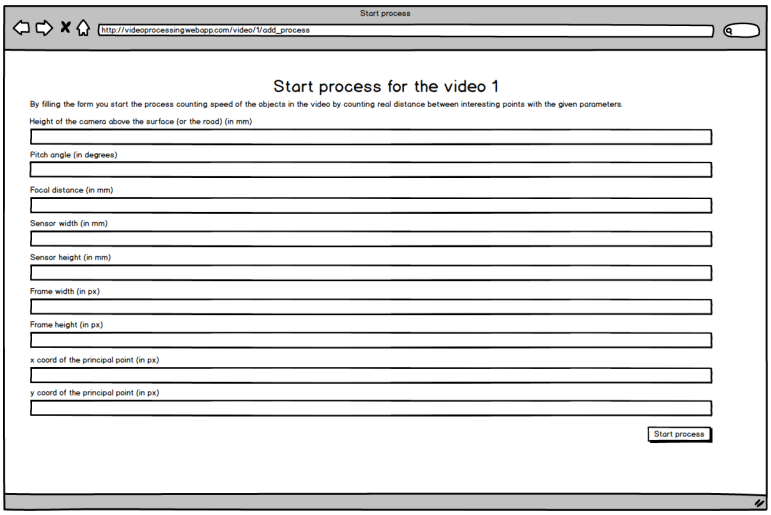
Ve třetím wireframu je načrtnuta obrazovka s formulářem pro přidání nového procesu. Nad formulářem se nachází krátký text o tom, k čemu formulář



Obrázek 3.6: Návrh hlavní obrazovky se seznamem videí a formulářem pro přidání videa



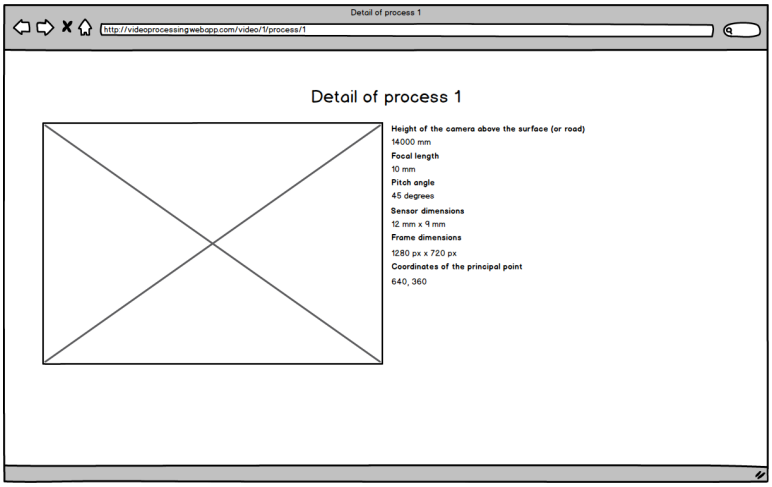
Obrázek 3.7: Návrh obrazovka se seznamem procesů, které zpracovaly, či zpracovávají dané video



Obrázek 3.8: Diagram tříd pro knihovnu pro měření rychlosti objektů ve videozáznamu

slouží a co se stane po jeho vyplnění. Pod tímto textem se nachází samotný formulář s tlačítkem pro odeslání. Formulář obsahuje políčka pro všechny povinné parametry pro algoritmus 2.2.2.3. Popisky, které vyjadřují, co se do daného políčka vyplňuje, jsou umístěny vlevo nad ním.

Poslední wireframe obsahuje návrh stránky s detailem procesu. Na stránce s detailem procesu se nachází samotné video, které je možné přehrát i zastavit. Vedle videa, či pod ním (pokud se stránka zobrazí na tabletu, či mobilu) se zobrazí seznam parametrů, se kterými byl daný proces spuštěn a jejich hodnot. Za každou hodnotou parametru jsou uvedeny i jednotky, aby byl seznam parametrů lidsky čitelný.



Obrázek 3.9: Diagram tříd pro knihovnu pro měření rychlosti objektů ve videozáznamu

Kapitola 4

Realizace

V této kapitole je popsána realizace aplikace, obsahující knihovnu s funkcemi pro zpracování videa, měření rychlosti objektů a generování anotací, REST API pro zpracování videa a webové aplikace. Je zde popsána implementace knihovny, REST API i webové aplikace. Podkapitoly se zaměřují zejména na vybrané metody, které jsou z hlediska implementace specifické.

4.1 Knihovna v Pythonu

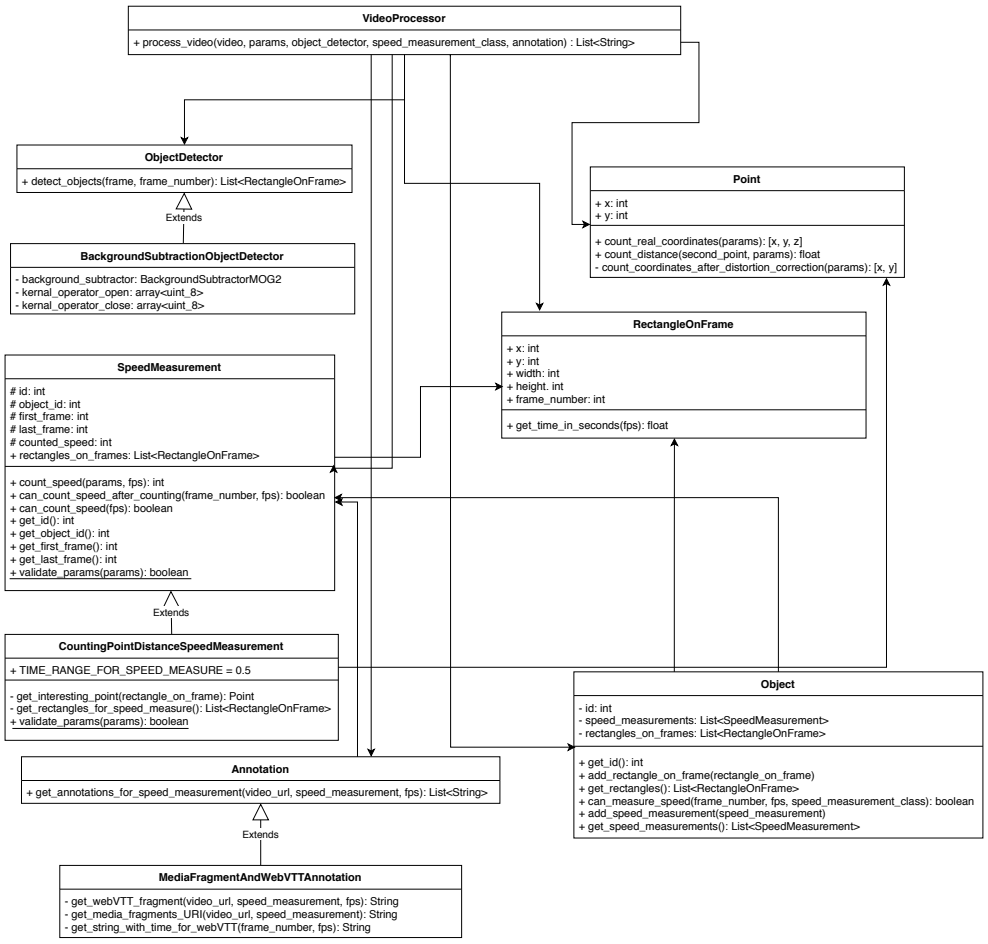
V této kapitole je popsáno, jak je implementována knihovna, jaké třídy funkce obsahuje a k čemu se jednotlivé třídy a metody používají. Jsou zde podrobněji popsány vybrané třídy a metody, které jsou z hlediska implementace zajímavé. Mezi zajímavé metody patří například metoda, která zpracovává video, či metoda, která detekuje objekty.

Knihovna v Pythonu obsahuje hlavní třídu `VideoProcessor`, která zpracovává video, měří rychlost objektů ve videozáznamu a generuje anotace. Knihovna dále obsahuje třídy `RectangleOnFrame`, `Point`, `Object`, `ObjectDetector` a jejího potomka `BackgroundSubtractorObjectDetector`, `SpeedMeasurement` a jejího potomka `CountingPointDistanceSpeedMeasurement`, `Annotation` a jejího potomka `MediaFragmentsAndWebVTTAnnotation`.

Hlavní třídou, ve které se nachází funkce pro zpracování vstupního videa, je třída `VideoProcessor`. `VideoProcessor` obsahuje metodu `process_video()`, která zpracuje video, detekuje objekty daným způsobem, změří rychlost objektů daným algoritmem a vygeneruje anotace pomocí daného nástroje.

Metoda `process_video(video, params, object_detector, annotation, speed_measurement_class)` zpracovává vstupní video, či proud dat snímek po snímku. Parametry metody jsou:

video
: řetězec s názvem videa



Obrázek 4.1: Diagram tříd pro knihovnu pro měření rychlosti objektů ve videozáznamu

params
: parametry kamery jako je šířka a výška snímače, souřadnice x, y principálního bodu, ohnisková vzdálenost, umístění kamery nad vozovkou, úhel naklonění na plochu, úhel otočení kamery a koeficienty radiálního a tangenciálního zkreslení kamery. Koeficienty radiálního a tangentového zkreslení umožňují přesnější výpočet rychlosti objektu, ale nejsou na rozdíl od zbylých parametrů v poli params povinné

object_detector objekt třídy `ObjectDetector`, či jejího potomka, který má detekovat objekty

annotation objekt třídy `Annotation`, který slouží pro vygenerování anotací

speed_measurement_class třída, která má reprezentovat jedno změření rychlosti objektu a zajistit samotný výpočet měření rychlosti

Metoda `process_video` nejdříve ověří pomocí volání statické metody `validate_params` z třídy pro výpočet rychlosti a v případě, že předané parametry nejsou validní, vyhodí výjimku typu `ValueError`. Poté načte video pomocí objektu třídy `VideoCapture` z knihovny OpenCV, který umožňuje načítat video snímek po snímku v cyklu, dokud je možné načíst nějaký další snímek. Poté je objekt třídy `VideoCapture` použit pro získání snímkové frekvence, výšky a šířky snímků v pixelech a výpočet principálního bodu, který se nachází ve středu snímku, pokud jeho souřadnice nejsou dány. Před cyklem, který prochází všechny snímky daného videozáznamu, se deklarují celočíselné proměnné `object_id` pro uložení identifikátoru objektu a `measurement_id` pro uložení identifikátoru měření rychlosti a pole již detekovaných objektů `objects_in_video`.

V cyklu, který prochází snímky videozáznamu, se nejdříve přečte snímek a uloží se do proměnné `frame`. Následně se zavolá metoda pro detekci objektů z dané instance třídy `ObjectDetector`, která vrátí obdélníky s pohybujícími se objekty z popředí. Následně se pro každý takto získaný obdélník kontroluje, zda není v dostatečné malé vzdálenosti od již uloženého obdélníku pro předchozí snímek u nějakého objektu. Pokud jsou souřadnice x, y daného obdélníku oproti obdélníku uloženém u nějakého již uloženého pohybujícího se objektu rozdílné o méně než čtvrtinu šířky pro souřadnici x a výšky pro souřadnici y, je tento obdélník uložen k již detekovanému objektu. V ostatních případech se vytvoří nový pohybující se objekt a uloží se k němu daný obdélník.

Poté se pro nový objekt, nebo dříve uložený objekt zkontroluje, zda je uloženo dostatečné množství obdélníků na snímcích pro měření rychlosti. Pokud je možné na základě uložených obdélníků měřit rychlost, rychlost objektu se vypočítá a následně se vygenerují anotace. Tyto vygenerované anotace jsou poté vráceny příkazem yield.

Třída `RectangleOnFrame` reprezentuje obdélník ve videu, do kterého je možné vepsat konturu některého z pohybujících se objektů na snímku s daným pořadím. Využívá se například pro ukládání URI pro Media Fragments. Obsahuje atributy `x` a `y` pro uložení souřadnic levého horního rohu obdélníku, atributy `width` a `height` reprezentující rozměry obdélníku a atribut `frame_number` pro uložení pořadí snímku, na kterém se v daném obdélníku nachází pohybující se objekt. Tato třída obsahuje metodu `get_time_in_seconds` pro získání čísla, udávajícího počet sekund od začátku videa po aktuální snímek.

Třída `Point` reprezentuje bod na snímku, který může být i zájmovým bodem nějakého pohybujícího se objekt. Její atributy jsou `x` a `y`, ve kterých jsou uloženy souřadnice. Obsahuje metodu `count_real_coordinates` pro získání

souřadnic v souřadném systému, kde centrum kamery leží v počátku, na základě daných souřadnic na snímku. Další její metodou je `count_distance`, která vypočítává reálnou vzdálenost v milimetrech daného bodu od aktuálního bodu. V neposlední řadě obsahuje metodu `count_coordinates_after_distortion_correction`, která vypočte souřadnice po korekci radiálního a tangenciálního zkreslení kamery.

Metoda `count_real_coordinates(params)` získává souřadnice bodu na ploše vůči centru kamery, kam se promítne daný bod na obrazovce. Vypočítá a vrátí souřadnice způsobem popsaným v kapitole 2.2.2.3. Metoda `count_distance (second_point, params)` vypočítá a vrátí reálnou vzdálenost dvou bodů tak, že vypočítá reálné souřadnice obou bodů. Metoda `count_coordinates_after_distortion_correction(params)` vypočte souřadnice na obrazovce po korekci radiálního a tangenciálního zkreslení dle vzorců 2.18 - 2.22.

Třída `Object` představuje objekt ve videu, jehož pozice na snímcích jsou průběžně zaznamenávány. Obsahuje proměnnou `id`, ve které je uložen jednoznačný číselný identifikátor pohybujícího se objektu. Další její proměnnou je pole objektů třídy `RectangleOnFrame` `rectangles_on_frames`, ve kterém se nachází pro každý snímek informace o tom, kde se daný pohybující se objekt nacházel. Další její proměnnou je pole objektů `SpeedMeasurement`, do kterého se ukládají měření rychlosti daného objektu.

Třída obsahuje metodu `add_rectangle_on_frame`, která přidá do pole `rectangles_on_frames` objekt třídy `RectangleOnFrame` s informacemi o pozici daného objektu na daném snímku. Další její metodou je `can_measure_speed`, která vrací, zda se již má změřit rychlost objektu, či nikoliv. Další metodou je `get_rectangles`, která vrací pole uložených objektů třídy `RectangleOnFrame` s informacemi o tom, kde se daný objekt na jednotlivých snímcích nacházel. Poslední metodou této třídy je `add_speed_measurement`, která přidá do pole `speed_measurements` záznam o měření rychlosti. Dále obsahuje gettery `get_id` a `get_speed_measurements`.

Metoda `can_measure_speed(fps, speed_measurement_class)` vrací pravdivostní hodnotu, která udává, zda se má změřit rychlost objektu, či nikoliv. Prvním parametrem je celé číslo `fps`, udávající snímkovou frekvenci. Druhým parametrem, který je ale nepovinný, je `speed_measurement_class`, ve kterém je uložena třída, která se používá pro měření rychlosti a kontrolu, zda je dostatek informací pro měření rychlosti. Nejdříve se zkontroluje, zda jsou již uložena měření rychlosti. Pokud ne, návratová hodnota se získá pomocí volání funkce `can_measure_speed` na objektu třídy, dané parametrem `speed_measurement_class` s polem objektů třídy `RectangleOnFrame` v třídní proměnné `rectangles_on_frames`. V ostatních případech projde uložené objekty třídy `SpeedMeasurement` a zavolá na každém z nich metodu `can_measure_speed_after_counting`. Pokud volání vždy vrátí `True`, metoda vrátí `True`, jinak vrátí `False`.

Třída `ObjectDetector` reprezentuje způsob detekce objektů na snímcích.

Jedná se o abstraktní třídu a je předkem všech tříd, které detekují objekty na snímcích určitým způsobem jako například ořezáváním pozadí a získáním popředí. Obsahuje metodu `detect_objects`, která v této nadtřídě vrací prázdné pole. Tato metoda je implementována v potomcích této třídy.

Potomkem třídy `ObjectDetector` je třída `BackgroundSubtractorObject-Detector`, která detekuje objekty tak, že odečte aktuální snímek od minulých snímků a získá popředí. Tento způsob detekce objektů je popsán v kapitole 2.2.1.1. Třída obsahuje objekt třídy `BackgroundSubtractorMOG2` `background_subtractor` pro odečítání pozadí a získávání popředí, `kernal_operator_open` pro odstranění šumu z pozadí a `kernal_operator_close` pro odstranění děr z kontur objektů na popředí. V této třídě je implementována metoda `detect_objects`.

Metoda `detect_objects(frame, frame_number)` detekuje objekty tak, že nejdříve odečte pozadí, získá popředí a poté najde kontury. Ke každé kontuře se najde obdélník, do kterého ji lze vepsat. Parametry metody jsou objekt, či pole `frame`, ve kterém je uložena reprezentace snímku, a celé číslo `frame_number`, udávající pořadí daného snímku. Nejdříve se odečte pozadí daného snímku od těch, které byly předtím zpracovány, a získá se tak i popředí. Popředí se získá tak, že pomocí určitého prahu se vyfiltruje pozadí a zvýrazní se popředí. Z pozadí je odstraněn šum a z kontur pohybujících se objektů jsou odstraněny díry. Následně se získají kontury objektů na popředí a ke každé kontuře se najde obdélník, do kterého je možné konturu vepsat. Pro každý obdélník, do něhož je možné vepsat konturu, se vytvoří objekt třídy `RectangleOnFrame` a uloží se do pole, pokud je jeho šířka i výška větší než 50. Nakonec vrátí pole objektů třídy `RectangleOnFrame`, které obsahují informace o pozicích objektů a jejich velikostech na snímku.

Třída `SpeedMeasurement` zajišťuje samotný výpočet rychlosti a reprezentuje algoritmus pro výpočet rychlosti. Jedná se o abstraktní třídu. Je předkem tříd, které reprezentují určitý algoritmus měření rychlosti jako například měření rychlosti na základě vypočtené vzdálenosti dvou bodů. Obsahuje celočíselnou proměnnou `id`, která je jednoznačným identifikátorem měření rychlosti objektu v rámci videozáznamu. Další její celočíselnou proměnnou je `object_id`, která obsahuje identifikátor objektu, jehož rychlost je změřena. Dalšími proměnnými této třídy je pole instancí `RectangleOnFrame`, které v sobě obsahuje pozice objektů na snímcích potřebné pro výpočet rychlosti, celá čísla `first_frame` a `last_frame`, udávající pořadí prvního a posledního snímku časového úseku, v němž je měřena rychlost objektu, a celé číslo `counted_speed`, do kterého se ukládá vypočtená rychlost objektu.

Její metodou je `count_speed`, která vypočítává rychlost algoritmem, implementovaným v potomcích této třídy. Dále obsahuje gettery `get_id`, `get_object_id`, `get_rectangles_on_frames`, `get_first_frame` a `get_last_frame`. Zde všechny metody kromě getterů vrací `0`, nebo `False`, čímž je naznačena abstraktnost třídy. Dále obsahuje statickou metodu `validate_params`, která v této nadtřídě vrací `True` a v podtřídách kontroluje parametry a vrací,

zda jsou předány všechny potřebné parametry, zda nebyly předány parametry navíc a zda mají parametry správný typ hodnoty.

Potomkem třídy `SpeedMeasurement` je třída `CountingPointDistanceSpeedMeasurement`, která reprezentuje algoritmus měření rychlosti popsaný v kapitole 2.2.2.3. Implementuje metody `count_speed`, `can_count_speed_after_counting` a `can_count_speed`. Další její metodou, která ale není v předkovi, je `get_interesting_point`, která získá zájmový bod pro daný obdélník, reprezentovaný objektem třídy `RectangleOnFrame`. Na rozdíl od předka obsahuje také metodu `get_rectangles_for_speed_measure`, která vrací obdélníky s objektem na začátku a na konci časového úseku. Třída zajišťuje výpočet vzdálenosti zájmových bodů objektu na začátku a na konci časového úseku a následný výpočet rychlosti dle vzorce 2.24. Rychlost objektu je zde měřena každých 0,5 sekundy, což je nastaveno pomocí konstanty `TIME_RANGE_FOR_SPEED_MEASURE`.

Metoda `count_speed(params, fps)` vypočítá rychlost objektu na základě daných parametrů a snímkové frekvence, pokud již nebyla vypočtena. Pokud rychlost objektu již byla vypočtena, vrátí hodnotu proměnné `counted_speed` s již vypočtenou rychlostí. Prvním jejím parametrem je slovník `params`, který obsahuje parametry kamery, výšku kamery nad plochou a úhel naklonění kamery na plochu. Dalším jejím parametrem je celé číslo `fps`, udávající snímkovou frekvenci. Nejdříve najde instance třídy `RectangleOnFrame`, jejichž zájmové body mají být použity pro výpočet vzdálenosti. Poté se nastaví proměnná `last_frame`, která udává poslední snímek časového úseku, v rámci něhož byla změřena rychlost objektu. Poté vypočítá vzdálenost zájmových bodů z obou instancí třídy `RectangleOnFrame`. Následně na základě vzdálenosti vypočte rychlost dle vzorce 2.24, uloží ji do proměnné `counted_speed`. Po výpočtu rychlosti se uloží do třídních proměnných i instance třídy `RectangleOnFrame`, jejichž zájmové body byly použity pro výpočet rychlosti. Nakonec metoda vrátí vypočtenou rychlost.

Metoda `can_count_speed_after_counting(frame_number, fps)` vrací, zda je možné znovu měřit rychlost po daném měření rychlosti, reprezentovaném daným objektem. Parametrem metody je celé číslo `frame_number` udávající číslo aktuálního snímku a celé číslo `fps` udávající snímkovou frekvenci. Zde se ověří, zda uběhlo od posledního měření alespoň 0,5 sekundy. Pokud od posledního měření uběhlo alespoň 0,5 sekundy, metoda vrátí `True`, jinak vrátí `False`.

Metoda `can_count_speed(fps)` zjišťuje, zda uložené informace v proměnné `rectangle_on_frames` jsou dostatečné. Parametrem metody je celé číslo `fps` udávající snímkovou frekvenci. Zjistí, zda je uložen v proměnné `rectangle_on_frames` aktuální snímek a snímek 0,5 sekundy před ním. Pokud jsou v proměnné `rectangle_on_frames` uloženy snímky z poslední 0,5 sekundy, metoda vrátí `True`, jinak vrátí `False`.

Metoda `get_interesting_point(rectangle_on_frame)` vrací zájmový bod, který se použije při výpočtu uražené vzdálenosti za daný časový úsek. Zájmový bod se nachází ve spodu uprostřed daného obdélníku. Jejím pa-

rametrem je instance třídy `RectangleOnFrame rectangle_on_frame`, která reprezentuje čtverec na snímku s pohybujícím se objektem.

Metoda `get_rectangles_for_speed_measurement` získá a vrátí dva objekty třídy `RectangleOnFrame`, které mají v sobě informace o pozici objektu na snímcích na začátku a na konci časového úseku. Předpokládá se, že pole objektu třídy `RectangleOnFrame rectangles_on_frames` je seřazené dle pořadí snímku. Metoda najde a vrátí pole, obsahující objekt třídy `RectangleOnFrame` s informacemi o pozici a velikosti objektu na začátku časového úseku a objekt třídy `RectangleOnFrame` s informacemi o pozici a velikosti objektu na konci časového úseku.

Třída `Annotation` zajišťuje generování anotací a reprezentuje nějaký způsob anotace. Jedná se o abstraktní třídu, které je nadtřídou všech tříd, které reprezentují určitý způsob anotace videa a generují anotace v určitém formátu jako je Media Fragments URI, či WebVTT. Obsahuje metodu `get_annotations_for_speed_measurement` pro získání anotací pro dané měření rychlosti objektu. V této třídě vrací prázdné pole, čímž je naznačena abstraktnost třídy.

Potomkem třídy `Annotation` je třída `MediaFragmentAndWebVTTAnnotation`, která pro jednotlivá měření rychlosti generuje URI pro Media Fragments a definice časových úseků s metadaty o rychlosti pro zápis do souboru s WebVTT metadaty. Třída implementuje metodu `get_annotations_for_speed_measurement` pro získání Media Fragments URI a fragmentů ve formátu WebVTT pro zápis metadat o měření rychlosti. Jejími dalšími metodami jsou `get_webVTT_fragment`, `get_media_fragments_URI` a `get_string_with_time_for_webVTT`. Metoda `get_media_fragments_URI` vrací řetězec s Media Fragments URI s identifikátorem měření rychlosti.

Metoda `get_annotations_for_speed_measurement(video_url, speed_measurement, fps)` získá anotace pro dané měření rychlosti, reprezentované objektem třídy `SpeedMeasurement speed_measurement`. Parametry metody jsou:

video_url
: řetězec s URL adresou videa

speed_measurement
: objekt třídy `SpeedMeasurement`, reprezentující měření rychlosti, pro které mají být vygenerované anotace

fps
: celé číslo udávající snímkovou frekvenci

Nejdříve získá Media Fragments URI pro dané měření rychlosti pomocí volání metody `get_media_fragments_URIs`. Následně získá pomocí volání metody `get_webVTT_fragment` řetězec ve formátu VTT, který v sobě obsahuje definici časového úseku, identifikátor měření i objektu i naměřenou rychlost. Metoda

vrátí pole, které obsahuje získané Media Fragments URI a řetězec ve formátu VTT s metadaty o daném měření rychlosti.

Metoda `get_webVTT_fragment(video_url, speed_measurement, fps)` získá a vrátí řetězec s metadaty o daném měření rychlosti, který se má zapsat do souboru s WebVTT metadaty. Parametry metody jsou řetězec `video_url` s URI videa, objekt `speed_measurement`, reprezentující měření rychlosti objektu a celé číslo `fps`, udávající snímkovou frekvenci. Nejdříve vytvoří řetězce reprezentující první řádek s identifikátorem měření rychlostí. Poté získá počáteční a koncový čas pomocí volání metody `get_string_with_time_for_webVTT` časy ve formátu minuty:sekundy.milisekundy získané z čísel prvního a posledního snímku daného časového úseku. Následně vytváří třetí řádek s metadaty, která obsahují identifikátor objektu, vypočtenou rychlost a seznam URI pro Media Fragments.

Při tvoření řetězce s metadaty se projde seznam všech objektů třídy `RectangleOn-`
`Frame`, které mají stejné, či vyšší číslo snímku než číslo prvního snímku daného časového úseku. Pro všechny takové objekty se vytvoří řetězec reprezentující Media Fragments URI s časem ve fragmentu t a pozicí a velikostí objektu ve fragmentu xywh. Tento řetězec s URI s fragmenty se přidá do hlavního řetězce s metadaty.

4.2 REST služba

V této kapitole je popsána implementace REST API jak z hlediska tříd a metod, tak z hlediska uložení dat. Zde je popsáno, jak jsou ukládána data, která REST API využívá. Další část kapitoly pojednává o tom, jak je implementováno REST API a jaké třídy a metody jsou implementovány.

4.2.1 Uložení anotací a dalších informací

V této kapitole je popsáno, jak jsou ukládána data, která využívá REST API. Vzhledem k tomu, že data jsou ukládána do Redisu, je zde popsáno, jaké klíče jsou použity, pod jakým klíčem je co uloženo a jakými datovými strukturami jsou data reprezentována.

V Redisu se používají klíče pro ukládání seznamu videí a seznamu procesů, aby bylo snadné získat identifikátor nově vloženého procesu, či videa. Jedná se o klíče `video` a `process`.

video seznam identifikátorů již nahraných videí, jedná se o datový typ list s identifikátory videí reprezentovanými řetězci

process seznam identifikátor již přidaných procesů, jedná se o datový typ list s identifikátory procesů reprezentovanými řetězci

Pro ukládání informací o videu a seznamu procesů se používají klíče obsahující identifikátor videa. Ke každému videu se ukládá relativní cesta souboru vůči definovanému adresáři (či název videa), token autora v Base64 a metadata.

video:video-id:path řetězec s relativní cestou k videu vůči definovanému adresáři, kam se ukládají videa

video:video-id:author přihlašovací údaje autora v Base64 reprezentované řetězcem

video:video-id:metadata hash s metadaty videa, který obsahuje rozměry snímku, snímkovou frekvenci, trvání, typ a MIME typ

video:video-id:processes množina procesů, které zpracovaly dané video, do které se ukládají identifikátory procesů jako řetězce

Informace o procesu a seznam objektů se ukládá pod klíči, které obsahují identifikátor videa i seznam procesů. Ke každému procesu se ukládají parametry při spuštění, cesta k souboru s WebVTT metadaty a jeho stav. Dále se k procesu ukládá seznam Media Fragments URI s identifikátorem měření rychlosti.

video:video-id:process:process-id:parameters hash s parametry, se kterými byl proces spuštěn

video:video-id:process:process-id:webVTTFile relativní cesta k souboru s WebVTT metadaty, který se má načíst do videa, reprezentovaná řetězcem, cesta je relativní vůči adresáři, kam se ukládají VTT soubory

video:video-id:process:process-id:status stav procesu reprezentovaný řetězcem "Running", nebo "Done"

video:video-id:process:process-id:objects množina s identifikátory objektů, které byly detekovány v rámci procesu

video:video-id:process:process-id:mediaFragmentsURIS - seznam řetězců s Media Fragments URI s identifikátorem měření rychlosti ve fragmentu id

Ke každému detekovanému objektu se ukládají URI pro Media Fragments s identifikátorem měření rychlosti objektu. Media Fragments URI jsou do seznamu ukládány jako řetězce. Tento seznam se ukládá pod klíčem video:video-id:process:process-id:object:object-id:mediaFragmentsURIS.

4.2.2 Třídy a metody

V této kapitole je popsána implementace REST API. Je zde popsáno, jaké třídy a metody jsou implementovány, k čemu slouží a co dělají. Jsou zde podrobněji popsány vybrané metody, které jsou z hlediska implementace zajímavé.

Třídy, které zajišťují zpracování požadavků, jsou uloženy v souboru *views.py*. V souboru *views.py* je uložena třída `BaseViewSet` a její potomci `VideoViewSet`, `ProcessViewSet` a `ObjectViewSet`. Třída `BaseViewSet` má uloženou instanci třídy `Storage` pro ukládání metadat a dalších informací o videích, procesech a detekovaných objektech. Dále je implementována třída `Storage` a její potomek `RedisStorage`. Třídy `Storage` a `RedisStorage` jsou uloženy v souboru *storage.py*. Používané konstanty jsou uloženy v souborech *constants.py* a *constants_objects.py*.

Třída `BaseViewSet` je předkem všech tříd, které zpracovávají požadavky na REST API a pracují s metadaty a dalšími informacemi o videích, procesech a detekovaných objektech. Obsahuje metody, které slouží k ověření, zda je HTTP požadavek validní.

video_and_process_exist
: Metoda zjistí, zda video a proces s danými identifikátory existují a zda proces zpracovával video s daným identifikátorem.

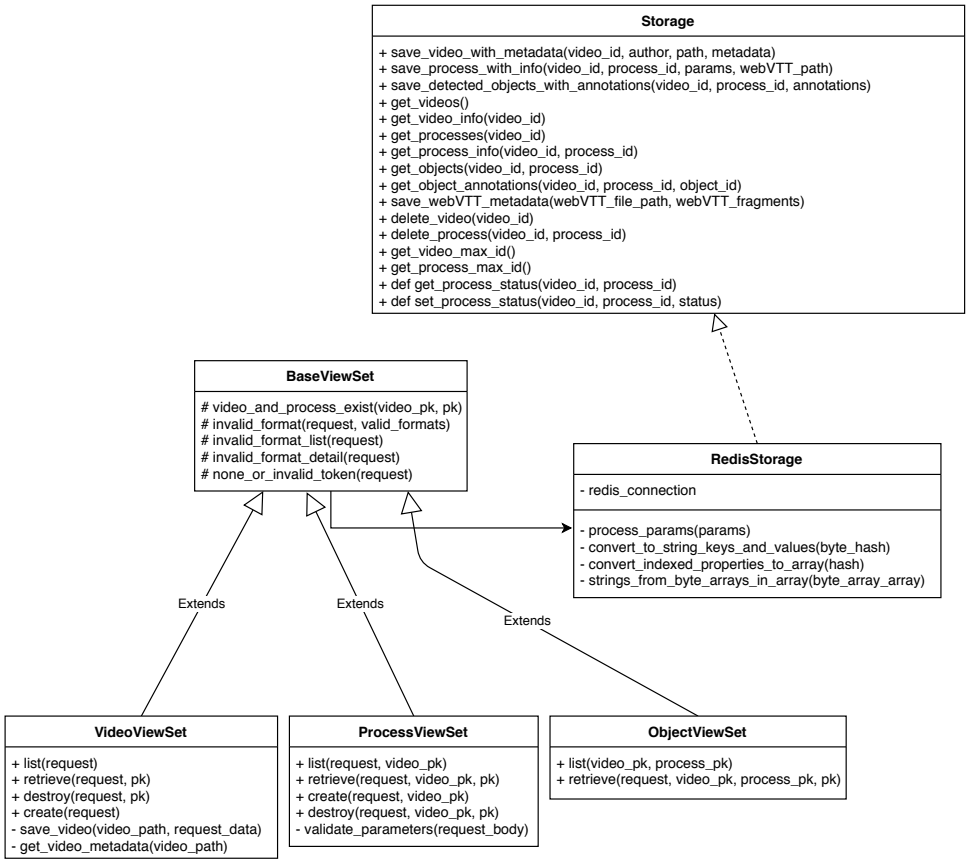
invalid_format
: Metoda zjišťuje, zda v požadavku je předán MIME typ, který není v seznamu akceptovaných MIME typů. V REST API je podporovaný zatím jen JSON, a proto metoda vrací True, když předaná hlavička Accept vyžaduje jiný formát než JSON.

invalid_format_list
: Metoda zjišťuje, zda v požadavku je předán MIME typ, který není podporován pro zobrazení seznamu entit. V REST API je podporovaný zatím jen JSON, a proto metoda vrací True, když předaná hlavička Accept vyžaduje jiný formát než JSON.

invalid_format_detail
: Metoda zjišťuje, zda v požadavku je předán MIME typ, který není v seznamu podporovaných MIME typů pro zobrazení detailu entity.

none_or_invalid_token
: Metoda zjišťuje, zda je předán autorizační token v hlavičce HTTP požadavku a zda je v hlavičce Authorization nastavena HTTP Basic autentizace

Jedním z potomků třídy `BaseViewSet` je třída `VideoViewSet`. Třída `VideoViewSet` obsahuje metody, představující jednotlivé metody GET pro získání seznamu videí, GET pro získání detailu videa, POST a DELETE nad zdrojem video.



Obrázek 4.2: Diagram tříd pro REST API, využívající funkce knihovny

list
Metoda zpracovávající požadavek s metodou GET nad zdrojem video pro získání seznamu videozáznamů

retrieve
Metoda zpracovávající požadavek s metodou GET nad zdrojem video/video-id pro získání detailu videozáznamu s metadaty a seznamem procesů, které toto video zpracovaly

create
Metoda zpracovávající požadavek s metodou POST nad zdrojem video pro nahrání videozáznamu na server a uložení metadat videozáznamu

destroy
Metoda zpracovávající požadavek s metodou DELETE nad zdrojem video pro odstranění videozáznamu ze serveru a odstranění jeho metadat a případné vymazání všech procesů, které toto video zpracovaly, a objektů, které byly detekovány

save_video
Metoda, která ukládá video do souboru s danou cestou. Obsah videa pro zapsání do souboru si získá z dočasného souboru, který je uložen v těle požadavku. Obsah souboru v těle požadavku zapíše do souboru s danou cestou.

get_video_metadata
Metoda, která s využitím knihovny a nástroje exiftool získá metadata videa - rozměry snímků v pixelech, typ a MIME typ, trvání a snímková frekvence. Nejdříve získá mapu se všemi metadaty daného videa. Poté se vybraná metadata (trvání, šířka a výška snímků v pixelech, snímková frekvence, typ a MIME typ) uloží do příslušných proměnných a vrátí se slovník s těmito metadaty. Vybraná metadata lze získat z pole všech metadat pod následujícími indexy:

File:FileType Typ videa

File:MIMEType MIME typ videa

index obsahující Duration Délka (trvání) videa (pro jiné formáty videa než MP4 lze získat dobu trvání pod indexem Composite:Duration)

index obsahující FrameRate Snímková frekvence videa (tento index je platný pro formát MP4)

index obsahující ImageSize rozměry snímku ve formátu šířkaxvýška, tyto rozměry se získají rozdělením řetězce dle znaku x.

Dalším z potomků třídy `BaseViewSet` je třída `ProcessViewSet`. Třída `ProcessViewSet` obsahuje metody, které slouží pro zpracování požadavků nad

zdrojem process s metodami GET pro získání seznamu procesů, které zpracovaly dané video, či pro získání detailu procesu s parametry a seznamem detekovaných objektů, POST pro spuštění nového procesu, PUT pro znovuspuštění procesu s jinými, či stejnými parametry a DELETE pro odstranění procesu a všech detekovaných objektů v rámci něj.

list
: Metoda zpracovává požadavek s metodou GET nad zdrojem process pro získání seznamu procesů, které zpracovaly videozáznam s daným identifikátorem video-id.

retrieve
: Metoda zpracovává požadavek s metodou GET nad zdrojem process pro získání procesu s identifikátorem process-id, který zpracoval video s identifikátorem video-id. Získá seznam objektů, které byly detekovány v rámci procesu a parametry, se kterými byl spuštěn.

create
: Metoda zpracovává požadavek s metodou POST nad zdrojem process pro spuštění procesu zpracování videozáznamu s identifikátorem video-id, který detekuje objekty a změří jejich rychlosti. Dále se v této metodě asynchronně zavolá funkce, která zpracuje video. Pro asynchronní volání funkce se zde používá knihovna DjangoQ.

update
: Metoda zpracovává požadavek s metodou PUT nad zdrojem process pro spuštění procesu zpracování videozáznamu s identifikátorem video-id, který detekuje objekty a změří jejich rychlosti. Proces se znovu spustí s jinými, či stejnými parametry. Tento proces detekuje objekty a měří jejich rychlosti. Na rozdíl od metody create vymaže všechny objekty, které byly detekovány v procesu s identifikátorem process-id. Stejně jako metoda create ukládá metadata do souboru ve formátu VTT a do Redisu ukládá seznam detekovaných objektů, informace o detekovaných objektech a parametry při spuštění.

delete
: Metoda zpracovává požadavek s metodou DELETE nad zdrojem process pro odstranění informací o procesu s identifikátorem process-id a detekovaných objektech v rámci něj. Z Redisu se vymažou informace o detekovaných objektech, seznam detekovaných objektů v rámci procesu a informace o procesu jako jsou parametry při jeho spuštění a cesta k souboru s WebVTT metadaty. Po vymazání informací z Redisu se smaže i samotný WebVTT soubor s metadaty.

validate_parameters
: Metoda validuje parametry v těle požadavku a kontroluje nejdříve, zda

obsahuje všechny povinné parametry. Poté kontroluje, zda tělo obsahuje jen ty povinné parametry a další povolené parametry - koeficienty zkreslení kamery, centrum zkreslení kamery a úhel otočení.

Dále je v souboru *views.py* implementována funkce `process_video_and_generate_annotations`, která je zavolána asynchronně. Nejdříve uloží do databáze, že daný proces běží. Následně převede parametry s požadavku pomocí volání funkce `converted_params_with_url`, která převede slovník s hodnotami, uloženými v poli o jednom prvku, na slovník s reálnými čísly, či řetězci a přidá parametr s URL adresou serveru. Poté zavolá metodu pro zpracování videa a zajistí uložení vygenerovaných anotací do Redisu. Po zpracování videa uloží do databáze, že daný proces již doběhl (tj. uloží do Redisu pro klíč reprezentující stav procesu hodnotu Done).

Posledním potomkem třídy `BaseViewSet` je třída `ObjectViewSet`. Třída `ObjectViewSet` slouží pro zpracování požadavků nad zdrojem object. Obsahuje metody, které zajišťují zpracování požadavků s metodou GET pro získání seznamu objektů, detekovaných v rámci procesu s identifikátorem process-id, který zpracovával video s identifikátorem video-id, či získání detailu objektu s URI pro Media Fragments a WebVTT fragmenty.

list
: Metoda zpracovává požadavek s metodou GET pro získání objektů, které byly detekovány během procesu s identifikátorem process-id, který zpracovával videozáznam s identifikátorem video-id. Seznam všech objektů si získá z Redisu pomocí klíče, který obsahuje identifikátor procesu i identifikátor videozáznamu.

retrieve
: Metoda zpracovává požadavek s metodou GET pro získání detailu objektu s WebVTT metadaty, týkajícími se daného objektu, a URI pro Media Fragments s časem, souřadnicemi a velikostí na snímku a identifikátorem. Seznam URI pro Media Fragments a seznam WebVTT fragmentů získá z Redisu.

Dále je implementována třída `Storage`, která představuje úložiště pro zápis, získávání a mazání informací o videozáznamech, procesech zpracování videozáznamů a detekovaných objektech a jejich rychlostech a jejich získávání. Obsahuje metody pro uložení informace o videu a jeho metadat, informací o spuštěném procesu a jeho parametrů, či detekovaných objektů v rámci daného procesů a jejich anotací. Dále jsou v ní metody pro získání metadat videa, procesů, které dané video zpracovaly, informací o procesu a jeho parametrů, seznamu objektů, detekovaných v rámci daného procesu, či anotací daného objektu. Pod klíčem videoMaxID je možné získat nejvyšší identifikátor videozáznamu a pod klíčem processMaxID je možné získat nejvyšší identifikátor procesu, zpracovávajícího nějaký videozáznam.

save_video_with_metadata
Metoda ukládá informace o videu a jeho metadata (délka, snímková frekvence, typ a MIME typ, šířka a výška snímků v pixelech)

save_process_with_info
Metoda ukládá informace o procesu jako například parametry při spuštění a cesta k WebVTT souboru s metadaty, ale neukládá seznam všech detekovaných objektů v rámci daného procesu.

save_detected_objects_with_annotations
Metoda ukládá seznam detekovaných objektů v rámci daného procesu a ke každému objektu uloží seznam URI pro Media Fragments a seznam WebVTT fragmentů s naměřenými rychlostmi. Dále zajistí zápis WebVTT metadat do souboru voláním metody save_webVTT_metadata

get_videos
Metoda získá z úložiště seznam všech nahraných videozáznamů na serveru.

get_video_metadata
Metoda získá z úložiště metadata daného videozáznamu.

get_processes
Metoda získá z úložiště seznam všech procesů, které zpracovaly dané video.

get_process_info
Metoda získá z úložiště informace o procesu (parametry při spuštění, seznam detekovaných objektů)

get_objects
Metoda získá z úložiště seznam všech detekovaných objektů v rámci daného procesu.

get_object_annotations
Metoda získá z úložiště anotace daného objektu - seznam URI pro Media Fragments, které obsahují čas, kdy se objekt objevil ve videu, souřadnice a velikost na daném snímku a jeho identifikátor a seznam WebVTT fragmentů s definicí časového úseku, kdy se pohyboval danou rychlostí, a naměřenou rychlost

delete_video
Metoda smaže daný videozáznam, všechny procesy, kterého zpracovaly i objekty detekované v rámci těchto procesů. Nejdříve smaže všechny detekované objekty v daném videozáznamu a jejich anotace (Media Fragments URI a WebVTT metadata). Poté smaže všechny informace o procesech, které toto video zpracovaly, včetně seznamu objektů a WebVTT soubory s anotacemi, které byly vygenerovány pro tyto procesy. Následně

smaže seznam procesů, které zpracovaly daný videozáznam, a všechny informace o daném videozáznamu jako je cesta k videozáznamu a metadata. Po vymazání všech informací vymaže i samotný videozáznam. Nakonec se videozáznam vymaže ze seznamu videozáznamů.

delete_process
Metoda smaže daný proces a všechny uložené informace o něm, všechny detekované objekty v rámci něj a jejich anotace. Nejdříve smaže všechny detekované objekty v daném videozáznamu a jejich anotace (Media Fragments URI a WebVTT metadata). Poté smaže všechny informace o daném procesu jako parametry, se kterými byl spuštěn, a cesta k WebVTT souboru. Následně odstraní tento proces ze seznamu procesů, které zpracovaly dané video. Nakonec smaže i samotný soubor s WebVTT metadaty, vygenerovaný pro tento proces.

save_webVTT_metadata
Metoda zapíše metadata do souboru ve formátu VTT, který se přidá do videa jako další stopa.

get_process_status
Metoda získá stav procesu s daným identifikátorem.

set_process_status
Metoda uloží danou informaci o stavu procesu.

Potomkem třídy `Storage` je třída `RedisStorage`, která využívá pro ukládání a získávání dat a jejich mazání Redis. Metadata videa a parametry při spuštění procesu ukládá do hashe s příslušnými vlastnostmi. Množina procesů, které zpracovávaly dané video a množina detekovaných objektů v rámci daného procesu se ukládá do typu set, seznam nahraných videozáznamů, seznam URI pro Media Fragments a seznam WebVTT fragmentů daného objektu se ukládá do typu list. Cesta k videozáznamu se uloží jako řetězec pod klíčem s identifikátorem videa a cesta k souboru s WebVTT metadaty se uloží jako řetězec pod klíčem s identifikátory procesu a zpracovaného videa.

Třída `RedisStorage` obsahuje proměnnou `redis_connection`, ve které je uložen objekt reprezentující Redis. Objekt uložený v proměnné `redis_connection` obsahuje metody, které vykonávají příkazy v Redisu, jmenují se stejně jako příslušné příkazy a mají stejné parametry. Metody tohoto objektu vrací místo řetězců pole bytů, či slovník, který obsahuje klíče i hodnoty, které jsou polemi bytů a je nutné je převést na řetězce. Třída obsahuje metody `process_params`, `convert_to_string_keys_and_values`, `convert_indexed_properties_to_array` a `strings_from_byte_arrays_in_array`.

process_params Metoda převádí slovník s polemi na slovník s klíči reprezentující prvky pole, který neobsahuje pole jako hodnotu.

convert_to_string_keys_and_values Metoda převádí slovník, který obsahuje klíče a hodnoty, které jsou polemi bytů, na slovník s řetězcovými klíči i hodnotami.

convert_indexed_properties_to_array Metoda převádí slovník, který obsahuje pole jako hodnoty, na slovník, který obsahuje pouze číselné a řetězcové hodnoty. Využívá se pro uložení parametrů do Redisu jako hash bez vnořených hashů.

strings_from_byte_arrays_in_array Metoda převádí prvky v poli z polí bytů na řetězce a vrací pole, kde prvky jsou řetězce.

4.3 Webová aplikace

V této kapitole je popsáno, jak je implementována webová aplikace a kde a jak jsou použity webové technologie. Je zde popsána implementace obsluhy požadavků na získání obsahu webových stránek, přihlašování a ukládání přihlašovacích údajů pro HTTP požadavky na REST API. V neposlední řadě je zde zmíněno, jaké frameworky jsou použity pro implementaci webové aplikace.

Metody, které zpracovávají požadavky na získání webové stránky a předávají proměnné a formuláře do HTML (Hypertext Markup Language) šablon, jsou definovány v souboru *views.py* ve složce pages. Ve všech metodách kromě `login_view` se nejdříve zkontroluje, zda je v session uložen token, který se používá pro autentizaci v REST API. Pokud token ještě není uložen, uloží do session aktuální URL adresu a přesměruje uživatele na stránku s přihlašovacím formulářem. V souboru *views.py* jsou implementovány metody `login_view`, `home_view`, `processes_view`, `add_process_view` a `process_detail_view`.

login_view

Metoda zpracuje přihlašovací formulář, který se zobrazí doposud nepřihlášeným uživatelům a následně vykreslí stránku s přihlašovacím formulářem. Metoda nejdříve zkontroluje, zda je formulář odeslán tak, že ověří, zda byl požadavek odeslán HTTP metodou POST. Následně získá vyplněné přihlašovací údaje a zjistí voláním metody `authenticate` z frameworku Django, zda uživatel s takovými přihlašovacími údajemi je uložen. Pokud uživatel se zadanými přihlašovacími údaji existuje, uloží se do session token v Base64 vzniklý z přihlašovacích údajů, který se poté používá pro autentizaci do REST API a přesměruje ho na URL naposledy navštívené stránky uloženou v session. Při prvním spuštění ho to přesměruje na domovskou stránku. V opačném případě se přidá chyba do formuláře, že přihlašovací údaje nejsou platné a vykreslí se stránka s přihlašovacím údajem a chybovou hláškou.

home_view
Metoda vykreslí domovskou stránku se seznamem videí a formulářem pro vložení nového videa, je-li uživatel přihlášený. Před vykreslením domovské stránky zobrazí hlášku, že video bylo úspěšně nahráno, pokud uživatel nahrál soubor s platnou koncovkou (mp4, ogg, webm) a odeslal formulář.

processes_view
Je-li uživatel přihlášený, metoda nejdříve zjistí, zda video s daným identifikátorem je uloženo. Pokud není, vrátí mu odpověď se stavovým kódem 404 a hláškou, že takové video neexistuje. V případě, že video s takovým identifikátorem existuje, vykreslí šablonu stránky se seznamem procesů.

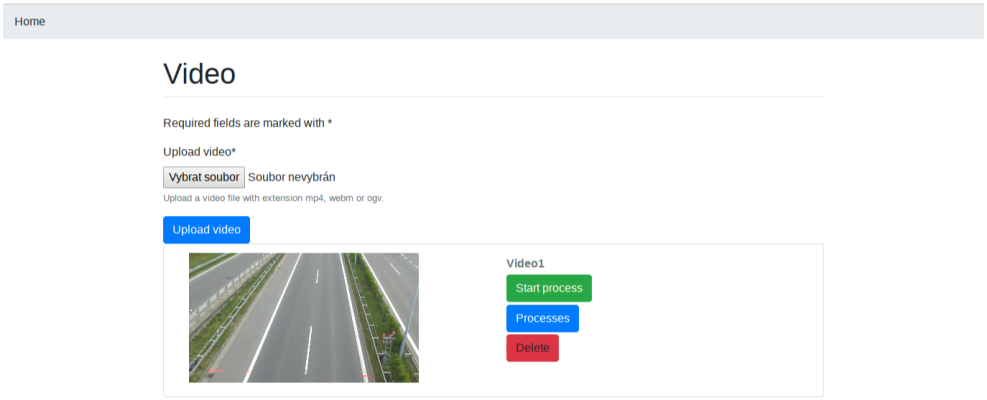
add_process_view
Pokud je uživatel přihlášen, metoda nejdříve zjistí, zda video s daným identifikátorem existuje. Pokud takové video neexistuje, vrátí mu odpověď se stavovým kódem 404 a chybovou hláškou. V opačném případě zjistí, zda je formulář odeslán stejným způsobem jako v metodě `login_view`. Následně zjistí, zda jsou údaje v daném formuláři validní. V případě, že údaje z formuláře jsou validní, přesměruje uživatele na seznam procesů a zobrazí hlášku, že proces je odeslán ke zpracování. Pokud uživatel není přesměrován, vykreslí šablonu s formulářem pro spuštění procesu zpracování videa s daným identifikátorem.

process_detail_view
Je-li uživatel přihlášený, metoda nejdříve zkontroluje, zda video a proces s danými identifikátory existují. Pokud neexistují, vrátí uživateli odpověď se stavovým kódem 404 s chybovou hláškou. Jinak získá z úložiště seznam MediaFragments URI a předá je do šablony.

V souboru *forms.py* jsou implementovány třídy, které reprezentují formuláře a používají se pro vykreslení formulářů v šablonách. Jsou zde implementovány třídy `VideoForm`, `LoginForm` a `ProcessForm`. Třída `VideoForm` reprezentuje formulář pro nahrání videa na server. Třída `ProcessForm` reprezentuje formulář pro spuštění nového procesu. Třída `LoginForm` představuje přihlašovací formulář.

HTML je generováno tak, že se zavolá metoda `render` v nějaké z metod ve views, předají se proměnné a předané proměnné se na příslušných místech v šabloně vykreslí. Webová aplikace obsahuje 5 šablon různých stránek a šablonu, která je základem pro ostatních 5 šablon.

Základní šablona je uložena v souboru base.html. V základní šabloně se importují pomocí JavaScriptové knihovny JQuery a Bootstrap a CSS framework Bootstrap. Základní šablona dále obsahuje element `title` s názvem stránky, drobečkovou navigaci, hlavní nadpis stránky v elementu s hlavním obsahem.



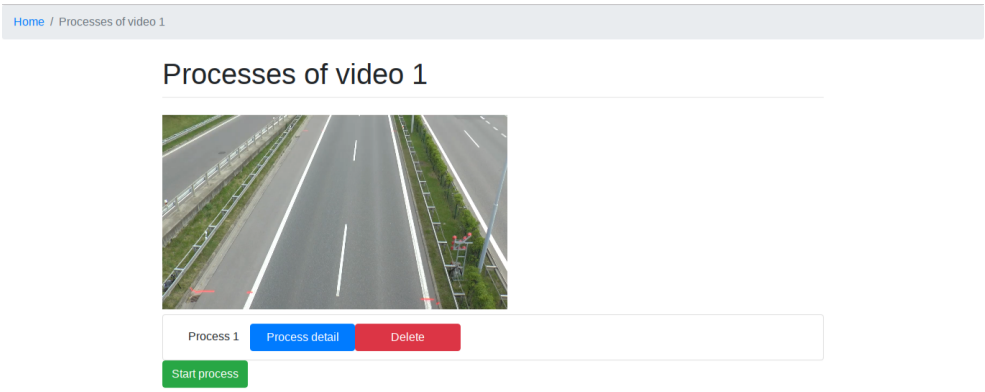
Obrázek 4.3: Ukázka domovské stránky se seznamem videí

V šabloně jsou definovány bloky *scripts*, *title*, *breadcrumb*, *heading* a *content*, které se používají v šablonách, které dědí tuto šablonu.

V potomcích této šablony se block *scripts* využívá pro vložení JavaScriptů, kde se například volají AJAXové požadavky. Block *title* se používá pro vložení titulku stránky v hlavičce. V bloku *breadcrumb* se nachází položky drobečkové navigace. Block *heading* se používá pro vložení hlavního nadpisu na danou stránku. V bloku *content* se nachází hlavní obsah stránky bez hlaviček, drobečkové navigace a hlavního nadpisu.

Šablona pro domovskou stránku je uložena v souboru *home.html*. Šablona obsahuje formulář pro vložení nového videa s CSRF (Cross Site Request Forgery) tokenem s tlačítkem pro odeslání formuláře (vygenerovaná stránka je na obrázku 4.3). Vzhled formuláře je upraven pomocí knihovny django-crispy-forms a filtru crispy. V šabloně se načtou elementy z knihovny django-crispy-forms pomocí příkazu `load crispy_form_tags`. Pod formulářem se nachází element pro seznam videí. Dále obsahuje JavaScript s AJAXovými požadavky na REST API pro načtení seznamu videí a pro nahrání videa.

JQuery zajistí, že po načtení stránky se zavolá AJAXový požadavek pro získání seznamu videí. AJAXový požadavek pro získání seznamu videí se odešle pomocí funkce `$.ajax`, kde jsou předány URL, metoda, hlavičky a callback `success`, který se zavolá po úspěšném zpracování požadavku na REST API. Hlavičky jsou nastaveny pomocí parametru `headers`, kde se nachází mapa s parametrem s autentizační hlavičkou a vyrenderovaným tokenem, předaným z views. Uvnitř callbacku `success` se volá metoda `addVideoToList`, která přidá pomocí JQuery položku s videem, jeho identifikátorem a odkazy na seznam procesů a na stránku s formulářem pro spuštění nového procesu a tlačítko pro smazání procesu.



Obrázek 4.4: Ukázka stránky se seznamem procesů

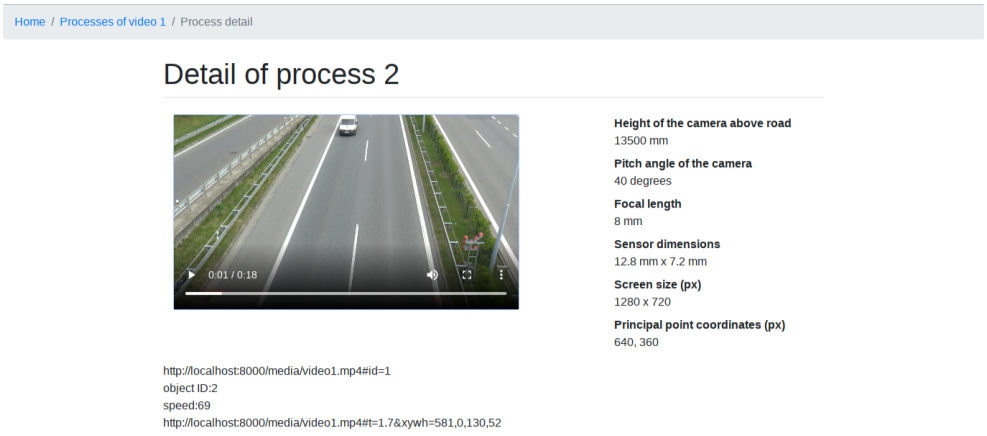
Formulář pro nahrání videa je zpracován pomocí funkce v JQuery `submit` pro selektor, který vybere formulář. Funkci `submit` je předán callback, který se zavolá po odeslání formuláře. Uvnitř callbacku se předávají pomocí třídy `FormData` data z formuláře, která se následně předají do REST API. Následně je zavolán HTTP požadavek pro vložení videa, který není asynchronní (to je zajištěno pomocí atributu `async` s hodnotou `false`).

V JavaScriptu je dále definována funkce `deleteVideo`, která zobrazí dialog pro potvrzení smazání video. Pokud uživatel potvrdí smazání video, odešle pomocí JQuery požadavek do REST API na smazání videa. Funkci `$.ajax` je nastaven callback `success`, uvnitř něhož se pomocí JQuery smaže položka s již smazaným videem.

Dále je implementováná šablona *login.html*, která obsahuje formulář pro přihlášení uživatele. Šablona pro stránku pro přihlášení uživatele obsahuje element s vyrenderovanými políčky formuláře, CSRF tokenem a tlačítkem pro odeslání formuláře.

Šablona pro zobrazení stránky se seznamem procesů pro dané video obsahuje náhled videa na URL adrese, která je předána z view, element představující seznam procesů a odkaz na stránku s formulářem pro spuštění nového procesu (lze vidět na obrázku 4.4). Dále je v šabloně skript, který odešle požadavek pomocí JQuery na REST API pro získání seznamu procesů. Po úspěsném zpracování požadavku se vytvoří pro každý proces ze získaného seznamu položka seznamu, která obsahuje název procesu, jeho stav, či odkaz na detail a tlačítko pro smazání.

Tlačítko pro smazání má pomocí atributu `onclick` nastavenou funkci `deleteProcess`, která se zavolá po kliknutí na něj. Funkce `deleteProcess` zobrazí potvrzovací dialog. V případě, že uživatel potvrdí, že chce smazat



Obrázek 4.5: Ukázka stránky s detailem procesu se zobrazenou rychlostí objektu, identifikátorem a příslušnou anotací

video, odešle AJAXový požadavek pro smazání videa a po úspěšném smazání videa v REST API se zavolá funkce, která pomocí JQuery smaže položku s již smazaným procesem.

Šablona pro zobrazení stránky s formulářem pro spuštění nového procesu pro existující video obsahuje element form se stylovaným formulářem a CSRF tokenem a skript pro zpracování formuláře. Ve skriptu se využívá knihovna JQuery pro zpracování již odeslaného formuláře. Pomocí třídy `FormData` se uloží data z formuláře a předají se do AJAXového požadavku na REST API pro spuštění procesu.

Šablona pro zobrazení stránky s detailem procesu obsahuje element pro zobrazení samotného videa, element pro zobrazení parametrů, se kterými byl proces spuštěn a element pro zobrazení rychlosti objektu na videu a příslušné Media Fragments URI s časem, pozicí a identifikátorem objektu (vygenerovanou stránku lze vidět na obrázku 4.5). Dále obsahuje skript, který po načtení stránky odešle pomocí JQuery AJAXový požadavek na REST API pro získání detailu procesu a zpracuje úspěšnou odpověď. Pokud požadavek vrátí odpověď s detailem procesu, přidá pomocí JQuery element s videem a stopou s WebVTT metadaty i parametry a jejich hodnoty v definičním seznamu. Pod element s videem a element s parametry při spuštění procesu se vloží skript, který zavolá funkci `addMetadataToVideo`. Po úspěšném zpracování požadavku se také uloží seznam Media Fragments URI do globální proměnné.

Funkce `addMetadataToVideo` najde element s videem a přidá mu eventListener, který po načtení WebVTT metadat zavolá předaný callback. V callbacku se načte element reprezentující stopu video a do proměnné se přiřadí callback, který se zavolá po změně času. Callback, který se volá po změně času ve vi-

deu, se přiřazuje pomocí události oncuechange elementu track, nebo události ontimeupdate elementu video. V tomto callbacku se načte seznam aktivních časových úseků a pro každý aktivní časový úsek se načtou příslušná metadata a převedou se z JSONu na objekt. Následně se do textu elementu pod videem přidá Media Fragments URI, odkazující na příslušný časový úsek, identifikátor objektu, jeho rychlost a Media Fragments URI s časem začátku časového úseku a pozicí daného objektu na začátku časového úseku.

Kapitola 5

Testování

V této kapitole je popsáno, jak byly otestovány knihovna, REST API a webová aplikace. V podkapitolách je popsáno, jaké testy a jak se spouští. Testy byly provedeny na notebooku Compaq s operačním systémem Debian 9. Notebook má procesor Intel(R) Core(TM) i3-2348M s frekvencí 2,3 GHz se 4 jádry a 4 GB operační paměti.

5.1 Testování funkčnosti knihovny unit testy

V této kapitole je popsáno, jakým způsobem je otestováno, zda knihovna funguje správně a vrací správné hodnoty. Funkcionalita knihovny je testována 6 různými unit testy, které testují dílčí funkcionality knihovny. Je zde popsáno také, jakými testy je co testováno, jak se testy spouští a jak dlouho trvají.

Zde je spuštěno celkem 6 unit testů, z nichž každý testuje funkcionalitu jedné třídy z knihovny. První z unit testů testuje třídu pro generování anotací, druhý testuje třídu reprezentující objekt, třetí testuje třídu reprezentující bod na obrazovce a výpočet reálných souřadnic ze souřadnic na obrazovce, čtvrtý testuje funkcionalitu třídy reprezentující obdélník s pohybujícím se objektem na snímku, pátý testuje třídu, která slouží pro výpočet rychlosti objektu a šestý testuje třídu s funkcí pro zpracování videa. Testují se zde jak správné scénáře, tak chybné scénáře jako například chybějící parametry při volání funkce pro zpracování videa, či měření rychlosti se zápornou, či nulovou snímkovou frekvencí.

První test ověřuje funkcionalitu třídy `MediaFragmentAndWebVTTAnnotation` a testuje, zda jsou anotace správně generovány, zda jich je vrácen správný počet a zda je vyhozena výjimka, pokud předaná snímková frekvence je záporná, nebo nulová. Testuje generování anotací pro jedno měření rychlosti objektu se dvěma uloženými obdélníky na 450. a 465. snímku. Test je proveden tak, že se nejdříve z těchto dvou obdélníků vypočte rychlost objektu a následně se vygenerují anotace. Ověřuje se, zda metoda pro generování anotací vygeneruje v tomto případě 2 URI pro Media Fragments a jeden třířádkový WebVTT fragment

s rychlostí objektu. Pro hodnoty parametru metody pro generování anotací představujícího snímkovou frekvenci 0 a -5, zda je vyhozena výjimka typu `ValueError`.

Druhý test ověřuje funkcionalitu třídy `Object`, reprezentující nějaký pohybující se objekt. Testuje, zda jednotlivé metody třídy reprezentující objekt vrací ty správné hodnoty. Ověřuje, zda metoda `can_measure_speed` skutečně vrací správný údaj o tom, zda může být změřena rychlost objektu, či nikoliv. Testuje se, zda metoda `can_measure_speed` vrací `False`, pokud není dostatek uložených informací o pozici a velikosti objektu, nebo právě proběhlo měření rychlosti, a zda v ostatních případech vrací `True`. Testuje se zde také, zda metoda pro zjištění, zda může být změřena rychlost daného objektu, vyhazuje výjimku typu `ValueError`, pokud předaná snímková frekvence je záporná, nebo nulová. Tento test byl proveden pro nevalidní hodnoty parametru představujícího snímkovou frekvenci 0 a -30.

Třetí test ověřuje funkcionalitu třídy `Point`, reprezentující bod na snímku. Testují se zde funkce pro výpočet reálných souřadnic a výpočet vzdálenosti dvou bodů. Ověřuje se správnost funkčnosti jak pro body, které zobrazují nějaký bod na ploše v určité vzdálenosti od kamery, tak pro bod, na kterém nemůže být zobrazena plocha a jeho reálné souřadnice x a y jsou nekonečné. V tomto testu se také ověřuje, zda je vyhozena výjimka v případě, že bod má mít libovolnou ze souřadnic x a y zápornou.

Čtvrtý test ověřuje funkcionalitu třídy `RectangleOnFrame`, reprezentující obdélník s pohybujícím se objektem na snímku. Testuje se zde funkce pro získání reálného času v sekundách, kdy se snímek s daným pořadím přehraje a na kterém se nachází daný obdélník. Ověřuje se zde také, zda je vyhozena výjimka, pokud je libovolná ze souřadnic x, y záporná, či šířka a výška je záporná, nebo nulová, nebo číslo snímku je záporné. Dále se testuje, zda je vyhozena výjimka, pokud se metoda pro získání reálného času, kdy se snímek s daným pořadím přehraje, zavolá se zápornou, či nulovou hodnotou parametru představujícího snímkovou frekvenci.

Pátý test ověřuje funkcionalitu třídy `SpeedMeasurement` a jejích potomků, které slouží pro výpočet rychlosti objektu na základě daných parametrů a uložených informací o jeho pozicích. Testuje se zde třída, reprezentující algoritmus popsaný v kapitole 2.2.2.3. Testují se zde metody pro samotný výpočet rychlosti a pro ověření, zda je uloženo dostatečné množství informací pro výpočet rychlosti. Metody pro výpočet rychlosti a pro ověření, zda je možné měřit rychlost objektu po tomto měření, jsou testovány pro různé snímkové frekvence od 20 do 50. Testuje se také, zda se při dalším zavolání metody pro výpočet rychlosti vrátí již vypočtená rychlost, která byla předtím uložena. V neposlední řadě se v tomto testu ověřuje také, zda je vyhozena výjimka, pokud se funkce pro výpočet rychlosti, či funkce pro zjištění, zda je možné vypočítat rychlost po daném měření rychlosti zavolá se zápornou, či nulovou hodnotou parametru představujícího snímkovou frekvenci.

Šestý test ověřuje funkcionalitu třídy a funkce pro zpracování videa, která

po každém výpočtu rychlosti objektu vrací vygenerované anotace. Test je proveden na videozáznamu, na kterém se pohybuje 18 vozidel a trvá 20 sekund. Zde se testuje, zda jsou při měření rychlosti generovány nějaké anotace a zda takových měření proběhlo více než 9. Nejdříve se tento test provede se správnými hodnotami parametrů. Poté se pro to samé video spustí proces s nevalidními parametry. V rámci dalších testů jsou předávány hashe s chybějícími parametry, parametry navíc, nečíselnými, nulovými, či zápornými hodnotami parametrů, jejichž hodnoty mají být kladné, nebo s parametrem, jehož hodnota má být pole a je například řetězec.

Testy se spouští pomocí příkazu `python -m unitttest discover -s tests`, který spustí všechny testy ve složce tests v aktuálním adresáři. Parametr `-s` slouží k předání složky, kde se mají hledat unit testy. Tento příkaz hledá a spouští testy v souborech s předponou test v daném adresáři. Pro implementovanou knihovnu s daným videem všechny unit testy doběhnou v pořádku v čase okolo 18,5 sekundy.

5.2 Testování REST služby pomocí scénářů

V této kapitole popisuji scénáře, které testují funkcionalitu REST API. Testuje se, zda scénáře vrací správné stavové kódy a zda odpovědi obsahují to, co mají a jsou ve správném formátu. Testují se jak scénáře, které mají proběhnout v pořádku a vrátit stavový kód 2xx, tak scénáře, které mají způsobit nějakou chybu a vrátit stavový kód 4xx. Zde je také popsáno, jak se testy spouští. Během testů je spuštěno celkem 10 různých scénářů.

První testovací scénář testuje, zda opravdu není možné získat seznam videí bez autorizace, či se špatnou autorizační metodou. Kontroluje se, zda požadavek bez hlavičky Authorization, či špatnou hodnotou hlavičky Authorization, vrátí statový kód 401 - Unauthorized.

Druhý testovací scénář testuje sekvenci operací nad videem a následně spuštěným procesem. V rámci této sekvence se nahrává a následně zpracovává videozáznam s 18 vozidly o délce 20 sekund. Tato sekvence je následující:

- Získání seznamu videí před nahráním
- Nahrání videa
- Získání seznamu videí po nahrání
- Spuštění nového procesu pro zpracování nově nahraného videa
- Získání detailu procesu 1 sekundu po spuštění
- Získání detailu procesu po jeho zpracování (simulováno čekáním po dobu 25 sekund)
- Získání seznamu objektů detekovaných v rámci procesu

- Získání detailu detekovaných objektů v různých formátech (testování nevalidních formátů)

- Smazání videa

Nejdříve se testuje, zda nahrání videa probíhá v pořádku a vrací stavový kód 201 - Created. Následně se testuje, zda je seznam videí o 1 delší, než před nahráním videa (tj. zda video je evidováno). Následně se testuje, zda je správně spuštěn nový proces a vrací se stavový kód 202. Po jedné sekundě se pošle požadavek na detail procesu a ověří se, zda se vrátí odpověď se stavovým kódem 202 - Accepted. Po dalších 30 sekundách, kdy je video již zpracováno, se ověří, zda další požadavek vrátí odpověď s informacemi o procesu a detekovanými objekty a stavovým kódem 200. Následně se testuje, zda se podaří získat seznam detekovaných objektů v rámci procesu. Pro každý detekovaný objekt se pošle požadavek na získání jeho detailu bez hlavičky Accept a poté se testuje požadavek na získání jeho detailu s hlavičkou Accept s nevalidním MIME typem. Na konci tohoto scénáře se testuje, zda se video s již spuštěným procesem a detekovanými objekty úspěšně smaže.

Třetí scénář testuje získání detailu neexistujícího videa s identifikátorem o 2 vyšším, než je nejvyšší identifikátor uloženého videa. Testuje se, zda požadavek získání videa s takto vysokým identifikátorem vrací odpověď se stavovým kódem 404 - Not Found.

Čtvrtý scénář testuje získání detailu neexistujícího procesu s identifikátorem o 2 vyšším, než je nejvyšší identifikátor uloženého a spuštěného procesu. Zde se stejně jako v druhém scénáři testuje, zda odpověď na tento požadavek bude mít stavový kód 404 - Not Found.

Pátý scénář testuje, zda při nesprávném předání identifikátoru videa a procesu se vrátí odpověď, že takový proces není nalezen. Testuje se zde požadavek na získání detailu existujícího procesu s identifikátorem existujícího videa, které tímto procesem zpracováno nebylo. Takový požadavek má vrátit odpověď se stavovým kódem 404 - Not Found.

Během pátého scénáře se nejdříve nahrají dvě videa a poté se pro první video spustí první proces a následně pro druhé video druhý proces. Po spuštění Následně se testuje, zda při předání identifikátoru prvního videa a prvního procesu, či druhého videa a druhého procesu se podaří správně získat detail procesu a zda při předání identifikátoru prvního videa a identifikátoru druhého procesu se vrátí, že takový proces nebyl nalezen. Tyto testy jsou také prováděny pro získání seznamu detekovaných objektů v rámci procesu.

Šestý scénář testuje, zda opravdu není možné spustit proces s chybějícími parametry, či parametry navíc. Pokud se pošle požadavek na spuštění procesu a chybí parametry, či naopak jsou předány nějaké parametry navíc, očekává se, že se vrátí odpověď se stavovým kódem 400 - Bad Request. V pátem scénáři se nejdříve nahraje video a poté je posláno 5 požadavků na spuštění procesů s chybějícími parametry, či parametry navíc.

Sedmý scénář testuje, zda opravdu není možné spustit proces s parametry s nečíselnou, zápornou, či nulovou hodnotou, pokud hodnotou parametru má být kladné číslo, nebo řetězcem, pokud se očekává, že hodnotou daného parametru bude pole. Testuje se zde, zda není možné spustit proces například s nulovou, či zápornou ohniskovou vzdáleností, šířkou snímku v pixelech. Očekává se, že při nesprávných hodnotách požadavek na spuštění procesu vrátí odpověď se stavovým kódem 400 - Bad request. V tomto scénáři se odešle 10 požadavků, kde mají parametry nesprávnou hodnotu (řetězec místo čísla, záporné číslo, či nula).

Osmý scénář testuje, zda se vrátí chyba, pokud máme v hlavičce Accept požadavku pro získání seznamu videí nějaký neplatný MIME typ jako například video, audio, PDF, či JavaScript. Pro několik vybraných neplatných formátu se odešle požadavek na získání seznamu videí s hlavičkou Accept a očekává se, že se vrátí odpověď se stavovým kódem 406 - Not Acceptable, který znamená, že daný MIME typ není podporován.

Devátý scénář testuje, zda se vrátí chyba, pokud máme v hlavičce Accept požadavku pro získání detailu nahraného videa nějaký neplatný MIME typ jako například text/uri-list, JavaScript, video, či obrázek. Nejdříve se nahraje video a ověří se, zda se nahrálo v pořádku. Následně se pro několik vybraných formátů testuje, zda se vrací při odeslání požadavku pro získání detailu videa s neplatným formátem v hlavičce Accept odpověď se stavovým kódem 406, pokud video s identifikátorem existuje a 404, pokud video s daným identifikátorem neexistuje.

Desátý scénář testuje, zda se vrátí chyba, pokud je v hlavičce Accept neplatný formát pro získání detailu procesu, či objektu, či seznamu procesů, či objektů detekovaných v rámci procesu. V tomto scénáři se nejdříve nahraje video a spustí se proces. Testuje se také, zda se 406 vrací jen pro existující videa, či procesy. Zde jsou definovány neplatné MIME mime typy, pro které bude ověřováno, zda požadavky vracejí odpověď se stavovým kódem 406 - Not Acceptable, pokud video a proces s danými identifikátory v URI existují a 404, pokud video, či proces s daným identifikátorem neexistuje. Pro neplatné formáty se po 30 sekundách od spuštění procesu testují tyto požadavky s neplatným MIME typem v hlavičce Accept:

- Získání seznamu procesů pro nově nahrané video
- Získání seznamu procesů pro neexistující video
- Získání seznamu objektů pro spuštěný proces
- Získání seznamu objektů pro neexistující proces
- Získání detailu spuštěného procesu
- Získání detailu neexistujícího procesu

Před spuštěním testů je potřeba spustit příkaz `python manage.py qcluster`, který spustí workery pro asynchronní zpracování videa. Tento příkaz nastartuje workery, kteří jsou připraveni zpracovávat asynchronní požadavky. Testy se spouštějí pomocí příkazu `python manage.py test`.

5.3 Testování webové aplikace

V této kapitole je popsáno testování webové aplikace jak z hlediska funkcionality, tak z hlediska použitelnosti. V první části je popsáno testování funkcionality webové aplikace manuálním průchodem. V druhé části je hodnoceno, jak je webová aplikace použitelná a jak splňuje jednotlivá kritéria Nielsenovy heuristické analýzy.

5.3.1 Testování funkcionality - manuální průchod

V této kapitole je popsáno testování funkcionality webové aplikace pomocí manuálního průchodu. Je zde popsáno, zda je webová aplikace funkční a jaká funkcionalita je testována.

V první části testu se zkouší, zda formulář pro přihlášení přesměruje uživatele na domovskou stránku jen, pokud jsou přihlašovací údaje správně vyplněné. Při testu byl otevřen prohlížeč, ve kterém webová aplikace ještě nebyla spuštěna. V takovém prohlížeči uživatel ještě není přihlášený. Při testu aplikace přesměrovala uživatele v novém prohlížeči na stránku s formulářem pro přihlášení.

Při prvním pokusu jsou zadány nesprávné přihlašovací údaje, a to konkrétně nesprávné heslo u účtu admin. První pokus skončí chybovou hláškou, že přihlašovací údaje jsou neplatné. Na druhý pokus je pro účet admin zadáno správné heslo. Po přihlášení k účtu admin se správným heslem aplikace přesměruje uživatele na domovskou stránku, kterou chtěl navštívit.

Ve druhé části testu se ověřuje, zda opravdu není možné nahrát soubor, který není videem a nemá koncovku mp4, webm, nebo ogg a zda se naopak podaří nahrát soubor, pokud se jedná o video ve správném formátu. Během testu se do formuláře nahraje soubor ve formátu PDF, textový soubor, obrázek a video s koncovkou mp4. Při nahrání souboru ve formátu PDF, textového souboru i obrázku s koncovkou jpeg se soubor nenahrál a zobrazila hláška, že soubor musí mít platnou koncovku. Při posledním pokusu se video úspěšně nahrálo a po úpravě kódu se nezobrazila žádná chybová hláška.

Ve třetí části testu se zkouší, zda je možné spustit proces jen se správnými hodnotami požadovaných parametrů. Při testu byly vyplňovány nulové a záporné hodnoty číselných parametrů, kde se očekávají kladné hodnoty. Během testu byla vyplněná záporná hodnota výšky kamery nad vozovkou, nulové rozměry snímače, nulová ohnisková vzdálenost, nulové rozměry snímku a záporné souřadnice principálního bodu. Všechny parametry, které byly nesprávně vyplněny, byly zvýrazněny a formulář nebylo možné odeslat. Poté, co byly

vyplněny kladné hodnoty všech parametrů, byl formulář úspěšně odeslán, a aplikace přesměrovala uživatele na seznam procesů. Nově vložený proces měl stav "Not started", což znamená, že ještě nebyl spuštěn. Po několika sekundách byla stránka obnovena a proces měl stav "Running", což znamenalo, že video se právě zpracovává. Po dalších zhruba třiceti sekundách byla stránka se seznamem procesů opět obnovena a u procesu se již nacházel odkaz na detail. Po rozkliknutí detailu procesu bylo možné vidět pod videem informace o pohybujícím se objektu a jeho rychlosti na základě již vygenerovaných anotací.

Ve čtvrté části testu se smaže proces přidaný v předchozí části testu a následně se přidá nový s jinými parametry. Po odeslání formuláře byl proces již ve stavu "Running". Po 20 sekundách byla přenačtena stránka a na stránce se nacházel odkaz na detail procesu, protože již doběhl. Po rozkliknutí procesu a následném spuštění videa se zobrazovaly stejné Media Fragments URI a identifikátory objektů jako v předchozím procesu, ale rychlosti se na základě parametrů lišily.

V páté části testu se nejdříve přidal proces. Po zhruba 5 sekundách od odeslání se běžící proces smazal. Následně byl opět přidán nový proces. Po zhruba 25 sekundách byla stránka se seznamem procesů opět přenačtena. Na přenačtené stránce se seznamem se již objevil odkaz na detail již přenačteného procesu. Po kliknutí na odkaz na detail procesu se spustilo video a byly opět vidět stejné Media Fragments URI a identifikátory objektů jako v předchozím procesů, ale rychlosti byly v závislosti na parametrech různé.

V šesté části testu bylo smazáno video, které bylo úspěšně přidáno v první části testu a obsahuje 2 procesy. Video se 2 procesy bylo úspěšné smazáno a příslušná položka v seznamu videí byla po kliknutí odstraněna.

V sedmé části testu bylo nahráno video a pro nově nahrané video byl spuštěn 1 proces. Toto video s jedním procesem bylo smazáno a poté bylo nahráno nové video. Pro nově nahrané video po smazání původního videa byl spuštěn opět jeden proces. Po zhruba 20 sekundách od odeslání formuláře pro spuštění procesu proces doběhl a na stránce se seznamem procesů byl odkaz pro zobrazení detailu procesu. Po kliknutí na detail procesu a spuštění videa se pod videem objevily správné identifikátory objektů a identifikátory měření rychlosti.

5.3.2 Nielsenova heuristika

V této kapitole je popsáno testování použitelnosti webové aplikace pomocí Nielsenovy heuristiky. Nielsenova heuristika je jednou z možností, jak simulováním uživatele zjistit, zda je aplikace použitelná a snadno ovladatelná. Zde je popsáno, co dané kritérium znamená a do jaké míry je implementovaná webová aplikace splňuje. Pokud aplikace z nesplňuje dané kritérium, je uveden důvod, proč ho nesplňuje.

5.3.2.1 Viditelnost stavu systému

Odstavec je převzatý ze zdroje [42]. Toto kritérium znamená, že stav systému musí být vždy viditelný. Systém musí informovat uživatele o tom, jakou akci vykonal a jaký je stav akce. Uživatel musí vědět, zda je akce již vykonána, nebo se čeká na nějaký vstup. Uživatel musí vždy vědět o tom, co vykonává, nebo právě vykonal.

Implementovaná webová aplikace toto kritérium do určité míry splňuje. Pokud uživatel spustí proces zpracování videa, jasně vidí, kdy proces ještě běží. Uživatel snadno pozná, zda proces doběhl, či nikoliv. Po vyplnění formuláře pro spuštění procesu, či nahrání videa je zřetelné, zda se video podařilo nahrát, či nikoliv. Po úspěšném odeslání formuláře se vždy objeví hláška, která uživateli říká, co se děje.

Na druhou stranu v systému není nijak zvýrazněno, že se video nahrává. Pokud uživatel nahrává video, není na první pohled vidět, že se ještě nahrává.

5.3.2.2 Propojení systému s reálným světem

V [42] je uvedeno, že propojení systému s reálným světem znamená, že systém používá slova z jazyka cílových uživatelů. Práce se systémem má připomínat úkony v reálném světě.

Implementovaná webová aplikace toto kritérium splňuje částečně. V aplikaci jsou použity slova z jazyka cílových uživatelů jako například spustit proces, či nahrát video. V detailu procesu se používají slova jako například ohnisková vzdálenost, či princiální bod. V aplikaci se ale používají i slovní jako například "detail procesu", která běžní uživatelé příliš nepoužívají. Aplikace neobsahuje žádné ikony jako například křížek, či koš pro smazání.

5.3.2.3 Uživatelská kontrola a svoboda

V [42] je uvedeno, že uživatelská kontrola a svoboda znamená, že systém musí dávat možnost uživateli vrátit se z určitého stavu zpět. Uživatel by měl mít možnost kliknout na odkaz Zpět, Storno, nebo Undo.

Webová aplikace toto kritérium splňuje toto kritérium s výhradami. Pokud uživatel například přidal video, či proces, který přidal nechtěl, může jej okamžitě smazat. Pokud uživatel spustil proces s nesprávně vyplněnými parametry, může jej ihned po spuštění smazat a nemusí čekat na jeho dokončení.

V implementované webové aplikaci se nenachází žádné odkazy jako například Undo, či Storno. Pokud se uživatel dostane na formulář pro spuštění procesu, nemá možnost smazat všechny hodnoty ve formuláři na jedno kliknutí, ale musí je ručně smazat.

5.3.2.4 Konzistence a standardy

V [42] se uvádí, že systém by měl být vzhledově i obsahově konzistentní. Popisky stejných akcí mají být stejné a jejich název se nemá lišit. V systému se má použít výchozí vzhled prvků.

Implementovaná webová aplikace toto kritérium splňuje bez problémů. Na všech stránkách se nachází drobečková navigace a pod ní nadpis. Akce pro smazání videa a akce pro smazání procesu je označena stejným termínem a tlačítka pro smazání videa, či procesu jsou červená, odkazy na přidání procesu jsou zelené a odkazy pro zobrazení seznamu procesů a pro zobrazení detailu procesu jsou modré. Tlačítka pro odeslání formuláře jsou také všude modrá.

5.3.2.5 Prevence chyb

V [42] se uvádí, že systém by neměl uživatele dostat do chybového stavu. Je vhodné nenechat uživatele úspěšně odeslat chybně vyplněný formulář. Prevence chyb se řeší pomocí potvrzovacích dialogů, či varování.

Implementovaná webová aplikace toto kritérium splňuje a nebyly nalezeny žádné problémy, související s tímto kritériem. Není možné nahrát soubor v jiném formátu, než umožňuje REST API. Pokud uživatel nahraje například PDF, či textový soubor, nepodaří se mu ho úspěšně odeslat. Vyplní-li uživatel nesprávné hodnoty pro spuštění procesu jako například zápornou, či nulovou ohniskovou vzdálenost, uživateli se nepodaří úspěšně odeslat formulář a zobrazí se mu chybová hláška.

Uživatel nemá možnost smazat video, či proces bez potvrzení. Po kliknutí na tlačítko pro smazání videa, či procesu se objeví potvrzovací dialog a video, či proces se smaže po potvrzení.

5.3.2.6 Rozpoznávání místo vzpomínání

V [42] se uvádí, že uživatel by měl být při používání systému co nejméně kognitivně zatížen. Uživatel by měl dobře vidět, kde se právě nachází a co může vykonat.

Implementovaná webová aplikace toto kritérium splňuje s výhradou. Na stránce se vždy nachází drobečková navigace, a tedy uživatel vždy ví, kde se nachází. Je-li uživatel na stránce se seznamem procesů, vidí, zda proces běží, či nikoliv. Uživatel si může zobrazit detail procesu jen v případě, že již doběhl. Uživatel ale nevidí, pod jakou identitou je přihlášen.

5.3.2.7 Flexibilní a efektivní použití

V [42] se uvádí, že aplikace by měla poskytovat uživateli dostatečné množství voleb i pro pokročilé. Vhodné je používat například klávesové zkratky či automatické doplňování hodnot pole.

Webová aplikace do určité míry splňuje toto kritérium. Umožňuje základní klávesové zkratky jako například odeslání formuláře stisknutím klávesy enter. Zjištěným problémem je, že ve formuláři pro spuštění nového procesu nejsou předvyplněny hodnoty z předtím spuštěného procesu pro to stejné video. Dalším problémem je, že i pokročilý uživatel musí vždy vyplňovat šířku a výšku snímku a souřadnice principálního bodu, které mohou být například poloviční oproti šířce a výšce snímku. V neposlední řadě uživatel nemá možnost smazat více videí, či procesů najednou a videa a procesy se musí mazat po jednom.

5.3.2.8 Estetický a minimalistický design

V [42] je uvedeno, že aplikace má poskytovat uživatelům minimální počet využívaných voleb. Další volby nemají být umisťovány na běžně používané obrazovky. Méně používané volby nemají být nabízeny, nebo mají být umístěny na méně používaných obrazovkách.

Webová aplikace nenabízí uživatelům žádné zbytečné volby. Například v seznamu videí se uživateli nabízí nahrání videa, zobrazení seznamu procesů daného videa, spuštění procesu a smazání videu. V seznamu procesů se u běžících procesů nabízí jen smazání. U procesů, které již doběhly, se nabízí jejich zobrazení a smazání. V detailu procesu se nabízí jen spuštění videa a pro každý objekt ve videu se zobrazuje jeho identifikátor, rychlost a příslušná URI pro Media Fragments s identifikátorem měření rychlosti daného objektu v aktuálním čase.

5.3.2.9 Pomoc uživatelům pochopit, poznat a vzpamatovat se z chyb

V [42] se uvádí, že systém má poskytovat uživatelům srozumitelné chybové hlášky. Z chybových hlášek má být uživateli jasné, jakou chybu udělal a jak ji může napravit.

Implementovaná webová aplikace toto kritérium splňuje bez problémů. Ve formulářích se pro každé chybně vyplněné pole zobrazuje, proč není správně vyplněno a jakou hodnotu má vyplnit.

5.3.2.10 Nápovědy a návody

V [42] je uvedeno, že je potřeba poskytnout dokumentaci, kde to uživatelé nejvíce potřebují. Nápověda se hodí například při vyplňování formulářů. Uživatel se z nápovědy dozví, co má do pole ve formuláři vyplnit.

Implementovaná webová aplikace toto kritérium splňuje s jednou výhradou. Formulář pro spuštění procesů obsahuje nápovědy pro každé políčko, ze kterých je patrné, co má uživatel vyplnit. Uživatel ve formuláři pro nahrání videa vidí, jaké koncovky souboru jsou podporovány.

Závěr

Tato práce se zaměřuje na návrh a následnou implementaci knihovny, umožňující detekci pohybujících se objektů ve videozáznamu, měření jejich rychlosti a generování anotací. Nejdříve byly analyzovány metody detekce pohybujících se objektů, algoritmy pro měření rychlosti objektů ve videozáznamů a způsoby anotace videa a jejich využití. Následně byl vybrán způsob využití anotací, který kombinuje Media Fragments a WebVTT a pro měření rychlosti objektů ve videozáznamu byl vybrán způsob, který nejdříve spočítá reálnou vzdálenost a poté na základě ní vypočte rychlost. Knihovna byla implementována v programovacím jazyce Python.

Dále bylo navrženo a implementováno REST API, které využívá funkce knihovny pro zpracování videa. Před implementací byly navrženy zdroje, operace nad zdroji, odpovědi a stavové kódy a výběr úložiště pro práci s daty o videích, procesech a detekovaných objektech, využívanými REST API. Na základě návrhu bylo implementováno REST API ve frameworku Django v Pythonu. REST API umožňuje nahrát video, získat informace o videu, spustit proces zpracování videa, v rámci něhož jsou detekovány objekty, je změřena jejich rychlost a jsou vygenerovány anotace, získat informace o procesech i detekovaných objektech.

V neposlední řadě byla navržena a implementována webová aplikace, která využívá implementované REST API. Nejdříve byly zvoleny technologie a bylo navrženo jejich použití. Poté bylo navrženo uživatelské rozhraní webové aplikace. Na základě návrhu byla implementována webová aplikace ve frameworku Django v Pythonu.

Knihovna byla otestována z hlediska funkcionality unit testy. Knihovna prošla unit testy bez problémů. REST API bylo testováno 10 scénáři, z nichž některé mají dopadnout dobře a jiné mají skončit chybou. REST API splňovalo požadovanou funkcionalitu a pro všechny scénáře vracelo správný stavový kód, odpověď i hlavičku. Webová aplikace byla manuálně otestována z hlediska funkcionality i z hlediska použitelnosti. Webová aplikace splňuje požadovanou funkcionalitu. Pro testování použitelnosti byla použita Nielsenova heuristika.

Webová aplikace splňuje většinu kritérií a zbylá kritéria splňuje s několika výhradami.

Teoretickou část práce je možné rozšířit o další algoritmy a způsoby detekce objektů. Je možné podrobněji analyzovat například měření rychlosti objektů s využitím strojového učení. Knihovnu je možné rozšířit o třídy pro další algoritmy pro výpočet rychlosti objektů ve videozáznamu jako například měření rychlosti zaznamenáváním návštěvy úseček na snímku, které jsou od sebe v určité vzdálenosti a třídy pro další způsoby anotace jako například MPEG-7.

REST API je možné upravit zejména v případě, že do knihovny budou doplněny další algoritmy. V REST API musí být v tomto případě upravena validace parametrů v těle požadavku i funkce, která zajišťuje zpracování videa. Do funkce pro zpracování videa se předá třída reprezentující předaný algoritmus měření rychlosti, která závisí na parametru v těle požadavku.

Webovou aplikaci je možné upravit tak, že formulář pro spuštění procesu bude obsahovat předvyplněné hodnoty. V případě, že v knihovně je implementováno více algoritmů, je možné do webové aplikace doplnit dynamický formulář, kde uživatel nejdříve zvolí algoritmus měření rychlosti a poté se dynamicky přidají políčka pro potřebné vstupy pro vybraný algoritmus. Dále je možné rozšířit JavaScript tak, aby byla zvýrazněna pozice objektu na aktuálním snímku například pomocí obdélníku a u daného obdélníku byla vykreslena rychlost objektu. Dále je možné ji upravit tak, že na všech stránkách bude zobrazeno uživatelské jméno aktuálně přihlášeného uživatele. V neposlední řadě je možné do webové aplikace doplnit indikátor, který uživateli říká, že se právě nahrává video. Do webové aplikace i REST API je možné doplnit registraci uživatelů s uživatelským jménem, heslem, přezdívkou, křestním jménem a dalšími údaji.

Literatura

[1] *Media Fragments URI 1.0 (basic)*. In: *World Wide Web Consortium (W3C)* [online]. [cit. 26.03.2020]. Dostupné z: https://www.w3.org/TR/media-frags/

[2] *Ontology for Media Resources 1.0*. In: *World Wide Web Consortium (W3C)* [online]. Copyright © 2012 [cit. 15.05.2020]. Dostupné z: https://www.w3.org/TR/mediaont-10/

[3] Yunjia LI, Mike WALD a další. (2012). *Synote: Weaving Media Fragments and Linked Data*. CEUR Workshop Proceedings. 937.

[4] *Protocol for Media Fragments 1.0 Resolution in HTTP*. In: *World Wide Web Consortium (W3C)* [online]. [cit. 26.03.2020]. Dostupné z: https://www.w3.org/TR/2011/WD-media-frags-recipes-20111201/

[5] *Accessibility/Video Media Fragments*. In: *MozillaWiki*. [online]. Dostupné z: https://wiki.mozilla.org/Accessibility/Video_Media_Fragments

[6] *WebVTT: The Web Video Text Tracks Format*. In: *World Wide Web Consortium (W3C)* [online]. ©2019 [cit. 26.03.2020]. Dostupné z: https://www.w3.org/TR/webvtt1/

[7] *Use cases and requirements for Media Fragments*. In: *World Wide Web Consortium (W3C)* [online]. Dostupné z: https://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-reqs

[8] *Synchronized Multimedia Integration Language (SMIL 3.0)*. In: *World Wide Web Consortium (W3C)* [online]. Copyright © 2008 [cit. 26.03.2020]. Dostupné z: https://www.w3.org/TR/SMIL3/

[9] *DocEng 2011: Timesheets - When SMIL Meets HTML5 and CSS3 - YouTube*. In: *YouTube* [online]. Dostupné z: https://www.youtube.com/watch?v=VKxDB4NHWdQ

[10] CAZENAVE, Fabien, Vincent QUINT a Cécile ROISIN, Cécile. *Timesheets.js: When SMIL Meets HTML5 and CSS3*. DocEng 2011, 11th ACM Symposium on Document Engineering, Sep 2011, Mountain View, United States. 10 p. ffhal-00619382

[11] *Use cases and requirements for Media Fragments*. In: *World Wide Web Consortium (W3C)* [online]. Dostupné z: `https://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-reqs/#MPEG-21`

[12] DAY, Neil a José M. MARTÍNÉZ. *Introduction to MPEG-7*. In: *World Wide Web Consortium (W3C)* [online]. [cit. 26.03.2020]. Dostupné z: `https://www.w3.org/2001/05/mpeg7/w4032.doc`

[13] *Use cases and requirements for Media Fragments*. In: *World Wide Web Consortium (W3C)* [online]. [cit. 22.05.2020]. Dostupné z: `https://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-reqs/#MPEG-7`

[14] *MPEG-7 and the Semantic Web*. In: *World Wide Web Consortium (W3C)* [online]. Dostupné z: `https://www.w3.org/2005/Incubator/mmsem/XGR-mpeg7/`

[15] CANU, Sergio. *Background Subtraction - OpenCV 3.4 with python 3 Tutorial 32 - Pysource*. In: *Learn computer vision with Opencv and Python - Pysource*. [online]. Copyright © Pysource LTD 2017 [cit. 22.05.2020]. Dostupné z: `https://pysource.com/2018/05/17/background-subtraction-opencv-3-4-with-python-3-tutorial-32/`

[16] *OpenCV: Cascade Classifier*. In: *OpenCV documentation index* [online]. Dostupné z: `https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html`

[17] MQ . *Lekce 10 - Detekce vlastních objektů v obrázku pomocí Haar Cascade*. In: *itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další*. [online]. Copyright © 2020. [cit. 26.03.2020]. Dostupné z: `https://www.itnetwork.cz/python/video/detekce-vlastnich-objektu-v-obrazku-pomoci-haar-cascade`

[18] VIOLA, Paul a Michael JONES. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Copyright ©2001 [cit. 26.03.2020]. Dostupné z: `https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf`

[19] *Object Detection with ImageAI in Python*. In: *Stack Abuse*. [online]. Copyright © 2020. [cit. 22.05.2020]. Dostupné z: `https://stackabuse.com/object-detection-with-imageai-in-python/`

[20] HÁJEK, Pavel, Bc. *Odhad rychlosti automobilů ve videu*. Brno. 2017. 55 s. Diplomová práce. VUT. Fakulta informačních technologií

[21] MAKWANA, Bhagyashri Ms. *Moving Vehicle Detection and Speed Measurement in Video Sequence* [online]. Copyright ©2013. Dostupné z: https://www.ijert.org/research/moving-vehicle-detection-and-speed-measurement-in-video-sequence-IJERTV2IS100920.pdf

[22] HUANG, Tingting. *Traffic Speed Estimation from Surveillance Video Data* [online]. Copyright ©2018. Dostupné z: http://openaccess.thecvf.com/content_cvpr_2018_workshops/papers/w3/Huang_Traffic_Speed_Estimation_CVPR_2018_paper.pdf

[23] *Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation.* In: *OpenCV documentation index* [online]. Copyright ©2011 - 2014 [cit. 26.03.2020]. Dostupné z: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[24] MYŠKA, Milan. *Rekonstrukce 3D modelu ze dvou snímků jedné scény.* Brno. 2014. 52 s. Bakalářská práce. VUT. Fakulta elektrotechniky a komunikačních technologií. Vedoucí diplomové práce Ing. Martin Hasmanda

[25] *Distortion Models.* In: *imatest — Image Quality Testing Software & Test Charts* [online]. Copyright ©2004 [cit. 26.03.2020]. Dostupné z: https://www.imatest.com/support/docs/pre-5-2/geometric-calibration/distortion-models/

[26] DOBROVOLNÝ, Petr. *FOTOGRAM_04* In: *Studijní materiály předmětu PřF:Z8101* [online]. Copyright @2005. Dostupné z: https://is.muni.cz/el/1431/jaro2005/Z8101/um/FOTOGRAM_04.pdf

[27] DUBSKÁ, Markéta, Jakub SOCHOR a Adam HEROUT. *Automatic Camera Calibration for Traffic Understanding* [online]. Copyright ©2014. Dostupné z: http://www.bmva.org/bmvc/2014/files/paper013.pdf

[28] DUBSKÁ, Markéta a další. *Fully Automatic Roadside Camera Calibration for Traffic Surveillance* [online]. Copyright @2014. Dostupné z: https://medusa.fit.vutbr.cz/traffic/data/papers/2014-IEEE-ITS-CameraCalibration.pdf

[29] ZHANG, Zhaoxiang a další. *Practical Camera Auto-Calibration Based on Object Appearance and Motion for Traffic Scene Visual Surveillance* [online]. Copyright @2008. Dostupné z: http://www.nlpr.ia.ac.cn/2008papers/gjhy/gh19.pdf

[30] DHAKAL, Bed. *Homogeneous coordinate.* In: *Share and Discover Knowledge on LinkedIn SlideShare* [online]. Copyright ©2013 [cit. 26.03.2020]. Dostupné z: https://www.slideshare.net/beddhakal9/homogeneous-coordinate

[31] NIE, Neil. *NeilNie/speed_estimation: Using deep learning to predict the speed of a moving vehicle.*. In: *GitHub* [online]. [cit. 03.04.2020]. Dostupné z: `https://github.com/NeilNie/speed_estimation`

[32] LUVIZON, Diogo, Bogdan NASSU a Rodrigo MINETTO. (2016). *A Video-Based System for Vehicle Speed Measurement in Urban Roadways. IEEE Transactions on Intelligent Transportation Systems*. PP. 1-12. 10.1109/TITS.2016.2606369.

[33] *OpenCV Vehicle Detection, Tracking, and Speed Estimation - PyImageSearch*. In: PyImageSearch - You can master Computer Vision, Deep Learning, and OpenCV. [online]. Copyright ©2020 [cit. 03.04.2020]. Dostupné z: `https://www.pyimagesearch.com/2019/12/02/opencv-vehicle-detection-tracking-and-speed-estimation/`

[34] ALEXIOU, John. *Using vector math to get point on perpendicular line from a point with the same Y*. In: *StackExchange* [online]. Dostupné z: `https://math.stackexchange.com/questions/2322333/using-vector-math-to-get-point-on-perpendicular-line-from-a-point-with-the-same`

[35] REYNOLDS, Paul. *pfr/VideoTracker: Track vehicles on one lane one way, or two lane bidirectional streets; record direction, compute speed and size*. In: *GitHub* [online]. [cit. 26.03.2020]. `https://github.com/pfr/VideoSpeedTracker`

[36] MOULANA, Azkar. *azkarmoulana/Speed-Detector: Vehicle speed detecting application using Java, openCV, php, MySQL*. In: GitHub [online]. [cit. 26.03.2020]. Dostupné z: `https://github.com/azkarmoulana/Speed-Detector`

[37] SOCHOR, Jakub a JURÁNEK Roman. *Automatic camera calibration from video*. In: *Faculty of Information Technology* [online]. Copyright © 2020 [cit. 22.05.2020]. Dostupné z: `https://www.fit.vut.cz/research/product-file/517/calibration.zip`

[38] DUBSKÁ Markéta a další. *Fully Automatic Roadside Camera Calibration for Traffic Surveillance. IEEE Transactions on Intelligent Transportation Systems*. 2014, vol. 2014, no. 1, pp. 1-10. ISSN 1524-9050.

[39] DUBSKÁ Markéta a HEROUT Adam. *Real Projective Plane Mapping for Detection of Orthogonal Vanishing Points*. In: *Proceedings of BMVC 2013. Bristol: The British Machine Vision Association and Society for Pattern Recognition*, 2013, pp. 1-10. ISBN 1-901725-49-9.

[40] ČÁPKA, David. *Lekce 1 - MySQL krok za krokem: Úvod do MySQL a příprava prostředí*. In: *itnetwork.cz - Ajťácká sociální síť a materiálová*

základna pro C#, Java, PHP, HTML, CSS, JavaScript a další. [online]. Copyright © 2020. [cit. 03.05.2020]. Dostupné z: `https://www.itnetwork.cz/mysql/mysql-tutorial-uvod-a-priprava-prostredi`

[41] *Introduction to Redis - Redis*. In: *Redis*. [online]. [cit. 03.05.2020]. Dostupné z: `https://redis.io/topics/introduction`

[42] NIELSEN, Jakub. *Usability Engineering*. San Diego: Academic Press. 362 s. ISBN 0-12-518406-9

[43] *Praktický úvod do Redis (3): cluster – Cloudsvět*. In: *Cloudsvět*. [online]. [cit. 21.05.2020]. Dostupné z: `https://www.cloudsvet.cz/?p=258`

Příloha **A**

Seznam použitých zkratek

URI Uniform Resource Identifier

RDF Resource Description Framework

OAC Open Annotation Collaborative

WebVTT Web Video Text Tracks

SMIL Synchronized multimedia Language

MPEG Moving Picture Experts Group

XML Extensible Markup Language

SIFT Scale Invariant Feature Transform

SURF Speeded up robust features

MEX MATLAB executable

GUI Graphical user interface

REST Representational state transfer

API Application programming interface

CRUD Create, Read, Update, Delete

JSON JavaScript object notation

MIME Multipurpose Internet Mail Extensions

CSV Comma Separated Values

SQL Structured Query Language

Redis Remote Dictionary Server

RDF Resource Description Framework

IRI Internationalized Resource Identifier

AJAX Asynchronous JavaScript and HTML

HTML Hypertext Markup Language

CSRF Cross Site Request Forgery

Příloha B

Návod k instalaci

V této příloze je popsáno, jak je možné nainstalovat aplikaci pro měření rychlosti objektů ve videozáznamu včetně jejích závislostí. Je zde popsáno, jak se instaluje knihovna, REST API a webová aplikace a jak se importuje knihovna do REST API a webové aplikace. Zde je popsán návod, jak instalovat knihovnu, REST API a webovou aplikaci a jak spustit REST API a webovou aplikaci.

Nejdříve je potřeba stáhnout zdrojové kódy z repozitáře `https://gitlab.fit.cvut.cz/supdanie/library-video-processing-and-speed-measurement` pomocí příkazu

```
git clone https://gitlab.fit.cvut.cz/supdanie/library-video-processing-and-speed-measurement.git
```

Do aktuálního adresáře se nahraje složka se stejným názvem jako repozitář na serveru (poslední část path). Chceme-li spustit webovou aplikaci, či REST API, je potřeba ještě nahrát zdrojové kódy z repozitáře `https://gitlab.fit.cvut.cz/supdanie/rest-api-and-web-application` příkazem

```
git clone https://gitlab.fit.cvut.cz/supdanie/rest-api-and-web-application.git
```

Poté si vytvoříme virtuální prostředí pomocí příkazu `python -m venv env`, nebo příkazu `virtualenv env`. Vytvořené prostředí pro Python, kam se instalují knihovny třetích stran, aktivujeme příkazem `source env/bin/activate`.

Po aktivaci prostředí přejdeme do adresáře `library-video-processing-and-speed-measurement` příkazem

```
cd library-video-processing-and-speed-measurement
```

Následně nainstalujeme knihovnu pomocí příkazu `python setup.py install`. Tento příkaz vygeneruje adresář s koncovkou egg-info, kde se nacházejí

vygenerované soubory s informacemi o závislostech, souborech atd.

Pro instalaci knihovny do REST API a webové aplikace je potřeba nejdříve se přemístit do adresáře s relativní cestou `../rest-api-and-web-application` vůči adresáři `library-video-processing-and-speed-measurement`, odkud byla instalována knihovna. Toto přemístění do adresáře s REST API a webovou aplikací se dá provést příkazem

```
cd ../rest-api-and-web-application
```

. V podadresáři `api/api` adresáře s REST API a webovou aplikací je potřeba vytvořit adresáře annotations a media. Pro instalaci závislostí do REST API potřebujeme mít nainstalovaný nástroj Exiftool.

Následně nainstalujeme potřebné závislosti pro REST API a webovou aplikaci příkazem `pip install -r api/requiremenets.txt` a knihovnu pomocí příkazu `pip install video_processing_library`. Před samotným spuštěním REST API a webové aplikace je potřeba spustit Redis server pomocí příkazu „`sudo systemctl start redis` na Linuxu, nebo `brew services start redis` na macOS.“[43].

Je-li aplikace nahraná jinam, než na localhost s portem 8000, je potřeba v kódu constants.py v adresáři `api/api` změnit konstanty `SERVER_URL`, `API_URL` a `MEDIA_URL` tak, aby místo `http://localhost:8000/` obsahovaly adresu a port serveru, odkud má být aplikace dostupná. Nahráváme-li na Apache, je potřeba přidat řádek `WSGIPassAuthorization On` do konfiguračního souboru wsgi.conf v adresáři mods-available, který se nachází v adresáři s konfigurací Apache. Apache bez přidání tohoto řádku filtruje hlavičku Authorization. Tento řádek povolí posílání hlavičky Authorization na Apachi a umožní tak autentizaci uživatele v REST API. Soubor wsgi.conf se nachází v adresáři `/etc/apache2/mods-available`, pokud je absolutní cesta k adresáři s konfigurací Apache `/etc/apache2`.

Před spuštěním REST API a webové aplikace je potřeba ještě spustit cluster pro knihovnu DjangoQ s workery příkazem `python api/manage.py qcluster`. Nakonec spustíme server příkazem `python api/manage.py runserver`, pokud nemáme aplikaci nahranou na Apache.

Příloha C

Implementace algoritmů pro měření rychlosti objektů ve videozáznamu

V této příloze jsou podrobně popsány vybrané existující cizí implementace algoritmů popsaných v kapitole 2.2.2. U každé implementace jsou rozebrány vybrané implementační detaily, zajímavé funkce a metody.

C.1 Implementace měření rychlosti zaznamenáváním průjezdu úsečkou - VehicleSpeedTracker

V této kapitole je podrobněji popsána existující implementace algoritmu, který měří rychlost objektů zaznamenáváním průjezdu úsečkami. Implementace je stručně popsána v kapitole 2.2.3.1. Zde jsou popsány jednotlivé třídy a metody včetně ukázek kódu.

Spouštěcí soubor videoSpeedTracker.cpp obsahuje funkce `main()`. Ve funkci `main()` se volá funkce `setup()`, která načítá konfigurační soubor s parametry jako je například cesta k souborům s videozáznamy, či k IP kameře. Funkce `setup()` umožňuje načíst parametry z konfigurace. Po načtení konfigurace a parametrů funkce `main()` provede cyklus, který zajišťuje postupné zpracování videozáznamů na základě konfigurace a dalších vstupů od uživatele jako je například informace, zda chce uživatel z daného adresáře zpracovat všechna videa, jedno video, či žádné video.

Listing C.1: Cyklus, který prochází všechna videa, která má zpracovat

```
bool moreFilesToDo = true;

int main(){
        setup();
```

```
        while (moreFilesToDo){
                if (yesNoAll == *){
                        if (getline(filesList, fileName)){
                        } else{
                                moreFilesToDo = false;
                                break;
                        }
                } else {
                        moreFilesToDo = false;
                        filesList.close();
                }
        }
}
```

Uvnitř tohoto cyklu se také volají funkce knihovny OpenCV, která zajišťuje odečítání aktuálního snímku od předchozího a zobrazení snímku. Pomocí prahování (thresholding) získáme reprezentaci snímku, kde popředí je bílé a pozadí černé. To znamená, že pokud hodnota pixelu je nad daným prahem, hodnoty se nastaví na maximum, jinak se hodnoty RGB nastaví na 0. Následně se odstraní šum a prahování se provede podruhé. Po odečtení pozadí a dvojím prahováním se volá funkce `manageMovers()`, která zajišťuje samotnou detekci pohybující se objektů, ukládání informací o objektech a jejich rychlosti.

Listing C.2: Procházení snímků a odečítání pozadí

```
while (capture.get(CV_CAP_PROP_POS_FRAMES) <
        capture.get(CV_CAP_PROP_FRAME_COUNT)){
        capture.read(frame1);
        cv::cvtColor(frame1, grayImage1, COLOR_BGR2GRAY);
        capture.read(frame2);
        cv::cvtColor(frame2, grayImage2, COLOR_BGR2GRAY);
        cv::absdiff(grayImage1, grayImage2, differenceImage);
        cv::threshold(differenceImage, thresholdImage, g.
            SENSITIVITY_VALUE, 255, THRESH_BINARY);
        cv::blur(thresholdImage, thresholdImage, cv::Size(g.
            BLUR_SIZE, g.BLUR_SIZE));
        cv::threshold(thresholdImage, thresholdImage, g.
            SENSITIVITY_VALUE, 255, THRESH_BINARY);
        objectDetected = manageMovers(thresholdImage, ROIFr2);
}
```

Funkce `manageMovers()` slouží k získávání informací o pohybujících se objektech ve videozáznamu a jejich rychlostech. Ve funkci manageMovers() se zjišťují optimální projekce vozidel na aktuálním snímku pomocí obdélníku. Optimální projekce vozidla je možné získat jako návratovou hodnotu metody `getBest-Projection()` ve třídě VehicleDynamics.

Třída `Globals` reprezentuje samotnou konfiguraci aplikace. Slouží pro ukládání parametrů konfigurace a načítání konfigurace ze souboru VTS.cfg. Samotné načítání parametrů z konfiguračního souboru VTS.cfg zajišťuje její metoda `readConfig()`, která nemá žádné parametry a vrací hodnotu typu bool,

reprezentující, zda se podařilo správně načíst konfigurační soubor. V metodě `readConfig()` se načítají parametry jako je cesta k souborům, či IP kameře, či souřadnice úseček, mezi kterými se zkoumá pohyb vozidel.

Třída `Globals` dále obsahuje pomocnou metodu `getSides()`, která umožňuje načíst řádek konfiguračního souboru, obsahující parametr a jeho hodnotu. Parametrem metody je řetězec, reprezentující řádku konfiguračního souboru. Pokud se podaří získat parametr a jeho hodnotu z daného řetězce, metoda vrátí hodnotu `true`. V ostatních případech metoda vrátí hodnotu `false`, která reprezentuje, že se nepodařilo získat z daného řetězce název parametru a jeho hodnotu.

Funkce nejdříve načte celou řádku z konfiguračního souboru do řetězce, který reprezentuje danou řádku. Následně bude na vstupu načtená řádka a načte se začátek řádky před prvním znakem = a druhá část řádky mezi prvním znakem = a prvním znakem #. Znak = zde symbolizuje přiřazení hodnoty nějakému klíči a znak # znamená, že ve zbytku řádky se nachází komentář.

Listing C.3: Funkce getSides(), která kontroluje validitu řádku z konfiguračního souboru

```
bool getSides(string inLine){
        string tempLHS, tempRHS;
        istringstream configLine(inLine);
        getline(configLine, tempLHS, '=');
        getline(configLine, tempRHS, '#');
        lhs = tempLHS;
        rhs = tempRHS;
        return true;
}
```

Třída VehicleDynamics reprezentuje samotný pohyb vozidla ve videozáznamu. Ukládá si informace o tom, jakou rychlosti vozidlo projelo, na jakém snímku vozidlo projelo první úsečkou a na jakém snímku projelo druhou úsečkou a kolik snímků dané vozidlo projíždělo mezi těmito dvěma úsečkami. Obsahuje metodu `computeFinalSpeed()`, která slouží k výpočtu rychlosti daného vozidla na základě předtím odhadované rychlosti, číslem snímku průjezdu první přímkou a číslem snímku průjezdu druhou přímkou.

Dále obsahuje metodu `estimateNextVehicleData()`, která slouží k aktualizaci informací o daném vozidle na snímku. Aktualizuje například obdélník, představující vozidlo na aktuálním snímku, pozici přední části vozidla a další údaje a zajišťuje výpočet rychlosti vozidla, pokud vozidlo projelo druhou úsečkou. Parametry metody je číslo snímku a objekt, ve kterém je uložená konfigurace, načtená ze souboru VTS.conf. Metoda vrací prvek výčtu `statusTypes`, který reprezentuje, zda se mají dále aktualizovat informace o vozidle a čekat, až projede druhou úsečkou, vymezující konec měřeného úseku. Uvnitř této metody se také volá metoda `computeFinalSpeed()`, jejíž návratová hodnota bude hodnotou proměnné, která udává rychlost vozidla. Zde se také kontroluje,

zda již byla rychlost vozidla změřená. Pokud již byla změřena rychlost vozidla, souřadnice přední části vozidla se dopočítají.

Parametry metody jsou objekt třídy `Globals`, prvek výčtu `direction` a celočíselné proměnné trackStartFrame a trackEndFrame. Objekt třídy `Globals` obsahuje proměnnou CalibrationFramesL2R, která udává, kolik snímků se bude pohybovat vozidlo jedoucí 25 MPH zleva doprava mezi dvěma danými úsečkami, a proměnnou CalibrationFramesR2L, která udává, kolik snímků se bude pohybovat vozidlo jedoucí 25 MPH mezi dvěma danými úsečkami zprava doleva. Výčtový typ `direction` reprezentuje možný směr pohybu vozidla. Tyto proměnné jsou pevně dané, pokud uživatel nezadá jejich hodnotu před zpracováním videozáznamů. Proměnná trackStartFrame a trackEndFrame vymezují, od kdy do kdy se vozidlo pohybovalo mezi dvěma danými úsečkami. Metoda obsahuje další parametry, ale ty nejsou v ukázce znázorněny.

Metoda vypočítá rychlost s ohledem na daný směr pohybu vozidla. Pokud vozidlo jede zleva doprava (tj. je-li daný prvek výčtu `direction` `L2R`), vypočte se poměr mezi hodnotou proměnné `CalibrationFramesL2R` ze třídy `Globals` a počtem snímků, po který se vozidlo pohybovalo mezi levou a pravou úsečkou na snímku. Tento poměr se vynásobí číslem 25, které udává maximální povolenou rychlost v MPH. Je-li dán prvek výčtu `direction` `R2L`), rychlost se vypočte podobným způsobem. Jen se místo proměnné `CalibrationFramesL2R` použije proměnná `CalibrationFramesR2L`.

Listing C.4: Funkce computeFinalSpeed(), která vypočítává rychlost objektu

```
int computeFinalSpeed(Globals& g, direction dir, int
    trackStartFrame, int trackEndFrame){

switch (dir){
        case L2R:
        return int(((double(g.CalibrationFramesL2R) / double(
            trackEndFrame - trackStartFrame)) * 25.0) + 0.4999);
        case R2L:
        return int(((double(g.CalibrationFramesR2L) / double(
            trackEndFrame - trackStartFrame)) * 25.0) + 0.4999);
        return -1;
}
```

V neposlední řadě je v této třídě definována metoda `getBestProjection()`, která nachází nejvhodnější projekci daného vozidla v daném snímku. Dále se kontroluje, která vozidla již projela druhou úsečkou na snímku a mají se vymazat z kolekce vozidel na aktuálním snímku. Po vymazání vozidel, která projela druhou úsečkou, se vymažou také vozidla, která předjíždí například cyklistu. Poté se zkoumá pohyb vozidel na snímku, která jsou mezi první a druhou úsečkou, umístěných v určité vzdálenosti od sebe. Tato funkce volá funkce `displayAnalysisGoingRight()` a `displayAnalysisGoingLeft()`, které vypisují informace o rychlosti vozidla na obrazovku, do snímku a ukládají snímek do objektů třídy VehicleDynamics, reprezentujících jednotlivá vozidla.

Tato třída také umožňuje ukládat si matice, které reprezentují snímek, na kterém se dané vozidlo nachází. Pro ukládání maticové reprezentace snímku se používá metoda `saveFrame()`. Další metoda `getSavedFrame()`, jejíž parametr je index snímku, slouží k získání maticové reprezentace snímku s daným indexem. Metoda `holdFrame()` slouží k uložení maticové reprezentace posledního snímku. Uložený poslední snímek je možné získat jako návratovou hodnotu metody `getHeldFrame()`.

Třída `Snapshot` slouží pro reprezentaci pohybujícího se objektu na daném snímku. Umožňuje ukládát informace o tom, v jakém obdélníku se nachází pohybující se objekt na daném snímku. Používá se při vyznačení pohybujících se objektů. Třída neobsahuje žádné výpočty a slouží jako struktura pro ukládání dat o objektech na snímku.

Implementace obsahuje třídu `Projection`, která ukládá údaje o vozidle jako je obdélník, ve kterém se nachází, jeho rychlost, číslo snímku a jeho stav, reprezentovaný prvkem výčtu `statusTypes`. Tato třída v sobě neobsahuje žádné výpočty a jen ukládá tyto parametry. Jedná se o strukturu, která umožňuje uložit informace o vozidle na snímku, jeho rychlosti a stavu.

Tato implementace umožňuje měřit rychlost vozidel, jejichž rychlost nepřekračuje určitou mez, která záleží na vzdálenosti kamery od vozovky. Pro vzorový videozáznam je možné měřit rychlost vozidel, jejichž rychlost nepřekračuje 71 mph. Pro tuto implementaci je vhodné mít kameru umístěnou dostatečně daleko od vozovky, pokud je rychlostní limit vysoký. Problémem pro tuto implementaci mohou být vozidla, která jedou několikanásobně rychleji, než je povolený rychlostní limit. Tato implementace nemusí odhalit vozidla, která jedou například 120 km/h tam, kde je rychlostní limit 40 km/h.

Tato implementace není použitelná pro videozáznamy frekventovaných a dlouhých úseků. Vzhledem k tomu, že každý objekt reprezentující vozidlo má v sobě uloženou kolekci snímků, je pro takové videozáznamy velmi paměťově náročná. Implementace není vhodná pro videozáznamy, na kterých se objevují kolony vozidel, či dlouhý úsek velmi frekventované silnice.

C.2 Implementace měření rychlosti zaznamenáváním průjezdu úsečkou - Speed detector

V této kapitole je podrobněji popsána existující implementace algoritmu, který měří rychlost objektů zaznamenáváním průjezdu úsečkami. Implementace je stručně popsána v kapitole 2.2.3.2. Zde jsou popsány jednotlivé třídy a metody včetně ukázek kódu.

Třída GUI slouží k vytvoření a zobrazení okna, ve kterém je možné si přehrát vstupní video i měřit rychlosti vozidel v něm. Tato třída reprezentuje samotné okno s tlačítky pro přehrání videa, či pozastavení videa. Okno obsahuje také needitovatelná textová pole s počtem osobních automobilů, počtem dodávek

a počtem nákladních vozů a průměrné rychlosti jednotlivých druhů vozidel (osobní automobil, dodávka, nákladní automobil). V tomto okně se také zobrazuje samotný videozáznam s vozidly uvnitř ImageView. V ImageView si uživatel vybere koncové body první úsečky, kterou vozidla projedou dříva, a koncové body druhé úsečky. Tato třída také zajišťuje odhadování rychlosti vozidel, která projedou druhou úsečkou.

Třída `GUI` obsahuje metodu `init()`, která zajišťuje vytvoření okna, tlačítek pro nahrání, či uložení videa a pro definování první a druhé úsečky, mezi nimiž se po určitou dobu pohybují objekty. Vytvoření okna, tlačítek a definice úseček je zajištěna pomocí volání metody `initGUI()` a volání metodu `createJFrame()` uvnitř metody `initGUI()`. Po zavolání metody `initGUI()` proběhne smyčka, která čeká na videozáznam od uživatele. Pokud je videozáznam načtený, zapisují se údaje o rychlosti objektů do souboru ve formátu xls, či do souboru ve formátu csv. Následně se vytvoří vlákno, které spustí cyklus, zajišťující přehrávání videa, zjišťování rychlosti objektů a jejich zobrazování. Tento cyklus je definován ve vnitřní třídě Loop.

Listing C.5: Ukázka vytvoření vlákna, které reprezentuje hlavní smyčku, ve které se zpracovává video

```
public void init(){
        initGUI();
        // obsluha tlacitek a~pripadny zapis informaci o~
            vozidlech do souboru
        Thread mainLoop = new Thread(new Loop());
        mainLoop.start();
}
```

Metoda `createJFrame()` vytváří instanci třídy `JFrame`, která vytvoří okno, v němž je možné přehrát video i zobrazit rychlosti objektů. Metoda také zajišťuje přidání titulků s informacemi, jako je reálná vzdálenost mezi zvolenými úsečkami. Následně se zavolá metoda `reset()`, aby uživatel měl možnost resetovat video. Po přidání možnosti resetovat video. se zajistí načtení videa po stisknutí příslušného tlačítka pomocí volání metody `loadFile()` a uložení videa po stisknutí tlačítka pro uložení voláním metody `saveFile()`. V neposlední řadě také zajišťuje přidání možnosti uživateli myší zvolit body první čáry pomocí volání metody `selectCountingLine()` a přidání možnosti zvolení dvou bodů druhé čáry pomocí volání metody `selectSpeedingLine()`.

Metoda `reset()` zajišťuje resetování videa a následné spuštění vlákna s objektem třídy `Reseting` s metodou `run()`. V této metodě se znovu načítá videozáznam a první snímek. Zde se také vynulují souřadnice koncových bodů první a druhé úsečky, kterými projíždějí vozidla. V neposlední řadě se vynulují počet osobních automobilů, počet dodávek a počet nákladních vozů a vynulují se průměrné rychlosti pro každou kategorii vozidel. Nakonec se vytvoří a následně spustí vlákno ze třídy Reseting.

Listing C.6: Ukázka vytvoření vlákna, které spustí samotný proces resetování videa včetně vynulování zadaných parametrů od uživatele jako jsou koncové

body dvou úseček, mezi kterými projíždí vozidla a jsou od sebe vzdáleny pevně daných 6 metrů

```
private void reset(JFrame frame) {
        lineCount1 = null;
        lineCount2 = null;
        lineSpeed1 = null;
        lineSpeed2 = null;
        // resetovani promennych, ktere se~pouzivaji
        // pri vypoctu rychlosti vozidel
        // a~prumernych rychlosti vozidel dle typu
        // (osobni auta, dodavky, nakladni vozy)
        Thread reseting = new Thread(new Reseting());
        reseting.start();
}
```

Metoda `loadFile()` přidává pole pro zvolení a nahrání videa pro zpracování. Tato metoda nic nevrací a jejím parametrem je objekt třídy `JFrame`, reprezentující již vytvořené okno. Následně je definována akce, která se spustí po kliknutí na tlačítko pro nahrání videa. Tato akce načte video do objektu třídy VideoCapture z knihovny OpenCV, zjistí jeho snímkovou frekvenci a načte první snímek videa.

Metoda `selectCountingLine()` přidává tlačítko, které umožňuje uživateli zvolit si myší dva koncové body první úsečky. Je zde přidána akce po stisknutí tlačítka, která umožňuje uložení vybraných souřadnic, které jsou koncovými body první úsečky. Je zde přidán `MouseListener`, který zajišťuje volání metody `call()`, v níž se ukládá počáteční, či koncový bod první úsečky.

Metoda `selectSpeedingLine()` slouží k podobnému účelu jako metoda `selectCountingLine()`, ale umožňuje uživateli zvolit si dva koncové body druhé úsečky, kterou vozidla projedou později. Po průjezdu touto úsečkou se vypočte rychlost daného vozidla. Stejně jako v metodě `selectCountingLine()` je přidán stejný `MouseListener`, který ale volá metodu `call2()`. Metoda `call2()` ukládá zvolené body druhé úsečky.

Metoda `call()` je volána z `MouseListeneru` po zvolení libovolného koncového bodu první úsečky, kterou vozidla projedou dříve. Parametry metody jsou číslo události a bod, na který uživatel klikl. Tato metoda nic nevrací. Slouží k uložení prvního koncového bodu, pokud nebyla předtím zavolána, jinak slouží k uložení druhého koncového bodu. Po uložení druhého koncového bodu se odstraní `MouseListener` z objektu třídy `ImageView`.

V metodě se kontroluje parametr event, který udává typ události jako je například kliknutí myší, či uvolnění tlačítka myši. Pokud uživatel klikne myší, hodnota parametru event bude 1 a uloží se jeden z koncových bodů první úsečky. Zkontroluje se, zda již zvolil první koncový bod první úsečky. Pokud uživatel má zvolený první koncový bod a volí druhý koncový bod, po uložení druhého koncového bodu první úsečky se zakáže `MouseListener` a `MouseMotionListener`, které umožňují vybrat si koncové body úsečky, kterou projíždí vozidla.

Listing C.7: Ukázka metody call(), která obsluhuje výběr koncového bodu první úsečky po kliknutí myší

```
public void call(int event, Point point) {
        if (event == 1) {
                if (!startDraw) {
                        lineCount1 = point;
                        startDraw = true;
                } else {
                        lineCount2 = point;
                        startDraw = false;
                        mouseListenertIsActive = false;
                        imageView.removeMouseListener(ml);
                        imageView.removeMouseMotionListener(ml2);
                }
        }
}
```

Metoda `call2()` slouží k podobnému účelu jako metoda `call()` a je i podobně implementována, ale slouží k ukládání koncových bodů druhé úsečky, kterou vozidla projedou později. Tato metoda má stejné parametry jako metoda `call()` a také nic nevrací. Ukládá první koncový bod druhé úsečky v případě, že nebyla předtím zavolána. V ostatních případech ukládá druhý koncový bod a odstraní `MouseListener` z objektu třídy `ImageView`. Tělo metody `call2()` je stejné jako tělo metody `call()` s tím rozdílem, že se místo koncových bodů první úsečky ukládají zvolené body druhé úsečky.

Metoda `count()` zjišťuje, zda se nenachází ve videozáznamu nějaký pohybující se objekt. Pokud je detekován nějaký pohybující se objekt, zvýší se proměnná `counter`, udávající počet vozidel, která projela první úsečkou. Do mapy `speed`, kde je ke každému identifikátoru uloženo, kolik snímků již jede vozidlo mezi první a druhou úsečkou, se uloží pro identifikátor vozidla, které právě navštívilo první úsečku, hodnota 0. Následně je vozidlo klasifikováno pomocí volání metody `classifier()`, která vrátí typ vozidla na základě obsahu čtverce, do kterého je možné vepsat jeho konturu. Podle vráceného typu vozidla se hodnota příslušné proměnné, která udává počet vozidel daného typu, zvýší o 1.

Listing C.8: Metoda count() ukládá identifikátory vozidel, které projely první zadanou úsečkou a klasifikuje vozidla dle typu

```
public synchronized void count(CountVehicles countVehicles) {
        if (countVehicles.isVehicleToAdd()) {
                counter++;
                lastTSM++;
                speed.put(lastTSM, 0);
                String vehicleType = countVehicles.classifier();
                switch (vehicleType) {
                        case "Car":
                                cars++;
                                break;
                        case "Van":
```

```
                                vans++;
                                break;
                        case "Lorry":
                                lorries++;
                                break;
                }
        }
}
```

Metoda `speedMeasure()` zajišťuje samotné měření rychlosti vozidel, která již projela druhou úsečkou. Pro každé vozidlo, které projelo první úsečkou a neprojelo druhou úsečkou, uloží počet snímků od průjezdu první úsečkou. Kontroluje se zde, zda nějaké vozidlo projelo druhou úsečkou. Všem takovým vozidlům se odhadne jejich rychlost na základě počtu snímku od průjezdu první úsečkou pomocí volání metody `computeSpeed()`. Po odhadnutí rychlosti vozidla se zjistí z příslušné buňky tabulky s identifikátorem vozidla, zda se jedná o osobní automobil, dodávku, či nákladní vůz, a na základě toho se přepočítá průměrná rychlost osobních automobilů, dodávek, či nákladních vozů. Nakonec se kontroluje, zda je jeho rychlost vyšší než 50 km/h. Pokud dané vozidlo jelo rychleji než 50 km/h, uloží se záznam o něm a jeho rychlosti do databáze.

V ukázce kódu je naznačeno, jak se měří rychlost vozidla, které projelo druhou úsečkou. Nejdříve se voláním metody `isToSpeedMeasure()` objektu třídy `countVehicles()` kontroluje, zda již nějaké vozidlo projelo druhou úsečkou. Pokud metoda `isToSpeedMeasure()` vrátí `true`, změří se rychlost prvního vozidla s identifikátorem `firstTSM`, který je identifikátorem prvního uloženého vozidla, které ještě nebylo vymazáno z mapy `speed` po změření rychlosti.

Následně se pro všechny identifikátory vozidel, která projela první úsečkou (tj. mají identifikátor menší než `lastTSM`), zvýší hodnota o 1. Po navýšení hodnot, které udávají, kolik snímků se dané vozidlo již pohybuje mezi první a druhou úsečkou, se změří rychlost vozidla s identifikátorem `firstTSM` voláním metody `computeSpeed()`. Pokud naměřená rychlost vozidla je vyšší než 50, uloží se informace o něm a o jeho rychlosti do databáze. Na konci metody se vymažou informace z mapy `speed` o vozidle s identifikátorem `firstTSM`, jehož rychlost je změřena.

Listing C.9: Metoda speedMeasure(), která navyšuje hodnoty proměnných o 1 v mapě speed s identifikátory vozidel a měří rychlost prvního vozidla z mapy speed

```
public synchronized void speedMeasure(CountVehicles countVehicles
    ){
        int firstTSM = speed.entrySet().iterator().next().getKey
            ();
        if (countVehicles.isToSpeedMeasure()) {
                for (int i~= firstTSM; i~<= lastTSM; i++) {
                        if (speed.containsKey(i)) {
                                speed.put(i, (speed.get(i) + 1));
                        }
                }
```

```
                double currentSpeed = computeSpeed(speed.get(
                    firstTSM));
                if (currentSpeed >= 50) {
                // ulozeni informaci o~vozidle do databaze
                }
                speed.remove(firstTSM);
        }
}
```

Metoda `computeSpeed()` spočítá rychlost na základě parametru, udávajícího počet snímků od průjezdu první úsečkou do průjezdu druho úsečkou. Nejdříve se z daného počtu snímků vypočítá reálný čas a poté se na základě něj vypočítá rychlost. Parametrem metody je hodnota typu int, která udává počet snímků od průjezdu první úsečkou do průjezdu druhou úsečkou. Metoda vrací rychlost vozidla, které se pohybovalo mezi první a druhou úsečkou zadaný počet snímků, v kilometrech za hodinu.

Třída `Loop` je vnitřní třídu třídy `GUI` a implementuje rozhraní `Runnable`. Reprezentuje smyčku, během které se přehrává videozáznam. Obsahuje metodu `run()`, ve které se nachází cyklus, který probíhá, dokud není video již přehrané. Metoda `run()` je volána z metody `init()` ve třídě `GUI` tak, že na objekt této třídy se zavolá metoda `start()`.

Na začátku metody `run()` se zajistí, že nebude změřená rychlost vozidel, která ujedou 6 metrů za více, než 2 sekundy. Poté se vytvoří objekt třídy `MixtureOfGaussianBackground`, který v sobě obsahuje objekt třídy `BackgroundSubtractorMOG2`. Objekt třídy `BackgroundSubtractorMOG2` vypočítává popředí a usnadňuje tak detekci pohybujících se objektů. Pokud se podařilo úspěšně nahrát videozáznam, probíhá cyklus, který skončí pokud je video přehrané.

Uvnitř cyklu se přečte aktuální snímek, který se následně zpracuje. V načteném snímku se pomocí objektu třídy `BackgroundSubtractorMOG2` detekuje popředí, které obsahuje pohybující se objekty. Toto popředí je reprezentováno pomocí objektu třídy Mat, představující matici pixelů. Poté se matice, v níž je uloženo popředí předá objektu třídy `CountVehicles`, který detekuje vozidla na tomto snímku s popředím reprezentovaným maticí. Na aktuálním snímku se také vyznačí pohybující se objekty.

Poté, co jsou vozidla ve snímku detekována, se zavolá metoda `count()` pro uložení informací o vozidlech, která projela první zadanou úsečkou. Po detekci vozidel se zavolá metoda `computeSpeed()`, která vypočítává rychlost vozidel, která projela druhou úsečkou. Po výpočtu rychlosti vozidel tato metoda zajistí aktualizaci času a případně zajistí uložení videa. Nakonec vykreslí aktuální snímek s detekovanými vozidly do `ImageView`.

Pokud se nepodaří načíst snímek z důvodu, že je video již přehrané, se uloží informace o pohybujících se objektech a jejich vypočtených rychlostech do souboru. Po uložení informací o pohybující se objektech do souboru se aktivují

tlačítko pro přehrání videa a tlačítko pro načtení videa. Naopak se zakáže tlačítko pro přehrání videa.

Třída `MixtureOfGaussianBackground` má v sobě uloženou instanci třídy `BackgroundSubtractorMOG2`, která slouží k detekování popředí na snímku a odečítání pozadí. Obsahuje konstruktor s celočíselným parametrem history, který udává, kolik snímků zpět se má při výpočtech uvažovat, a s parametrem typu double, udávajícím práh τ_1. V konstruktoru se vytvoří instance třídy `BackgroundSubtractorMOG2` se zadanými parametry, která ignoruje stíny.

Třída `MixtureOfGaussianBackground` obsahuje metody `process()` pro detekci popředí, `setImageThreshold()` pro nastavení prahu a `setHistory()` pro nastavení počtu snímků zpět, které se budou uvažovat. Metoda `process()` slouží k detekci popředí a usnadňuje tak detekci vozidel. Parametrem metody je matice reprezentující aktuální snímek a metoda vrací reprezentaci snímku, na kterém je popředí zvýrazněno bílou barvou.

Třída `CountVehicles` slouží ke zjišťování, zda nějaké vozidlo projelo první úsečkou a má být uložen jeho identifikátor. Dále umožňuje zjistit, zda je možné změřit rychlost některých vozidel. V neposlední řadě umožňuje klasifikovat vozidla ve videozáznamu. Metoda `classifier()` klasifikuje nalezená vozidla ve videozáznamu a vrací, zda dané vozidlo je osobní automobil, dodávka, či nákladní vůz.

Metoda `isVehicleToAdd()` zjišťuje, zda se ve videozáznamu nachází vozidla, jejichž identifikátor zatím nebyl uložen. Hledají se zde vozidla, která právě projela první úsečkou. Vrací pravdivostní hodnotu, která udává, zda takové vozidlo existuje, či nikoliv. Uvnitř metody je cyklus, který prochází nalezenými konturami objektů z popředí. Pro každou konturu získá pomocí volání metody `boundingRect()` ze třídy `ImgProc` obdélník, ve kterém se nachází. Následně se kontroluje, zda se úsečka nachází uvnitř obdélníku. Pokud se úsečka nachází uvnitř obdélníku, metoda vrátí pravdivostní hodnotu `true`, která značí, že se mají uložit informace o pohybujících se objektech ve videu.

V ukázce kódu je naznačen průchod přes všechny nalezené kontury objektů, uložené v proměnné `goodContours`. Pro každou konturu se získá obdélník pomocí pomocí volání metody `boundingRect()` ze třídy `ImgProc`. Pro získaný obdélník se kontroluje, zda obsahuje první úsečku, uloženou v proměnné `line`. V případě, že je první úsečka uvnitř obdélníku, hodnota `countingFlag` se nastaví na `true`.

Poté se kontroluje, zda byl nalezen obdélník, obsahující konturu vozidla. Pokud takový obdélník byl nalezen (tj. hodnota proměnné `countingFlag` je `true`), kontroluje se, zda nebylo v předchozím volání metody nalezeno vozidlo, které zrovna projíždí první úsečkou (tj. hodnota proměnné `crossingLine` je `false`). Pokud nebylo v předchozím volání metody nalezeno vozidlo, které zrovna projíždí první úsečkou, metoda vrátí `true`. V ostatních případech metoda vrací `false`.

Listing C.10: Metoda isVehicleToAdd() kontroluje, zda projíždí nějaké vozidlo první úsečkou

```
public boolean isVehicleToAdd() {
        for (int i~= 0; i~< goodContours.size(); i++) {
                Rect rectangle = Imgproc.boundingRect(
                    goodContours.get(i));
                if (checkRectLine.rectContainLine(rectangle)) {
                        countingFlag = true;
                        break;
                }
        }
        if (countingFlag == true) {
                if (crossingLine == false) {
                        crossingLine = true;
                        return true;
                } else {
                        return false;
                }
        } else {
                crossingLine = false;
                return false;
        }
}
```

Metoda `isToSpeedMeasure()` zjišťuje, zda se nachází ve videozáznamu objekt, jehož rychlost je možné aktuálně spočítat. Metoda nemá žádné parametry a vrací pravdivostní hodnotu, která udává, zda se má nějakému objektu vypočítat jeho rychlost. Prochází v cyklu nalezené kontury vozidel, kde pro každou z nich získá pomocí metody `boundingRect` třídy `ImgProc` obdélník, ve kterém se nachází. Následně se kontroluje, zda se druhá přímka nachází uvnitř získaného obdélníku. Pokud je přímka uvnitř obdélníku, metoda vrátí `true`, která udává, že ve videu se nachází objekty, jejichž rychlost je možné aktuálně spočítat.

Metoda `findAndDrawContours()` nachází kontury pohybujících se objektů na daném snímku a zajišťuje zvýraznění obdélníků, uvnitř kterých se nachází nějaký pohybující se objekt. Parametry metody jsou objekt třídy `Mat`, reprezentující aktuální snímek, a objekt třídy `Mat`, ve kterém je uloženo popředí s pohybujícími se objekty, které mají být nalezeny. Metoda vrací aktuální snímek, na kterém jsou vyznačeny obdélníkem pohybující se objekty. Třída `ImgProc` z knihovny OpenCV zajistí nalezení objektů z popředí. Následně se prochází již nalezené kontury objektů z popředí a ohraničí se obdélníkem.

Ohraničení kontury objektu obdélníkem se provede voláním metody `drawBoundingBox()`, která volá metodu `boundingRect()` z třídy `ImgProc`. Metoda `drawBoundingBox()` také zajistí vykreslení takového obdélníku na aktuální snímek.

Třída `Reseting` je vnitřní třídou třídy `GUI`, která implementuje rozhraní `Runnable` a používá se pro resetování videa. Při resetování videa si musí uživatel znovu zvolit koncové body první a druhé úsečky, kterými budou

projíždět vozidla. Obsahuje metodu `run()`, v níž se nachází smyčka, která skončí ve chvíli, kdy uživatel zvolí koncové body první a druhé přímky, kterými projíždí vozidla. Pokud jsou již vybrány koncové body první a druhé úsečky, vytvoří se vlákno se třídou Loop. Vytvořené vlákno se třídou `Loop` se spustí a poté se tento cyklus ukončí.

Tato implementace je použitelná jen pro videozáznamy, na kterých jedou vozidla jedním směrem. Pokud jede vozidlo opačným směrem a nejdříve projede úsečkou, jejíž body uživatel vybíral později, neuloží se záznam o něm a jeho rychlost zůstane nezměřená. Změří se jen rychlost vozidel, která jedou předpokládaným směrem od první úsečky přes druhou úsečku. Tato implementace tedy není vhodná například pro videozáznamy, na nichž jedou vozidla po dvouproudé silnici a v každém jízdním pruhu jedou vozidla jiným směrem. Dá se použít například pro měření rychlosti vozidel v jednosměrné víceproudé ulici. Pokud se změní konstanta, udávající reálnou vzdálenost mezi úsečkami, bude použitelná pro měření rychlosti vozidel na dálnicích a víceproudých rychlostních silnic.

Implementace dále obsahuje několik pevně daných parametrů jako je rychlostní limit a reálná vzdálenost dvou zadaných úseček, mezi kterými projíždí vozidla. Bez změny těchto konstant je tato implementace použitelná jen pro víceproudé silnice, kde jedou vozidla jedním směrem, s rychlostním limitem 50 km/h. V případě změny konstant, udávajících rychlostní limit a reálnou vzdálenost, na parametry zadané uživatelem bude tato implementace použitelná pro libovolný záznam silnice, po které vozidla jedou jedním směrem.

C.3 Implementace automatické kalibrace kamery

V této kapitole je podrobněji popsána existující implementace algoritmu, který měří rychlost objektů tak, že detekuje objekty a na základě pozic zájmových bodů vypočítá parametry kamery. Implementace je stručně popsána v kapitole 2.2.3.3. Zde jsou popsány jednotlivé třídy a metody včetně ukázek kódu.

Implementace obsahuje spouštěcí soubor s funkcí `calibRoad`, která zajišťuje správné načtení videozáznamu a jeho zpracování. Dále obsahuje balíček s třídou `RoadCalibration`, která slouží k uchovávání parametrů kamery a samotnou kalibraci kamery, výpočet úběžníků a dalších parametrů kamery. Ve třídě `RoadCalibration` se pro reprezentaci gradientů, které se používá objekt třídy `GradientBGModel` pro detekci viditelných hran a jejich směru při výpočtu druhého úběžníku. Třída `RoadCalibration` také vytváří objekty třídy `Diamond-Space` pro reprezentaci kosočtvercového prostoru, ve kterém se hlasováním vybírají první a druhý úběžník. Pro detekci zájmových bodů pohybujících se objektů se používá `PointTracker`, který používá výše zmíněné objekty ve třídě `Road-Calibration`. Třída `PointTracker` má také nastavený algoritmus pro detekci hran.

Funkce `init` je konstruktorem třídy `RoadCalibration`, který se volá před zpracováním prvního snímku videozáznamu. Zajišťuje samotné vytvoření objektu třídy `RoadCalibration` a nastavuje parametry jako je velikost snímku, principální bod, funkci, která se využívá jako normalizátor bodů. Zde je nastavená funkce `normalizePoints` z balíčku geometry jako normalizátor bodů. Pokud není principální bod nastaven, je principálním bodem střed snímku. Po nastavení parametrů se v konstruktoru aktualizuje objekt třídy `GradientBGModel`.

Funkce `setupImpl` inicializuje samotný výpočet parametrů kamery a její kalibrace. Zde se vytvoří point tracker, který detekuje zájmové body vozidel pomocí KLT algoritmu. Dále se zde vytvoří objekt třídy `CornerDetector` z Computer Vision System Toolboxu, který detekuje rohové body ve snímcích. Poté se vytváří objekt třídy `GradientBGModel`, který slouží pro výpočet gradientů nalezených hran vozidel, které se používají při výpočtu druhého úběžníku. V neposlední řadě se vytvoří dvě instance třídy `DiamondSpace`. První instance se používá pro akumulaci krátkých fragmentů trajektorií vozidel a následný výběr prvního úběžníků. Druhá instance akumuluje nalezené hrany a následně hlasováním vybere druhý úběžník. Nakonec se vytvoří instance třídy `EdgeDetector` z Computer Vision System Toolboxu, která slouží pro detekci hran, jejichž gradienty se počítají při hledání druhého úběžníku.

Funkce `stepImpl` provádí operace, které jsou potřeba při přechodu na další snímek jako je aktualizace úběžníků a množiny zájmových bodů, které se mají sledovat. Ve funkci se nejdříve zvýší pořadí aktuálního snímku o 1, pokud se video již zpracovává. Při prvním volání metody se proměnná, udávající pořadí snímku, nejdříve inicializuje. Po inkrementaci proměnné, udávající pořadí snímku, se zavolá funkce `trackPoints` pro nalezení zájmových bodů vozidel, na základě kterých se určuje směr pohybu vozidel. Následně se pro aktuální snímek aktualizuje instance třídy `GradientBGModel`. Poté se upraví úběžníky a odhadnutá ohnisková vzdálenost pomocí volání metod `updateVP1`, `updateVP2` a `updateVP3AndFocal`. Po úpravě úběžníků a ohniskové vzdálenosti se zavolá funkce `getNewPointsForTracking` pro aktualizaci množiny bodů, které se mají sledovat.

Listing C.11: funkce stepImpl, kde se zvyšuje číslo, udávající pořadí aktuálního snímku, a aktualizují úběžníky, ohnisková vzdálenost, model pozadí

```
function stepImpl(obj, frame)
        if obj.numFrames == 0, init; end;
        obj.numFrames = obj.numFrames+1;

        [edgeMask,edgeMag,edgeGrad] = step(obj.B, frame);

        [trackPts, oldPts] = trackPoints;
        updateVP1;
        updateVP2;
        updateVP3AndFocal;
```

```
getNewPointsForTracking;
```

Funkce `trackPoints` hledá zájmové body na aktuálním snímku a vrací body, které byly zájmovými body v předchozím snímku. Nejdříve zavolá funkci `step` point trackeru a získá tak všechny zájmové body a vybere z nich ty validní. Poté se vyberou ze všech nalezených zájmových bodů body, které jsou od sebe navzájem vzdálenejší než určitá mez, aby bylo možné najít směr pohybu vozidel. Funkce nakonec vrátí pole validních bodů včetně bodů, jejichž pozice se oproti předchozímu snímku změnila, a body, které byly validní v minulém snímku.

Funkce `updateVP1` aktualizuje první úběžník na základě detekovaného pohybu vozidel. Nejdříve se na základě aktuálních zájmových bodů a bodů, které byly zájmovými body v minulém snímku, vytvoří fragmenty trajektorií vozidel. Následně se využívá první instance třídy `DiamondSpace` pro akumulování těchto fragmentů do kosočtvercového prostoru. Následně se pomocí volání metody `findMaximum` na první instanci třídy `DiamondSpace` vybere z akumulačního prostoru maximum, jehož souřadnice v kosočtvercovém prostoru jsou vráceny. Po získání souřadnic prvního úběžníku v kosočtvercovém prosoru se převedou tyto souřadnice do kartézského souřadného systému pomocí statické funkce `backproject` ve třídě `DiamondSpace`. Pokud se nově nalezený první úběžník nachází od dříve nalezeného prvního úběžníku ve vzdálenosti vyšší než 1 pixel, souřadnice prvního úběžníku se změní a poté se aktualizují masky pro hledání druhého úběžníku.

Funkce `updateVP2` hledá a případně aktualizuje souřadnice druhého úběžníku, pokud souřadnice prvního úběžníku nebyly aktualizovány alespoň po daný počet snímků. Následně se aktualizuje práh pro výběr hran a získají se hrany z aktuálního snímku pomocí volání metody `getVP2EdgePixels`. Je-li dán region zájmu, z nalezených hran se vyberou ty, které se nachází uvnitř regionu zájmu pomocí volání funkce `inpolygon`. Po získání hran se tyto hrany transformují na úsečky, které jsou poté akumulovány do kosočtvercového prostoru. Pokud vznikla transformací bodů alespoň jedna úsečka, použije se druhá instance třídy `DiamondSpace` pro akumulaci těchto úseček do kosočtvercového prostoru. Dále je druhá instance třídy `DiamondSpace` použita pro výběr maxima z akumulačního prostoru a získání souřadnic druhého úběžníku. Poté se tyto souřadnice převedou na souřadnice v kartézském souřadném systému. Pokud se nachází nově nalezený druhý úběžník od dříve nalezeného úběžníku ve vzdálenosti vyšší než 1 pixel, uloží se pořadí aktuálního snímku do proměnné, která udává pořadí posledního snímku, pro který byl aktualizován druhý úběžník.

Funkce `updateVP3AndFocal` spočítá třetí úběžník na základě prvních dvou úběžníků, pokud se souřadnice prvních dvou úběžníků právě změnily. Pokud třetí homogenní souřadnice libovolného z prvních dvou úběžníků je nulová, bude třetím úběžníkem bod s homogenními souřadnicemi (0, -1, 0). V ostatních případech se homogenní souřadnice třetího úběžníku dají vypočítat dle vztahu

2.30 s rozdílem, že výsledek bude uložen do proměnné V3. Složky výsledného vektoru V3 se znormalizují a následně se vynásobí ohniskovou vzdáleností f. Funkce vrací homogenní souřadnice třetího úběžníku s váhou 1. První dvě složky vráceného vektoru se rovnají součtu souřadnic V3 a souřadnic principálního bodu.

Listing C.12: Funkce updateVP3AndFocal aktualizuje souřadnice třetího úběžníku, pokud se změnil první, nebo druhý úběžník

```
function updateVP3AndFocal
        if obj.lastVp1Update==obj.numFrames ||
                obj.lastVp2Update==obj.numFrames
                if (obj.vp{1}(3) == 0 || obj.vp{2}(3) == 0)
                        obj.vp{3} = [0,-1,0];
                else
                        V3 = cross([obj.vp{1}(1:2),obj.focal] - [
                            obj.pp,0],
                        [obj.vp{2}(1:2),obj.focal] - [obj.pp,0]);
                        V3 = V3 ./ V3(3) .* obj.focal;
                        obj.vp{3} = [V3(1:2) + obj.pp,1];
                end
        end
end
```

Funkce `getNewPointsForTracking` aktualizuje pro každý snímek objektovou proměnnou pts, do které je uloženo pole zájmových bodů. Nejdříve se kontroluje, zda je potřeba aktualizovat body a zda pořadí aktuálního snímku je alespoň o nějakou konstantu (zde o 1) vyšší než pořadí snímku, pro který byly naposled aktualizovány zájmové body. Pokud je tato podmínka splněná, zavolá se funkce step objektu třídy `PointDetector` pro nalezení zájmových bodů na aktuálním snímku. Ze zájmových bodů se vybere množina bodů, které jsou od sebe vzdáleny alespoň nějaký počet pixelů, který je zde nastaven na 2. Pokud je dán region zájmu, vybere se podmnožina nalezených bodů, které se nachází v daném regionu zájmu. Po výběru podmnožiny bodů, které jsou od sebe dostatečně vzdálené, se uloží tato množina bodů do objektové proměnné pts. Tyto nalezené body se následně předají `PointTrackeru`, který je bude následně sledovat. Nakonec se upraví proměnná, udávající pořadí snímku, pro který byly naposled aktualizovány zájmové body.

Funkce `getVP2EdgePixels` získává viditelné hrany na pohybujících se objektech, které se používají pro výpočet druhého úběžníku. Nejdříve se získá maska pro velikosti gradientů z detektoru hran. Poté se vypočítá maska pro první úběžník na základě zadaného gradientu a obrazové masky, ve které jsou pro každý pixel uloženy směry k prvnímu úběžníku. Následně se pro videa, která například nejsou vysoko nad vozovkou a první úběžník se nachází na snímku, vypočte maska pro druhý úběžník tak, že se pro každý index, který je kladným číslem, udávajícím souřadnici y, která je na snímku a není vzdálená více než 100 pixelů od souřadnice y prvního úběžníku, uloží hodnota 1. Po vytvoření masky pro druhý úběžník se vytvoří maska pro třetí úběžník, kam

se uloží hodnota 1, pokud je cosinus složky gradientu menší než 0.6.

Na základě získaných mask a masky se získanými hranami lze spočítat výslednou masku Q. Následně se vypočte konvoluce získané masky Q a vypočte se vektor Q tak, že nenulové zůstanou jen složky vyšší než 2. Následně se do vektoru Q uloží jen hrany s velikostí gradientu vyšší než 1.

Po filtrování hran dle velikosti se seřadí gradient dle velikostí. Ve chvíli, kdy je gradient již seřazený, se vrátí prvních n bodů, kde $n = min(length(mag), 100)$, prvních n nejvyšších velikostí gradientu a prvních n gradientů.

Třída `GradientBGModel` se využívá pro výpočet modelu pozadí, který se aktualizuje, a pro výpočet hran a jejich směrů. Vlastnostmi třídy je koeficient alfa od 0 do 1, kterým se násobí reprezentace obrázku H, vypočtená z gradientů a orientací hran, počet různých směrů a počet přihrádek, do kterých je diskretizována orientace hrany. Zde je počet přihrádek nastaven na 2, počet orientací na 6 a koeficient alfa na 0.95.

Funkce `stepImpl` aktualizuje model pro výpočet modelu pozadí. Nejdříve se aktualizuje instance třídy `Convolver`, která se používá pro konvoluci. Poté se vypočítají velikosti gradientů a orientace jednotlivých hran. Na základě velikostí gradientů a orientací hran se vypočítá reprezentace pozadí pro výpočet v dalším volání funkce `stepImpl`. Funkce zajistí pomocí volání metody `imresize` změnu velikosti obrázku reprezentovaného maskou tak, aby měl stejné rozměry jako originální snímek z videozáznamu. Tato funkce vrátí masku, velikosti gradientů a vektor směrů.

Třída `DiamondSpace` reprezentuje kosočtvercový prostor, do kterého je možné transformovat body, či krátké fragmenty trajektorií vozidel. Používá se pro akumulaci úseček z 2D prostoru do kosočtvercového prostoru a hledání maxima v kosočtvercovém prostoru hlasováním. Statická mfunkce `backproject` najde a vrátí souřadnice daného bodu v kartézském systému. Další statická funkce `project` najde a vrátí souřadnice bodu v kosočtvercovém prostoru. Tato třída obsahuje také statickou metodu `emptySpace`, která vrací prázdný kosočtvercový prostor.

Funkce `accumulateLines` transformuje úsečky a převede je do kosočtvercového prostoru. Nejdříve kontrolue, zda argument `lines` je maticí o velikosti 4 x N. Pokud je předána matice 4 x N jako argument, tato matice se převede na matici reálných čísel s jednoduchou přesností (datový typ float). Tato metoda vypočte pomocí Mex-Wrapperu, který volá funkce z C++ souboru, funkci `mx\_raster\_space` pro samotnou akumulaci daných úseček do kosočtvercového prostoru.

Mex-Wrapper umožňuje ve třídách napsaných v jazyce Matlab použít funkci mx_raster_space, která volá C++ funkci `mexFunction`. V C++ souboru se nachází `mexFunction`, která zajišťuje samotnou akumulaci pole úseček do kosočtvercového prostoru.

Listing C.13: Metoda accumulateLines, která volá wrapper pro transformaci

úseček do kosočtvercového prostoru

```
function accumulateLines(obj, lines)
        if ~isfloat(lines) || ~ismatrix(lines) || size(lines,1)
            ~=4
                error('...');
        end
        if ~isa(lines,'single'), lines = single(lines); end;
                obj.ds = obj.ds +
                double(mx_raster_space(size(obj.ds,1), lines));
end
```

Funkce `findMaximum` hledá maximum v akumulačním prostoru a vybírá tak hlasováním kandidáta pro první, či druhý úběžník. Nejdříve upraví proměnnou reprezentující akumulátor a její složky případně vynásobí zadanou maskou. Po úpravě akumulátoru se zvolí kandidát na první, či druhý úběžník tak, že se najde první maximální prvek v akumulátoru. Nakonec se vrátí vygenerované souřadnice v kosočtvercovém prostoru, které odpovídají vybranému maximu.

Funkce `backproject` je statickou funkcí třídy `DiamondSpace`, která hledá a vrátí souřadnice daného bodu v kosočtvercovém systému. Parametrem funkce je pole bodů se souřadnicemi x a y v kosočtvercovém prostoru. Zkontroluje se, zda je vstupní argument reprezentující bod polem N bodů se 2 souřadnicemi. Poté se vypočítají homogenní souřadnice bodů v kosočtvercovém prostoru. Jsou-li již homogenní souřadnice vypočtené, zkontroluje se, zda váha bodu není menší než epsilon, které se rovná 10^{-6}. Pokud je váha bodu nulová, funkce vrátí bod, jehož první dvě souřadnice jsou znormované. V ostatních případech funkce vrátí bod, jehož první dvě souřadnice jsou vydělené původní vahou.

Listing C.14: Funkce backproject hledá souřadnice daného bodu v kosočtvercovém prostoru

```
function c = backproject(p)
        assert(size(p,2)==2);
        p = num2cell(p,1); [u,v] = deal(p{:});
        c = [v, sign(v).*v + sign(u).*u - 1, u];
        eps = 1e-6; reg = abs(c(:,3)) > eps;
        c( reg,:) = bsxfun(@rdivide, c(reg,:), c(reg,3));
        c(~reg,:) = normr(c(~reg,:)); c(~reg,3) = 0;
end
```

Funkce `project` vrací homogenní souřadnice daných bodů v kosočtvercovém prostoru. Parametrem je pole souřadnic bodů x a y v kartézském systému. Funkce zkontroluje, zda je argumentem opravdu pole bodů se dvěma, či třemi souřadnicemi. V metodě se získají homogenní souřadnice zadaných bodů v kartézském systému. Následně se na základě homogenních souřadnic vypočítají homogenní souřadnice v kosočtvercovém prostoru. Funkce vrátí pole, kde mají všechny body v kosočtvercovém prostoru váhu 1 a zbylé souřadnice vydělené původní vahou.

Balíček geometry obsahuje funkce, které se používají pro různé operace se zájmovými body a jiné geometrické operace. Mezi funkce patří například

`linesFromEdges`, která z detekovaných hran na základě gradientů a jejich velikostí získává úsečky. Další funkcí je `linesFromEndpoints`, která hledá přímky, procházející danými body. Další funkcí je `lines`, která transformuje body s danými souřadnicemi a vahami na úsečky. Funkce `normalizePoints` normalizuje body tak, aby jejich váha byla 1 a zbylé dvě souřadnice se převedou na jeho reálné souřadnice x a y. Obsahuje i funkci `pointToPointDist`, která vrací vzdálenost mezi zadanými body. V balíčku je i funkce `uniquePoints`, která vrací ze dvou zadaných množin body, které jsou dostatečně vzdálené od sebe.

Funkce `linesFromEdges` slouží pro získání úseček z detekovaných hran na základě gradientů a jejich velikostí. Jejímy parametry jsou matice se souřadnicemi bodů, velikosti gradientů a gradienty. Následně se vypočítají souřadnice normálových vektorů k zadaným rohům. Funkce vrátí přímky se získaným normálovým vektorem a vahám odpovídajícím velikostem gradientu.

Funkce `linesFromEndpoints` vrací přímky, které prochází zadanými body. Nejdříve se vypočítá vektor vzdáleností mezi danými body d. Po výpočtu vektoru vzdáleností d se vypočítá normálový vektor, který budou mít přímky, vedoucí zadanými body. Funkce vrací přímky, které vedou body, které jsou uloženy v prvním argumentu a mají již vypočtený normálový vektor.

Funkce `lines` transformuje body s danými vahami a normálové vektory na úsečky, či přímky. Pomocí volání funkce size(pts, 1) se získá počet zadaných bodů. Tato funkce vrací normálové vektory, směrové vektory vzniklých úseček, či přímek a jejich váhy.

Funkce `normalizePoints` převádí body do souřadného systému, kde počátkem je zadaný bod `o` a jednotka byla buď vynásobena, nebo vydělena hodnotou parametru `Z`. Používá se zpravidla pro vyjádření relativních souřadnic vůči středu snímku, kde jednotkou je místo pixelu jedna polovina šířky snímku. Prvním parametrem množina bodů `p`, jejichž souřadnice mají být normalizovány. Dalším parametrem je bod `o`, který má být považován za počátek, vůči němuž mají být vypočítány souřadnice. Třetím parametrem je číslo `Z`, které udává, kolik původních jednotek odpovídá jedné nově zavedené jednotce, je-li jako poslední parametr `backward` předána hodnota false, nebo tato hodnota není předána. V ostatních případech parametr `Z` udává, kolik nově zavedených jednotek odpovídá jedné původní jednotce. Poslední parametr udává, zda mají být převedeny relativní souřadnice například v polovinách šířky snímku na souřadnice v pixelech (hodnota True), nebo naopak.

Funkce nejdříve zkontroluje, zda první parametr je matice bodů. Poté pro každý bod zjistí, zda je jeho třetí homogenní souřadnice větší než 10^{-5} (tj. má-li kladnou váhu). Všechny body s kladnou vahou vydělí jejich třetí homogenní souřadnicí a získá jejich souřadnice x a y. Následně se převedou na základě parametru `backward` získané souřadnice s váhou 1 na relativní souřadnice, či naopak.

Listing C.15: Funkce normalizePoints, normalizující, či denormalizující body

```
function p = normalizePoints(p, o, Z, backward)
        assert(ismatrix(p));
        if nargin == 3, backward = false; end;
                [nPts,d] = size(p);

        if d == 2, p = [p, ones(nPts,1)]; end;

        regular = p(:,3) > 1e-5;
        p(regular,:) = bsxfun(@rdivide, p(regular,:), p(regular
            ,3));

        if ~backward
                p(regular,1:2) = bsxfun(@minus, p(regular,1:2), o
                    ) ./ Z;
        else
                p(regular,1:2) = bsxfun(@plus, p(regular,1:2).*Z,
                     o);
end
```

Funkce `pointToPointDist` vypočítá vzdálenosti bodů z první množiny od bodů z druhé množiny. Vypočítá se rozdíl souřadnic bodů z první množiny a souřadnic bodů z druhé množiny. Vrací druhou odmocninu ze skalárního součinu vypočteného vektoru s rozdílem souřadnic se sebou.

Funkce `uniquePoints` vrací ze dvou zadaných množin bodů podmnožinu bodů, jejichž vzájemná vzdálenost je vyšší než zadaná mez. Funkce vypočte euklidovskou vzdálenost mezi zadanými body. Poté vybere ze dvou vstupních množin p1 a p2 bodů takové body z množiny p2, jejichž vzdálenost od bodů z množiny p1 je vyšší než daná mez. Funkce vrátí matici bodů, jejichž vzájemná vzdálenost je vyšší než daná mez.

Dále se tam nachází přehrávač videa, který umožňuje zobrazit první snímek a následně skákat na další snímky. Videozáznam může být zadán také jako sekvence obrázku. Balíček obsahuje třídu `ImageSequenceReader`, která se používá pro načtení sekvence obrázků z adresáře, reprezentující nějaký záznam z kamery.

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `video-processing-library` zdrojové kódy knihovny pro zpracování videa
    - `rest-api-and-web-application` ..... zdrojové kódy REST API a webové aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS