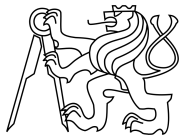
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Analýza pohybu pro staticky umístěnou kameru
Student:	Lubomír Baloun
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Nastudujte knihovny OpenCV [1] a/nebo SimpleCV [2] a patřičné algoritmy a jejich použití v analýze pohybu pro staticky úmístěnou kameru.
Navrhněte vhodné použití algoritmů pro detekci a sledování pohybu humanoidů ve video záznamu.
Navrhněte vhodné použití algoritmů pro detekci a sledování nehumanoidních objektů.
Navrhněte vhodné použití algoritmů pro analýzu míst s největším množstvím pohybu (heat mapy).
Na základně návrhů vytvořte funkční prototypy aplikací pracující pod operačními systémy Linux a Windows.
Navržené aplikace otestujte s vhodnými video záznamy.

Seznam odborné literatury

[1] OpenCV. Dostupné z: http://opencv.org/
[2] SimpleCV. Dostupné z: http://simplecv.org/

L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 24. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Bakalářská práce

Analýza pohybu pro staticky umístěnou kameru

Lubomír Baloun

Vedoucí práce: Ing. Radomír Polách

16. května 2016

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Radomíru Poláchovi za průběžné konzultace a připomínky. Dále bych chtěl poděkovat své rodině za finanční i psychickou podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. § 46 odst. 6 tohoto zákona 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 16. května 2016 .....

Odkaz na tuto práci

Baloun, Lubomír. *Analýza pohybu pro staticky umístěnou kameru.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zabývá implementací algoritmů pro sledování jednotlivých osob, rozhodnutí o přítomnosti určitého objektu a analyzování pohybu do teplotní mapy. V práci je proveden rozbor technik pro vhodné řešení této problematiky a dostupných knihoven pro použití v počítačovém vidění. Dále práce obsahuje popis postupu realizace jednotlivých aplikačních prototypů.

Klíčová slova počítačové vidění, OpenCV, C++, analýza pohybu, sledování osob, lokalizace objektů, teplotní mapa

Abstract

The bachelor thesis deals with implementation of algorithms used for tracking of individual people, determining the presence of certain objects and analysis of motion into heat maps. The thesis explores techniques suitable for solving such issues and libraries available for use in computer vision. The work further describes methods employed while implementing the individual application prototypes.

Keywords computer vision, OpenCV, C++, motion analysis, people tracking, object localization, heatmap

Obsah

Seznam obrázků

Úvod

Jak se postupem času technika neustále zlepšuje, počítače nahrazují lidi stále ve více segmentech. Jedním z nich je i počítačové vidění.

Počítačové vidění je snaha lidí naučit počítače pomocí vědy a techniky vidět a rozpoznávat jednotlivé předměty, jako to umí člověk. Své využití pak nalézá ve všech možných odvětvích, jako je vojenství, marketing, bezpečnost, zábava a další. Existuje však stále mnoho faktorů, které tuto snahu ztěžují.

S jednoduchou formou počítačového vidění se lze v dnešní době setkat v aplikacích pro přidání různých žertovných efektů do videa, jako je například efekt ohně, který se objeví v místech, kde se uživatel pohybuje. Také na ni lze narazit v případě ovládání počítače gesty rukou.

Díky stále se zvyšujícímu výkonu výpočetní techniky lze některé metody, které v době svého vzniku vyžadovaly čas nebo výkonný počítač, již používat pohodlně i na mobilních zařízeních. Počítačové vidění tak získává zcela nový rozměr a objevují se nová využití. Jedná se však o velmi náročné odvětví a má-li být snaha o řešení některého problému úspěšná, je potřeba plného seznámení s danou problematikou.

Cíl práce

- Navrhnout algorimty pracující v reálném čase umožňující:
  - Detekci pohybu chodců
  - Lokalizaci určitého objekt
  - Zpracování pohybu do teplotní mapy
- Vytvořit podle návrhů funkční prototypy aplikací a popsat postup jejich tvorby
- Zhodnotit použité metody a analyzovat prostor pro jejich zlepšení

Struktura práce

Tato práce je rozdělena do 4 kapitol.

V kapitole 1 vysvětluji základní pojmy z oblasti počítačového vidění, na které lze během čtení této bakalářské práce narazit.

V rámci kapitoly 2 podrobně vymezuji jednotlivé požadavky všech tří částí této práce.

Obsah kapitoly 3 zaměřuji na rozbor existujících knihoven vhodných k implementaci práce a popisu jednotlivých nabízených metod.

V poslední kapitole 4 popisuji implementaci a návrh všech aplikačních prototypů, které jsem během psaní této práce vytvořil.

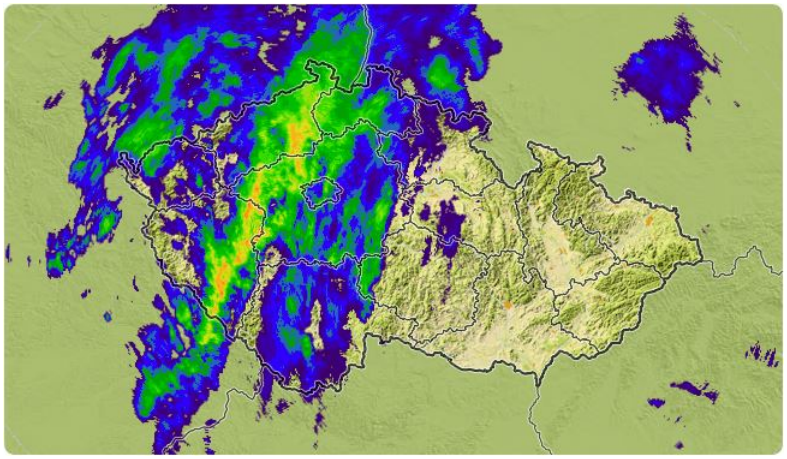
Kapitola 1

Seznámení s problémem

1.1 Vymezení pojmů

1.1.1 Teplotní mapa

Teplotní mapy slouží ke grafickému zobrazení 2D dat. Narozdíl od tabulek, které vyobrazují hodnoty pomocí jejich textové formy, teplotní mapy používají barvy. Své uplatnění nalézají, pokud je potřeba rychlé orientace a nalezení nejvyšších (resp. nejnižších) hodnot ve velkém množství dat. Barvy mohou být zvoleny libovolně s plynulým přechodem mezi barvou pro nejnižší a nejvyšší hodnotu. Nejčastěji se používá kombinace studených, modrých barev pro nízké hodnoty, a teplých, sytě červených barev pro vysoké hodnoty, od toho pochází název „Teplotní mapa“. Často se s nimi lze setkat například při sledování předpovědi počasí. Příkladem je obrázek 1.1.



Obrázek 1.1: Jeden ze způsobů využití teplotní mapy. Převzato z [1].

1.1.2 Klasifikace

Klasifikace doslova znamená třídění, zařazování do různých tříd. V oblasti počítačového vidění se s tímto pojmem lze setkat při rozpoznávání objektů v obraze.

Způsob, kterým se objekty rozpoznávají, záleží vždy na konkrétním problému. Pro klasifikaci objektů do nízkého počtu tříd (v řádu jednotek), které mají mezi sebou dobře odlišitelnou určitou základní vlastnost, lze použít pouhou podmínku. Pokud však počet tříd roste a/nebo odlišnost vlastností je nejednoznačná, musí být pro jejich spolehlivější klasifikaci použit specializovaný algoritmus.

Všechny standardní algoritmy pro komplexnější klasifikaci byly vytvořeny pro řešení obecných problémů a před každým použitím se musí tzv. „naučit“. Řadí se mezi ně rozhodovací stromy, shlukovací algoritmy, regresní algoritmy, neuronové sítě a další.

1.1.3 Klíčový bod, blob a feature

Klíčové body (angl. keypoints) představují malé a unikátní oblasti v obraze. Jejich cílem je označit místa, která lze od sebe snadno odlišit a zároveň není problém určit jejich přesnou pozici. Vhodnými kandidáty pro výběr klíčových bodů jsou části s kontrastními přechody překrývajících se objektů, tedy například hrany a rohy. Naopak nekontrastní olasti, například plochy objektů, bývají pro umístění klíčových bodů nevhodné, neboť by se v takovém případě nedala jednoznačně určit jejich pozice.

K označení ploch se používá vhodnější objekt nazývaný blob. Bloby zastávají podobné funkce jako klíčové body, ale mimo přesné pozice umožňují popsat také tvar a vlastnosti obrysu představované oblasti.

Úskalím klíčových bodů a blobů je nízká schopnost vzájemného porovnání. Pro zlepšení této schopnosti se pomocí specializovaných algoritmů obě struktury převádí do jedné společné označované jako feature. Features zastávají obecnější pojem. V případě počítačového vidění označují specifické oblasti v obraze vhodné například k rozhodování, zda-li je jiný odlišný obraz obsahuje, či nikoli. Přesná struktura a složení samotných features však záleží na konkrétním řešeném problému a algoritmu, který s nimi pracuje. Svou funkci plní například při automatickém skládání fotek do panoramatických záběrů nebo při sledování pohybujících se objektů ve videozáznamech.

1.1.4 Histogram

Histogram slouží k zobrazení distribuce numerických dat grafickou formou. Zpravidla se pro tento účel používá sloupcový graf. Narozdíl od sloupcového grafu, který může zobrazovat hodnoty libovolně, histogram musí zachovávat stejnou šířku všech svých sloupců, čímž je definovaný interval. Svou výškou pak sloupec vyjadřuje četnost sledované veličiny v tomto intervalu.

V oblasti počítačové grafiky jsou histogramy využívány pro zobrazení množství barev v určitém barevném spektru či jiné barevné vlastnosti (například jas) nebo průměr barevných složek. Levá část histogramu reprezentuje četnost tmavých odstínů sledované složky a pravá část světlých odstínů sledované složky. Příklad histogramu lze najít na obrázku 1.2. Lze si všimnout například špičky hodnot ve světlých odstínech modré složky, která je způsobena značnou plochou nebe z obrázku.

Obrázek 1.2: Ukázka histogramů pro základní barevné složky obrázku. Převzato z [2].

Kapitola 2

Analýza problému

V této kapitole se zabývám rozborem problémů, které se mohou vyskytnout při analyzování obrazu a vymezením metod, na které bude potřeba se zaměřit při jeho následném zpracování.

Každé části zaměření bakalářské práce je zde věnovaná sekce, ve které vždy definuji její požadavky, aby bylo podrobně jasné, jak by měl vypadat výstup algoritmu, jenž jej bude řešit.

2.1 Detekce humanoidů

Detekce humanoidů, tedy lidí, je v oblasti počítačového vidění nejrozšířenějším segmentem. Podle požadavků se s ní lze setkat například při sledování pohybu více osob pomocí průmyslových kamer v rámci bezpečnosti nebo sledování jednotlivce pomocí osobní kamery kvůli interakci s počítačem. V dnešní době však stále neexistuje algoritmus, který by bylo možné univerzálně použít pro jakýkoli případ.

V případě mé bakalářské práce se zaměřím na detekci chodců zaznamenaných pomocí staticky umístěných kamer na stěnách budov. Záznamy mohou pocházet jak z venkovních, tak z vnitřních prostorů. Cílem algoritmu bude, aby s co nejmenší možnou chybou spočítal počet chodců, kteří se na vstupním záznamu objeví. Také by měl být schopný řešit situace, které mohou negativně probíhající detekci ovlivňovat. Například zakryje-li překážka část či celou sledovanou osobu, zastaví-li se chodec uprostřed své chůze nebo potká-li se více osob najednou a dojde na krátký okamžik k jejich vzájemnému překrývání.

2.2 Detekce nehumanoidních objektů

Sledovaní nemusí být jenom lidé, schopnosti počítačového vidění lze využít ke sledování jakéhokoli objektu nebo úkazu. Příkladem může být například auto-

matické pozorování chování buněk [3] nebo rozpoznávání poznávacích značek vozidel [4].

Pro zachování pestrosti se v této části nebudu zaměřovat na sledování typu objektů, ale na jeden vybraný objekt. Cílem bude určit, zda se na sledované scéně nachází nebo ne. Algoritmus by měl být také schopný objekt rozpoznat, i když nebude plně viditelný, například pokud jej bude částečně zakrývat překážka. Bude-li to možné, rozpoznávání by nemělo být ovlivněno změnou světelných podmínek ve sledovaném prostoru.

2.3 Zpracování pohybu do teplotní mapy

Posledním zaměřením je zpracování vstupního videa a přenesení informace o množství probíhajícího pohybu ve sledovaném prostoru do teplotní mapy. Pomocí této metody lze snadno odhalit místa, kde se sledované objekty nejčastěji zdržují, či naopak pohybují nejrychleji. Své využití nalézá například v marketinkové oblasti. Obchodník může pomocí znalosti o pohybu zákazníků upravit rozmístění svých produktů tak, aby zboží, o které je větší zájem, bylo dostupnější.

Mým cílem bude zpracovat z videa veškerý pohyb a z něj vytvořit teplotní mapu, na které budou vyobrazená nejrušnější místa. Narozdíl od předchozích případů se zde nebudu zaměřovat na určování typů objektů. Navrhovaný algoritmus by měl umožnit analyzovat pohyb do teplotní mapy jak z celého záznamu, tak i z jeho části. Výsledným produktem bude statická bitmapa teplotní mapy, do které bude přidán snímek ze sledované oblasti jako pozadí.

Kapitola 3

Dostupná řešení

3.1 Knihovny a frameworky

V této kapitole představím a porovnám možné knihovny a frameworky, mezi kterými jsem se rozhodoval, když jsem hledat vhodné řešení k implementaci vlastní práce.

3.1.1 FFmpeg

Jako první bych představil framework FFmpeg. Jedná se o balík programů a knihoven sloužící k veškeré práci s audio a video záznamy, včetně nahrávání a streamování. Balík obsahuje několik samostatných programu, které slouží převážně ke konverzi, přehrávání a streamování jednotlivých formátů multimédií. Hlavní součástí jsou však knihovny, které veškerou funkcionalitu zajišťují.

Framework je k dispozici pod licencí LGPL. Oficiálně jej autoři vyvíjí v jazyce C a pro operační systémy Windows, Linux i MacOS. Díky své oblíbenosti a aktivní komunitě ji lze také získat pro další operační systémy, včetně těch pro mobilní zařízení.

Framework však neobsahuje žádné algoritmy cílené pro využití v počítačovém vidění. Hodí se tak k řešení jednoduchých problému nebo pokud je potřeba implementovat požadované algoritmy vlastním způsobem.

3.1.2 OpenCV

Jedna z nejobsáhlejších knihoven pro počítačové vidění se nazývá OpenCV [5], kterou spravuje společnost Itseez. Obsahuje rozsáhlou nabídku algoritmů pro počítačové vidění a strojové učení.

Knihovna je dostupná pod licencí BSD. Již od první verze byla knihovna navržena jako multiplatformní a napsána v jazyce C. Od druhé verze vývojáři přidali rozhraní pro jazyk C++ a od třetí verze (současná verze) se pro nové algoritmy používá výhradně C++. Existují také rozhraní pro jazyky Python a

Java, které obsahují většinu používaných algoritmů. Knihovnu lze používat jak v operačních systémech pro osobní počítače (Windows, Linux, MacOS, FreeBSD, OpenBSD), tak pro mobilní zařízení (Android, Maemo, iOS). U některých algoritmů lze urychlit výpočty s použitím grafické karty, která podporuje rozhraní OpenCL nebo CUDA.

3.1.3 SimpleCV

Framework SimpleCV [6], který je vyvíjen společností Sight Machine, Inc., využívá ke své funkci více knihoven určených pro počítačové vidění včetně předchozí OpenCV. Za cíl si klade zjednodušení řešení úloh způsobem, kdy se programátor nepotřebuje učit o bitových hloubkách, souborových formátech, správě vyrovnávací paměti nebo rozdílech mezi datovými strukturami pro matice a obrazová data.

Framework lze používat pod BSD licencí. Přestože OpenCV podporuje více programovacích jazyků, SimpleCV se zaměřuje pouze na Python, a tak jej nelze provozovat na mobilních operačních systémech.

Na úkor jednoduchosti SimpleCV disponuje omezenější sadou funkcí a neumožňuje využití grafických karet pro urychlení výpočtů. Nestandardním způsobem je však možné volat funkce z knihovny OpenCV a převádět jednotlivé datové struktury mezi sebou.

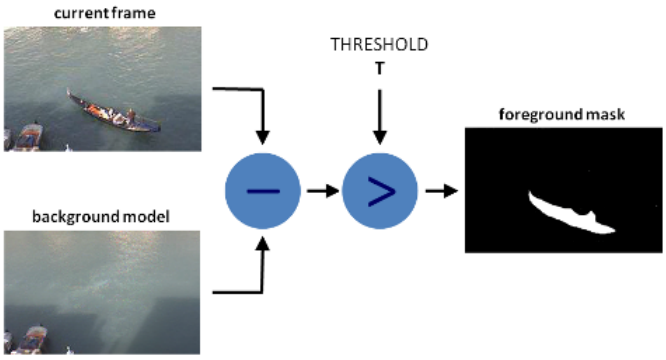
3.1.4 Accord.Net

Další dostupný framework, vhodný k vypracování potřebných algoritmů, se jmenuje Accord.Net [7]. Oproti algoritmům pro počítačové vidění a strojové učení nabízí také algoritmy pro zpracování zvuku a statistickou analýzu dat.

Licence tohoto frameworku je LGPL. V porovnání s ostatními frameworky se liší především v podporovaném programovacím jazyce, kterým je C#. Spolehlivé používání se tak omezuje pouze na operační systémy Windows.

3.1.5 Další knihovny

Knihoven a frameworků existuje více, všechny však poskytují stejné funkce pro rozpoznávání objektů, sledování objektů a práci s obrazem. Liší se pouze v dodatečných funkcích, podporovaném programovacím jazyce a nabízenou licencí. Mezi dalšími zdarma nabízenými řešeními lze najít CCV[8], VXL[9], LibCVD[10] nebo FastCV[11]. Vzhledem k jejich velkému množství neexistuje mnoho komerčních řešení, přesto jsem narazil například na framework SentiSight od společnosti Neurotechnology[12], který již zdarma nabízen není.



Obrázek 3.1: Princip odečítání pozadí [13].

3.2 Detekce a sledování pohybu

3.2.1 Odečítání pozadí

Nejjednodušší formy detekce a sledování pohybu ve videu lze dosáhnout použitím metody nazývané odečítání pozadí. Princip spočívá v porovnávání vstupního obrazu, v němž je potřeba hledat pohybující se objekty, s referenčním obrazem scény, na kterém není zachycen žádný pohybující se objekt (nazývaný právě pozadí). Výstupem je poté binární bitmapa, kde pixely náleží právě pohybujícím se objektům. Graficky znázorněný princip odečítání pozadí lze najít na obrázku 3.1.

Pro zajištění správné funkčnosti musí být záznam zachycen pevně umístěnou kamerou, se kterou se nesmí během nahrávání manipulovat. Klíčovým prvkem se také stává způsob výběru pozadí. Pro scény, u kterých se nepředpokládá, že by se měnilo osvětlení, či by se nějakým jiným způsobem měnilo složení pozadí, které by mělo být z analýzy vyloučeno, lze použít předem získaný obraz scény bez detekovatelných objektů. Naopak pokud se předpokládá, že se světelné složení pozadí bude postupem času měnit a analýza by měla být schopná se změnám přizpůsobovat, pak je vhodné pozadí postupně skládat z aktuálně sledovaného záznamu. Tato metoda s sebou přináší i jistá úskalí. Zastaví-li se nějaký z pohybujících se objektů po dostatečně dlouhou dobu, algoritmus jej začne považovat za pozadí a objekt „zmizí“.

3.2.2 Sledování features

Alternativní metodou pro sledování pohybujících se objektů ve scéně může být sledování jejich features. Popis features jsem již zmínil v úvodní kapitole 1.1.3. Myšlenka této metody se zakládá na hledání podobných rysů mezi dvěma obrazy, které se však nemusí nacházet na stejném místě. V prvním kroku se pro každý obraz najdou nejvhodnější klíčové body, bloby, případně jiné označující prvky. Poté se z nich utvoří features, které se následně popíší a v posledním

kroku porovnají mezi oběma obrazy. Většina algoritmů si pamatuje, jaké klíčové body náleží features, které se spolu porovnávaly, a lze tak sledovat pohyb přímo podle prvotních klíčových bodů.

Schopnosti této metody nejvíce ovlivňují algoritmy v prvních dvou fázích. Vzniklo tak mnoho algoritmů pro detekci označujících prvků a jejich následné popisování. V první fázi se liší výběrem označujících prvků. Některé vybírají pouze klíčové body, jiné pouze bloby. V druhé fázi je poté rozlišují schopnosti stejně popisovat features, které však byly ovlivněny ještě jiným způsobem než posunem. Příkladem může být algoritmus SIFT (Scale-invariant feature transform), který již z názvu vypovídá, že features, jež byly ovlivněny změnou velikosti, popisuje stejným způsobem.

3.3 Detekce objektů

Při sledování určité scény se někdy místo pohybu potřebuje detekovat určitý typ objektů, a to i v případě, že se nepohybuje. Vhodných kandidátů pro řešení tohoto problému se nabízí hned několik. Nejjednodušším řešením je použití klasifikátoru, o kterém jsem se zmiňoval v kapitole 1.1.2. To však není jednoduché, neboť základní algoritmy pro klasifikaci objektů umí pouze rozhodnout o vstupních datech, do jaké třídy patří. Pro spolehlivé použité by bylo nezbytné nejprve detekovat všechny objekty a až pak by se mohlo přejít ke klasifikaci. Lepšího řešení lze docílit použitím algoritmů pro sledování features o kterém jsem psal v předchozí sekci 3.2.2. Pro toto použití by se vybral jeden obraz s požadovaným objektem k detekci, ve kterém by se vytvořily features, a ty by se poté porovnávaly s obrazem ve kterém by se vyhledávalo. Problém však nastává při vyhledávání více objektů, typicky při vyhledávání určitého typu objektů, ne jednoho konkrétního (například obličeje, či těla lidí). Existují však některé algoritmy, přizpůsobené právě pro tento případ. Jejich původní cíl je sice detekovat tělo člověka, či jeho část (nejčastěji obličej), nic však nebrání jejich použití pro detekci jiných objektů.

3.3.1 Kaskádový klasifikátor

Jedním druhem algoritmů pro detekci objektů jsou kaskádové klasifikátory. Průběh, během kterého se vyhledávají objekty v zadaném obraze, se skládá z několika částí a jsou založeny na kombinování jednoduchých klasifikátorů (například rozdělují data do tříd pouze na základě jednoduché podmínky). Použití těchto klasifikátorů samostatně by mělo za následek velmi chybné výsledky, ale právě jejich kombinováním lze tak sestavit robustnější a mnohem spolehlivější klasifikátor. Každá část průběhu je zkonstruována tak, aby dokázala co nejrychleji zamítnout hodnocený vzorek. Během každého rozhodování mohou nastat čtyři případy:

- Vzorek byl správně přijat (v angličtině označován jako „true positive“)

- Vzorek byl správně odmítnut („true negative“)
- Vzorek byl chybně přijat („false positive“)
- Vzorek byl chybně odmítnut („false negative“)

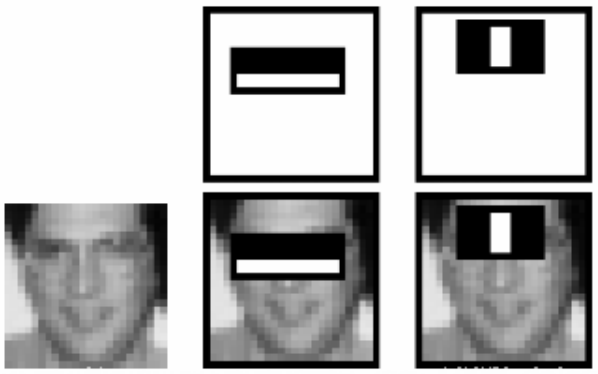
Pokud v některé části nastane odmítnutí, cyklus klasifikace vzorku se zastaví. Důležité tedy je, aby míra chybných odmítnutí byla co nejmenší. V případě, že algoritmus přijme vzorek ve všech částech klasifikace, obsahuje hledaný objekt.

Výběr vzorků k testování probíhá pomocí tzv. plovoucího okénka. Jedná se o menší části obrazu, které jsou vybírány postupně. Začíná se vždy od rohu obrazu (typicky levý horní roh). Po dokončení klasifikace vzoru, se okénko posune o část v rámci jedné osy (například vodorovné, tedy doprava) a opět se vzor klasifikuje. Když se okénko posune až na kraj obrazu, přesune se o kus v druhé ose a v rámci té první se posune na začátek. Tímto způsobem se vyberou vzory ke klasifikaci z celého obrazu. Problém nastane pokud je hledaný objekt větší než plovoucí okénko. Pro tento případ se proces s posouváním okénka opakuje, avšak plovoucí okénko se zvětší. Proces skončí v momentě, kdy je okénko natolik velké, že pokrývá celý obraz najednou.

Velikost okénka zachovává poměr stran hledaného objektu. Proto pro spolehlivé a přesné výsledky by objekt, ve své vyhledávané formě, neměl příliš měnit svůj tvar.

V praxi se lze setkat s dvěma implementacemi těchto algoritmů:

- **Haar** Starší varianta, pomalejší, ale přesnější. Vnitřní výpočty provádí pomocí čísel s plovoucí řádovou čárkou. V částech klasifikace porovnává vstupní data v určitých úsecích se vzory, které obsahují hodnoty podle Haarovy vlnky. Podle toho tato varianta získala svůj název. Princip porovnávání této metody znázorňuje obrázek 3.2.

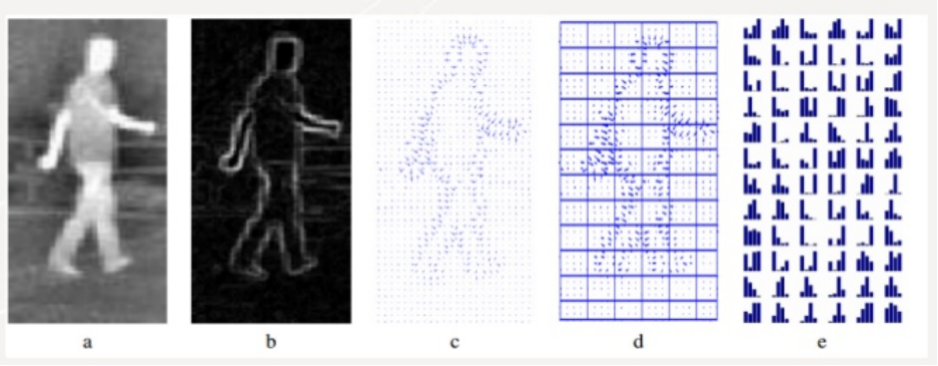


Obrázek 3.2: Porovnávání obrazu pomocí Haar kaskádového klasifikátoru [14].

- **LBP** Novější varianta, rychlejší na úkor menší přesnosti. Protože vnitřní výpočty provádí pomocí celých čísel, je vhodný i pro použití v mobilních zařízeních. Vzory použité při klasifikaci se velmi podobají těm, které používá předchozí Haar varianta. Bývají však více členité a jsou zpravidla uspořádány do čtvercové matice. Podle této vlastnosti také tato implementace získala svůj název Local Binary Pattern.

3.3.2 HOG

Druhým vhodným algoritmem pro detekci objektů je algoritmus HOG (Histogram of oriented gradients). Nejedná se přímo o klasifikátor, ale o popisovač a porovnávač features v obraze. Stejně jako předchozí algoritmy používá k prohledání celého obrazu plovoucí okénko a také pro zajištění všech možných velikostí hledaných objektů svůj proces opakuje se změnou velikosti okénka. Princip vyhodnocení, zda se ve vybraném vzorku nachází objekt, či nikoli, je už pochopitelně jiný. Algoritmus při porovnávání předloženého vzoru upraví jeho barevné hladiny pro získání maximálního kontrastu. Následně jej rozdělí na více malých oblastí. Z každé oblasti analyzuje barevný přechod. Především se zaměřuje na jeho orientaci. Následně z každé oblasti zhotoví histogram (o histogramech jsem již pojednával v kapitole Vymezování pojmů 1.1.4), a ten poté porovnává pomocí klasifikátoru podle předem naučeného modelu. Díky tomuto procesu nese algoritmus svůj název. Graficky znázorněný tento proces si lze prohlédnout na obrázku 3.3. [15] [16]



Obrázek 3.3: Průběh analyzování vzorku pomocí algoritmu HOG [17].

Algoritmus HOG používá k analýze vzorků druh klasifikátoru nazývaný SVM (Support vector machine). Jeho úlohou je rozdělit klasifikovaná data v určitém prostoru pomocí nadroviny do patřičných tříd. Efektivně zvládá řešit tento úkon v prostorech s vysokým počtem dimenzí. Během klasifikace využívá podmnožinu klasifikovaných bodů, je tak efektivní i při využívání paměti. Tato podmnožina bodů se nazývá podpůrné vektory (angl. support vectors), díky tomu algoritmus získal svůj název. Ve většině případů se separace dat provádí

pomocí lineární nadroviny, může však nastat situace, kdy se lineární nadrovina ukáže jako nevhodná. Pro tento případ nabízí SVM možnost změny jádra rozhodovací funkce. SVM však neumí určit míru pravděpodobnosti, se kterou rozhodl o klasifikovaném vzorku. [18] [19]

Kapitola 4

Realizace

V předchozích kapitolách jsem představil a definoval problematiku této bakalářské práce. Zde v každé sekci detailně popíši implementaci příslušné prototypové aplikace, která řeší zadaný problém. Zároveň také vysvětlím, jakým způsobem jsem se vyrovnal s případnými úskalími a jaké další metody by byly vhodné k jejich řešení, pokud se jich bude nabízet více. Na konci každé sekce poté bude následovat závěr s rekapitulací zhotoveného prototypu a případnými návrhy pro zlepšení.

4.1 Použité nástroje

K implementaci algoritmů jsem použil programovací jazyk C++ s pomocí knihovny OpenCV [5] pro následné zpracování videa. Protože knihovna již obsahuje vše potřebné pro práci s videem, jiná knihovna použita nebyla.

4.2 Detekce pohybu chodců

V první části se budu zabývat detekcí pohybujících se chodců. Jak jsem definoval tento problém v kapitole 2.1, důležitým kritériem pro tuto část bude sledování jejich pohybu během jejich výskytu ve sledované scéně.

4.2.1 Základní práce s videem

OpenCV používá pro načítání a práci s videem framework FFmpeg, o kterém jsem se již v této práci zmiňoval v kapitole 3.1.1. Pro obrazová data využívá jednotné třídy `cv::Mat`. Ta představuje datový typ pro ukládání hodnot v libovolné matici. Může tak obsahovat i data, která nepředstavují žádnou bitmapu. Vše záleží na předem zvoleném typu matice. Ve většině případů si jej knihovna OpenCV spravuje sama a stačí si pouze pamatovat, k jakému účelu matice slouží. Případ, kdy je potřeba se důkladněji zaměřit na vnitřní typ

matice nastává, pokud programátor musí například zasahovat vlastnoručně přímo do jejích hodnot.

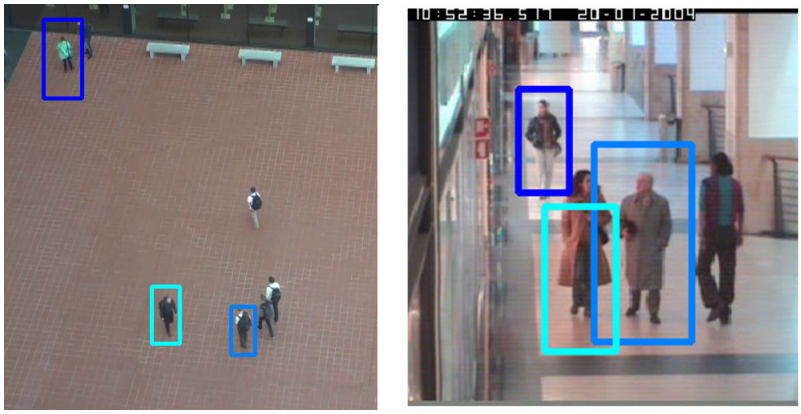
OpenCV také nabízí základní funkce k tvorbě jednoduchého GUI. Pro potřeby mé práce jsem si vystačil ve všech případech pouze s vytvořením libovolného okna `cv::namedWindow()` a následným zobrazením matice s obrazovými daty `cv::imshow()`. Nepředpokládal jsem, že by navrhované prototypy algoritmů měly být schopny měnit své hodnoty nastavení během provozu. Všechny potřebné hodnoty ovlivňující běh algoritmu jsem umožnil nastavit buď pomocí parametru při spuštění prototypu aplikace nebo skrze konstantu ve zdrojových kódech. Pouze v případě přepínání různých režimů zobrazení jsem použil ovládání pomocí čtení stisknutých kláves.

Hlavní třídou, která v knihovně OpenCV spravuje načítání obrazových dat z videozáznamů, je `cv::VideoCapture`. Ta umožňuje univerzálně otevřít jak soubor z disku, tak datový proud z případné kamery. Vše záleží na parametrech předaných v konstruktoru. Funkce `VideoCapture::read()` poté umožňuje získání snímku. Viditelným poznatkem u takto jednoduchého programu, který umí pouze otevřít předem zvolený videosoubor a následně jej snímek po snímku zobrazit, je fakt, že knihovna nepředpokládá své využití jako přehrávač videozáznamů. Všechny videozáznamy dekóduje maximální možnou rychlostí. Videozáznamy s menším rozlišením se tak mohou dekódovat až nepřiměřeně rychle.

Abych tedy zajistil, že se přehrávaná videosekvence bude dekódovat přijatelnou rychlostí, přidal jsem na konec cyklu po zobrazení aktuálního zpracovaného snímku funkci umožňující čekání na případný stisk klávesy uživatele po zvolenou dobu. Díku tomu lze také přidat různé drobné ovládání prototypu (například ukončení pomocí stisku klávesy `Q`). Tento způsob patří mezi velmi jednoduché a trpí nedokonalostí, že se prodleva, během které čeká na stisk uživatele, připočítává k celkovému trvání běhu cyklu.

4.2.2 Detekce pohybu

Ze získaného videa nyní potřebuji získat pohybující se chodce. Zde se nabízí několik možností. Testování jsem prováděl vždy na dvojici videozáznamů, na kterých je zachycen pohyb chodců, avšak se liší v umístění kamery, snímaném úhlu, snímaném prostoru a rozlišením videa. První záznam zachycuje pohyb chodců ve venkovním prostoru na nádvoří před vstupem do budovy. Kamera je umístěna ve větší vzdálenosti včetně vyšší výšky nad zemí. Rozlišení videa je 1280×720. Druhý záznam obsahuje pohyb chodců chodbou v obchodním centru. Kamera byla umístěna níže než v předchozím případě, ale díky velikosti chodby se v záznamu objevují jak malí chodci na konci, tak velcí, blízko před kamerou. Tento záběr byl snímán s rozlišením 384×288 (obsahuje tedy zhruba devětkrát méně obrazových bodů).



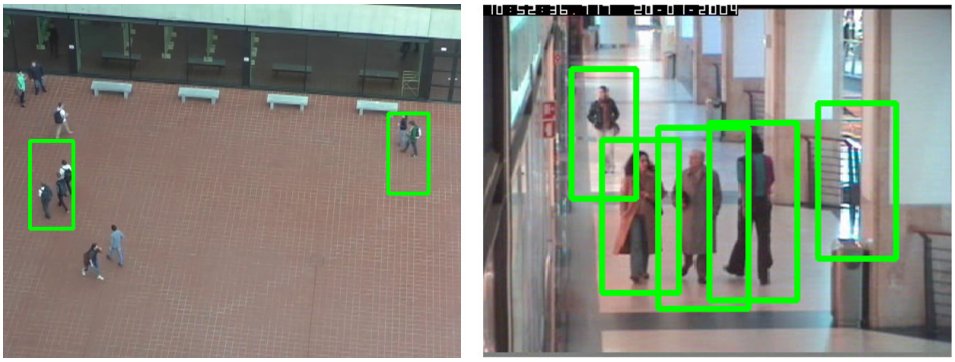
Obrázek 4.1: Vlevo scéna z venkovních prostorů, vpravo scéna z interiéru obchodního domu

Kaskádové algoritmy

Prvními vhodnými kandidáty k řešení tohoto problému jsou kaskádové algoritmy. Jak fungují, jsem rozebíral v kapitole 3.3.1. OpenCV spolu s implementací těchto algoritmů nabízí i předem naučené modely pro vyhledávání tváří, očí, ale i celých těl lidí. Všechny tyto modely se nachází v příslušným souborech.

Implementaci umožňuje v knihovně třída `cv::CascadeClassifier`. Samotné použití je poté jednoduché. Metodou `CascadeClassifier::load()` algoritmus nahraje naučený model do paměti a následně voláním metody `CascadeClassifier::detectMultiscale()` analyzuje zadaný obraz. Oblasti s hledanými objekty vrací v podobě seznamu obdélníků.

Z praktického hlediska se však tato metoda prokázala jako nevhodná. V prvním záznamu venkovních prostorů většinu chodců nebyla schopna detekovat a pokud nastala správná detekce, trvala vždy jenom na krátkou chvíli. V druhém záznamu již poskytovala lepší výsledky i tak docházelo často k situacím, kdy pohybující se osoba nebyla detekována. Občas nastal také případ, kdy byla označena oblast, na které se žádná osoba nevyskytovala. K ale docházelo k tomu výjimečně. Nevýhodou této metody je také její výpočetní náročnost. Detekce osob v záznamu z vnitřního prostoru obchodního centra probíhala přijatelnou rychlostí, ale snímek venkovních prostorů dokázala analyzovat pouze několikrát za sekundu. Pro urychlení by bylo možné video předem zmenšit, avšak vzhledem k pozici a vzdálenosti kamery jsou chodci již v originálním rozlišení malí. Zmenšení by detekci více ztížilo. Výsledek si lze prohlédnout na obrázku 4.1.



Obrázek 4.2: Vlevo scéna z venkovních prostorů, vpravo scéna z interiéru obchodního domu

HOG

Alternativní metodu HOG, představenou v kapitole 3.3.2, jsem se také rozhodl vyzkoušet. I zde knihovna nabízí spolu s implementací předem naučené modely pro detekci chodců, pro tuto metodu dokonce dva.

- **Default** Základní model naučený podle osob s počáteční velikostí okénka 64×128. Učení probíhalo podle datasetu INRIA [20], který obsahuje přes 1 300 snímků osob.

- **Daimler** Dodatečný model naučený podle osob s počáteční velikostí okénka 48×96. Pro tento model byl použit dataset DAIMLER [21]. Počet snímků osob dosahuje zhruba 21 tisíc. Vzhledem k takovému množství jsou však všechny zachyceny pouze ve stupních šedi.

Samotné použití této metody se podobá té předchozí. Nejprve je potřeba vytvořit instanci třídy `cv::HOGDescriptor`. Především důležité je správné nastavení velikosti plovoucího okénka, které závisí na modelu, jenž bude použit. Pro jeho nastavení poté slouží metoda `HOGDescriptor::setSVMDetector()`. Předem naučené modely lze získat ve třídě `cv::HOGDescriptor` pomocí metod `getDefaultPeopleDetector()` případně `getDaimlerPeopleDetector()`.

V praktických testech model Daimler vykazoval lepší výsledky, přesto však byly srovnatelné s výsledky při použití kaskádových algoritmů. Pouze v druhé interiérové scéně zvládala metoda HOG detekovat lépe chodce. Jako chodce také chybně označovala oblast, na které žádný chodec nebyl a při zvýšení požadované přesnosti docházelo spíše k horší detekci skutečných chodců, než k odstranění chybně označené oblasti. Výsledek si lze prohlédnout na obrázku 4.2.

Odečítání pozadí

Protože všechny snímky byly pořízeny pomocí staticky umístěné kamery, nabízí se jako další možnost pro detekci chodců metoda odečítání pozadí. O principu této metody jsem již psal v kapitole 3.2.1. Pro zajištění této funkce knihovna disponuje třídou `cv::BackgroundSubstractor`. Nabízí také dvě implementace, které používají odlišný algoritmus pro tvorbu pozadí a rozlišování pohybujících objektů.

- **MOG2** Pravděpodobnostní funkce využívá v této metodě Gausovských směsí.
- **KNN** Alternativní metoda založená na principu K-nejbližších sousedů.

Obě implementace umožňují sestavování pozadí z aktuálně analyzované videosekvence. Nabízí také pomocí parametrů nastavení z kolika snímků se bude pozadí skládat a hodnotu prahu potřebnou pro označení pixelu za pohybující se objekt. Speciální schopností obou implementací je detekce možného stínu, který pohybující se objekt vrhá. Tím lze odhalit oblasti pohybujících se objektů, které by mohly negativně ovlivnit případnou analýzu.

Jak jsem zmiňoval, výstupem této metody je pouze binární bitmapa, na které pixely označují černou barvou pozadí, bílou barvou pohybující se objekty a případně šedou barvou stíny pohybujících se objektů. Dalším krokem je lokalizace bílých objektů a určení jejich oblasti v rámci bitmapy. Pro tento účel slouží metody k vyhledávání kontur v obraze.

Často se ve videu objevuje i mnoho nedokonalostí. Výsledkem jsou tak chybně předpokládané oblasti s pohybujícími se objekty. Naštěstí ve většině případů se jedná o oblasti ovlivňující pouze několik pixelů pospolu. Proto jsem přidal podmínku, že pohybující se objekt musí být složen z kontury s velikostí minimálně 10×10 pixelů.

Praktické otestování této metody dosahovalo v porovnání s předchozími dvěma nejlepších výsledků. Na první scéně ve venkovních prostorech metoda úspěšně lokalizovala všechny pohybující se chodce včetně pobíhajících dětí, které jsou také na tomto záznamu zachyceny. I zde se objevilo několik chybně označených oblastí. Metoda například označovala za pohyb odraz chodce ve skleněné zdi. V druhém videozáznamu podané výsledky byly poněkud horší. Vzhledem k menšímu úhlu kamery se zde více projevují odrazy chodců. A označené oblasti zabírají více prostoru. Častěji zde také dochází k překrývání za sebou jdoucích chodců, které pak metoda označí jako jeden objekt. Výsledek si lze prohlédnout na obrázku 4.3.

Vzhledem k výsledkům jsem se rozhodl, že pro detekci objektů zvolím tuto metodu.

Obrázek 4.3: Vlevo scéna z venkovních prostorů, vpravo scéna z interiéru obchodního domu

4.2.3 Odstranění šumu

Záznam videosekvencí pomocí kamer není v žádném případě dokonalý a občas je zachycen nepatrný šum. Především se tak stává v prostředí s nízkým osvětlením scény, typicky v interiérech budov. Tento šum člověk nepostřehne, avšak algoritmy pro odečítání pozadí na něj dokáží zareagovat. V binární bitmapy se tak objevují náhodné pixely, které mohou ve větším množství negativně ovlivňovat detekované oblasti.

Abych se zbavil těchto chybných bodů, aplikoval jsem na binární bitmapu erozní obrazový filtr. Algoritmus tohoto filtru prochází vstupní bitmapu a všechny světlé pixely, které sousedí alespoň s jedním tmavým pixelem obarví na tmavé pixely. Tímto způsobem zmizí z binární bitmapy všechny bílé oblasti, které se skládají pouze z několika pixelů a oblasti s chodci se pouze zmenší. V případě potřeby lze tento filtr několikrát opakovat, ale hrozí riziko, že zmizí i oblasti s chodci. V mém případě jsem si vystačil pouze s jednou iterací tohoto filtru.

Předchozí filtr trochu zmenšil oblasti s chodci. Abych navrátil velikosti oblastí na původní hodnoty, aplikoval jsem ještě opačný, dilatační filtr. Ten funguje stejně jako erozní, ale místo nahrazování světlých pixelů tmavými, nahrazuje tmavé pixely světlými. Zde se opakování filtru již vyplatí, spojí se tak případné rozdělené oblasti, které vznikly, pokud například barva určité části oblečení pohybující se osoby byla stejná jako pozadí. Pro prototyp mé práce jsem použil tři iterace. Více iterací způsobovalo, že se častěji oblasti spojovaly do velkých celků, což by ztížilo následné sledování jednotlivých osob.

4.2.4 Sledování osob

Z vyčištěné sekvence pohybujících se osob lze začít sledovat jednotlivé osoby. Pro zachování přehlednosti ve svém zdrojovém kódu jsem vytvořil dvě třídy, které budou tuto funkci spravovat. První jsem pojmenoval `EntityManager`.

Jejím úkolem je správně aktualizovat pohyb sledovaných osob tak, aby každá nová pozice byla přiřazena vždy k příslušné osobě a nenastávala situace, že se například dvěma osobám při jejich střetu zamění jejich ID. Druhá třída `Entity` představuje objekt pro samotnou sledovanou osobu. Slouží k uchování historie pohybu, aktuální pozice a vypočítání aktuálního směru pohybu. Každý objekt této třídy má přiražené unikání identifikační číslo, které přísluší osobě ve sledované scéně.

Proces sledování a přiřazování pozic jsem rozdělil na dva případy. První případ nastává, pokud je sledovaný objekt plně viditelný. V takovém případě předpokládám, že se pozice osoby mění pouze minimálně. Pro každý objekt s aktivně sledovanou osobou porovnávám všechny pozice z nového snímku podle formule:

$$distance = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (2 \times \Delta w)^2 + (2 \times \Delta h)^2}$$

Proměnná *distance* vyjadřuje vzdálenost odlišností mezi aktuálně porovnávanými oblastmi. Proměnné x a y určují pozice oblastí v rámci obou os a w a h šířku a výšku oblastí. Poslední dvě proměnné ještě násobím konstantou 2. Tím jsem zvýšil růst hodnoty v případě rozdílu mezi velikostmi porovnávaných oblastí. Nepředpokládá se, že by člověk během své chůze měnil svou velikost. Lze se tak vyvarovat situacím kdy docházelo k záměně ID při střetu dvou osob.

K druhému případu dochází, pokud proměnná *distance* přesáhne nastavenou mez. V takovém případě nebyla nalezena dostatečně blízka oblast v novém snímku a předpokládá se, že se objekt „ztratil“.Tento případ nastával zejména při střetu dvou a více osob, pokud osobu zakrývala celkově překážka nebo pokud osoba opustila sledovaný prostor. Pokud k této situaci došlo z posledního důvodu, bylo by dobré objekt smazat a uvolnit nepotřebnou paměť. Třída `Entity` pro tento případ obsahuje proměnnou *timeout*, jejíž hodnota udává počet snímků, během kterých nebyla nalezena žádná vhodná oblast. Pokud přesáhne zvolenou hodnotu předpokládám, že se sledovaná osoba již nevrátí a smažu ji z paměti. V případě zbývajících dvou důvodu je však dobré dopočítat předpokládanou pozici, neboť je velmi pravděpodobné, že se pohybující osoba nevychýlí ze své trasy a opět se objeví.

4.2.5 Určení směru pohybu

Abych mohl úspěšně určit směr pohybu osoby, musím si pamatovat, jaká byla její předchozí pozice. Pro tento účel uchovávám ve třídě `Entity` historii všech oblastí zahrnující jejich pozice a velikost. Podle charakteristiky sledovaných scén můžu předpokládat, že nenastane situace, kdy počet aktivních chodců překročí hodnotu sto. Nemusím tak omezovat historii oblastí a z paměti se vždy uvolní celá až po té, co osoba opustí sledovanou oblast.

Historie pohybu může sloužit také jako rychlý přehled o pohybu aktuálních chodců. Přidal jsem proto možnost zobrazit pro všechny vyskytující se chodce

ve scéně historii jejich pohybu v podobě barevných cest. Pro snadnou odlišnost od sebe bylo potřeba vybrat dobře rozlišitelné barvy. Pro tento účel jsem použil paletu 62 barev, kterou sdílel uživatel s přezdívkou Tatarize na svém blogu [22] (Paleta obsahuje přesně 64 barev, v mém prototypu jsem však bílou a černou barvu vynechal). Barvy poté přiřazuji jednotlivým osobám podle hodnoty přidruženého ID.

Pro vypočítání směru pohybu v ose x využívám jednoduchý průměr hodnot z historie:

$$\frac{1}{n}\sum_{i=1}^{n}(x_{i-1} - x_i)$$

Stejným způsobem i pro pohyb v ose y. Počet hodnot, z kterých se průměr sestavuje, závisí na počtu hodnot, které jsou uloženy v historii sledované osoby. Aby se však směr pohybu dokázal přizpůsobit případným změnám, pro výpočet se jich použije nanejvýš padesát.

Hodnoty směru pohybu představují vzdálenost, kterou urazí sledovaná osoba mezi dvěma snímky. Často se stává, že se pohybuje velmi pomalu a vypočítané hodnoty jsou nižší než jedna. Přesnost těchto hodnot hraje klíčovou roli a jakékoli zaokrouhlení by způsobilo chybu s fatálním důsledkem.

4.2.6 Předvídání pozice osob

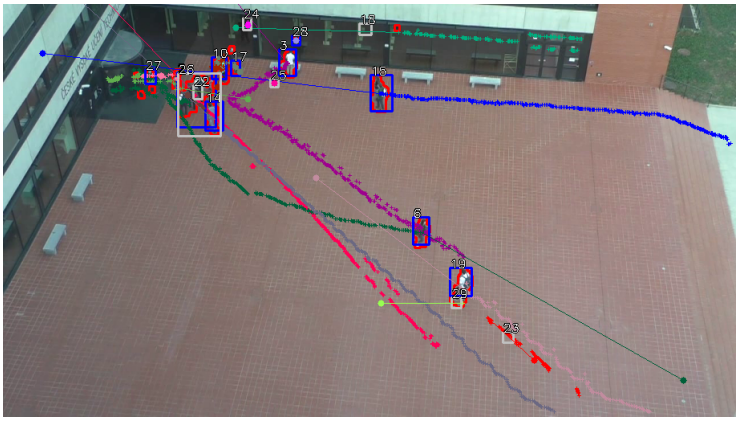
V předchozí sekci ohledně sledování osob 4.2.4 jsem popisoval případ, kdy osoba přestane být sledována a bude potřeba dopočítat její předpokládanou pozici. Díky znalosti o jejím aktuálním směru pohybu tuto vlastnosti již lze implementovat. Kvůli zajištění přesnosti však nemůžu přičítat každý snímek hodnoty směru pohybu k aktuální pozici, neboť její datová struktura umožňuje pouze zápis celých čísel.

Použil jsem tedy způsob, kdy k poslední známé pozici (x_s, y_s) přičítám hodnotu směru pohybu (x_d, y_d) vynásobenou počtem snímků n, během kterých nebyla nalezena správná oblast osoby. Lze jej vyjádřit například:

$$(x, y) = (x_s, y_s) + n(x_d, y_d)$$

Do třídy `Entity` jsem přidal metodu `getLocation()`, která podle situace vrátí buď oblast s aktuální pozici, pokud proměnná *timeout* bude rovna jedné, nebo předpokládanou oblast, pokud se hodnota zvýší. Této metody poté využívá třída `EntityManager` při porovnávání aktuální oblasti osob, když vyhledává a přiřazuje nejbližší z nového snímku. V případě, že se „ztracená“ osoba opět „objeví“ lineárně se do historie dopočítají chybějící pozice.

Tato vlastnost s předvídáním pozice osob značně zvýšila schopnost správného udržení hodnot ID k příslušným osobám.



Obrázek 4.4: Konečná podoba algoritmu pro sledování pohybu osob, barevná cesta znázorňuje aktuální trasu chodce, číslo nad ohraničením přiražené ID

4.2.7 Další drobná vylepšení

Předvídání pozice osob funguje ve většině případů spolehlivě. Na videozáznamu lze však najít osoby, které během své chůze mění velikost své oblasti a ovlivňují tak výsledný směr pohybu. Pro zmírnění tohoto jevu jsem se rozhodl upravit algoritmus pro výpočet směru pohybu. Pozice oblasti osoby v základním tvaru vyjadřuje pozici levého horního bodu obdélníku, která tuto osobu zachycuje. Lepších výsledků jsem dosáhl, když jsem jako pozici použil střed oblasti. Ten více koresponduje se skutečnou pozicí osoby a zpřesňuje vypočítaný směr pohybu.

Změna velikosti oblastí sledovaných osob také ovlivňovala funkci vyhledávání nejbližší oblasti pro nové snímky. Abych zmírnil tento jev upravil jsem funkci pro předání aktuální pozice `getLocation()`. Ta nyní vrací oblast s velikostí vypočítanou pomocí průměru velikostí z historie (nejvýše z deseti hodnot).

Konečnou podobu algoritmu si lze prohlédnout na obrázku 4.4

4.2.8 Shrnutí

Problém s udržováním správných hodnot ID je hlavní překážkou při návrhu toho algoritmu. V rámci této bakalářské práce vznikl prototyp s algoritmem, který zvládne v základních případech správné ID přiřadit. Stále existuje mnoho případů, které řešit neumí. Problémy nastávají při sledování chodců blízko vchodů do budov, kde se v jeden okamžik může pohybovat mnoho osob a algoritmus založený na odečítání pozadí vše označí jako jeden objekt. Pro řešení takového problému by mohl sloužit soubor alternativních robustnějších metod aplikovaných pouze na tyto oblasti, které se však z důvodu efektivity nedají použít na celou sledovanou scénu. Dodatečnými řešeními se ještě nabízí

Obrázek 4.5: Přední strana krabičky zápalek slouží jako referenční obraz objektu, který má být lokalizován

sledování scény pomocí více kamer, či jednoduše upravit pozici té stávají.

Prototyp také nerozlišuje, zda-li je pohybující se objekt skutečně člověk. Tato vlastnost nebyla naimplementována, neboť z povahy testovacích videozáznamů by ji nebylo jak vyzkoušet. Pokud by to bylo nutné, nabízí se sledování podobností tvarů nalezených objektů s případnými objekty z předem připraveného seznamu. Bohužel knihovna OpenCV žádný předem připravený seznam tvarů pro porovnávání nenabízí a bylo by tedy nutné si jej vytvořit vlastnoručně.

4.3 Detekce nehumanoidních objektů

V druhé části se budu zabývat detekcí jednoho určitého objektu ve videu. Podrobněji jsem tuto část definoval v kapitole 2.2. Kritériem, na které se zaměřím v této části, bude rozhodnout, zde se sledovaný objekt nachází v obraze či nikoli. V rámci mé bakalářské práce sledovaný objekt bude reprezentovat krabička zápalek, respektive její přední strana, jak zobrazuje obrázek 4.5.

4.3.1 Kaskádové algoritmy a HOG

Jako řešení tohoto problému se zde opět nabízí využití kaskádových algoritmů a HOG. V tomto případě se však nevyhnu procesu učení na mnou definovaný objekt. Popíši tedy v této kapitole, jak takové učení těchto metod probíhá, jaké s sebou přináší úskalí a jak následně naučený model použít. Protože se proces pro obě zmíněné metody velmi podobá, zaměřím se v této části pouze na kaskádové algoritmy. V rámci této kapitoly se pokusím odpovědět na otázku, zda-li je možné tyto metody naučit vyhledávat libovolně definovaný objekt z malého množství vstupních obrazů. Ve většině prací, které používají k řešení tyto metody, byly vlastní modely sestaveny z velkého množství vstupních obrazů (řádech tisíců).

4.3.2 Učení kaskádových algoritmů

První model budu sestavovat pro kaskádové algoritmy. Knihovna OpenCV nabízí několik programů, které slouží k tomuto účelu. Předtím, než však mohu začít nějaké programy používat, musím si připravit dva druhy obrázků:

- **Pozitivní vzorky** (angl. positive samples) - na těchto obrázcích se vyskytuje hledaný objekt popř. typ objektu. Pro pevné objekty, které nemění svůj tvar a stačí, aby je metoda detekovala v takovém rozpoložení, v jakém je zachycen na určitém obrázku, poslouží jeden vzorek. Pokud bych však cílil na typ objektů jako jsou například tváře lidí, těla zvířata, poznávací značky vozidel apod. musel bych mít těchto vzorků mnohem více. Vzorky by měly také zachycovat vyhledávaný objekt z nejrůznějších úhlů, aby bylo možné jej lokalizovat i v případě, že se na vyhledávaném snímku nevyskytuje kolmo ke kameře.

- **Negativní vzorky** (angl. negative samples) - tyto obrázky zachycují scény, na kterých by se mohl vyhledávaný obrázek vyskytoval, ale v žádném případě se nevyskytuje. Díky tomu metoda pozná, jaká část z pozitivních vzorku náleží hledaným objektům. Negativní vzorky již není možné nijak předem vygenerovat. Těchto vzorků budu potřebovat mnohem více než pozitivních.

Příprava vzorků

Postup jak naučit vlastní model pro kaskádové algoritmy z knihovny OpenCV jsem čerpal ze stránek [23] a [24]. Především na stránce [24] autor popisuje i jeho praktickou zkušenost. V případě jeho práce použil k tvorbě funkčního modelu 40 pozitivních vzorků a 600 negativních vzorků. V rámci mé práce jsem se rozhodl zaměřit se na pětkrát nižší počet než je tento. Tedy 8 pozitivních vzorků a 120 negativních vzorků. Při tvorbě pozitivních vzorků jsem krabičku od zápalek vyfotil vždy na určitém místě, přičemž abych získal i negativní vzorky, vyfotil jsem stejné místo i bez zápalek. Takto vzniklých negativních vzorků nebylo mnoho, proto jsem doplnil negativní vzorky obrázky z datasetu INRIA [20], o kterém jsme se již zmínil v sekci s tvorbou předchozího prototypu při popisování metody HOG 4.2.2.

Všechny obrázky pro pozitivní vzorky jsem ořezal a zmenšil na velikost, při které krabička zápalek zabírá oblast přibližně 96×64 pixelů. Polovinu těchto hodnot (tedy 48×32 pixelů) nastavím jako výchozí velikost vzorků a plovoucího okénka. Při prvních pokusech o trénování modelu Obrázky pro negativní vzorky jsem pouze zmenšil na velikost 320×180 pixelů, aby nezabíraly příliš prostoru a učení tak bylo rychlejší. Obrázky jsem poté rozdělil do příslušných složek.

Generování vzorků

Po ruční přípravě všech obrázků mohu přejít k dalšímu kroku. Program pro trénování klasifikátoru neumí pracovat s obrázky pozitivních vzorků v běžných formátech pro bitmapy a před samotným učením se musí převést. Tento mezikrok vznikl, protože pozitivní vzorky mohou obsahovat více objektů, které jsou předmětem detekce. K tomuto účelu musím vytvořit dodatečný textový soubor umístěný vedle složek s příslušnými vzorky. Každý řádek poté obsahuje cestu k obrázku představující pozitivní vzorek, počet cílových objektů na tomto obrázku a jejich souřadnice. V mém případě všechny vzorky obsahují krabičku na stejném místě a tak se budou se lišit pouze cesty k jednotlivým obrázkům.

Vytvořený soubor poté použiji v programu `opencv_createsamples`, který je součástí zdrojových kódů knihovny. Umožňuje z definovaných obrázků pro pozitivní vzorky vytvořit standardizovaný soubor vhodný pro proces učení.

Generování modelu

Stejně jako jsem musel vypsat všechny obrázky pro pozitivní vzorky do samostatného soubory, musím tento krok opakovat i pro obrázky představující negativní vzorky. Protože se však na nich nevyskytuje hledaný objekt, skládá se tento soubor pouze z cest k příslušným souborům.

S takto vytvořenými soubory již mohu přejít k samotnému generování naučeného modelu. Pro tento účel slouží program `opencv_traincascade`, který je také součástí zdrojových kódů. Nabízí mnoho parametrů, kterými se proces učení ovlivňuje. Především lze určit, zda výsledný model bude využívat metodu LBP nebo HAAR.

Důležitým parametrem je počet částí (angl. stages), během kterých se model učí. Ten ovlivňuje přesnost a celkovou dobu trvání procesu učení. Pokud je částí mnoho a dataset se vzorky velký, může tato operace trvat i několik dní. V mém případě však tato úloha tak dlouho netrvala, neboť mých vzorků bylo skutečně málo. Konečným výstupem učení je XML soubor, které již lze použít pro detekci objektů v obraze.

Výsledky

K otestování naučeného modelu jsem se rozhodl použít kód předchozího prototypu aplikace ve fázi testování kaskádových algoritmů. Nahradil jsem však vstupní video za proud z webkamery.

Schopnosti mého naučeného modelu nebyly dokonalé, přesto v některých případech dokázal krabičku sirek úspěšně lokalizovat. Nejlepších výsledků dosahoval, pokud byla dostatečně blízko kamery a dobře čitelná bez odlesků světla. Neúspěch v detekci nastal, pokud jsem krabičku příliš vzdálil. Bohužel nebyl schopný krabičku lokalizovat ani při slabém natočení či při zakrytí

Obrázek 4.6: Lokalizace objektu pomocí algoritmu HAAR

určité části rukou. Ukázku prototypu lokalizující krabičku zápalek pomocí tohoto algoritmu lze nalézt na obrázku 4.6

4.3.3 Sledování Features

Jako odlišná metoda pro detekci přítomnosti objektu ve scéně může sloužit algoritmus pro sledování features mezi dvěma obrazy. První obraz bude obsahovat objekt, který má být lokalizován a druhým bude videozáznam nebo statická bitmapa, ve které je potřeba daný objekt najít. Princip sledování features jsem popisoval v kapitole 3.2.2.

V rámci mé práce porovnám a otestuji schopnosti algoritmů:

- **AKAZE** (Accelerated KAZE) - Efektivnější varianta algoritmu KAZE, který také slouží pro detekci klíčových bodů a popis features. Vzhledem k zachování stejných struktur pro klíčové body, lze pro popis features použít body získané algoritmem KAZE a naopak. Původní algoritmus KAZE využívá pro detekci klíčových bodů stejných metod jako algoritmy SIFT a SURF. [25]

- **BRISK** (Binary Robust Invariant Scalable Keypoints) - Tento algoritmus se řadí do skupiny binárních popisovačů features. Vyznačují se způsobem v popisování features podle určitého vzoru. Za cíl si kladou zjednodušit tento proces a zefektivnit následující párování featurs.[26]

- **ORB** (Oriented BRIEF) - Algoritmus využívá k detekci klíčových bodů již existujících metod z jiných algoritmů, především z algoritmu FAST. Jsou však upraveny, aby výsledné features byly stabilnější. Pro popis features využívá algoritmus BRIEF. Stejně jako předchozí algoritmus BRISK se řadí do binárních popisovačů features. [27]

Implementace prototypu

Všechny výše zmíněné algoritmy jsou v knihovně implementovány jako podtřídy k třídě `cv::Feature2D`. Ta definuje názvy funkcí pro základní operace vyhledávání klíčových bodu a jejich popis. Při vytváření instancí každé implementace lze určit míru kvality pro jednotlivé features, která posléze ovlivňuje jejich počet. U implementace algoritmu ORB lze tuto hodnotu ovlivnit ještě samotným parametrem pro maximální počet popsaných features.

Pro testování této metody jsem vytvořil jednoduchý prototyp, který načte podle zvoleného parametru obrázek vyhledávaného objektu. Dalším parametrem lze zvolit požadovaný algoritmus pro detekci featrues. Poslední dostupný parametr umožňuje vybrat druh zdroje obrazu ve kterém prototyp bude vyhledávat zadaný objekt. Vstupem může být soubor s bitmapou, videosoubor nebo obraz z webkamery.

Postup jak využít video s použitím knihovny OpenCV jsem vysvětlil v sekci s realizací prvního prototypu. Pro načtení samotného obrázku ze souboru slouží metoda `cv::loadimg()`.

Prvním krokem, pro využití této metody, je detekce klíčových bodů v obraze. Pomocí třídy `cv::Feature2D` tuto funkci zajišťuje metoda `detect()`. Výsledný seznam nalezených bodů se ukládá do matice.

V druhém kroku nastává popis features podle nalezených klíčových bodů z předešlého kroku. O tento proces se stará metoda `compute()`. Výsledek se opět zapíše do matice. Tyto dva kroky se poté opakují pro druhý obraz.

Posledním krokem je porovnání a párování features z obou obrazů. K tomuto účelu slouží speciální třída `cv::DescriptorMatcher`. Ta nabízí několik metod jak mezi sebou jednotlivé features párovat:

- **BruteForce** - Metoda využívá pro výpočet odlišnosti převod vektorů do L^2-normy (Eukleidovská norma).
- **BruteForce-L1** - Metoda využívá pro výpočet odlišnosti převod vektorů do L^1-normy.
- **BruteForce-Hamming** - Metoda využívá pro výpočet odlišnosti Hammingovu vzdálenost.
- **BruteForce-Hamming(2)** - Algoritmus této metody se shoduje s předchozí BruteForce-Hamming, liší se však vnitřní implementací. Využívá určitých rozšíření procesoru pro dosažení rychlejších výpočtů.

Protože Hammingova vzdálenost pracuje s binárními řetězci využívá se pro porovnávání featrues popsaných pomocí algoritmů ORB a BRISK. Features popsané pomocí algoritmu AKAZE se porovnávají zbývajícími metodami BruteForce a BruteForce-L1.

Na konci každého porovnávání features vznikne seznam, ve kterém jsou dvojice features z obou obrazů a k nim vždy příslušná hodnota vyjadřující

míru odlišnosti. Vyplývá z toho, že ke každé feature algoritmus přiřadí jiný do páru. Musím tak pro každý algoritmus zvolit vhodnou konstantu, která bude představovat maximální možnou hodnotu odlišnosti. Pokud některé z features budou mít hodnotu nižší než je tato konstanta, budu je považovat za správně spárované. V opačném případě půjde o chybu. Jako rozhodující kritérium, zda se hledaný objekt nachází ve vyhledávaném obraze či nikoli, jsem zvolil minimální počet správně spárovaných features. Jako vhodná konstanta se ukázala hodnota pěti párů. Rozhodování o přítomnosti na základě jednoho správného páru se ukázalo jako nevhodná podmínka. Vzhledem k velkému množství features se náhodně stávalo, že se některá hodnota odlišnosti vyskytla za požadovanou konstantou a byla nesprávně uznána.

Výsledky

V praktických testech se algoritmus AKAZE prokázal jako nejméně vhodný pro řešení tohoto problému. Krabičku nedokázal lokalizovat, pokud byla na snímaném obraze otočená. Stejně tak ji nerozpoznal v případě, že se lišila ve velikostech. Při zachování rotace a velikosti, jako měla krabička na referenčním vzoru, již algoritmus úspěšně krabičku rozpoznal. V porovnání s ostatními algoritmy byl tento nejvíce náročný na výpočetní výkon a docházelo ke snižování snímkové frekvence webkamery.

Algoritmus BRISK dosahoval podstatně lepších výsledků. Rotace krabičky ve sledované scéně její detekci nijak neovlivňovalo. V případě, že krabička na sledované scéně zabírala více prostoru, než byla velikost krabičky na referenčním obraze, algoritmus jej také dokázal správně lokalizovat. Problém nastával, pokud byla krabička menší než na referenčním obraze. Algoritmus v takovém případě nedokázal správně popsat features, a tedy ji i úspěšně lokalizovat. Nabízelo se řešení zmenšit velikost krabičky na referenčním obraze. Tento krok však nebyl vhodný, neboť přílišným zmenšením krabičky na referenčním obraze došlo opět ke špatnému popsání jejích features, a tím se celkově zvýšla neúspěšnost detekce krabičky za všech podmínek.

Poslední varianta s použitím algoritmu ORB poskytla ze zkoušených algoritmů nejlepší výsledky. Díky jednoduchému zvýšení požadováného maxima features byl schopný lokalizovat krabičky i v případě, že se vzdálila od kamery a její velikost ve sledované scéně zabírala menší část, než na referenčním obraze. Vzhledem k této úpravě vzrostla výpočetní náročnost, stále však byla nižší než v případě prvního zkoušeného algoritmu AKAZE. Grafická ukázka prototypu využívající této metody se nachází na obrázku 4.7.

4.3.4 Shrnutí

Návrh správného a přesného detektoru objektů ve scéně vždy bude záviset na požadavcích a typu objektu, který má být detekován. V této práci se nejlépe osvědčilo použití algoritmů pro sledování features. To bylo zajištěno díky

Obrázek 4.7: Lokalizace objektu pomocí sledování features. Na levém obrázku se nachází stav, když algoritmus hledaný objekt lokalizoval. Na pravém obrázku je poté stav, když jej nenalezl.

kontrastnímu písmu na obalu krabičky od sirek. Sloužilo jako dobrý zdroj pro tvorbu klíčových bodů a následný popis featrues. Algoritmy pro sledování features však nejsou pro takovýto účel stavěny a neumožňují například lokalizaci více stejných objektů ve sledované scéně. Problém by také nastával pokud by pozadí na sledované scéně obsahovalo mnoho kontrastních objektů. V takovém případě by algoritmy popsaly mnoho features na objektech v pozadí a počet samotných features na objektu, který má být lokalizován, byl se mohl snížit až na nedostatečnou úroveň.

Jako možné vylepšení pro dosažení lepších výsledků se nabízí použití plovoucího okénka, které využívají například kaskádové klasifikátory. Bylo by tak možné lokalizovat více objektů na scéně a díky rozdělení obrazu na méně částí by se zvýšila šance na detekci i ve scénách s mnoha objekty v pozadí. Samozřejmě tato úprava by měla značný dopad na celkovou efektivitu algoritmu. V neposlední řadě poté hraje důležitou roli kvalita samotného záznamu. Výhodou použitých metod je nezávislost na nehybnosti pozadí. Lze je tak využít i například s videosekvencemi zaznamenanými pomocí kamery držené v ruce.

4.4 Zpracování pohybu do teplotní mapy

Poslední částí této práce je zpracování pohybu ze sledované scény do teplotní mapy. Tuto část jsem představil v kapitole 2.3. Výsledek analýzy bude poté uložen v podobě obrázku do složky s prototypem této aplikace.

4.4.1 Detekce pohybu

Pro detekci pohybu použiji, jako v případě první části, metodu odečítání pozadí. Zde však budu pracovat přímo s binární bitmapou a nemusím tak zjišťovat pozice oblastí pohybujících se osob pomocí vyhledávání kontur. Pro pohodlnější práci a přehlednější kód jsem vytvořil třídu `Recorder`. Ta se bude zaměřovat na sběr dat a následné generování výsledného obrázku.

4.4.2 Sběr dat

Sběr dat probíhá do matice s datovým typem umožňující uchovat vysoké hodnoty. Z každého snímku videosekvence se analyzuje pohyb osob. Vzniklá binární bitmapa se následně přičte do matice pro sběr dat. K tomuto účelu jsem vytvořil ve třídě `Recorder` metodu `insertFrame()`. V prvním kroku bude potřeba převést binární bitmapu, neboť představuje obraz a každý pixel je zde reprezentován třemi hodnotami. Pro převod typů matice slouží ve třídě `cv::Mat` příkaz `convertTo()`. Aby bylo možné matice sečíst, musí se také rovnat jejich velikost. Tuto korekci a jakoukoli jinou změnu velikosti umožňuje metoda `cv::resize()`.

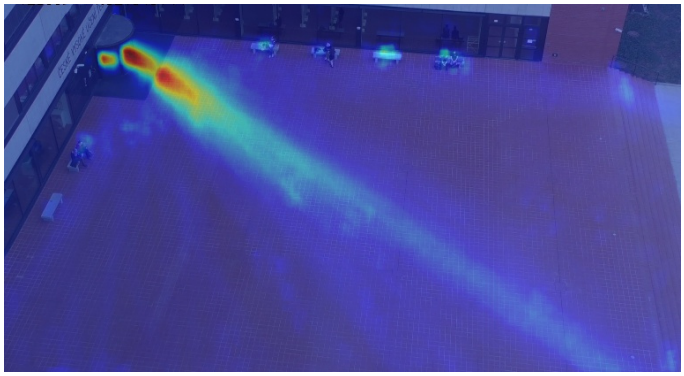
Bílá barva (pohybující se osoby) je v takto převedené matici reprezentována hodnotou 255 a černá barva (nehybné pozadí scény) hodnotou 0. Tyto hodnoty již mohu přičíst do matice pro sběr dat, nicméně abych zajistil její efektivní využití zmenším hodnotu pohybujících se objektů na hodnotu 1. Tuto úlohu snadno provedu pomocí procesu normalizace a k tomu určenou metodou `cv::normalize()`. Finální sečtení dvou matic poté umožňuje ve třídě `cv::Mat` metoda `add()`.

4.4.3 Převod hodnot

Poté co algoritmus umožňuje ukládat hodnoty pohybů zbývá implementovat způsob, jak tyto hodnoty zobrazit. Tento úkon zajišťuje ve třídě `Recorder` metoda `drawImage()`. Princip této funkce bude transformace hodnot matice z rozsahu od nuly do maximální hodnoty v matici na rozsah od nuly do 255 (hodnota bílé barvy). Rozšířením takto transformované matice na typ pro obrázková data vniká obraz ve stupních šedi. Černá barva v tomto obraze poté reprezentuje místa s minimálním výskytem pohybu a bílá barva naopak místa s nejvyšším místem pohybu.

Obrázek ve stupních šedi však není vhodný pro reprezentaci takových dat. Lidské oko dokáže lépe reagovat na změny v barevné složce, než na změny v jasu. Kvůli této vlastnosti musím všem pixelům přiřadit barvu podle hodnoty jejich jasu. Barva se vybírá na základě zvoleného barevného přechodu. Tím vznikne teplotní mapa, která je popsána v úvodní kapitole 1.1.1. Pro jednoduchý převod disponuje knihovna metodou `cv::applyColorMap()`. Na výběr má programátor z několika možných barevných přechodů a je tak možné pro jisté případy použít i jiných barevných kombinací než typické modro-červené pro standardní teplotní mapy.

Abych zachoval přehlednost o rušných místech ve sledované scéně, použiji vygenerovanou teplotní mapu jako poloprůhlednou vrstvu nad vybraným snímkem ze vstupního videozáznamu. Jako snímek jsem použil pozadí, které si uchovává implementace algoritmu pro odečítání pozadí. Díky tomu nebudou ve výsledném obraze pohybující se osoby, které by mohly rušit uživatele, jenž si jej bude prohlížet. Dvě matice stejného typu lze snadno sečíst po-



Obrázek 4.8: Výsledná tepelná mapa s lineárním měřítkem

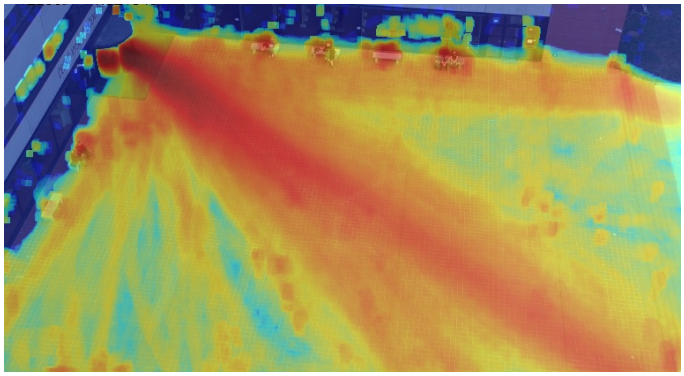
mocí specializované metody `cv::addWeighted()`. Ta umožňuje před sečtením pomocí dodatečných parametrů vynásobit dočasně obě sčítané matice. Nastavením obou parametrů na hodnotu 0,5 jsem následně docílil požadovaného efektu poloprůhledné vrstvy. Míru průhlednosti lze samozřejmě upravit podle potřeb, důležité je, aby se součet těchto parametr rovnal hodnotě 1.

Na obrázku 4.8 si lze prohlédnout výslednou tepelnou mapu, která vnikla po analýze 90minutového videa. Již na první pohled se může zdát, že neobsahuje příliš vypovídající hodnotu. Veškerý detekovaný pohyb je soustředěn na oblast před vchodem nacházející se v levém horním rohu sledované scény. Přesto však najde své uplatnění. Červená barva v této tepelné mapě znázorňuje místa, kterým projde většina pohybujících se osob. Tuto informaci lze uplatnit, pokud je potřeba provedení změn, které by měly ovlivnit co nejvíce osob.

Pro získání tepelné mapy, ve které jsou oblasti pohybu více viditelné, je nutné upravit metodu pro převod hodnot. Před transformací hodnot je nutné změnit jejich měřítko. V rámci mé práce jsem všechny hodnoty zlogaritmoval. Vysoké hodnoty v matici již tolik neovlivňovaly výslednou tepelnou mapu a nižší hodnoty mohly lépe vyniknout. Výsledný obrázek 4.9 představuje tepelnou mapu, která tímto procesem vznikla. Již je jasně patrné, že osoby se pohybují často od vchodu směrem k pravému dolnímu rohu sledované scény, případně rovně doprava. Více se však také projevují chyby vzniklé nedokonalým odečítáním pozadí.

4.4.4 Analýza úseků videosekvence

V některých případech může být užitečné vytvářet teplotní mapy po určitých časových úsecích a sledovat, jak se pohyb chodců mění postupem času. Jednoduchým řešením pro tento problém může být vytvoření instance třídy `Recorder` pro každý časový úsek. Takové řešení není efektivní. Implementoval jsem proto do zmíněné třídy novou metodu `deflate()`, která slouží ke snížení



Obrázek 4.9: Výsledná tepelná mapa s logaritmickým měřítkem

všech hodnot v matici pro sběr dat. O jak velký rozsah se sníží lze definovat pomocí parametru. Díky tomu lze pozorovat změny pohybu chodců v závislosti i na předchozích, již uplynulých úsecích. V testovaném videozáznamu se však pohyb chodců neměnil a teplotní mapy měřené z malých úseků byly prakticky stejné jako při měření na celém záznamu.

4.4.5 Shrnutí

Implementace tohoto prototypu byla poměrně přímočará. Více než hledání speciálních metod zabralo hledání správných metod pro efektivní počítání s maticemi v rámci knihovny OpenCV, aby nebyly zbytečně implementované znovu neznalým programátorem. Pro správné zpracování pohybu do tepelné mapy je obzvlášť důležitá pozice kamery. Výsledky získané pomocí mé metody by mohly být matoucí na videosekvencích zaznamenaných kamerami umístěnými blízko země. V těchto případech by ovlivňovali více prostoru osoby pohybující se blíže ke kameře než vzdálené osoby. V takovém případě by bylo potřeba použít jiné metody pro detekci pohybu ve videu.

Další případné vlastnosti prototypu by mohly být implementované na základně požadavků. Jako možné vylepšení stávajícího řešení se nabízí uložení rozpracovaného stavu matice pro sběr dat tak, aby bylo možné pokračovat ve zpracování videa v případě náhlého přerušení. Jako výborná dodatečná informace by mohl sloužit převládající směr pohybujících se osob v určitém místě. Vhodným algoritmus pro zprostředkování této funkce by mohl být z prototypu pro sledování osob.

Závěr

Zhodnocení

Cílem této bakalářské práce bylo analyzovat a seznámit se s dostupnými metodami počítačového vidění a navrhnout algoritmy pro sledování pohybu osob, lokalizaci objektu a zpracování informací o pohybu do teplotní mapy. V rámci této bakalářské práce jsem vytvořil pro každou rozebranou část v kapitole 2 prototyp aplikace, který splňuje zadané požadavky.

V každé části, která nabízela více metod řešení, jsem tyto metody vyzkoušel a na základně výsledků uvedl jejich vhodnost k řešení daného problému. Popsal jsem také případné procedury, které bylo nutno vykonat, pokud si to využívaná metoda vyžadovala. Každý navržený algoritmus představuje dobrý začátek pro případáné aplikace, které by vznikly na základě podobných požadavků.

Výhled do budoucna

Do budoucna by bylo určitě možné každý prototyp dále rozšiřovat o funkce, které byly navrženy v souhrnu na konci každé sekce v kapitole 4, kde byla popsána realizace. Všechny vytvořené prototypy v rámci této práce nebyly cílené pro běžné používání, přesto jsou schopné jisté problémy úspěně řešit. Jako možný krok se nabízí jejich začlenění do větších celků, které umožní pohodlnější práci s předáváním vstupních dat.

V průběhu realizace jsem si osvojil práci s knihovnou OpenCV a rozšířil své znalosti o metody využívané v odvětví počítačové vidění. Věřím, že získaných zkušenost jistě využiji i v budoucnu.

Literatura

[1] MAFRA a.s.: Přehled počasí - srážky, pátek 19.12.2014 17:30. [online], [cit. 2016-05-02]. Dostupné z: http://pocasi.idnes.cz/?d=19.12.2014

[2] Schnell, I.: Ilan Schnell - RGB Histogram. [online], [cit. 2016-05-08]. Dostupné z: http://ilan.schnell-web.net/prog/histogram/

[3] Hur, I. A.: *Novel computer vision algorithms for automated cell event detection and analysis*. Dissertation, University of Iowa, 2012.

[4] Černá, T.: Detekce a rozpoznávání registrační známk vozidla pro analýzu dopravy. [online], 2015, [cit. 2016-05-02]. Dostupné z: http://excel.fit.vutbr.cz/submissions/2015/031/31.pdf

[5] Itseez: OpenCV. [online], [cit. 2016-04-30]. Dostupné z: http://opencv.org/

[6] Sight Machine, I.: SimpleCV. [online], [cit. 2016-04-30]. Dostupné z: http://www.simplecv.org/

[7] Accord.Net. [online], [cit. 2016-04-30]. Dostupné z: http://accord-framework.net/

[8] CCV. [online], [cit. 2016-04-30]. Dostupné z: http://libccv.org/

[9] The VXL Project. [online], [cit. 2016-04-30]. Dostupné z: http://vxl.sourceforge.net/

[10] Rosten, E.: libCVD. [online], [cit. 2016-04-30]. Dostupné z: http://www.edwardrosten.com/cvd/

[11] Qualcomm Technologies, I.: FastCV. [online], [cit. 2016-04-30]. Dostupné z: https://developer.qualcomm.com/software/fastcv-sdk

[12] Neurotechnology: SentiSight. [online], [cit. 2016-04-30]. Dostupné z: http://www.neurotechnology.com/sentisight.html

[13] Itseez: How to Use Background Subtraction Methods. [online], [cit. 2016-04-30]. Dostupné z: http://docs.opencv.org/3.1.0/d1/dc5/tutorial_background_subtraction.html

[14] Mordvintsev, A.: Face Detection using Haar Cascades. [online], 2013, [cit. 2016-05-04]. Dostupné z: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

[15] Intel: Histogram of Oriented Gradients (HOG) Descriptor. [online], [cit. 2016-05-08]. Dostupné z: https://software.intel.com/en-us/node/529070

[16] Dalal, N.; Triggs, B.: Histograms of Oriented Gradients for Human Detection. In *International Conference on Computer Vision & Pattern Recognition*, ročník 2, editace C. Schmid; S. Soatto; C. Tomasi, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005, s. 886–893. Dostupné z: http://lear.inrialpes.fr/pubs/2005/DT05

[17] CEVA: Challenges in Object Detection on Embedded Devices. [online], [cit. 2016-05-08]. Dostupné z: http://www.slideshare.net/embeddedvision/2-c01-cevapaz

[18] Scikit-learn developers: Support Vector Machines. [online], [cit. 2016-05-08]. Dostupné z: http://scikit-learn.org/stable/modules/svm.html

[19] Introduction to Support Vector Machines. [online], [cit. 2016-05-09]. Dostupné z: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[20] Dalal, N.: INRIA Persons dataset. [online], [cit. 2016-05-09]. Dostupné z: http://pascal.inrialpes.fr/data/human/

[21] Gavrila: Daimler Pedestrian Segmentation Benchmark Dataset. [online], [cit. 2016-05-09]. Dostupné z: http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/daimler_pedestrian_benchmark_d.html

[22] Tatarize: Color Distribution Methodology. [online], [cit. 2016-05-09]. Dostupné z: http://godsnotwheregodsnot.blogspot.cz/2012/09/color-distribution-methodology.html

[23] Cascade Classifier Training. [online], [cit. 2016-05-09]. Dostupné z: http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[24] Ball, T.: Train Your Own OpenCV Haar Classifier. [online], [cit. 2016-05-09]. Dostupné z: `http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html`

[25] Alcantarilla, P. F.; Nuevo, J.; Bartoli, A.: Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In *British Machine Vision Conf. (BMVC)*, 2013.

[26] Leutenegger, S.; Chli, M.; Siegwart, R. Y.: BRISK: Binary Robust invariant scalable keypoints. *Computer Vision, IEEE International Conference on*, ročník 0, 2011: s. 2548–2555, doi:http://doi.ieeecomputersociety.org/10.1109/ICCV.2011.6126542.

[27] Rublee, E.; Rabaud, V.; Konolige, K.; aj.: ORB: An efficient alternative to SIFT or SURF. In *ICCV*, editace D. N. Metaxas; L. Quan; A. Sanfeliu; L. J. V. Gool, IEEE Computer Society, 2011, ISBN 978-1-4577-1101-5, s. 2564–2571. Dostupné z: `http://dblp.uni-trier.de/db/conf/iccv/iccv2011.html`

Příloha **A**

Seznam použitých zkratek

CV Computer Vision

GUI Graphical user interface

ID Identifikátor

XML Extensible markup language

Příloha B

Uživatelská příručka

Všechny implementace vyžadují ke spuštění knihovnu OpenCV 3.1.0

B.1 BackgroundSubtraction

Příručka k prototypu umožňující sledování pohybu osob.

Parametry

−vid <cesta> – Cesta k videozáznamu na kterém bude provede detekce a sledování chodců.

Klávesové zkratky

Q – ukončení aplikace.

B.2 FeatureObjectLocator

Příručka k prototypu, který lokalizuje objekty pomocí sledování features.

Parametry

<cesta> – První parametr určuje cestu pro refrenční obrázek, který bude představovat vyhledávaný objekt.

−vid <cesta> – Cesta k videozáznamu, ve kterém bude objekt vyhledáván, nelze kombinovat s parametrem −img.

−img <cesta> – Cesta k bitmapovému souboru, ve kterém bude objekt vyhledáván, nelze kombinovat s parametrem −vid.

Pokud není zadaný parametr −img ani −vid bude objekt vyhledáván z proudu výchozí webkamery.

Klávesové zkratky

Q – ukončení programu.

B.3 Heatmap

Příručka k prototypu umožňující zpracování pohybu do teplotní mapy.

Parametry

−vid <cesta> – Cesta k videozáznamu ve kterém bude pohyb zpracováván

Klávesové zkratky

Q – ukončení aplikace.

S – uložení aktuálních tepelných map do souboru.

C – nastavení matice pro sběr dat na počáteční hodnoty.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `samples` ..... adresář s testovacími videozáznamy
- `data` ..... adresář s dodatečnými daty pro všechny prototypy
- `bin` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `background_subtraction` ..... zdrojové kódy pro sledování osob
    - `object_locator` ..... zdrojové kódy pro lokalizaci objektu
    - `heatmap` ... zdrojové kódy pro zpracování pohybu do teplotní mapy
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Baloun_Lubomír_2016.pdf` ..... text práce ve formátu PDF