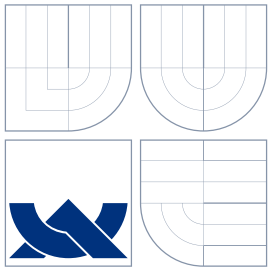
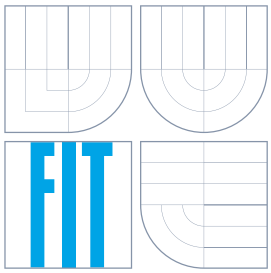


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČASOVÝ SNÍMEK Z OBRAZU STACIONÁRNÍ KAMERY

TIME LAPSE FROM STATIONARY CAMERA IMAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ TUREK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá problematikou časového snímku ze stacionárních kamer. Jsou zde analyzovány nežádoucí jevy, které v časovém snímku vznikají, a navrženy algoritmy pro překonání těchto omezení. Jednotlivé algoritmy byly implementovány a následně porovnány na pořízené sadě testovacích dat. Výsledná aplikace vytváří ze vstupního videa pokročilý časový snímek a obsahuje možnost výběru techniky zpracování včetně nastavení příslušných parametrů.

Abstract

The topic of this master's thesis is the time lapse from stationary camera images. Unwanted phenomena, which arise in time lapse, were analyzed and algorithms to overcome these limitations were designed. The algorithms were implemented and compared using the captured dataset. The resulting application creates time lapse from the video input and allows users to choose the processing technique including the setting of appropriate parameters.

Klíčová slova

časosběrné video, stacionární kamera, zpracování obrazu, pokročilý časový snímek, lokální filtrace, neuniformní vzorkování, Markovovská náhodná pole

Keywords

time lapse video, stationary camera, image processing, advanced time lapse, local filtering, nonuniform sampling, Markov random fields

Citace

Lukáš Turek: Časový snímek z obrazu stacionární kamery, diplomová práce, Brno, FIT VUT v Brně, 2015

Časový snímek z obrazu stacionární kamery

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Turek
22. května 2015

Poděkování

Děkuji vedoucímu mé práce doc. Ing. Adamu Heroutovi, Ph.D. za cenné rady a náměty, které mi v začátcích pomohly vidět možné přístupy k danému tématu z větší perspektivy, a za konstruktivní kritiku a ochotu na konzultacích v průběhu řešení.

Obsah

Kapitola 1

Úvod

Časosběrná videa jsou pořizována za účelem zachycení dlouhodobých změn v obraze, které při sledování scény v reálném čase snadno uniknou pozornosti. Typickým příkladem je pohyb mraků či putování slunce po obloze. Naopak krátkodobé změny ve zrychleném videu narušují celkovou plynulost a rozptylují pozornost diváka. Tato práce si klade za cíl sumarizovat nežádoucí jevy, kterými časosběrná videa trpí, navrhnout a implementovat postupy pro jejich eliminaci a na pořízené sadě testovacích dat provést srovnání těchto postupů.

Jednou z možností, jak časosběrné video vytvořit, je snímkování fotoaparátem s konstantní časovou prodlevou mezi expozicemi (tzv. *time-lapse photography*). Druhým možným způsobem pro dosažení téhož výsledku je zpracovávat video tak, že s danou periodou vzorkujeme sekvenci snímků. Středem zájmu této práce je právě druhá zmíněná metoda a hlavní myšlenka tkví v efektivním využití okolních snímků u vybraného vzorku tak, aby se výstupní video jevilo jako více ucelené. Narozdíl od jiných úloh z oblasti počítačové grafiky, jako je například rozpoznávání a klasifikace, se tedy v tomto případě nejedná o výstup s jasně měřitelnými výsledky. Důraz je kladen na zprostředkování estetického zážitku.

Následující kapitola definuje pojem pokročilého časového snímku, shrnuje současný stav v této oblasti a ukazuje možné přístupy. Popis použitých algoritmů, které byly nastudovány v rámci teoretické přípravy, čtenář nalezne v kapitole 3. Další text je již věnován samotnému řešení. Kapitola 4 je o způsobech získávání dat pro analýzu nežádoucích jevů a získaných poznatcích. Následuje návrh algoritmů pro odstranění těchto jevů a dále návrh a implementace výsledné aplikace. V poslední části textu diskutuji dosažené výsledky, srovnávám výstupy navržených algoritmů s naivním časovým snímkem a hodnotím celkovou výkonnost.

Tato diplomová práce navazuje na můj semestrální projekt (viz [12]), ze kterého čerpají kapitoly 2, 3, 4 a 5. Zde jsou obsaženy části původního textu doplněné o nové poznatky.

Kapitola 2

Pokročilý časový snímek a současné možnosti jeho tvorby

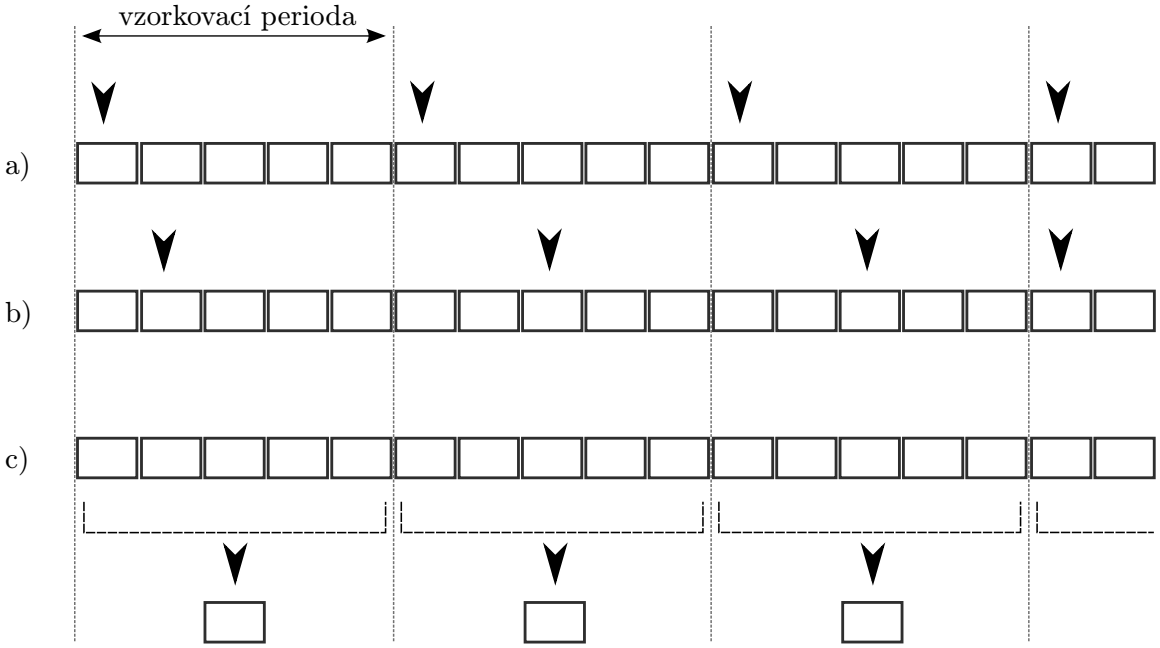
Jak vzniká klasický časový snímek? Jaká je motivace pro jeho vylepšování? A jakými způsoby toho lze dosáhnout? To jsou otázky, na které odpovídá následující kapitola, kde čtenář získá náhled do řešené problematiky. Jsou zde vytyčeny hlavní cíle této diplomové práce a naznačeny cesty, které mohou vést k jejich naplnění. Část textu je věnována analýze současných řešení, produkčních i akademických, z nichž některé byly inspirací pro tuto práci. Zároveň zde také naznačuji, jakým způsobem se chci od těchto metod odlišit.

2.1 Od primitivního časového snímku k pokročilému

V první řadě je potřeba si uvědomit, že ať už vzniká časový snímek z jakéhokoli důvodu (marketing, monitorování, zábavní účely, apod.), většinou se autor snaží o zachycení nějakého konkrétního dlouhodobého jevu - svítání/stmívání, změna počasí, vývoj dopravní situace, či ještě dlouhodobější změny jako je například práce na stavbě. V každém případě je klíčové správné nastavení vzorkování videa (či prodlev mezi expozicemi v případě snímání fotoaparátem) v závislosti na prvku scény k němuž upínáme svou pozornost. Ovšem s prodlužováním periody mezi snímky zpravidla přibývá krátkodobých změn, jejichž vývoj ve výsledném videu nezachytíme. Při dlouhodobém snímání scény v řádech hodin až měsíců při tomto zrychlení dochází k poblikávání částí obrazu, náhodnému objevování se a mizení objektů a nepřirozeným trhavým pohybům. V oblasti zpracování signálů je tento jev označován jako aliasing a abychom mu předešli, je nutné mít vzorkovací frekvenci vyšší než je dvojnásobek nejvyšší frekvence vzorkovaného signálu, viz Shannonův vzorkovací teorém [11]. To je však protichůdný požadavek u časosběrných videí, které potřebují vzorkovací periodu v řádech sekund až hodin. Článek s názvem „How fast is too fast?“ [14] se zabývá nastavením optimální rychlosti videa v přehrávači při zrychleném převíjení záznamu a z provedených experimentů na uživatelích vyhodnocuje rychlost 64x jako horní hranici, kdy je ještě rychlost videa subjektivně přijatelná a lze pojmout změny, které se v obraze dějí. Hlavní motivací pro pokročilé zpracování časového snímku je tedy možnost vytvořit velmi zrychlené video a současně předcházet nežádoucím jevům ve výsledné sekvenci snímků. To vše s využitím původního videa pořízeného v reálném čase, jeho analýzou a následným zpracováním.

Předmětem zkoumání je tedy otázka, jakým způsobem zpracovat okolní snímky pro dosažení optimálního výsledku. Jednou z možností je provést výběr optimálního snímku,

který bude z daného okolí nejlépe reprezentovat danou scénu, tj. upřednostní snímek zachycující pouze statické objekty. Jak se však později ukázalo na provedených experimentech, řešení založené na tomto principu vede jen k nepatrnému vizuálnímu zlepšení ve srovnání s primitivním časovým snímkem. Sofistikovanější variantou je sekvenci snímků zkombinovat, přičemž nalezení k tomu vhodného algoritmu je právě tou největší výzvou. Schematicky jsou jednotlivé skupiny metod naznačeny na obrázku 2.1.



Obrázek 2.1: Různé varianty, jak vytvořit časový snímek z videa v reálném čase: a) Naivní časový snímek, b) Výběr nejvhodnějšího snímku, c) Využití všech okolních snímků

2.2 Současné metody

V rámci teoretické přípravy na tvorbu pokročilého časového snímku jsem analyzoval současný stav v tomto odvětví. Vydal jsem se dvěma základními směry, a to hledáním aktuálně dostupného softwaru a studiem publikací z akademického výzkumu v této oblasti.

2.2.1 Analýza trhu

Velkou inspirací bylo diskusní fórum věnující se problematice časového snímku [2], které mi pomohlo udělat si představu o technikách, které používají autoři videí pro dosažení vizuálně atraktivních výsledků. Velká část návodů a rad z této oblasti se týká přímo nastavování času expozice a intervalu mezi jednotlivými expozicemi. Prodloužením expozičního času lze například docílit rozmazání pohybujících se objektů, které by ve zrychlené sekvenci způsobovaly efekt přeblikávání, a zlepšit tak zdánlivou kontinuitu videa.

Co se týče samotné postprodukce, kombinací programů pro editaci fotografií a videa lze výše zmíněné jevy manuálně odstranit. Existují i komplexní řešení, například software LRTimelapse zpracovávající snímky dávkově. Specificky zaměřené jsou aplikace Panolapse (pro přidání pohybu do videa ze stacionární kamery) a RAWBlend (interpolace nastavení

expozice, vyvážení bílé apod. pro odstranění efektu problikávání snímků). Ve všech případech se však počítá s interakcí s uživatelem, který výstup vytváří manuálně. Navrhovaná aplikace by měla vstupní data zpracovat automaticky bez nutnosti dodání explicitní znalosti o nežádoucích jevech.

2.2.2 Výzkumy a odborné publikace

Na podobné téma již byla vytvořena diplomová práce v Edinburghu pana J. Ortize [8], která řeší jeden konkrétní příklad scény snímané několikrát denně po dobu tří let. Autor zde používá metodu jádrového odhadu hustoty pravděpodobnosti pro vytvoření jednoho denního snímku ze všech nasnímaných v daný den. Ze sekvence denních snímků se dále snaží detekovat a odstranit popředí, přičemž porovnává použití naivního Bayesovského klasifikátoru, vzájemné korelace a neuronových sítí. Toto řešení je ovšem navrženo jen pro tuto jednu konkrétní scénu a nejedná se o obecně použitelné postupy.

Odlišný pohled nabízí vědecký článek *Motion Denoising with Application to Time-lapse Photography* [10], který zpracovává již vytvořené časosběrné video. Krátkodobé změny v obraze jsou odfiltrovány přeskupením pixelů v čase a prostoru a na formální úrovni je tento problém definován jako optimalizace energie na Markovovském náhodném poli (MRF) s využitím algoritmu *Loopy belief propagation* (LBP). Podstatnou nevýhodou tohoto řešení je velká výpočetní náročnost. V závěru je uvedeno, že sekvenci snímků o velikosti 300^3 trvalo zpracovat 50 hodin na stroji s dvoujádrovým procesorem na frekvenci 2.66 GHz a 26 GB pamětí RAM.

Práce s názvem *Computational time-lapse video* [4] se opírá o vzorkování s proměnnou periodou, které zohledňuje dynamiku scény. Vstupem je video v reálném čase a počet snímků ve výsledném zrychleném videu. Výstupem jsou snímky vybrané tak, aby byla minimalizována celková penalizace výstupního videa. Představeny jsou dvě metriky pro určení penalizace mezi dvěma snímky - jedna zachovávající maximální množství pohybu z původního videa a druhá vyřazující snímky, které se výrazně liší od průměru. V rámci diplomové práce jsem vyzkoušel implementaci této metody a dále jsem navrhl algoritmus pro neuniformní vzorkování, který nevynucuje pevný počet snímků ve výsledném videu, ale pracuje mnohem rychleji.

2.3 Možné využití a přínos

Výsledek této diplomové práce slouží jednak jako přehled různých metod, jak efektivně zpracovat video z IP kamery pro vytvoření časového snímku, ale výstupem je také sada programů, které demonstrují jednotlivé techniky a lze je přímo použít pro vytvoření pokročilého časového snímku z živého vysílání nebo videa nahraného na disku.

Jednou z hlavních charakteristik výsledné aplikace je co největší zautomatizování procesu zpracování videa tak, aby po uživateli nebyla vyžadována interakce při běhu programu. Na druhou stranu by ale měl mít možnost ovlivnit některé parametry jako například vzorkovací frekvenci. Algoritmické zpracování vstupních dat bez nutnosti manuální editace snímků je tedy hlavní změnou proti současným řešením dostupným na trhu. Další specifikum je zaměření na online streamy, tj. živé vysílání kamer na internetu. Tím se otevírají možnosti pro zpracovávání dat v reálném čase již při pořizování záznamu (například výběr klíčových snímků, první fáze filtrace nevyhovujících snímků, tvorba metadat,...). V navrhované podobě výsledné aplikace se počítá se statickými kamerami snímající stále ze stejného bodu pod stejným pozorovacím úhlem. Do budoucna by navíc mohly být možnosti používaných

vstupních dat rozšířeny o otočné kamery, které zabírají širší pozorovací úhel v horizontálním směru až do 360°. Výsledkem by pak byl panoramatický časový snímek.

Aplikaci může využít kdokoli se znalostí adresy daného streamu. Odlišnou motivaci budou mít zřejmě přímo majitelé kamer než ostatní uživatelé, kteří by si časový snímek z veřejných kamer pořizovali pravděpodobně pro zábavní účely. Pro majitele kamer to může být účinný marketingový nástroj, který podpoří návštěvnost daného místa. Může se jednat o lyžařské středisko či horskou chatu disponující výhledem do okolí a tématem může být například východ Slunce nad obzor. Dlouhodobě může být perspektivní použití pro monitorování vývoje krajiny, např. růst nových stromů po vysázení či dokumentace vzniku nové stavby. V současné podobě je aplikace navržena jako uživatelský program, ale v případě migrace na server a zpřístupnění jako online služba by byla tvorba těchto dlouhodobých časových snímků pro koncového uživatele mnohem jednodušší, neboť by stačilo zadat úlohu na server a vyzvednout si hotové video ze zaslaného odkazu bez nutnosti běhu vlastního počítače po dobu sběru dat.

Kapitola 3

Algoritmy pro zpracování obrazových dat

V této kapitole jsou blíže popsány algoritmy, které jsem ve své diplomové práci využil a implementoval. Konkrétní použití včetně dalších modifikací a rozšíření uvádím v kapitole 5. Zde se jedná o obecný popis použitých technik, které zahrnují optický tok, Markovovská náhodná pole a dynamické programování.

3.1 Optický tok

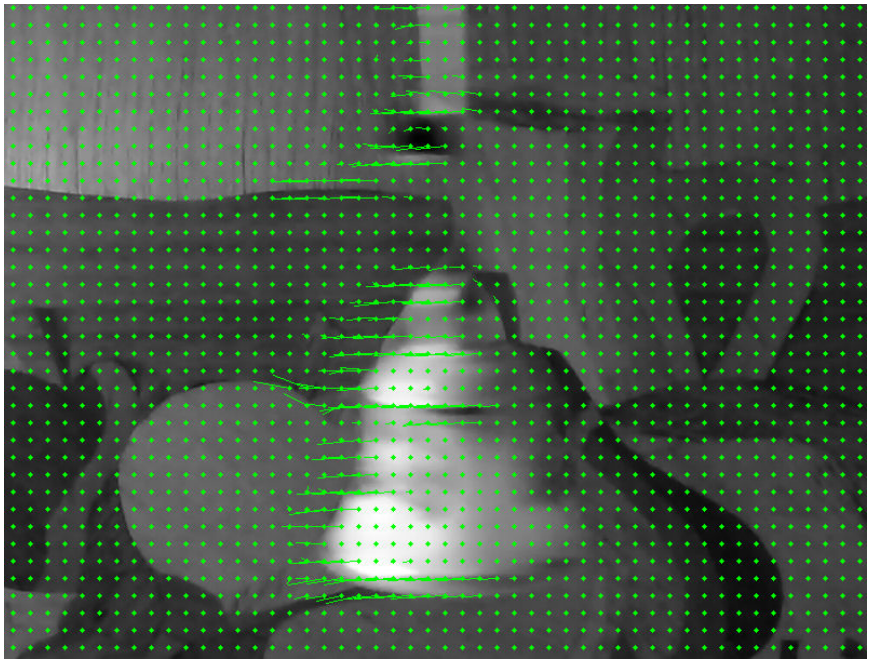
Optický tok vyjadřuje zdánlivý pohyb ve scéně, který vzniká pohybem objektů nebo pozorovatele (kamery), viz [13]. Metody pro odhad optického toku se dále dělí na řídké a husté. Toto označení je odvozeno od počtu bodů, které mají přiřazeny vektor toku. Zatímco pro hustou verzi tento vektor existuje pro každý pixel v obraze, řídká verze sleduje pouze několik význačných bodů (např. metoda Lucas-Kanade).

Výstupem metody hustého optického toku je tedy vektorové pole přiřazující každému bodu v obraze směr a velikost zdánlivého pohybu na základě rozdílu sousedních snímků. Nejedná se o reprezentaci skutečného pohybu. Jedním z důvodů je projekce třírozměrného prostoru do 2D obrazu, a tedy například pohyb v ose kamery nelze identifikovat. Dalším důvodem je, že tyto metody ze svého principu pracují s gradienty v obraze a tudíž např. pohyb jednobarevné plochy nebude odhalen. Příkladem, kdy skutečný pohyb není reprezentován správně, je také např. otáčení spirály, kde vektor optického toku bude kolmý na skutečný vektor pohybu. Pro účely aplikace zpracovávající časový snímek však není určení reálného pohybu stěžejní.

Jedním z algoritmů pro výpočet hustého optického toku je Farneback [6]. Optický tok je počítán mezi dvěma sousedními snímky. V prvním kroce je okolí obou snímků aproximováno kvadratickými polynomy. Hlavní myšlenkou je z původního signálu f_1 zkonstruovat nový signál $f_2(x) = f_1(x-d)$ a řešením rovnice nalézt globální posun d. Na obrázku 3.1 je ukázka použití tohoto algoritmu, kde je výsledek vizualizován mřížkou a vektory z ní vycházející znázorňují velikost optického toku v těchto bodech.

3.2 Markovovská náhodná pole

Pokročilejší technikou, kterou jsem se zabýval v rámci výběru vhodných algoritmů z oblasti počítačové grafiky, jsou Markovovská náhodná pole (MRF - Markov random fields), viz [5].



Obrázek 3.1: Vizualizace okamžitého optického toku ve videu.

V následujícím textu vysvětlím základní myšlenku a ukážu konkrétní použití pro filtraci videa.

3.2.1 Obecné principy

Motivací ke studiu tohoto formalismu bylo omezení lokálních filtrů, kde vznikaly artefakty kvůli ztrátě kontextu v časoprostoru. Nyní připouštíme, že okolní pixely jsou na sobě statisticky závislé. Hlavní myšlenky formalismu MRF jsou následující:

- Obraz je rozdělen na skupinu uzlů, které odpovídají jednotlivým pixelům nebo skupinám pixelů.
- Uzlům jsou přiřazeny tzv. „skryté proměnné“, které vysvětlují význam hodnot pixelů. Např. při segmentaci na popředí/pozadí nabývají skryté proměnné hodnoty 0 nebo 1.
- Nad hodnotami pixelů a hodnotami skrytých proměnných je vytvořen pravděpodobnostní model.
- Přímá závislost mezi skrytými proměnnými je znázorněna jejich slučováním, v grafu znázorněným hranami spojujícími uzly (tj. skryté proměnné).

Statistická závislost skrytých proměnných je explicitně znázorněna jen mezi sousedními uzly. Korelace vzdálenějších uzlů jsou vyjádřeny implicitně propojením celé sítě. Nejjednodušším Markovovským modelem je Markovovský řetězec, což je sled po sobě následujících stavů (příklad: předpověď počasí). Pokud známe matici $M_i(x, x') = P(X_i = x \mid X_{i-1} = x')$ určující pravděpodobnost následujícího stavu při znalosti předchozího, pak můžeme pravděpodobnost tří po sobě následujících hodnot vyjádřit násobením matic:

$$P(X_i = x \mid X_{i-2} = x'') = \sum_{x' \in L} M_i(x, x') M_{i-1}(x', x'') \tag{3.1}$$

Hodnoty proměnných X_i mohou být odvozeny z odpovídající množiny pozorování $z = (z_1, z_2, ..., z_i, ..., z_N)$ s použitím Bayesova pravidla:

$$P(X = x \mid Z = z) \propto P(Z = z \mid X = x) P(X = x) \tag{3.2}$$

kde $P(X = x)$ je apriorní rozdělení pravděpodobnosti, tedy pravděpodobnost jednotlivých stavů X, pokud nemáme k dispozici žádná pozorování. Ve většině případů předpokládáme, že hodnota pozorování ve stavu i závisí pouze na odpovídající proměnné X_i.

Typickou úlohou nad Markovovským náhodným polem v oboru počítačového vidění je odhad maximální aposteriorní pravděpodobnosti sekvence stavů x na základě dat z, tedy $\hat{x} = \text{argmax}_x P(x \mid z, \omega)$. Algoritmy, které toto řeší, jsou založeny na principu propagace zpráv mezi uzly, zástupcem je algoritmus „max-product belief propagation“. Dopřednou rekurzí lze spočítat pravděpodobnost aktuálního stavu ze znalosti předchozích stavů a pozorování:

$$P(x_i) = P(z_i \mid x_i, \omega) max_{x_{i-1}} P(x_i \mid x_{i-1}, \omega) P_{i-1}(x_{i-1}) \tag{3.3}$$

Po dopředném průchodu následuje zpětná rekurze, na jejímž konci získáme konečný vektor x stavů s maximální aposteriorní pravděpodobností.

Markovovský řetězec lze dále zobecnit na obecný graf, tedy Markovovské náhodné pole. Pro vyjádření statistických závislostí v obraze se nejčastěji používá čtyř-okolí, tedy každý uzel (reprezentující pixel nebo skupinu více pixelů) je propojen se čtyřmi okolními uzly. Odhad maximální aposteriorní pravděpodobnosti $\hat{x} = \text{argmax}_x P(x \mid z, \omega)$ lze také ekvivalentně zapsat jako minimalizaci energie $\hat{x} = \text{argmin}_x E(x, z, \omega)$. Funkce pro výpočet energie je rovna sumě unárních potenciálů (uzly) a párových potenciálů (hrany mezi uzly).

3.2.2 Filtrace krátkodobých změn ve videu

Konkrétní aplikaci Markovovských náhodných polí na filtraci videa představuje článek „Motion Denoising with Application to Time-lapse Photography“ [10]. Vstupem není video v reálném čase, ale již vytvořený časový snímek. Autoři této práce se snaží o rozdělení videa na dvě složky - krátkodobé a dlouhodobé změny v obraze, přičemž ve výsledku ponechávají pouze ty dlouhodobé. V rámci mé diplomové práce jsem jejich řešení implementoval a porovnal s ostatními algoritmy, více v podkapitole 5.3.

Vstupem tohoto systému je videosekvence $I(x, y, t)$ s RGB intenzitami v rozsahu [0,255], výstupem je sekvence $J(x, y, t)$ vzniklá přeuspořádáním pixelů z původního videa. Chceme, aby se výstupní sekvence podobala vstupní a zároveň mělo výstupní video hladké přechody v čase, což při definování problému jako minimalizace energie vede k následující formulaci:

$$E(J) = \sum_{x,y,t} |J(x, y, t) - I(x, y, t)| + \alpha \sum_{x,y,t} |J(x, y, t) - J(x, y, t + 1)| \tag{3.4}$$

Výstupní video lze definovat pomocí vstupního videa následovně:

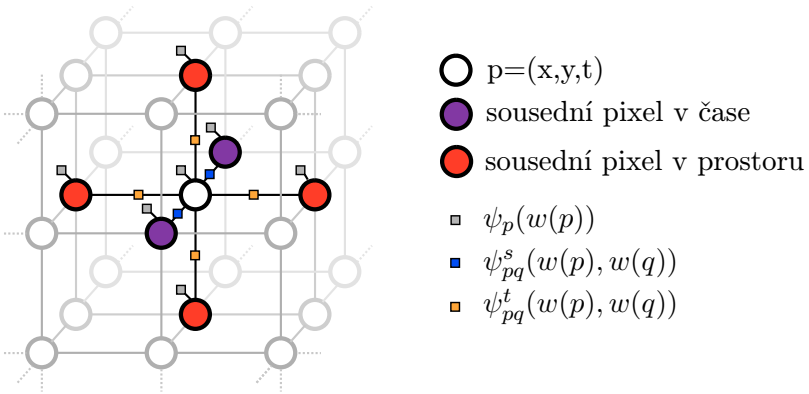
$$J(x, y, t) = I(x + w_x(x, y, t), y + w_y(x, y, t), t + w_t(x, y, t)) \tag{3.5}$$

pro posunutí pixelů v časoprostoru $w(x, y, t) \in (\delta_x, \delta_y, \delta_t) : |\delta_x \leq \Delta_s|, |\delta_y \leq \Delta_s|, |\delta_t \leq \Delta_t|$, kde Δ_s a Δ_t jsou parametry definující velikost prohledávané oblasti v prostoru a čase.

Konečně dosazením vztahu 3.5 do vztahu 3.4, parametrizací $p = (x, y, t)$ a přidáním dalšího členu pro konzistenci posunutí, dostáváme finální vzorec:

$$\begin{aligned} E(w) = \sum_{p} |I(p + w(p)) - I(p)| + \\ \alpha \sum_{p,r \in \mathcal{N}_t(p)} ||I(p + w(p)) - I(r + w(r))||^2 + \\ \gamma \sum_{p,q \in \mathcal{N}(p)} \lambda_{pq} |w(p) - w(q)| \end{aligned} \tag{3.6}$$

kde $\mathcal{N}(p)$ jsou sousedé pixelu p v časoprostoru a $\mathcal{N}_t(p) \subseteq \mathcal{N}(p)$ jsou sousedé pixelu p v čase. Parametry α a γ regulují vliv jednotlivých složek.



Obrázek 3.2: Pixel v kontextu 3D MRF mřížky. Obrázek znázorňuje okolí pixelu v časoprostoru s unárními a párovými potenciály. Grafický model odpovídá vztahu 3.6.

Tento vztah je optimalizován na trojrozměrném MRF, kde každý uzel odpovídá jednomu pixelu videa. Stavový prostor je množina možných posunutí jednoho pixelu v trojrozměrném prostoru (x,y,t). Technikou, kterou je možné použít pro výpočet posunutí všech pixelů s ohledem na minimalizaci výsledné energie, je Loopy Belief Propagation (LBP) [15].

3.3 Neuniformní vzorkování s využitím dynamického programování

Vzorkování s proměnnou periodou je alternativní přístup k tomu, jak se vypořádat s rychlými změnami ve scéně. E. P. Bennett a L. McMillan navrhli dvě metriky pro výpočet optimálního vzorkování při zadané délce výstupního videa a požadovaných vizuálních charakteristikách, viz [4]. V následujícím textu popisuji jejich řešení, které jsem implementoval a otestoval a které mě inspirovalo pro vytvoření vlastního algoritmu pro neuniformní vzorkování (více v kapitole 5.4). Níže zmíněné metriky lze dále doplnit o penalizaci za příliš velké rozdíly ve vzdálenosti mezi snímky, pokud chceme, aby se výsledek alespoň do určité míry blížil uniformnímu vzorkování:

$$\Upsilon(i, j) = \frac{j - i - T_d}{T_d} \tag{3.7}$$

kde T_d je požadovaná perioda. Tuto penaltu pak se zvolenou váhou můžeme přičíst k jedné ze zvolených metrik.

3.3.1 Metrika min-error

Metrika min-error definuje jako optimální video takový výstup, který zachovává co nejvíce z původního pohybu ve scéně. Tato úloha v praxi znamená nalézt M snímků, jejichž lineární interpolace aproximuje původní video s nejmenší chybou.

Definujeme tedy metriku $\Delta(i,j)$, která měří chybu mezi pixely z původního videa a pixely vzniklými interpolací mezi počátečním snímkem i a koncovým snímkem j:

$$\Delta(i,j) = \sum_x \sum_y [\sum_{k=i}^{j} ((a_{ij}^{xy} + b_{ij}^{xy} k) - s_k^{xy})^2] \tag{3.8}$$

$$a_{ij}^{xy} = s_i^{xy} - b_{ij}^{xy} i \qquad b_{ij} = (s_j^{xy} - s_i^{xy})/(j-i) \tag{3.9}$$

Autoři dále přidávají další parametry α, β, γ pro modifikaci této metriky, kde α a β slouží pro tvarování průběhu a γ jako práh pro scény s výrazným šumem:

$$\Delta'(i,j) = (\beta MAX(O, \Delta(i,j) - MAX(0,(j-i-1)\gamma)))^{\alpha} \tag{3.10}$$

3.3.2 Metrika min-change

Metrika min-change definuje jako optimální video takový výstup, který vybere navzájem si nejpodobnější snímky, tedy vyřazuje z výběru snímky výrazně se lišící od ostatních. Tato metrika je součtem čtverců odchylek:

$$\delta(i,j) = \sum_x \sum_y [(s_j^{xy} - s_i^{xy})^2] \tag{3.11}$$

3.3.3 Dynamické programování

K nalezení optimálního řešení s ohledem na zvolenou metriku je možné využít dynamické programování, konkrétně algoritmus Perez and Vidal [9]. Původní použití algoritmu je pro výpočet optimální aproximace křivky přímkami, ale je aplikovatelný také na hledání optimálního vzorkování videa. Změní se pouze použitá metrika na jednu z výše uvedených.

Dynamické programování obecně spočívá v rozdělení komplexního problému na jednodušší podproblémy, které jsou řešeny samostatně a jejich výsledky jsou pak sloučeny. Při správném použití dynamického programování se zkrátí výpočetní doba eliminací výpočtů, které by se jinak zbytečně prováděly vícekrát.

Algoritmus Perez and Vidal pracuje s množinou po sobě následujících bodů NP. Cílem je nalézt podmnožinu bodů $NS+1$, které minimalizují globální chybu. V následujícím pseudokódu algoritmu je použito pole reálných čísel $g[1..NP, 0..NS]$ pro zapamatování si minimální chyby při dosažení jakéhokoli bodu s jakýmkoli počtem segmentů. $Father[1..NP, 1..NS]$ je pole, které ukládá počáteční bod předchozího segmentu pro zpětnou rekonstrukci celé cesty. Následuje pseudokód původního algoritmu Perez and Vidal:

Algoritmus 1 Perez and Vidal

g[1,0] = 0;
for n=2 TO NP **do**
  g[n,0] = MaxReal;
end for
for m=1 TO NS **do**
  for m=2 TO NP **do**
    $g[n,m] = min(g[i,m-1] + error(i,n) | i \in [m,n-1])$;
    $father[n,m] = i_{min}$;
  end for
end for
TotalError = g[n,m];

Z pseudokódu lze odvodit, že složitost tohoto algoritmu je $O(n^2.m)$, což je výrazně lepší než řešení hrubou silou, které by procházelo všechny možné kombinace, kterých je $\frac{(NP+NS-1)!}{NS!(NP-1)!}$.

Kapitola 4

Sběr dat a analýza nežádoucích jevů

Před započetím implementační fáze řešení bylo zapotřebí důkladně analyzovat klasický časový snímek a na reprezentativní množině testovacích dat klasifikovat nejčastější jevy, které působí rušivě. Za tímto účelem bylo vybráno a následně nahráváno 58 veřejně dostupných internetových kamer s důrazem na rozmanitost typů scén. Pořízené časové snímky byly dále podrobeny vizuálnímu zkoumání.

Určení, které aspekty v konkrétním videu působí rušivě, je ze své podstaty subjektivní. Proto jsem nejprve začal hledat informace o tom, co považují za problém lidé zabývající se technikou tvorby časosběrných snímků dlouhodobě. Inspirací bylo mimo jiné také diskusní fórum [2], zvláště pak témata věnující se následnému zpracování pořízených snímků. Toto jsem dále doplnil o vlastní poznatky z pořízených dat.

4.1 Snímky z IP kamer

První data byla získána připojením k živému vysílání internetových kamer přes protokol RTMP, resp. RTSP. Za tímto účelem jsem vytvořil jednoduchou aplikaci, která s využitím knihovny FFMpeg analyzuje jednotlivé rámce a vybírá pouze ty, které mají v hlavičce příznak označující snímek jako klíčový (tzv. *I-frame*). Vybrané snímky se ukládají na disk k dalšímu zpracování.

Použity byly veřejně dostupné statické IP kamery, typicky umístěné na budovách. Scény byly voleny s důrazem na různorodost, aby bylo možné později porovnat účinnost algoritmů na videu z odlišného prostředí.

Od poloviny listopadu 2014 do konce března 2015 jsem prováděl pravidelný sběr dat z vybraných IP kamer, které jsem následně použil jako podklady pro řešení problému dlouhodobého časového snímku. Tento proces jsem zautomatizoval vytvořením konzolové aplikace, která jako vstupní parametry dostává adresy streamů, ke kterým se připojí a ukládá na disk klíčové snímky. Takto bylo třikrát denně sbíráno 100 snímků z vybraných kamer. K zautomatizování tohoto procesu jsem využil příkazu *at* na školním linuxovém serveru *merlin*, který vykoná požadovanou operaci v zadaný den a čas.

4.2 Vlastnoručně pořízená videa

Reálné scény jsou často příliš komplexní a obsahují celou škálu nežádoucích jevů dohromady. Abych se mohl zaměřit na jednotlivé kategorie vyjmenované v následujícím textu, začal jsem natáčet svá vlastní videa fotoaparátem upevněným na stativu. Omezujícím faktorem ze strany fotoaparátu byla maximální možná délka záznamu 30 minut, to se však ukázalo jako dostačující pro střednědobé jevy jako je například pohyb mračen. Konkrétní využití fotoaparátu je demonstrováno u případové studie s kyvadlem, viz kapitola 5.2.

4.3 Inspirace odjinud

Většina veřejných diskusí se týká spíše technik vlastního snímání fotoaparátem a nastavování parametrů při pořizování snímků. Například i trhavé pohyby vzniklé zrychlením scény s pohybujícími se objekty bývají eliminovány použitím kratší prodlevy mezi expozicemi, a tedy menším výsledným zrychlením. Pro zlepšení vizuálního dojmu se často používá i delší expozice s větším zacloněním objektivu, což vyústí v sérii fotografií s rozmazaným pohybujícím se objektem, která v rychlém sledu za sebou vypadá lépe.

Často skloňovaným pojmem je tzv. „flickering“, což je poblikávání obrazu jako následek automatické měření expozice a vytvoření sekvence snímků s odlišnou dobou, po kterou byla závěrka otevřena. Běžným řešením je použití softwaru, který ze sekvence snímků spočítá průměrný jas a u všech snímků jejich jas upraví na tuto hodnotu. S tímto problémem jsem se však v případě sekvence snímků ze stacionárních kamer nepotýkal.

4.4 Klasifikace nežádoucích jevů v časovém snímku

Z vybraných dohledových kamer jsem pořídil sekvence snímků při různých rozptylových podmínkách a v různou denní dobu za účelem provedení analýzy primitivního časového snímku při různém zrychlení, přičemž hlavně faktor zrychlení měl zásadní vliv na povahu výsledného videa a jeho subjektivní plynulost. Podpůrným materiálem byla i volně dostupná videa na internetu. Ukázalo se, že i když se v různých scénách vyskytují naprosto odlišné objekty, lze pozorovat jisté podobnosti chování ve zrychleném videu. Jako příklad lze uvést kouř vycházející z komínu a prapor vlající ve větru. Po této analýze jsem často se opakující jevy, které narušují estetický dojem z časového snímku, rozdělil do následujících kategorií:

- periodické pohyby a změny (např. blikající semafor)
- náhodné pohyby a změny stálých objektů (např. strom ve větru)
- náhodný výskyt objektů a jejich proměny (např. procházející člověk)
- globální změny scény (osvětlení, počasí)

Popis jednotlivých kategorií v následujícím textu je vždy doplněn o ilustrativní příklady reprezentující sekvenci po sobě následujících snímků ve videu.

4.4.1 Periodické pohyby a změny

Periodické pohyby se v reálné scéně nevyskytují příliš často. Do této kategorie řadím všechny změny v obraze, které lze eliminovat výběrem vhodného snímku bez nutnosti dalšího zpra-

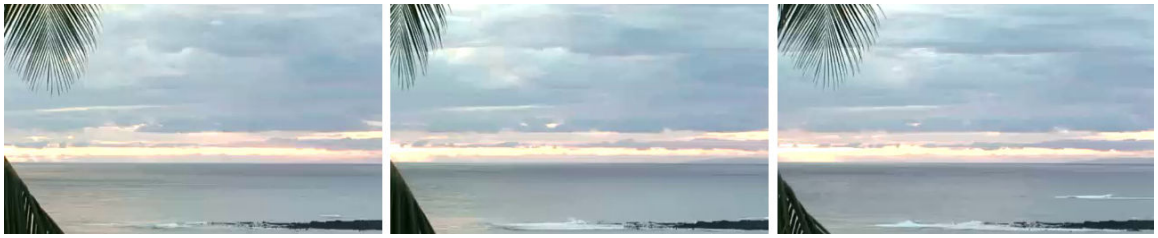
cování. Příkladem periodického pohybu mohou být hodinové ručičky, kolotoč či kyvadlo. V sekci 5.2 se podrobněji zabývám řešením scény s kyvadlem.

Do této kategorie řadím i změny, které nejsou přímo pohybem, ale vedou na periodickou změnu jasu pixelů. Příkladem je semafor blikající s konstantní periodou. Tyto jevy se vyznačují tím, že je lze vhodným převzorkováním časového snímku zpětně rekonstruovat.

Obrázek 4.1: Ukázka periodických změn v časovém snímku - větrná turbína v popředí.

4.4.2 Náhodné pohyby a změny stálých objektů

Druhá kategorie, kterou jsem označil jako náhodné pohyby a změny statických objektů, je jednou z největších výzev, protože pokud neexistuje žádný ustálený stav (např. bezvětří), je potřeba kromě volby vhodných snímků začít manipulovat i s obrazovými daty. Častým jmenovatelem těchto jevů je vítr a ovlivněnými objekty jsou často rostliny, stromy a vodní hladina.



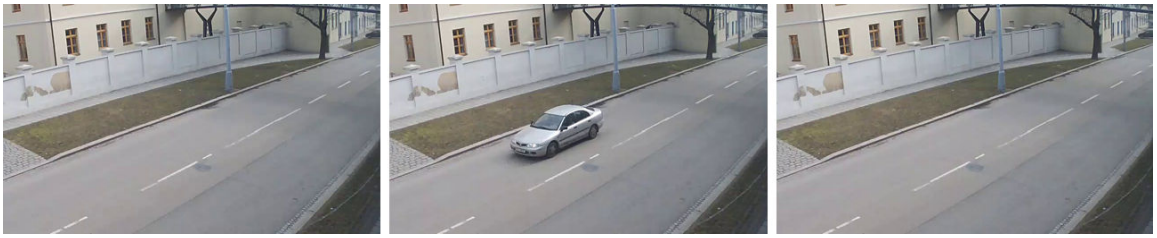
Obrázek 4.2: Ukázka náhodných změn ve scéně - v horním rohu je palma pohybující se vlivem větru.

4.4.3 Náhodný výskyt objektů a jejich proměny

Náhodným výskytem objektů jsou myšleny scény, kde se v nepravidelných intervalech objevují lidé či dopravní prostředky. Jiným příkladem může být přelet ptáků po obloze. V primitivním časovém snímku se toto projeví jako krátké probliknutí. Při nízké frekvenci výskytu náhodných objektů lze postižené snímky odfiltrovat. Komplikovanější jsou scény, kde k pohybu dochází neustále, např. křižovatka nebo náměstí. Zde se nabízí možnost kombinovat části snímků stávající pouze z pozadí scény.

4.4.4 Globální změny

Velkým problémem je poslední kategorie, kterou jsem nazval globální změny. Dynamikou osvětlení a počasí může při pomalém vzorkování snímků docházet ke skokovým změnám,



Obrázek 4.3: Ukázka náhodných výskytů objektů - projíždějící auto, které se při dlouhé vzorkovací periodě objevilo jen v jednom snímku.

které je obtížné eliminovat, nejedná-li se pouze o ojedinělé extrémy, které by bylo možné odstranit vhodným výběrem snímku. Pozorované globální změny lze dále rozdělit na krátkodobé a dlouhodobé. Ukázkou krátkodobé globální změny scény je přechod oblačnosti, kdy místy prosvítá slunce a změna stínů a osvětlení zcela mění charakter prostředí. Při časovém snímku v řádech dní až měsíců vstupuje do hry také střídání dne a noci, v širším měřítku např. střídání ročních období.

Obrázek 4.4: Ukázka globální změny - proměnné osvětlení ve scéně způsobuje přeblikávání celého videa.

Jak se potvrdilo ze získané kolekce dat, problémem zcela se odlišujícím od zpracování časového snímku pořízeného v průběhu jednoho dne je dlouhodobý časový snímek v řádech týdnů až měsíců. I v případě, že se podaří vybrat snímek dokonale reprezentující konkrétní den, největším úskalím zůstávají změny počasí, které jsou při přechodu mezi jednotlivými dny často skokové. Další nežádoucí efekty působí změny ve scéně s dlouhou setrvačností, např. několik hodin zaparkované auto.

Obrázek 4.5: Signifikantní globální změny lze pozorovat v dlouhodobém časovém snímku. Častým problémem těchto videí jsou výrazné výkyvy počasí. Snímky na obrázku jsou pořízené ze stejného místa tři po sobě následující dny.

Na obrázku 4.5 lze vidět, jak výrazné rozdíly se objevily mezi jednotlivými dny v datech

určených pro dlouhodobý časový snímek. V tomto konkrétním případě jsou všechny ostatní uložené snímky z druhého dne redundantní, neboť jsou téměř totožné a žádnou jejich kombinací nelze zmírnit ostrý přechod ve výsledné sekvenci. Nápadem, jak si poradit se změnami počasí, je zmapovat nejtypičtější povětrnostní podmínky pro danou scénu, ponechat pouze tyto snímky a následně vyřazené snímky interpolovat z okolních.

Kapitola 5

Navrhované postupy pro zpracování snímků

Vlastní zadání práce je experimentálního charakteru, proto pro vytvoření počáteční vize, jakým způsobem data zpracovávat, následovala série pokusů s metodami používanými při zpracování obrazu, od jednodušších po složitější. V následujícím textu jsou popsány konkrétní algoritmy, z nichž některé vychází z obecné teorie popsané v kapitole 3.

5.1 Lokální filtrace

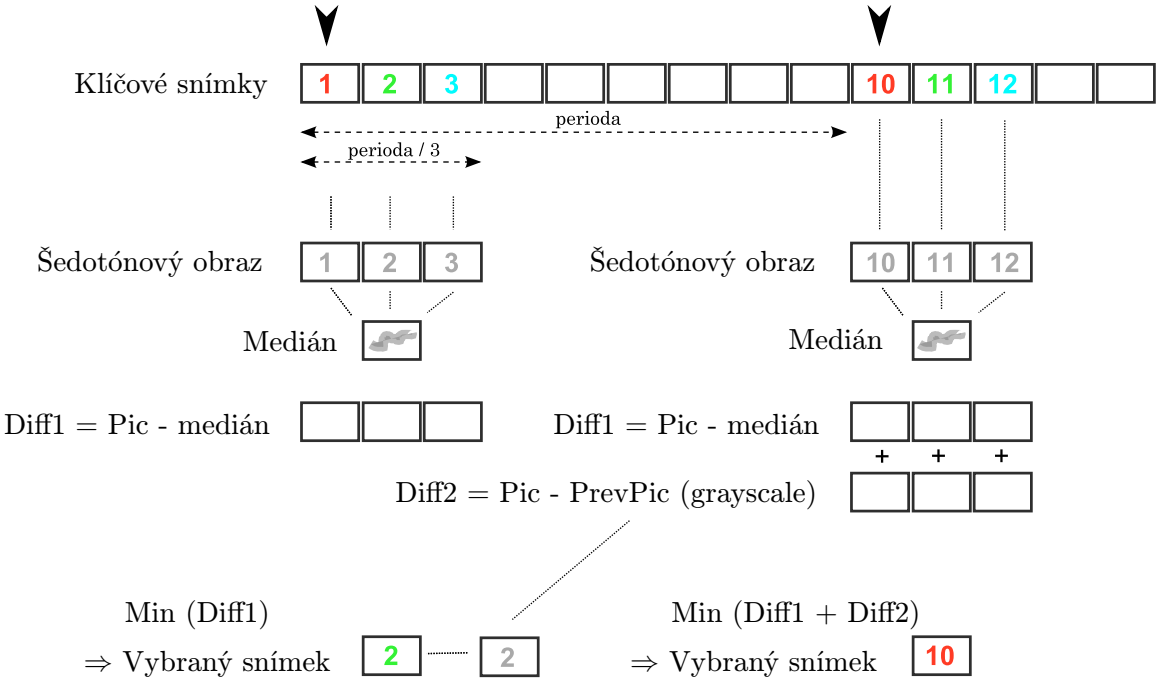
Lokální filtrací videa je v tomto kontextu myšleno zpracování videa „per-pixel“, kde je o každém jednotlivém pixelu $p_{x,y}$ uvažováno jako o jednorozměrném signálu v čase t. Jednalo se o první intuitivní řešení zadaného problému, kdy jsem ze sekvence snímků $0..n$ vytvořil mediánový snímek s použitím mediánu na každé sekvenci pixelů $p(x, y, 0)..p(x, y, n)$. Později jsem se k této myšlence znovu vrátil a rozvinul ji přidáním dalších filtrů - průměr, max-filtr, min-filtr a jejich kombinace. Tyto filtry byly dále zkoušeny ve dvou variantách:

- tříkanálová filtrace - každá ze tří barevných složek RGB se počítá nezávisle
- jednokanálová filtrace - snímky jsou nejdříve převedeny na stupně šedi, výsledný pixel se zpětně dohledá z původního barevného snímku

5.1.1 Medián

První návrh aplikace používal mediánový snímek k výběru nejvíce reprezentativního vzorku ze sekvence snímků v okolí periody časového snímku. Schematicky je algoritmus naznačen na obrázku 5.1. Ze skupiny sousedních snímků převedených do odstínů šedi je vytvořen mediánový snímek. V dalším kroku se počítá rozdíl všech snímků od tohoto mediánového. Vybrán je ten, který má nejmenší součet rozdílů od naposledy zvoleného a od mediánu.

Druhá modifikovaná verze ukládá přímo mediánový snímek převedený zpětně do barev. Podstatným rozdílem tedy je, že výsledkem není reálný obraz, ale kombinace snímků. Zatímco předchozí verze, která vybírala nejvhodnější snímek, nedokázala eliminovat například pohybující se objekt s výskytem v celé sekvenci, zde se přirozeně vyfiltruje za předpokladu, že v místě jeho aktuálního výskytu je na většině zbylých snímků statické pozadí. Základním nedostatkem tohoto řešení je vznik artefaktů a rozostření obrazu.



Obrázek 5.1: Algoritmus pro výběr snímku na základě mediánu

5.1.2 Průměr

Průměrování snímku je alternativou k výpočtu mediánu. Výpočetně je to jednodušší varianta, protože není potřeba seřazovat pole čísel jako v případě mediánu, a proto je také průměrování snímku rychlejší. Zanechává však více artefaktů při filtrování náhodně se vyskytujících objektů, protože jsou do výsledku promítnuty pixely z celé sekvence snímků. To také vyplynulo z provedených pokusů při porovnávání s mediánem. S malými změnami ve scéně a rostoucím počtem snímků se ale výsledky blíží natolik, že jsou vizuálně nerozeznatelné. Průměrování snímků se osvědčilo například pro odstranění šumu z nočních scén.

5.1.3 Max-filtr a min-filtr

Max-filtr vybírá vždy nejvyšší hodnotu z celé sekvence, podobně min-filtr vybírá vždy nejnižší hodnotu. Jako efektní použití max-filtru se ukázaly noční scény s projíždějícími auty, jejichž světla při filtrování sekvence snímků zanechávají zářivou stopu. Protože je v těchto scénách typicky velký šum generovaný čipem kamery, případně posílený nízkou přenosovou rychlostí, vyzkoušel jsem dále kombinaci max-filtru s průměrem snímků. K tomu vytvářím pomocný snímek, který je rozdílem průměru a max-filtru a nastavení prahu z tohoto pomocného snímku vznikne maska, která slouží pro výběr pixelu ze snímku průměru resp. max-filtru. Mimo jiné jsem také zkoušel kombinaci min-filtru a max-filtru pro vytvoření konstrastního videa, ale výsledek nepůsobí přirozeně.

5.2 Případová studie - kyvadlo

Kyvadlo bylo vybráno jako ukázkový příklad periodického jevu. Mediánový filtr tento typ scény zpracuje tak, že původní kyvadlo zcela odstraní a vyobrazí pouze stálé pozadí. Zbavíme se tedy aliasingu, ale zároveň se vytratí i původní povaha scény. Záměrem proto bylo algoritmicky vybrat takovou sekvenci snímků, aby výsledný pohyb kyvadla odpovídal jeho pohybu v reálném čase, zatímco pozadí scény bude zrychleno.

Použil jsem metodu Farneback, jejíž implementace je zabudovaná v knihovně OpenCV. Jedná se o metodu hustého optického toku, která je pro tento účel vhodnější než řídký optický tok, kde se sledují pouze vybrané body v obraze. Výsledné vektorové pole reprezentující optický tok jsem použil při výpočtu rozdílu snímků jako čtvrtou složku:

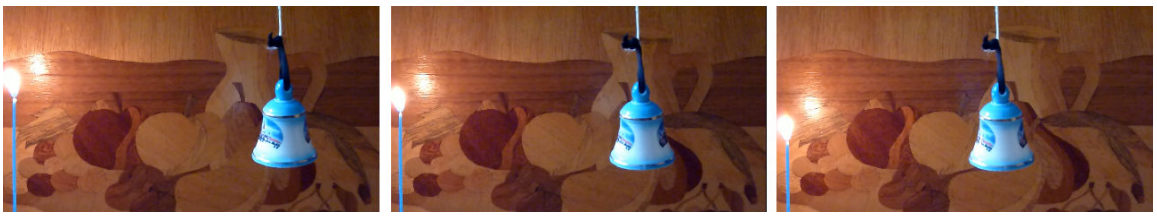
$$f(X, M) = \underset{X}{\operatorname{argmin}} \left(|R_X - R_M| + |G_X - G_M| + |B_X - B_M| + |F_X - F_M| \right) \tag{5.1}$$

kde X je porovnávaný snímek, M je mediánový snímek; R, G, B jsou matice s intenzitami jednotlivých barevných složek a F je vektorové pole optického toku.

Toto porovnávání snímků jsem úspěšně použil pro řešení problému periodického pohybu, avšak pro obecné použití se ukázal jako nevhodný, neboť nepřináší výrazné zlepšení proti mediánovému snímku bez výpočtu optického toku a navíc je tento výpočet časově náročný. Poznamenejme, že tento výpočet je nutné provést ještě před filtrací klíčových snímků na původní sekvenci v reálném čase, protože při řídkém vzorkování nelze pohyb ve scéně detekovat.

První pokus byl proveden s modelem kyvadla vytvořeným v softwaru Blender a následně přidaným do hodinového videa zachycující východ Slunce. V případě tohoto matematického modelu kyvadla lze kýženého výsledku dosáhnout výběrem snímků s periodou $x = pn + 1$, kde p je perioda kyvadla a n je kladné celé číslo. Takto jsem vytvořil referenční řešení pro porovnání výsledků pro variantu s výpočtem diference barevných složek a optických toků.

Přestože výsledky s vymodelovaným kyvadlem dodatečně přidaným do videa byly uspokojivé, bylo potřeba tento pokus ověřit také na reálné scéně. Zde jako kyvadlo posloužil zavěšený zvonek a jako orientační časomíra zapálená svíčka. Na výsledném časosběrném videu je tedy plynulý pohyb zvonku a zrychlené hoření svíčky, viz obrázek 5.2.



Obrázek 5.2: Kyvadlo v reálné scéně. Obrázek ukazuje rekonstrukci periodického jevu nalezením vhodného vzorkování snímků, zatímco hoření svíčky demonstruje zrychlené pozadí scény. (Video na přiloženém CD: `video/experiments/pendulum.avi`)

Ukázalo se, že periodické jevy ve scéně lze zpracovávat poměrně jednoduše s využitím již zmíněné metody diference hodnot pixelů a optického toku. V tomto případě bez informace o optickém toku metoda selhává, jak se potvrdilo při pokusu s pouhým odečítáním obrazových dat. Do stejného místa se totiž kyvadlo dostává při cestě tam i zpět a při změně tohoto

kontextu se mění i směr pohybu ve výsledné sekvenci, což při častém střídání vede na stochastické kmitání na dráze kyvadla.

5.3 Použití Markovovských náhodných polí

I když se ukázalo, že s výše uvedenými základními filtry lze dosáhnout výrazného vylepšení časového snímku, mají tyto metody svá úskalí vyplývající z faktu, že se jedná pouze o lokální filtraci bez kontextu okolních pixelů v prostoru. To se projevuje častými artefakty u scén s velmi frekventovaným výskytem náhodných objektů a dále rozmázáním pohybujících se stálých objektů. A právě proto se zde nabízí použití Markovovských náhodných polí, kde je o pixelech smýšleno jako o statistických datech vzájemně spolu souvisejícími.

Vzhledem k paměťové náročnosti (detaily v kapitole 7) je filtrace aplikována až na výsledný časový snímek, nikoli na celý vstup v reálném čase. Předzpracováním videa základními filtry (např. mediánem) by se ztratila výhoda zachování větší ostrosti pohybujících se objektů, proto je vhodnější pracovat s naivním časovým snímkem jakožto vstupem a nebo použít jako předstupeň zpracování některou z metod pro výběr vhodného snímku, například mnou navrhovaný algoritmus pro výběr snímku na základě mediánu popsaný v kapitole 5.1.

5.4 Neuniformní vzorkování

V rámci experimentování s různými metodami jsem implementoval algoritmus pro neuniformní vzorkování (viz [4]) popsaný v kapitole 2.2. I s optimalizací výpočtů dynamickým programováním je časová náročnost v řádech desítek minut i pro malá testovací data o několika desítkách snímků. Navrhnul jsem tedy vlastní jednodušší algoritmus, který zachovává myšlenku proměnné periody, ale přepíná pouze mezi dvěma zvolenými hodnotami. Existuje tedy jedna výchozí perioda pro vzorkování částí vstupního videa bez výrazných změn a kratší perioda pro hustější vzorkování živějších částí. Příkladem vhodného použití může být záběr na silnici, kde jen občas projede auto.

Velikost změny vypočítávám diferencí snímků jako průměrný absolutní rozdíl dvou sousedních snímků ve všech barevných složkách. Pomocí zvoleného prahu a výsledné hodnoty je snímek zařazen do skupiny malých, resp. velkých změn. Další nastavitelnou hodnotou je počet po sobě následujících snímků s velkými změnami potřebný k tomu, aby byla snížena vzorkovací perioda. Jinými slovy je to nejmenší délka trvání změny ve scéně, kvůli které chceme zpomalit video.

Celý proces převzorkování se pak dělí do čtyř fází:

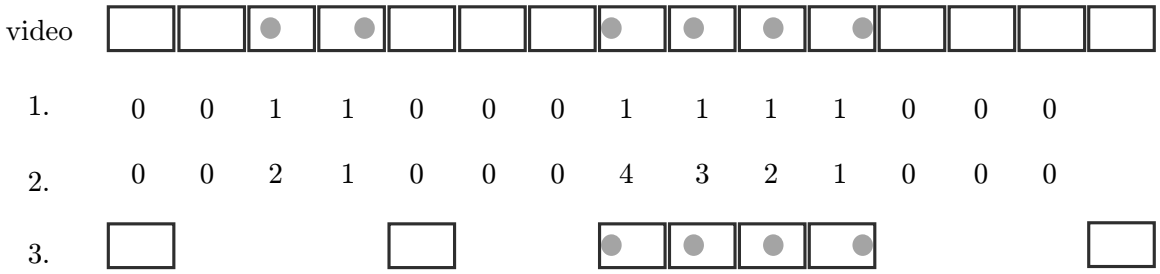
1. Snímky jsou označeny nulou, resp. jedničkou v závislosti na tom, zda obsahují změny menší resp. větší než zvolený práh

2. Vytvořená metadata jsou procházena zprava a přepisují se počtem po sobě následujících jedniček.

3. Video je vzorkováno s výchozí periodou, dokud se v následující periodě neobjeví snímek s hodnocením vyšším než zvolená hodnota nebo se nedojde na konec původního videa.

4. Video je vzorkováno s nižší periodou, dokud je hodnocení snímku vyšší než 0. Návrat na bod 3.

Obrázek 5.3 ukazuje použití tohoto algoritmu na ilustrativním příkladě. Výhodou mého řešení je, že obrazová data jsou zpracována jen jedním průchodem, protože rozdíl snímků je počítán vždy jen mezi dvěma sousedními snímky. Ve srovnání s výše zmíněným algoritmem pro výpočet optimálního vzorkování, který má kvadratickou složitost, má tedy tato zjednodušená verze s lineární složitostí značnou výhodu.



Obrázek 5.3: Ukázka použití navrhovaného algoritmu pro neuniformní vzorkování. Výchozí vzorkovací perioda je zde 4, pomalejší 1 (shodná s originálem), práh pro přepnutí periody jsou čtyři snímky. Nahoře je původní video, následuje první fáze detekce změn, přepočítání na počet po sobě následujících snímků a dále vzorkování videa s přepínáním periody.

5.5 Neosvědčené přístupy

Při řešení diplomové práce jsem experimentoval s různými metodami, z nichž některé se nakonec ukázaly jako nevhodné pro danou problematiku. Několik nápadů, které byly slepou uličkou, bych rád zmínil v následujícím textu.

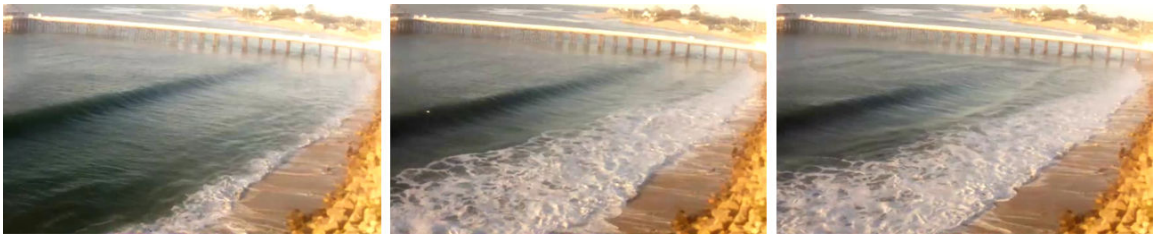
5.5.1 Zobecnění periodických jevů

Po úspěšném vyřešení případové studie s kyvadlem jsem se pokusil myšlenku rekonstrukce periodických jevů rozvinout a experimentoval jsem s dalšími typy scén. Správným výběrem vzorkovací periody se například podařilo dosáhnout dobrých výsledků u nočních záběrů z křižovatky, kde jsem se zaměřil na blikající oranžové světlo semaforu. Avšak v případě komplexních jevů, které se jeví jako periodické, tento postup selhává.

Příkladem neúčinného použití metody pro hledání periodicity je záběr na pobřeží a mořské vlny omývající pláž. Ačkoli tyto přílivy vln nastávají v pravidelných intervalech, kontury nabývají rozličných tvarů a tudíž tento jev spadá do kategorie náhodných pohybů. Pokus o převzorkování a rekonstrukci pohybu vln nedopadl dobře ani při manuálním zpracování, kde jsem hledal největší překryv dvou snímků. Výsledek působí stále velice nepřirozeně a neliší se velmi od naivního časového snímku. Ukázka je na obrázku 5.4.

5.5.2 Porovnávání histogramů

Výpočetní náročnost výše navrhovaných metod byla motivací k hledání jednodušších postupů, které mohou být méně přesné při hledání optima, avšak pracují v kratším čase. Histogram reprezentuje distribuci dat, jednotlivé hodnoty odpovídají četnostem dat v daném intervalu. Pro barevný model RGB pak získáme tři histogramy pro červenou, zelenou a modrou složku. Hlavním rozdílem v kontextu předchozích metod je použití metadat místo



Obrázek 5.4: Neúspěšný pokus o převzorkování pohybu mořských vln u pobřeží. Zatímco zadní blížící se vlnu se povedlo zrekonstruovat, přední část u pobřeží se mění zcela náhodně. (Video na přiloženém CD: `video/experiments/ocean_resampled.avi`)

práce s vlastními obrazovými daty. Navrhované použití bylo nahradit metodu výběru ideálního snímku na základě mediánu zjednodušeným výběrem na základě metadat.

Seřazením hodnot pro všechny intervaly histogramu a vybráním prostřední hodnoty je vytvořen mediánový histogram a princip hledání nejreprezentativnějšího snímku dále zůstává stejný jako v případě mediánu nad obrazovými daty - vybírá se snímek, který má histogram nejpodobnější tomu s mediánovými hodnotami. Knihovna OpenCV poskytuje funkci pro porovnání histogramů s různými metrikami. V tomto případě jsem použil vzájemnou korelaci:

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}} \tag{5.2}$$

kde $\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$ a N je celkový počet intervalů v histogramu. Výsledná hodnota je v intervalu $< 0; 1 >$, přičemž 1 znamená totožnost obou histogramů. Po sečtění výsledku ze všech tří barevných složek je tedy hodnota v rozmezí $0 - 3$.

Ukázalo se, že v reálném prostředí jsou rozdíly těchto hodnot úplně minimální a pohybují se běžně v rozmezí $2.95 - 3$. Detekce odlišných snímků tak funguje pouze pro výrazné změny jako je globální osvětlení nebo výskyt velkých objektů o relativní velikosti alespoň v řádu desetin procent celkové velikosti obrazu. Porovnávání histogramů tedy neuspělo jako účinná metoda pro výběr optimálního snímku. Je možné tímto způsobem detekovat výrazně se odlišující snímky v sekvenci a ty pak následně odfiltrovat před dalším zpracováním, avšak s extrémními hodnotami pixelů si dobře umí poradit i mediánový filtr bez nutnosti předzpracování.

5.5.3 Použití diference snímků pro lokální filtraci maskou

V tomto experimentu byl rozdíl snímků počítán z posledního vybraného rámce na indexu i a následujícího rámce na indexu $i + p$, kde p je perioda vzorkování. Nastavením prahu se z rozdílového snímku vytvoří maska pro filtraci. Zamaskované pixely znamenají přiřazení hodnoty z původního rámce i a nezamaskovaným pixelům je přiřazena hodnota z rámce $i + p$. Tím je dosaženo zachování pouze malých změn hodnot jednotlivých pixelů, zatímco velké skokové změny jsou potlačeny. Dále jsem tuto metodu vylepšoval přidáním morfologických operací. Konkrétně bylo použito morfologické uzavření, které přetvoří masku na více konzistentní. Pro zohlednění pohybu (tedy přeskupování pixelů, nejen lokálních změn jasu pixelů) jsem počítání rozdílu rozšířil na výpočet při různém vzájemném posunutí dvou obrazů v definovaném okolí.

Přes zmíněná vylepšení této metody se nepodařilo dosáhnout uspokojivých výsledků. Video obsahovalo spoustu artefaktů, zejména velkých ploch, které prošly nastaveným prahem. Například zůstaly viditelné stopy projíždějících světlých aut na pozadí šedého asfaltu.

Kapitola 6

Návrh a implementace výsledné aplikace

Po fázi experimentování přichází na řadu návrh a implementace systému, který bude vytvářet ze vstupního videa pokročilý časový snímek a uživateli bude umožněno měnit jeho nastavení. V následujícím textu je souhrn použitých technologií, návrh aplikace a jejího ovládání a také pohled do samotné implementace.

6.1 Použité technologie

Všechny části této diplomové práce jsou implementovány v jazyce C++ s použitím knihoven OpenCV (verze 2.4.10), FFmpeg (verze 56.13.100) a Boost (verze 1.57). Tyto prostředky v následujícím textu stručně popisuji. Testování probíhalo na školním linuxovém serveru *merlin*, kde byly zdrojové kódy přeloženy překladačem *gcc* verze 4.7.4.

6.1.1 OpenCV

OpenCV je knihovna pro počítačové vidění a strojové učení. Vydáním pod BSD licencí se stává volně použitelná pro akademické i komerční účely. Knihovna má rozhraní pro použití v programovacích jazycích C, C++, Python, Java a MATLAB a je také multiplatformní, takže je možné ji přeložit na operačních systémech Windows, Linux, Android i Mac OS.

OpenCV jsem si zvolil pro jeho komplexnost a množství již implementovaných funkcí, které nabízí pro manipulaci s obrazovými daty a které bych jinak musel naprogramovat svépomocí. Proti MATLABu má výhodu ve vyšší rychlosti a menších nárocích na systémové zdroje. Kromě oficiální dokumentace je k dispozici i literatura provázející základními koncepty formou návodů, viz [7].

6.1.2 FFmpeg

FFmpeg je multimediální framework a obsahuje kolekci programů a knihoven. Vydáním pod licencí „GNU Lesser General Public License“ se stává volně použitelným. Právě tyto obsažené knihovny pro práci s videem jsem použil ve svém programu jako první vrstvu, která dekóduje video a vybírá klíčové snímky k dalšímu zpracování. Konkrétně byly použity knihovny *libavcodec* (kódování/dekódování), *libavformat* (muxer/demuxer pro rozlišení obsahu v multimediálním kontejneru) a *libswscale* pro práci s obrazem. K zápisu videa nyní

používám třídu *VideoWriter* z OpenCV, do budoucna přichází v úvahu toto nahradit voláním funkcí z FFmpeg stejně jako při načítání vstupu. Výhodou by byla větší kontrola nad parametry výstupního videa.

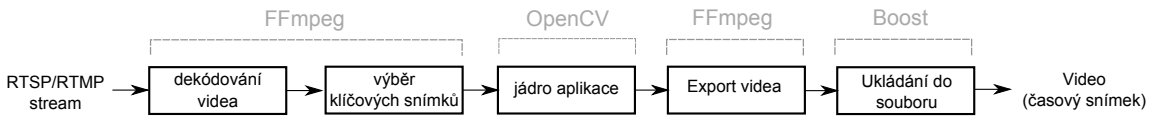
6.1.3 Boost

Boost je sada knihoven určená pro použití v programovacím jazyce C++. Obsahuje implementaci datových struktur, funkce pro práci s řetězci, správu paměti, souborový vstup/výstup a mnoho dalšího.

V mé aplikaci používám knihovnu *Boost.Filesystem* při zápisu do souboru, kdy je potřeba vytvořit zadanou cestu, pokud dosud neexistuje. Výhodou použití této knihovny je abstrakce nad operačním systémem. Další částí z kolekce Boost, kterou používám pro vytvoření asynchronního časovače, je knihovna *Boost.Asio*.

6.1.4 Programovací jazyk C++

Jazyk C++ podporuje více programovacích paradigmat - procedurální, generické a objektové programování, z nichž právě objektově orientovaný styl koresponduje s mým návrhem aplikace. Také výše zmíněná kniha o openCV [7] vede čtenáře k užívání objektového návrhu a ukazuje některá základní obecně použitelná schémata. Výhodou tohoto jazyka je jeho silná vyjadřovací síla na všech úrovních abstrakce, od práce s pamětí až po konstrukce na vyšších úrovních. Mnou použité knihovny navíc mají rozhraní pro C++ bez nutnosti používání tzv. wrapperů jako mezikódu (např. kombinace FFmpeg a jazyku Java).



Obrázek 6.1: Způsob využití externích knihoven v navrhované aplikaci.

6.2 Architektura

Cílový produkt je navrhován jako konzolová aplikace, která demonstruje jednotlivé možnosti zpracování videa jako časového snímku. Při návrhu jsem zohlednil odlišnou úroveň flexibility při práci s videozáznamem a s živým vysílání z internetové kamery. Některé z výše představených algoritmů vyžadují opakovaný průchod celou videosekvencí, což v případě načítání dat v reálném čase není možné. Avšak i pro algoritmy operující s časovým oknem kolem aktuálního snímku je klíčová jejich výpočetní náročnost, aby bylo možné na cílovém stroji vytvářet výstupní data stejnou rychlostí jako přijímat vstupní. V opačném případě by vstupní data byla shromažďována v paměti, jejíž kapacita by se při delším běhu programu postupně vyčerpala. Z toho tedy vychází koncept rozdělení na dvě aplikace, které jsem nazval *timelapse-online* (zpracování videa v reálném čase) a *timelapse-offline* (zpracování nahraného video záznamu). Obě aplikace obsahují nastavitelné parametry, které jsou

načítány ze stejnojmenných souborů s koncovou „.ini“ - *timelapse-online.ini* a *timelapse-offline.ini*. Kromě těchto dvou hlavních aplikací jsem dále vytvořil pomocný nástroj *collect*, který ze zadaného streamu ukládá snímky pro pozdější zpracování.

6.2.1 timelapse-online

Verze pro práci s živým vysíláním kromě tvorby klasického časového snímku umožňuje zpracování základními filtry, které jsou aplikovány na kolekci snímků v okolí periody. Nepovinně lze také nastavit velikost posuvného okna pro druhou fázi filtrace snímků vytvořených v první fázi. Příkladem použití je dlouhodobý časový snímek, kde se osvědčil mediánový filtr pro snímky z jednoho dne a dále průměrování denních snímků posuvným oknem.

Vstupem je tedy adresa streamu IP kamery, výstupem video soubor se zvolenou lokací v souborovém systému. Dalšími nastavitelnými parametry je kodek pro výsledné video, vzorkovací perioda, délka trvání sběru snímků v rámci jedné periody a počet těchto snímků. Dále celkový čas běhu programu (jak dlouhý úsek chceme z kamery nasnímat), typ použitého filtru a délka posuvného okna pro druhou fázi filtrace. Ta se provede vždy při délce větší než jedna. Typ filtru lze zvolit v rozmezí hodnot 0 - 8 s následujícím významem:

- 0 (no filter) - vytvoří klasický časový snímek bez zpracování okolních snímků
- 1 (median) - medián počítaný ve všech barevných složkách RGB
- 2 (grayscale median) - medián počítaný ze sekvence převedené do odstínů šedi
- 3 (max-filter) - max-filtr ve všech barevných složkách RGB
- 4 (grayscale max-filter) - max-filtr počítaný ze sekvence převedené do odstínů šedi
- 5 (min-filter) - min-filtr ve všech barevných složkách RGB
- 6 (grayscale min-filter) - min-filtr počítaný ze sekvence převedené do odstínů šedi
- 7 (average filter) - průměrování snímků
- 8 (average + max filter) - kombinace průměru a max-filtru se zvoleným prahem

Práh pro poslední typ filtru je hodnota v rozmezí 0-255 nastavitelná v konfiguračním souboru a určuje, o kolik se musí lišit výstup z max-filtru od výstupu z průměrovacího filtru, aby byla do výstupního snímku uložená hodnota z max-filtru, jinak se vezme hodnota z výsledku průměrování.

6.2.2 timelapse-offline

Aplikace s názvem *timelapse-offline* je určena pro práci s uloženým videozáznamem a ve své podstatě je rozšířením předchozí verze o algoritmy, které není možné aplikovat na obrazová data v živém přenosu. Rozšířené možnosti zpracování však začínají už u základních filtrů. Můžeme například nastavit velikost filtrované kolekce snímků v rámci jedné periody větší než je perioda samotná. Důvodem, proč byla tato aplikace vyčleněna samostatně, je jiná architektura hlavního programu, protože není potřeba žádná synchronizace časovačem pro sběr snímků z živého vysílání. Všechna data jsou k dispozici již v době spuštění programu a jejich zpracování tedy může proběhnout zcela asynchronně.

Kromě základních filtrů jsou zde implementovány také oba představené algoritmy pro neuniformní vzorkování a optimalizace na 3D Markovovských náhodných polích (MRF). Algoritmus MRF je specifický tím, že očekává jako vstup naivní časový snímek, nikoli video v reálném čase. Kvůli vysoké paměťové náročnosti jej lze aplikovat pouze na velmi malá vstupní data. Uživatel je na požadovanou velikost paměti upozorněn po načtení vstupního videa.

Program lze spustit s jedním z těchto parametrů:

- *--basic* Použití základních filtrů
- *--nonuni* Neuniformní vzorkování - nalezení optima
- *--nonuni-simple* Zjednodušené neuniformní vzorkování - dvoustavové
- *--mrf* Markovovská náhodná pole

Nastavení lze opět měnit v konfiguračním souboru, kde k parametrům pro lokální filtry přibyly další proměnné pro ostatní algoritmy. Pro dvoustavové neuniformní vzorkování je to základní perioda a druhá kratší perioda použitá k zachycení výrazných změn. Dále lze měnit dvě proměnné reprezentující prahové hodnoty minimální velikosti a délky změny, pro kterou se použije menší vzorkovací perioda. U Markovovských náhodných polí lze měnit parametry α a γ, viz vzorec 3.6.

6.2.3 Běh aplikace na serveru

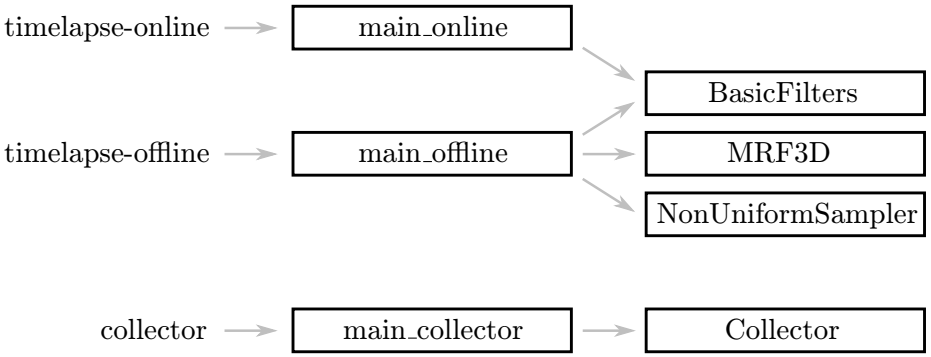
Jedním z diskutovaných témat v rámci mého semestrálního projektu (viz [12]) byla možnost migrace aplikace na server. Tato potřeba vyplývá z nutnosti mít aplikaci spuštěnou dlouhodobě pro sběr dat z internetové kamery, navíc uživatel musí mít stabilní a rychlé internetové připojení a v neposlední řadě dostatek systémových zdrojů na svém lokálním stroji, kde aplikaci spouští.

V rámci diplomové práce byly algoritmy pro lokální filtraci zasazeny do reálného prostředí knihoven pro počítačové vidění na jeden ze serverů společnosti Angelcam, viz [3]. Jedná se o upravenou verzi navrhované aplikace *timelapse-online*, ze které bylo extrahováno její funkční jádro pro zpracování obrazu. Aplikace běží v prostředí *docker*, které zapouzdřuje aplikaci do vlastního kontejneru. Za tímto účelem byl vytvořen repozitář *lukasturek/timelapse* na dockerhubu, odkud je možné si zdrojový kód stáhnout.

Příprava konfiguračních souborů je zautomatizovaná skriptem v jazyce *bash*, který pro každou URL adresu v souboru *streams.txt* vygeneruje soubor s příponou „.ini“. Ten se v zásadě podobá konfiguračnímu souboru pro lokální verzi aplikace. Dále se pro každou adresu streamu vygeneruje nový „engine“ v souboru *timelapse.yml*. Tímto způsobem lze spustit paralelně více instancí aplikace a vytvářet tak časový snímek z několika kamer současně. Výsledné snímky se ukládají do zvoleného adresáře jako obrázky, pro jejichž spojení je k dispozici skript na bázi volání aplikace FFmpeg.

6.3 Implementace

Struktura programu byla navržena tak, aby jednotlivé moduly byly použitelné i samostatně v jiných aplikacích. Uplatnil jsem principy objektově orientovaného programování, zejména zapouzdření. Tam, kde to z principu dává smysl, byl použit návrhový vzor singleton (např. pro modul *Collector* ukládající data z kamery). Hlavní programy jednotlivých aplikací začínají vždy předponou *main_* a volají funkce z jednotlivých modulů.



Obrázek 6.2: Struktura aplikace na úrovni jednotlivých modulů a jejich provázání.

6.3.1 main_online.cpp

Hlavní program aplikace *timelapse-online* načte parametry z konfiguračního souboru a vypíše aktuální nastavení na standardní výstup. Dále se pokusí otevřít vstupní datový proud videa a vytvořit výstupní soubor. Spouští se asynchronní časovač s opakovaným voláním funkce *getFrame()*, kde probíhá celý proces vytvoření výstupního snímku. Posuvné časové okno je implementováno jako kruhový buffer, kde se aktuální index pro uložení snímku počítá operací modulo: $i \mod n$, kde i je číslo aktuálního snímku a n je velikost posuvného okna.

Problémem, který se při implementaci objevil, je interní buffer knihovny OpenCV, do nějž jsou řazeny snímky při čtení z objektu *cv::VideoCapture*. Při opakovaném čtení jednoho snímku jednou za několik sekund totiž funkce postupně vrací snímky uložené v bufferu z prvního volání. Řešením tedy bylo v cyklu načíst vždy více snímků najednou, a to i v případě tvorby naivního časového snímku, kdy je potřeba pouze jeden obrázek.

6.3.2 main_offline.cpp

Aplikace *timelapse-offline* taktéž začíná načtením parametrů z konfiguračního souboru a dále podle režimu spuštění (parametr z příkazové řádky) přechází do jedné ze čtyř větví pro příslušný algoritmus. Volitelně je na standardní výstup vypisován poměr již zpracovaného vstupu v procentech.

6.3.3 main_collector.cpp

Nástroj pro sběr snímků má svůj vlastní konfigurační soubor, který je však o poznání jednodušší a obsahuje pouze základní parametry: vstup, výstup, perioda, celkový počet snímků a boolovská proměnná určující zda budou sbíraný pouze klíčové snímky nebo všechny. Po načtení parametrů z konfiguračního souboru je volána funkce *collect()* z modulu *Collector*, která zajistí sběr dat.

6.3.4 BasicFilters.cpp

Modul základních filtrů v sobě obsahuje implementaci funkcí pro lokální filtraci sekvence snímků. Návratovou hodnotou všech těchto funkcí je matice (objekt knihovny OpenCV *cv::Mat*) s výsledným obrázkem. Protože implementace mediánu, max-filtru a min-filtru se

liší pouze vybraným prvkem ze seřazeného pole, byla vytvořena společná funkce *sortingFilter()* pro omezení duplicity kódu. Zde jsou pro každou souřadnici (x,y) seřazeny pixely v časové ose podle intenzity ve všech třech barevných složkách a filtrace je provedena odděleně v každé z nich. Pro filtry s koncovkou „Grayscale“ je to funkce *sortingFilterGrayscale()*, která nejdříve převede všechny snímky na jednokanálové (stupeň šedi), následně seřazuje pixely v tomto jednom kanále a zpetně podle indexu výsledku najde korespondující RGB pixel, který zapíše na výstup. Pro uchování dvojice $(hodnota, index)$ je použita datová struktura *std::pair*.

6.3.5 Collector.cpp

Modul *Collector* byl navržen pro zapouzdření funkcionality týkající se sběru dat z internetové kamery. V rámci jeho rozhraní je přístupná pouze funkce *collect()*, která je přetížená a lze ji volat dvěma způsoby: pro sběr snímků do vektoru matic nebo pro ukládání snímků na disk. Připojování na zadanou adresu je blokující operace, proto bylo přidáno asynchronní přerušení, které po uplynutí tří minut ukončí běh funkce s chybovou návratovou hodnotou. Dekódování datového proudu je zajištěno funkcemi z knihovny *libavcodec* (součást kolekce FFMpeg), která disponuje většími možnostmi pro práci s daty na nižší úrovni. Z hlavičky rámce lze například určit, jestli se jedná o klíčový snímek. Díky tomu mohou být pro zvýšení kvality výstupu vybírány ke zpracování pouze klíčové snímky, což lze nastavit parametrem *onlyKeyFrames* při volání funkce *collect()*.

6.3.6 Compositor.cpp

Compositor je modul, který při provádění experimentů s různými technikami sloužil k vytvoření kompozice z více videí pro porovnání výsledků. Metoda *composeImages()* umožňuje spojování na úrovni jednotlivých obrázků, *composeVideos()* pak kombinuje celé sekvence. Rozhraní příslušné třídy umožňuje definovat počet snímků na výšku a na šířku, mezery mezi snímky a popisky jednotlivých videí.

6.3.7 MRF3D.cpp

V tomto souboru je implementován algoritmus pro Markovovská náhodná pole (MRF) popsaný v kapitole 3. Zdrojový kód vychází z návodu na použití 2D MRF pro úlohu stereo vidění, kde je ze dvou obrázků z levé a pravé kamery počítána hloubková mapa scény, viz [1]. Tento kód byl transformován rozšířením 2D mřížky na 3D a funkce pro výpočet unárních a párových potenciálů byly nahrazeny funkcemi odpovídajícími této úloze.

Jádrem výpočtů je funkce *SendMsg()* pro posílání zpráv v rámci algoritmu „Loopy Belief Propagation“ (LBP). Podle aktuálního směru průchodu mřížkou se počítá párový potenciál (*PairwiseTemporal()* pro směr průchodu v čase, *PairwiseSpatioTemporal()* pro všechny směry). Z kódu funkce *SendMsg()* je patrná i výpočetní náročnost algoritmu, neboť se jedná o šest zanořených cyklů. - Pro každé z možných posunutí pixelu p se počítá pravděpodobnost posunutí sousedního pixelu $p2$ v trojrozměrném prostoru. Po ukončení poslední iterace LBP je ve funkci *MAP()* (maximum a posteriori) určeno výsledné posunutí pro každý pixel ve videosekvenci.

6.3.8 NonUniformSampler.cpp

Rozhraní modulu pro neuniformní vzorkování obsahuje metodu *getOptimalSampling()*, která dynamickým programováním počítá optimální vzorkování pro zadaný počet vzorků. Jedná se o algoritmus *Perez and Vidal* popsaný výše v pseudokódu. Neveřejná metoda *errorMetric()* počítá penalizaci dvou snímků podle metriky min-error. Implementace metriky min-change v tomto případě příliš nemá význam, neboť odstranění odlišných rámců ze sekvence úspěšně řeší již implementované základní filtry s menší výpočetní náročností. Návratovou hodnotou z hlavní funkce je vektor celočíselných hodnot, které mají význam indexů vybraných vzorků z původní sekvence.

Zjednodušená verze algoritmu pro neuniformní vzorkování je implementována ve funkci *getSamplingWithMotion()*, která nejdříve počítá pohyb ve scéně z rozdílu sousedních snímků. Do pomocného pole *ratings* jsou ukládány značky, pokud je překročen nastavený práh a zpětným průchodem pole je dosaženo stavu, kdy na pozici každé značky je číslo reprezentující počet po sobě následujících značek, viz obrázek 5.3. Funkce opět vrací vektor celočíselných hodnot s indexy vybraných vzorků.

Kapitola 7

Diskuse výsledků

V této kapitole prezentuji dosažené výsledky formou porovnání výstupů jednotlivých algoritmů s klasickým časovým snímkem. Pozornost je věnována také paměťové a časové náročnosti. V neposlední řadě je tato kapitola také zhodnocením současného stavu a nastiňuje možné cesty pro budoucí rozšiřování projektu.

7.1 Srovnání s naivním časovým snímkem

Hlavním úkolem této práce bylo vylepšit vizuální charakteristiky videa, proto nejprve hodnotím, jak jednotlivé techniky přispěly k odstranění nežádoucích jevů. Lokální filtrace, neuniformní vzorkování a optimalizace naivního časového snímku na 3D MRF mřížce - to jsou metody, které byly na sadě pořízených dat testovány. Všechny pracují zcela jiným způsobem a proto se také liší výsledky a vhodnost jejich použití. V závěru diskuse jednotlivých metod uvádím příklady vhodných a nevhodných scén pro jejich aplikaci.

Pro vizuální prezentaci výsledků mé práce jsem vybral sekvence tří po sobě následujících snímků, na kterých lze vidět rozdíl mezi naivním časovým snímkem a použitou metodou. V popisu obrázku je vždy uvedeno umístění příslušného videa na CD disku odevzdaného spolu s touto diplomovou prací. Čtenář si tak může dohledat a pustit celé video, odkud byla tato sekvence snímků vyňata.

7.1.1 Odstranění nežádoucích jevů základními filtry

Z navržených základních filtrů měl nejuniverzálnější využití mediánový a průměrovací filtr. Min-filtr a max-filtr byl spíše experimentální a pro běžné scény tyto filtry neprodukují užitečné výstupy. Min-filtr však například dokázal eliminovat kouř na popředí v některých scénách a s pomocí max-filtru jsem dosáhl zajímavého efektu na nočních snímcích (viz následující podkapitola). Nejlepší výsledky podávalo dvoufázové filtrování, kdy byl na sekvenci snímků aplikován medián a výsledek dále zpracován průměrovacím posuvným oknem. V úvodu této práce byly na pořízené sadě dat definovány kategorie nežádoucích jevů, které se v klasickém časovém snímku vyskytují. Jak si s jednotlivými problémovými skupinami poradily tyto základní filtry?

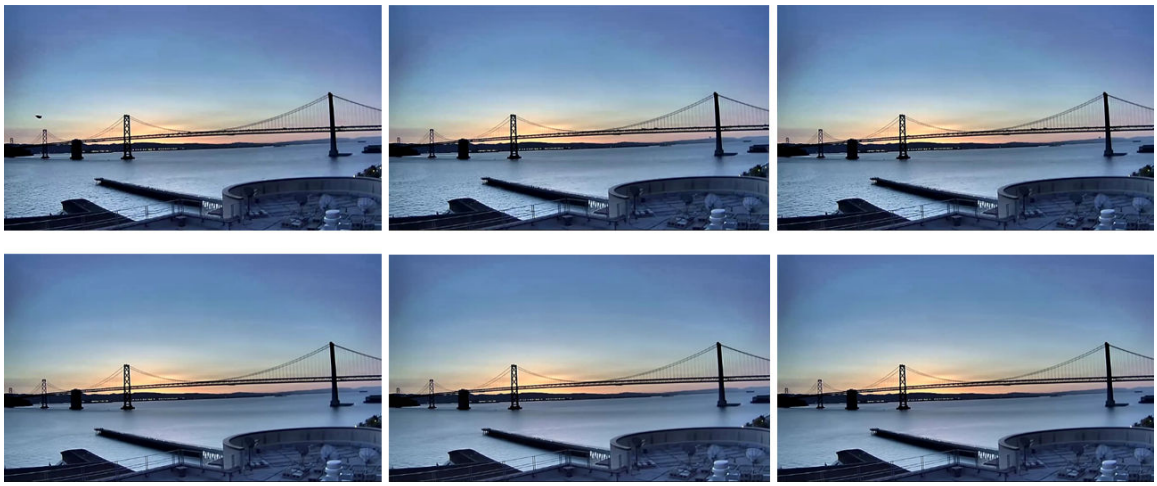
Periodické pohyby a změny v kaskádě mediánového a průměrovacího filtru eliminuje zpravidla již medián. V případě pohyblivých částí (např. turbína) jsou tyto zcela eliminovány a zůstává pouze pozadí scény. U periodických změn jasu (např. semafor) bude výsledkem stav objektu, ve kterém se nacházel po nejdelší dobu. Základní filtry tedy peri-

odické pohyby a změny zcela odstraní. Pokud by byla žádoucí jejich rekonstrukce, pak je nutné použít techniku převzorkování, což bylo ukázáno výše na příkladu kyvadla.

Náhodné pohyby a změny stálých objektů jsou největším oříškem, neboť je nelze eliminiovat ani rekonstruovat pouhým převzorkováním. Mediánový filtr tyto části scény vyhladí, čímž je video zbaveno vizuálně nepříjemného mihotání obrazu, ale zároveň se vytrácí původní kontury. Z rozvlněné vody v oceánu se tak stává hladká plocha a obrysy stromů ve větru jsou rozmazané. V řešení této kategorie nežádoucích jevů tedy základní filtry příliš neuspěly a stále zde vidím prostor pro zlepšování jinými metodami.

Náhodný výskyt objektů a jejich proměny se také daří úspěšně filtrovat v první fázi zpracování mediánem, ovšem podmínkou je malý počet objektů a jejich malá setrvačnost. Artefakty v jednotlivých pixelech vznikají v případě, kdy se na dané pozici větší část periody nachází nějaký objekt v popředí. S velkým počtem lidí pohybujících se na náměstí si například tato metoda již zcela neporadí. Avšak stačí i jeden člověk ve scéně, který se na chvíli zastaví a na příslušném místě v obraze vzniknou artefakty. Zde se uplatní druhá fáze filtrování průměrovacím posuvným oknem, která výsledný dopad dokáže zmírnit.

Globální změny scény jsou zpravidla dlouhodobějšího charakteru, a proto na ně první fáze filtrování nemá velký vliv, protože má jako vstup snímky z krátkého časového úseku. Zde se nejvíce uplatní posuvné okno pro druhou fázi filtrování, které je aplikováno na výsledné mediánové snímky. Experimentoval jsem také s mediánem jako filtrem pro posuvné okno, který lépe odstraní extrémy z dané sekvence (například jeden snímek ve stínu mezi snímky s jasnou oblohou). Častější však bývají skokové změny osvětlení trvalejšího charakteru. Zde přichází ke slovu právě průměrovací filtr, který z těchto skokových změn udělá plynulé. Například místo okamžité změny ze stavu „zataženo“ na „slunečno“ ve dvou po sobě následujících snímcích je výsledným efektem plynulé zvýšení jasu ve scéně. I při častých změnách osvětlení v rámci jednotek snímků, které v naivním časovém snímku způsobují nepříjemné blikání celého videa, jsou v případě průměrovacího filtru omezeny hranice minima a maxima těchto změn a výsledek působí mnohem plynuleji.



Obrázek 7.1: Východ slunce nad Baylights, San Francisco. Aplikací mediánového filtru byla odstraněna auta na mostě a prolétávající ptáci (viz první snímek). Výsledné video také obsahuje méně šumu. Shora dolů: 1) naivní časový snímek, 2) filtrace mediánem a posuvným průměrovacím oknem. (Video na přiloženém CD: `video/results/shortterm/baylights.avi`)

7.1.2 Max-filtr v nočních scénách

Osvědčenou kombinací základních filtrů pro noční scény s projíždějícími auty se stal průměr a max-filtr. Průměrováním lze eliminovat šum z čipu kamery, který je typicky v neosvětlených scénách výrazný, a max-filtr vytváří efekt světelných stop za projíždějícími auty. Tato technika produkuje vizuálně pěkné výsledky v tmavých scénách s vysokým kontrastem světel, jak je vidět například na obrázku 7.2. Jako méně efektní se použití max-filtru ukázalo ve scénách se silným pouličním osvětlením, kde méně kontrastu znamená větší citlivost na nastavený práh a více artefaktů ve výsledném videu.

V každé periodě je potřeba aplikovat max-filtr na všechny snímky. Je možné provést i řídké vzorkování v rámci jedné periody, avšak na úkor kvality výsledku. Obecně tedy metoda není použitelná dlouhodobě na video s velkým rozlišením v reálném čase, neboť zpracování vstupu trvá déle než je rychlost jeho načítání. Ukázkou je graf na obrázku 7.6 pro video 640x400px, kde je porovnáván medián patřící do stejné skupiny algoritmů s řazením prvků.

Obrázek 7.2: Použití max-filtru v noční scéně s projíždějícími auty. Vzorkovací perioda je 50 snímků. Shora dolů: 1) naivní časový snímek, 2) průměr snímků, 3) průměr + maxfiltr. (Video na přiloženém CD: `video/experiments/maxfilter-zlin.avi`)

7.1.3 Vyhodnocení dlouhodobého časového snímku

Od listopadu 2014 do března 2015 byla z pěti vybraných IP kamer pravidelně sbírána data pro dlouhodobý časový snímek. Každý den ve stejný čas bylo uloženo 100 klíčových snímků. K poslednímu dni v březnu byl tento projekt zastaven a získaná data zpracována. Zamýšlená perioda pro výsledný časový snímek byla 24 hodin. Nejprve jsem vytvořil referenční video naivního časového snímku, kde byl z každého dne použit první uložený obrázek. Druhé pokusné video vzniklo transformací všech 100 snímků mediánovým filtrem na jeden

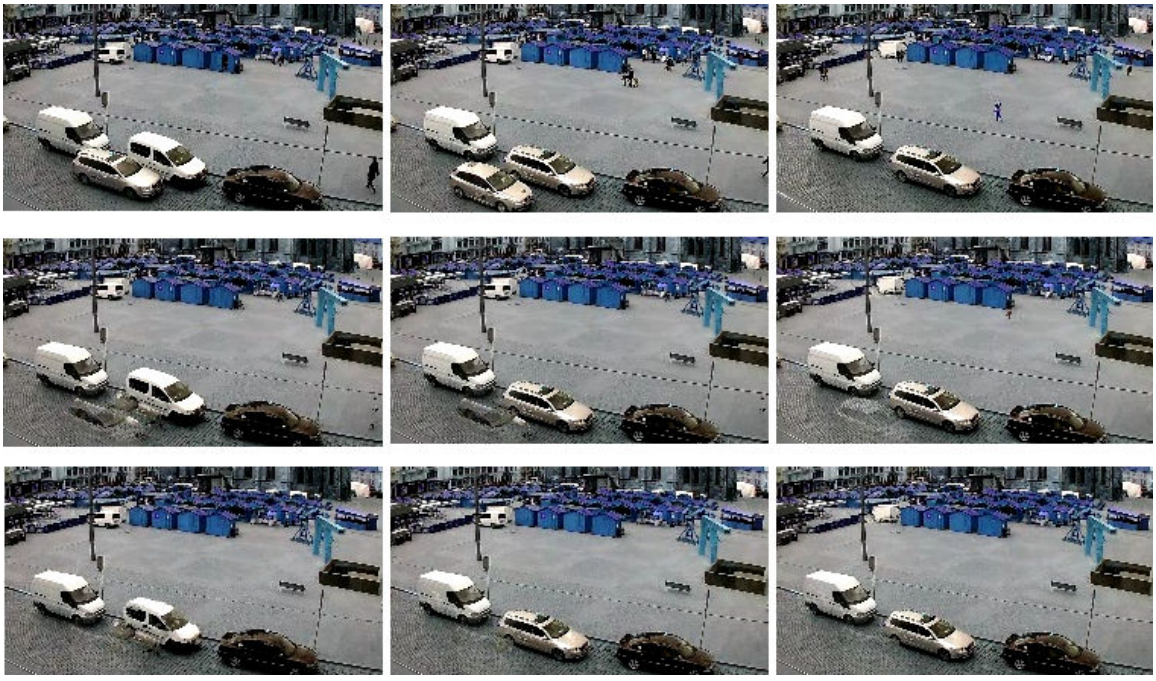
výsledný. Ve třetím videu jsem dále na mediánové snímky aplikoval průměrovací posuvné okno velikosti pět. Výstup byl ve všech třech případech generován s frekvencí 25 snímků za sekundu.

Na obrázku 7.3 je ukázka vytvořená z jedné ze sledovaných kamer. Lze si povšimnout, že mediánový filtr odstranil z původních snímků projíždějící auta a procházející lidi. Stále však zůstávají ostré přechody mezi snímky způsobené parkujícími auty a různými rozptylovými podmínkami v jednotlivých dnech. Toto dokáže rapidně vylepšit průměrovací posuvné okno, které vytvoří efekt prolínání u měnících se aut a zároveň zajistí plynulý přechod změn osvětlení. Výrazný přechod počasí mezi druhým a třetím dnem je tak přeměněn na postupné snižování intenzity stínů.

Obrázek 7.3: Dlouhodobý časový snímek z kamery na VUT FIT. Snímky jsou pořízeny tři následující dny po sobě. Shora dolů: 1) naivní časový snímek, 2) medián z denních snímků, 3) medián s aplikací posuvného průměrovacího okna. (Video na přiloženém CD: `video/results/longterm/Cam1day.avi`)

7.1.4 Markovovská náhodná pole

Optimalizace energie na Markovovských náhodných polích (MRF) se od ostatních implementovaných algoritmů liší tím, že jako vstup očekává již navzorkované snímky z původního videa v reálném čase. V ukázce na obrázku 7.4 je tedy první video s naivním časovým snímkem vstupem pro druhé a třetí video, kde je porovnáván výsledek filtrace posuvným oknem s výsledkem při použití MRF. Pro účely tohoto porovnání byla zvolena stejná velikost 3 pixely pro posuvné okno i každý z rozměrů prohledávacího prostoru MRF. Na srovnávaných videosekvencích lze vidět, že v případě lokální filtrace posuvným oknem zůstává v obraze více artefaktů než v případě MRF, které pracuje s kontextem okolních pixelů v časoprostoru.



Obrázek 7.4: Srovnání optimalizace naivního časového snímku na Markovovských náhodných polích s filtrováním posuvným oknem. Shora dolů: 1) naivní časový snímek, 2) posuvné mediánové okno, 3) optimalizace na MRF. (Video na přiloženém CD: `video/experiments/mrf-plzen.avi`)

Obecně lze říci, že MRF mají větší sílu než lokální filtrace v kontextu odstranění všech kategorií zmíněných nežádoucích jevů. Artefakty vznikají v menší míře a výstup je vizuálně kvalitnější. Výhoda počítání se statistickou závislosti pixelů v časoprostoru se nejvíce uplatňuje při vyhlazování náhodných pohybů stálých objektů, kde jsou více patrné původní obrysy a nedochází k takovému rozmazání jako v případě lokální filtrace.

Přestože implementovaný algoritmus pracuje s časovým snímkem jako vstupem, není vhodné použít jako předstupeň základní filtry, protože tím dáváme vznik artefaktům již v této první fázi a výhody MRF se tím ztrácejí. To platí zejména pro zmíněné pohyby stálých objektů jako například stromy ve větru, které by byly rozmazané již ve vstupním časovém snímku.

Velkou nevýhodou tohoto algoritmu je výpočetní i paměťová náročnost. Testován byl pouze na velmi krátkých videosekvencích s malým rozlišením a pro reálné vstupy se použitelnost přesouvá spíše do teoretické roviny, více v podkapitole 7.2 o paměťových a výpočetních nárocích. Zanedbáme-li toto omezení na velikost vstupu, pak je vhodné použití pro komplexní scény obsahující mnoho rušivých prvků, kde základní filtry neprodukují uspokojivé výsledky.

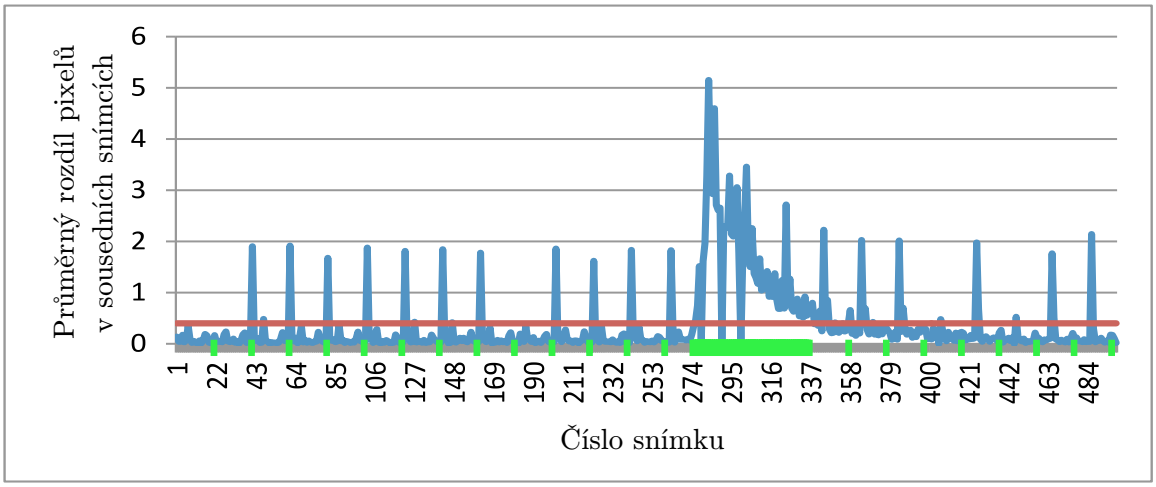
7.1.5 Neuniformní vzorkování

Neuniformní vzorkování má velmi specifické použití, a proto musí být vhodně zvolena scéna, aby byl výsledek efektní. Zvolená kamera by měla snímat statické prostředí, kde pouze zřídka proběhne nějaká změna, kterou chceme zachytit tím, že zvýšíme hustotu vzorkování snímků. Proto nemá smysl aplikovat proměnlivé vzorkování na scény s neustálými změnami

a pohybem. Správným příkladem výběru scény je záběr na málo frekventovanou cestu, kde občas projíždí auto. Průjezd auta ve výsledku bude zpomalen a ostatní části videa budou vzorkovány s delší periodou.

První verze algoritmu pro výpočet optimálního vzorkování zachovává celkový počet snímků stejný jako při uniformním vzorkování. To se dá považovat za výhodu i nevýhodu zároveň, neboť je sice předem známá délka výsledného videa, ale na druhou stranu se při nahuštění vzorků na jednom místě prodlouží perioda ve zbytku videa. To lze korigovat omezením na maximální vzdálenost dvou vzorků (které zároveň sníží výpočetní náročnost), ale pak nebude výsledek optimální vzhledem ke stanovené chybové metrice. Při vizuálním porovnání s původním videem jsou rozdíly jen nepatrné (tím spíše při omezení maximální délky periody) a kvadratická složitost algoritmu je tudíž příliš vysokou cenou za nepříliš užitečné výsledky.

Zjednodušená verze, která pracuje pouze se dvěma konstantními délkami periody, na základě provedeného testování produkuje vizuálně přitažlivější výsledky. Nevýhodou je, že není předem známá délka výsledného videa. Víme pouze, že počet rámců bude v rozmezí dvou extrémů - délka při vzorkování celého videa kratší periodou a délka při vzorkování delší periodou. Úspěšné použití je závislé na správném nastavení prahu průměrné změny pixelů mezi sousedními snímky, který se může pro různé scény lišit. Při mých experimentech se osvědčila prahová hodnota 0,4. Na obrázku 7.5 je ukázka použití dvoustavového vzorkování, kde byl správně detekován velký pohyb (v tomto případě projíždějící auto) a dočasně snížena vzorkovací perioda.



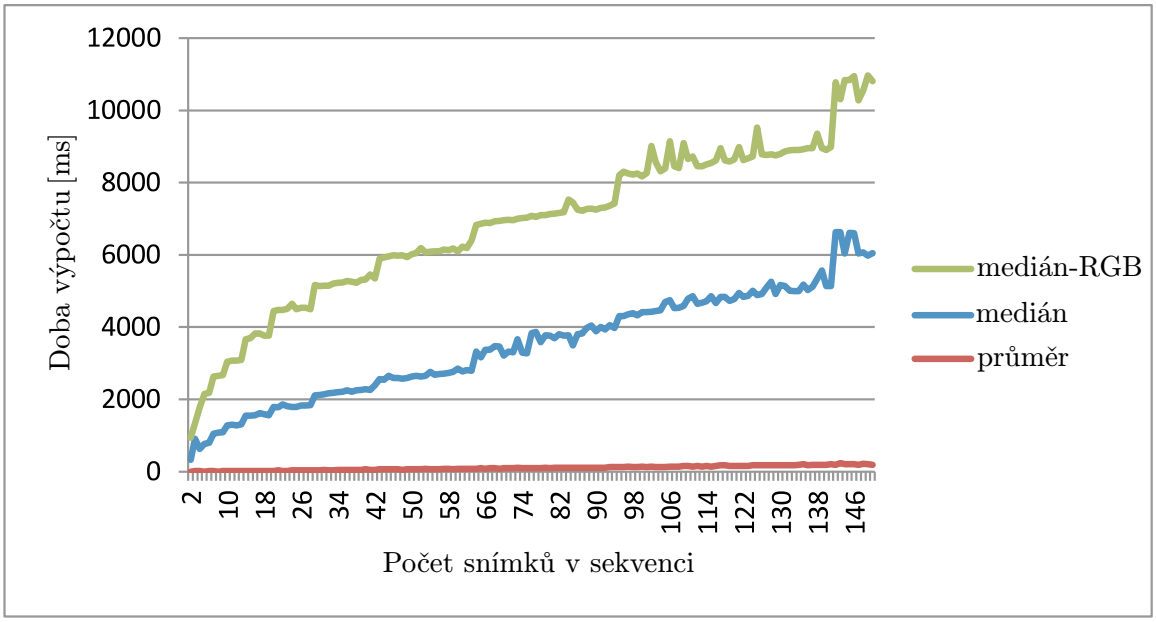
Obrázek 7.5: Ukázka použití dvoustavového neuniformního vzorkování. Modře je znázorněna momentální hodnota rozdílu hodnot pixelů, červeně nastavený práh pro přepnutí periody a zeleně je vyznačeno výsledné vzorkování. Na tomto grafu je vidět, že perioda se mění jen při dlouhodobém překročení prahu, které odpovídá projíždějícímu autu. Pravidelně rozmístěná lokální minima korespondují s klíčovými snímky ve videu.

7.2 Výpočetní a paměťová náročnost

Dosud byly jednotlivé metody pro vytvoření pokročilého časového snímku diskutovány převážně z vizuálního hlediska. Nyní se pojďme zaměřit na to, jaké mají nároky na zdroje. Jak

bylo předesláno již v předchozím textu, nejlépe z tohoto srovnání vychází lokální filtrace, zatímco optimalizace na 3D MRF má výpočetní i paměťové nároky ze všech implementovaných algoritmů největší.

Na obrázku 7.6 je graf znázorňující nárůst výpočetní náročnosti lokálních filtrů v závislosti na rostoucím počtu snímků v sekvenci. Medián zde reprezentuje celou skupinu filtrů včetně max-filtru a min-filtru, které se od mediánu liší pouze výběrem prvku ze seřazeného pole pixelů. V porovnání s průměrovacím filtrem je doba výpočtu mnohonásobně vyšší, a to právě kvůli řadícímu algoritmu volanému pro každý pixel výsledného snímku zvlášť. Linearitmická složitost tohoto algoritmu se promítá do tvaru křivky grafu. Co se týče paměťových nároků, potřebuje jádro algoritmu pouze danou sekvenci snímků a pracovní pole dvojic $(index, pixel)$ pro řazení. V aplikaci *timelapse-online* se vstupním videem o rozměrech obrazu 640x400 pixelů se pro sekvenci o velikosti 30 snímků a velikost posuvného okna 5 alokuje 58 MB operační paměti.



Obrázek 7.6: Srovnání výpočetní náročnosti průměrovacího filtru, jednokanálového mediánu a RGB mediánu. Video: 640x400 px; CPU: 2x 2,4 GHz; 4GB RAM.

U Markovovských náhodných polí je pro výpočetní náročnost rozhodující složitost algoritmu *Loopy Belief Propagation* pro rozesílání zpráv napříč 3D mřížkou. Tato složitost je lineární vzhledem k velikosti videa, ale kvadratická vzhledem k velikosti stavového prostoru (tj. prohledávací oblast všech možných posunutí pixelu). Pro každý pixel videa je potřeba uchovat ohodnocení každého z možných stavů přicházející ze všech směrů (6 sousedních pixelů). Například pro video o velikosti 500^3 a prohledávací prostor 10^3 je to $500^3 \times 10^3 \times 6 \times 4$ B, tedy přibližně 3 TB. V článku Motion Denoising [10] je uvedeno, že sekvenci 300^3 s prohledávací oblastí o velikosti 7^3 trvalo vyřešit 50 hodin na čtyřjádrovém CPU s frekvencí 2.66 GHz a operační pamětí 28 GB. Já jsem ve svých experimentech používal dvoujádrové CPU s frekvencí 2,4 GHz a operační paměť 4 GB. Video o velikosti obrazu 266x144 a počtu snímků 50 trvalo zpracovat 2,5 hodiny.

7.3 Budoucí vývoj

Tato práce byla pojata jako experimentální a při řešení byla vyzkoušena celá řada nápadů, z nichž některé nakonec vyhasly stejně rychle jako vzplanuly, jiné byly úspěšně použity pro vytvoření pokročilého časového snímku. Zároveň se však začaly rodit i myšlenky na další pokračování tohoto projektu v širším obraze. Na závěr této kapitoly bych tedy rád představil svou vizi do budoucna v kontextu dalšího rozšiřování mé práce.

Ačkoli výsledná aplikace je navržena pro lokální spouštění, jednou z odnoží tohoto projektu byla i implementace a testování lokálních filtrů na straně serveru. V rozvíjení tohoto serverového řešení vidím největší potenciál pro komerční využití. Uživatel by mohl z webového rozhraní spustit nahrávání pokročilého časového snímku se zvolenými parametry, veškeré výpočty by probíhaly na straně serveru a nakonec by mohl být elektronickou poštou zaslán odkaz pro stažení výsledného videa. Zde se nabízí možnost začlenit do uživatelského prostředí vizuální prvky pro větší interakci s uživatelem (například nastavení masky pro části videa, které mají být ponechány bez filtrace), ale zároveň lze propracovat i větší automatizaci při zpracování (například rozpoznání noční scény, která je vhodná pro použití max-filtru).

Výše zmíněné nápady jsou vesměs nástavbami vedoucími ke zvýšení uživatelského komfortu. Prostor pro vylepšení však nabízí i jádro aplikace. Pokročilejší algoritmy, které byly implementovány, potřebují mnoho výpočetních i paměťových zdrojů a nejsou schopny pracovat v reálném čase s živým vysíláním kamery. Lokální filtrace je sice efektivní, ale neobejde se bez artefaktů ve výsledném obraze. Další bádání v této oblasti by mohlo přinést objevení nových metod, které by vizuální aspekty časového snímku ještě více vylepšily.

Systém byl navrhován pro vytvoření časového snímku ze statických kamer, přičemž se počítá s tím, že vstup je již stabilizován. V praxi se ukázalo, že velké množství kamer, byť nejsou otočné, nejsou příliš pevně uchyceny a při náporu větru se vychýlí. Do další verze aplikace plánuji začlenit stabilizační mechanismus, který tuto vadu z výsledného videa odstraní. Další možnou cestou pro rozšiřování funkcionality je umožnit zpracování otočných kamer a spojováním obrázků vytvářet panoramatický časový snímek.

Při pořizování ukázkových videí jsem často čekal na příhodné povětrnostní podmínky, přičemž mě napadlo další možné rozšíření do budoucna, a to přímé propojení s meteorologickými daty. Uživatel by tak mohl jako požadavek zadat například, že chce z dané kamery natočit časový snímek zachycující průběh bouřky a snímání by bylo automaticky spuštěno na základě práce s aktuální předpovědí počasí a daty z radaru. Stejným způsobem by mohla být zahrnuta také astronomická data (např. východ slunce, měsíc v úplňku, konjunkce jasných planet apod.). Podmínkou pro správnou funkčnost by bylo přesné zadání zeměpisných souřadnic dané kamery.

Kapitola 8

Závěr

Tato diplomová práce definuje problematiku časového snímku ze stacionární kamery, analyzuje současné techniky jeho tvorby a na pořízené sadě dat z 58 veřejně dostupných kamer se zaměřuje na nepřirozené jevy, které vznikají při vzorkování obrazu v čase. Tyto jevy byly následně klasifikovány do čtyř základních kategorií. Protože běžně se vyskytující scény jsou příliš komplexní, byla následně pořízena fotoaparátem na stativu videa s předem připravenými kompozicemi pro oddělení jednotlivých skupin nežádoucích jevů a jejich hlubší analýzu.

Na základě zjištěných faktů o klasickém časovém snímku byly navrženy algoritmy pro pokročilé zpracování obrazu vylepšující vizuální aspekty výsledného videa ve smyslu subjektivní plynulosti pohybů ve scéně. Tyto algoritmy se liší způsobem použití i výpočetní a paměťovou náročností. Při prováděných experimentech byly nejvíce efektivní lokální filtry, které lze použít i pro zpracování živého vysílání IP kamer v reálném čase a eliminují většinu nežádoucích jevů s minimálním vznikem artefaktů. Specifické použití má neuniformní vzorkování pro zvýraznění občasných změn ve statické scéně. Dalším představeným přístupem je optimalizace naivního časového snímku na Markovovských náhodných polích a bylo ukázáno, že se jedná spíše o teoreticky použitelný algoritmus, neboť pro běžně dostupnou výpočetní techniku lze aplikovat pouze na velmi malé videosekvence.

Výsledná aplikace byla implementována ve dvou verzích - pro živé vysílání v reálném čase a pro nahraný záznam, který může být zpracován pokročilejšími technikami s podstatně delším trváním než je délka samotného videa. Navržené algoritmy byly také testovány na jednom ze serverů Angelcam, kde bylo paralelně spuštěno několik procesů pro nahrávání více streamů současně.

V poslední části textu jsem nastínil možné cesty pro pokračování tohoto projektu s vyzdvižením potenciálu v dalším rozvoji výše zmíněného serverového konceptu. Další navrhovaná vylepšení spočívají v rozvoji interakce s uživatelem, tvorby panoramatického časového snímku či propojení s dalšími systémy (např. meteorologická data).

Literatura

[1] Loopy belief propagation, Markov Random Field, stereo vision — Nghia Ho. [online], 2012.
URL http://nghiaho.com/?page_id=1366

[2] Timescapes - Digital Timelapse Discussion. [online], 2014.
URL http://forum.timescapes.org/phpBB3/

[3] libcve Documentation. [online], 2015.
URL https://www.angelcam.com/computer-vision/documentation/

[4] Bennett, E. P.; McMillan, L.: Computational Time-Lapse Video. In *SIGGRAPH 2007*, ročník 26, ACM New York, 2007, s. 102–107, iSSN: 0730-0301.

[5] Blake, A.; Kohli, P.; Rother, C.: *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011, iSBN 0262015773.

[6] Farnebäck, G.: Two-Frame Motion Estimation Based on Polynomial Expansion. In *Image Analysis*, Springer Berlin Heidelberg, 2003, s. 363–370, iSBN 978-3-540-40601-3.

[7] Laganiere, R.: *OpenCV 2 Computer Vision Application Programming Cookbook: Over 50 Recipes to Master this Library of Programming Functions for Real-time Computer Vision*. Packt Publishing Ltd, 2011, iSBN 1849513252.

[8] Ortiz, J. L. R.: *Probabilistic Time Lapse Video*. Diplomová práce, University of Edinburgh, 2008.

[9] Perez, J.-C.; Vidal, E.: Optimum polygonal approximation of digitized curves. *Pattern Recognition Letters*, ročník 15, 1994: s. 743–750.

[10] Rubinstein, M.; Liu, C.; Sand, P.; aj.: Motion Denoising with Application to Time-lapse Photography. Technická zpráva, červen 2011.

[11] Shannon, C. E.: Communication in the presence of noise. *Proc. Institute of Radio Engineers*, ročník 37, č. 1, 1949: s. 10–21.

[12] Turek, L.: *Časový snímek z obrazu stacionární kamery*. Semestrální projekt, FIT VUT v Brně, Brno, 2014.

[13] Wedel, A.; Cremers, D.: *Stereo Scene Flow for 3D Motion Analysis*. Springer US, 2011, iSBN 978-0-85729-964-2.

[14] Wildemuth, B. M.; Marchionini, G.; Yang, M.; aj.: How Fast Is Too Fast? Technická zpráva, 2003.

[15] Yedidia, J. S.; Freeman, W. T.; Weiss, Y.: Understanding belief propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, Morgan Kaufmann Publishers Inc. San Francisco, 2003, s. 239–269, iSBN: 1-55860-811-7.

Dodatek A

Obsah CD

Přiložené CD má následující adresářovou strukturu:

- **app** - Aplikace
  - **bin** - Binární soubory přeložené ze zdrojových kódů pro běh na školním serveru merlin
  - **src** - Zdrojové soubory
- **text** - Text diplomové práce ve formátu PDF a původní zdrojové kódy pro LaTeX
- **video** - Ukázky výstupů
  - **experiments** - Prováděné experimenty odkazované z textu diplomové práce
  - **results** - Pokročilý časový snímek z různých kamer
    - **longterm** - Dlouhodobý časový snímek (výsledky 4,5 měsíčního snímání)
    - **shortterm** - Krátkodobý časový snímek ($<$ 24 hodin)