VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

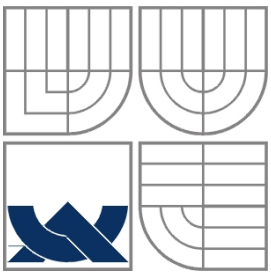
POTLAČENÍ VLIVU ATMOSFÉRICKÉ TURBULENCE V REÁLNÉM ČASE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

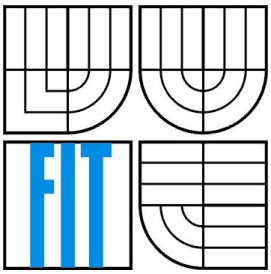
AUTOR PRÁCE
AUTHOR

JIŘÍ MACKŮ

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

POTLAČENÍ VLIVU ATMOSFÉRICKÉ TURBULENCE V REÁLNÉM ČASE

Real Time Suppression of Atmospheric Turbulence

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ MACKŮ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ MARVAN

BRNO 2014

Abstrakt

Tato práce se zabývá odstraněním nežádoucích deformací obrazu způsobených turbulencemi v zemské atmosféře. Úkolem je navrhnout a implementovat algoritmus potlačení vlivu atmosférických turbulencí ve videosekvencích pro použití v dálkovém kamerovém sledovacím systému. Tato práce se také zaměřuje na optimalizaci stávajících algoritmů pro běh v reálném čase. Algoritmus bude implementován v programovacím jazyce C++ s využitím knihovny pro počítačové vidění OpenCV.

Abstract

This paper deals with removing undesirable image deformations caused by turbulences of Earth's atmosphere. The goal is to design and implement algorithm for suppression of atmosphere turbulence in video sequences to use in long-distance camera monitoring system. This paper also focuses on optimization of existing algorithms for running in real-time. Algorithm will be implemented in C++ programming language with use of library for computer vision OpenCV.

Klíčová slova

Atmosférická turbulence, počítačové vidění, optický tok, registrace obrazu, OpenCV, reálný čas, OpenCL, hardwarová akcelerace

Keywords

Atmospheric turbulence, computer vision, optical flow, image registration, OpenCV, real-time, OpenCL, hardware accceleration

Citace

Macků Jíří: Potlačení vlivu atmosférické turbulence v reálném čase, bakalářská práce, Brno, FIT VUT v Brně, 2014

Potlačení vlivu atmosférické turbulence v reálném čase

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Marvana. Další informace mi poskytl Ing. David Herman.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Macků
20. květen 2014

Poděkování

Chtěl bych zde poděkovat svému konzultantovi Ing. Davidu Hermanovi za odbornou pomoc, praktické připomínky a trpělivost během vzniku této práce. Také bych chtěl poděkovat své rodině za psychickou a materiální podporu.

Obsah

1 Úvod

Atmosférická turbulence je přirozený jev přítomný v zemské atmosféře. Je to neuspořádaný pohyb vzduchu způsobený například nerovnoměrným zahříváním zemského povrchu - termická turbulence [1, 2]. Tento pohyb způsobuje náhlé a náhodné změny indexu lomu a ovlivňuje tak světelné paprsky procházející atmosférou. Toto se projevuje zejména při snímání obrazu na velké vzdálenosti a negativně tak ovlivňuje výsledný záznam [3].

Z pohledu počítačového vidění obsahuje optický záznam, ovlivněný atmosférickou turbulencí, geometrickou deformaci, časově a prostorově proměnné rozostření a při dlouhé expozici i pohybové rozostření [3]. Všechny tyto jevy zhoršují nebo přímo znemožňují jakékoli další analytické využití záznamu. Například v astronomii, při steroskopickém měření vzdálenosti hvězd, může již drobná odchylka v záznamu způsobit velké rozdíly v naměřené vzdálenosti oproti skutečné.

Odstranění těchto nedostaků je možné řešit dvěma způsoby: hardwarově za pomoci adaptivní optiky nebo softwarově na bázi zpracování signálů [3]. Tato práce se zabývá výhradně softwarovými přístupy. Tyto přístupy se snaží získat samotný vysoce kvalitní snímek z videosekvence nebo série snímků za předpokladu, že scéna i snímač jsou oba statické a pohyb vyskytující se v záznamu je způsoben pouze vzdušnou turbulencí. V této práci budeme vycházet ze stejných předpokladů.

V následující kapitole si nastíníme možná řešení a současné metody redukce atmosférické turbulence. Ve třetí kapitole se pokusíme navrhnout možný algoritmus pro řešení našeho problému s využitím přístupů představených v první kapitole. Následně ve čtvrté kapitole popíšeme implementaci námi navrženého algoritmu v prostředí knihovny OpenCV. Pátá kapitola bude popisovat možné optimalizace algoritmu za účelem zrychlení jeho provádění a dosažení běhu v reálném čase. V šesté kapitole budeme testovat náš algoritmus na vygenerovaných i reálných datech a budeme zkoumat jeho efektivitu a věrohodnost získaných výsledků. V závěrečné kapitole zhodnotíme dosažené zlepšení obrazu a budeme diskutovat možné další návrhy a zlepšení, které by bylo možné provést.

2 Současné přístupy

Jak bylo naznačeno v úvodu, atmosférická turbulence má za následek několik typů zhoršení obrazové kvality. Jedná se zejména o geometrickou deformaci a časově a prostorově proměnné rozostření [3]. Současné přístupy řeší odstranění postupně: nejprve se redukuje pomocí nějaké nerigidní registrace geometrická deformace a následně se obraz zostří metodou „deblurringu“ [3-7].

Všechny tyto přístupy pracují s předpokladem statické scény. V našem případě se, kvůli zjednodušení algoritmu a tím zvýšení rychlosti výpočtu, zaměříme především na odstranění geometrické deformace způsobené atmosférickou turbulencí.

2.1 Odstranění atmosférické turbulence

Jak již bylo uvedeno atmosférická turbulence má v obrazových systémech za následek geometrické deformace obrazu a časově a prostorově proměnné rozostření. Vznik těchto nechtěných efektů je možné modelovat jako 2D konvoluci ideálního snímku s proměnnou rozptylovou funkcí turbulence a neměnnou difrakčně limitovanou rozptylovou funkcí optiky, sečtenou s nějakým dodatečným šumem způsobeným snímačem [3]. Protože neznámá rozptylová funkce turbulence je prostorově a časove proměnná, obnova původního vysoce kvalitního obrazu není triviální.

K řešení tohoto problému existuje několik různých přístupů, ale protože informace z jednoho snímku není většinou dostatečná k jeho rekonstrukci, všechny tyto přístupy jsou založeny na zpracování videí nebo alespoň obrázkových sekvencí, za předpokladu statické scény [3-7]. Současné algoritmy pro odstranění atmosférické turbulence mohou být obecně rozděleny do dvou hlavních kategorií. První kategorie je založena na rekonstrukci obrazu z více snímků. Druhá kategorie využívá výběru snímků a fúzní metody pro redukci rozmazání způsobených turbulencí.

2.1.1 Metody vícesnímkové rekonstrukce

Tento přístup nejdříve vyžaduje použití techniky nerigidní obrazové registrace pro registraci všech zpracovávaných snímků a využití parametrů registrace k odhadu deformační matice. Potom je za pomoci bayesovské obrazové rekonstrukce získán ostrý snímek.

Například v článku M. Shimizua [4] je předveden experiment ukazující, že lokální turbulentní pohyb má charakter normálního rozložení s nulovou střední hodnotou, a tudíž je možné odstranit geometrickou deformaci prostým zprůměrováním zpracovávaných snímků. Takový snímek je sice ještě rozostřenější něz originální data, ale přesto může sloužit jako referenční snímek pro následnou registraci zpracovávaného snímku. Registrací obrazu

rozumíme nalezení geometrických transformací mezi souřadnicemi jednotlivých snímků a zvoleného referenčního snímku.

Ve svém článku [4] dále Shimizu a kol. navrhli vylepšenou metodu nerigidní registrace založenou na B-splajnech pro odhad turbulentního rozostření. Představili také stabilizační člen vložený do váhové funkce a prohlásili, že tento člen umožňuje přesnější odhad v oblastech s menší strukturou. Tento stabilizační člen mění hodnoty deformačních parametrů v závislosti na lokálních gradientech obrazu. Ale atmosférická turbulence je ve skutečnosti nezávislá na snímaném obsahu, tudíž tento přístup může mít za následek nechtěnou odchylku přidanou k odhadu [6].

Hlavním problémem těchto algoritmů je to, že neodhadují skutečnou rozptylovou funkci. V přístupech popsaných v článcích [4] a [5] je používán pevný gaussovský model pro aproximaci rozptylové funkce, což značně limituje jejich výkon [3].

2.1.2 Metody výběru snímků

Technika výběru snímků hledá snímky s nejvyšší kvalitou („šťastné snímky“) ze sekvence snímků s krátkou expozicí. Výstupní obraz je tvořen složením těchto šťastných snímků dohromady. Tato metoda je založena na pozorování, že pro snímky s krátkou expozicí vytváří turbulence mutace v obrazové kvalitě a náhodně tvoří dostatečně ostré snímky [6]. Tato strategie je oblíbená v mnoha astronomických aplikacích, kde je objekt zájmu snímán v isoplanatickém úhlu, což je velmi malý úhel , v kterém je možné nahlížet na rozmazání obrazu jako prostorově neměnné [6]. Problém tohoto přístupu je ve velmi malé pravděpodobnosti výskytu takovýchto šťastných snímků [5].

Modifikací tohoto přístupu je hledání „šťastných oblastí“ v anisoplanatických záběrech. Malé (isoplanatické) „šťastné oblasti“ jsou detekovány pomocí metriky lokální ostrosti a dále jsou skládány do celku tvořícího vysoce kvalitní obraz o původní velikosti [5, 6]. Potíž s touto metodou je opět nízká pravděpodobnost výskytu takovýchto oblastí. I když se vysoce kvalitní oblasti vyskytují mnohem častěji než vysoce kvalitní snímky, tak tato metoda vyžaduje velké počty (běžně stovky) snímků pro vygenerování jednoho ostrého snímku [5].

Díky isoplanatismu turbulentních efektů v malých oblastech se metody „šťastných oblastí“ soustředí na obnovu původního obrazu pomocí malých vzorků. Nevýhodou je, že omezená velikost těchto vzorků omezuje také jejich výkon [3]. Odhad rozptylové funkce může být výrazně zlepšen s větší velikostí vzorků. Jinak řečeno, když budeme schopni odhadnout rozptylovou funkci z velkých oblastí (třeba celého snímku), budeme schopni dosáhnout vyššího výkonu. K dosažení tohoto cíle je však potřeba redukovat prostorovou změnu v rozptylové funkci [3].

Další důležitý faktor který může ovlivnit obnovu obrazu je šum snímače. Krátký expoziční čas je hojně využívaný v systémech dálkového snímání obrazu pro vyvarování se pohybovému rozmazání způsobenému turbulencí, ale má neodvratný vliv na zvýšení úrovně šumu. Šum může výrazně ovlivnit výkon „deblurringu“, protože zkresluje detaily snímku

a může vytvářet „duchy“ v obraze. Naštěstí jsou snímky v sekvenci mezi sebou vzájemně vysoce korelované a takovou korelaci je možné využít ku prospěchu při redukci šumu v časové doméně [3].

2.1.3 Metoda prostorově nezávislé dekonvoluce

V článku [3] je navrhnut nový přístup k redukci prostorové změny rozptylové funkce skrz celý snímek a rozmazání obrazu tedy může být zhruba bráno jako časově nezávislé a skrytý obrazový obsah může být odhadnut globálně místo lokální dekonvoluce. Tímto lze vylepšit přesnost odhadu a difrakčně limitované rozmazání může být efektivně odstraněno. Snímek s redukovanou proměnlivostí rozptylové funkce (zde nazáván jako „téměř difrakčně limitovaný snímek“) je generován skrze fúzní proces. Pro vyvarování se šumových efektů v následné dekonvoluci je při redukci šumu použita temporální jádrová regrese.

Postup dle [3] spočívá ve třech krocích. V prvním kroku je každý zpracovávaný snímek registrován na pevnou referenční mřížku, čímž je generována nová sekvence snímků bez geometrické deformace. Tento proces zajišťuje, že odpovídající pixely jsou v čase zarovnány na stejných pozicích. Tento krok také zlepšuje výkon následných časových filtrací.

Druhý krok vrací jeden samostatný snímek vytvořený ze zkoumané sekvence. Pro každou lokální oblast je detekován nejostřejší vzorek ze sekvence vzorků. Dále je provedena temporální jádrová regrese v rámci vzorku, čímž je redukována úroveň šumu. Výsledný snímek je vytvořen složením všech vzorků dohromady. Tento obrázek je zatím ještě rozmazaný difrakčně limitovanou rozptylovou funkcí, kterou je možné považovat za prostorově nezávislou.

Konečně, ve třetím kroku je proveden algoritmus slepé dekonvoluce založený na přirozených statistikách obrazu. Tímto je odstraněno difrakčně limitované rozmazání a je zvýšena výsledný kvalita snímku.

2.2 Registrace obrazu

Tato práce vychází především z článku Zhua a Milanfara [3] a je zaměřena hlavně na odstranění geometrické deformace způsobené atmosférickou turbulencí. Tudíž si nyní probereme úvodní část jejich algoritmu více do podrobna.

Už ve svém předchozím článku [5] představili Zhu a Milanfar registrační algoritmus založený na B-splajnech pro odhad pohybového pole v každém sledovaném snímku. Tato metoda může být implementována pro odstranění geometrické deformace. Metoda na vstupu očekává dva snímky. První představuje sledovaný zkreslený snímek a druhý je referenční snímek bez turbulentní deformace, který může být získán zprůměrováním sekvence snímků [4].

Pružný deformační model poté může být použitý pro popsání geometrického zkreslení mezi těmito dvěma snímky. Komplexní pohyb je v tomto modelu reprezentován pohybem několika kontrolních bodů, jejichž počáteční pozice jsou rovnoměrně rozmístěny

nad referenčním snímkem. Posun všech kontrolních bodů na sledovaném snímku je popsán deformačním vektorem, který obsahuje horizontální i vertikální posuny (také nazývány deformační parametry) oproti počátečním pozicím všech kontrolních bodů.

Deformovanou pozici kteréhokoliv pixelu z referenčního snímku lze získat přičtením pohybového vektoru k počáteční pozici pixelu. Pohybový vektor je lineární kombinací pohybu všech kontolních bodů a lze jej získat vynásobením deformačního vektoru s bázovou maticí. Váhy (neboli báze splajnů) jsou určeny vzdáleností mezi počáteční a deformovanou pozicí s použitím B-splajnových funkcí. Parametry těchto funkcí jsou horizontální a vertikální intervaly mezi jednotlivými kontrolními body. Tento model zajišťuje lokální hladkost turbulentního pohybového pole.

V klasické B-splajn registraci je deformační vektor odhadován minimalizací určité nákladové funkce. Protože taková funkce pozbývá nějaké počáteční omezení, je takový odhad nestabilní a může uváznout v lokálním minimu. Ke zlepšení této metody bylo v článku [4] představeno stabilizační omezení, které drží odhadnuté deformační parametry v oblastech s menší texturou (s malým gradientem obrazu) v nízkých hodnotách. Avšak deformace způsobená atmosférickou turbulencí je ve skutečnosti nezávislá na obsahu obrazu. Jinak řečeno, takové stabilizační omezení může do odhadu vnést nechtěnou odchylku.

Pro přesnější odhad deformačního vektoru z referenční mřížky bodů do sledovaného snímku představili Zhu a Milanfar [3] přirozenější symetrické omezení B-splajn registračního algoritmu. Toto omezení je založeno na důležité vlastnosti, že registrace by měly být symetrická neboli inverzně souhlasná. Toto omezení využívá dvou navzájem inverzních deformačních vektorů. První popisuje již zmíněnou deformaci bodů referenční mřížky do deformovaných pozic ve sledovaném snímku (dopředná deformace). Druhý deformační vektor popisuje zpětnou deformaci bodů ze sledovaného snímku do původních pozic v referenčním snímku (zpětná deformace). Tyto dva vektory by měly být alespoň přibližně inverzní, protože pixely v obraze neodpovídají přesně reálným objektům a tudíž se mohou odpovídající body obou obrazů mírně lišit. Kombinací těchto dvou vektorů lze získat onu nákladovou funkci, kterou je třeba minimalizovat. Jakmile je získán odhad deformačního vektoru pro sledovaný snímek, je možné vygenerovat registrovaný snímek pomocí bilineární interpolace.

2.3 Shrnutí

Odstranění atmosférické turbulence z videosekvence je netriviální problém. V této kapitole jsme si představili některé současné přístupy k řešení tohoto problému. Tyto přístupy budou využity při návrhu algoritmu v následující kapitole.

3 Návrh algoritmu

Algoritmus popisovaný v této kapitole je navržen pro zpracování videosekvencí v reálném čase. Pro zjednodušení budeme předpokládat, že na začátku výpočtu máme k dispozici celou sekvenci a realným časem budeme chápat, když výpočet algoritmu bude trvat stejný časový úsek jako je doba trvání sekvence. Toto může být zásadní problém, protože operace nad obrazovými daty jsou obecně poměrně výpočetně náročné. Navíc většina současných metod odstraňování atmosférické turbulence je založena na statistických metodách, které ke své práci vyžadují běžně stovky snímků [5].

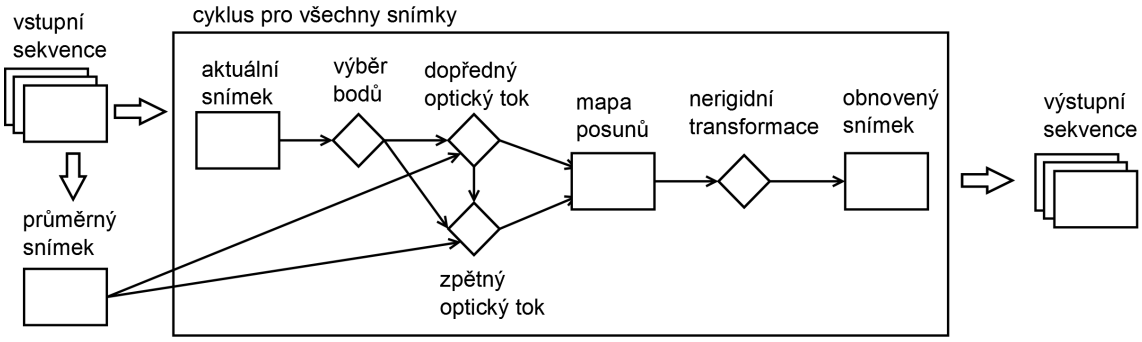
Navrhovaný algoritmus by měl být pokud možno robustní vzhledem k pohyblivým objektům. Pokud bychom uvažovali navíc možnost pohyblivé kamery, vznikl by další problém: snímky v sekvenci by byly mezi sebou navzájem posunuty. Toto by znemožňovalo použití statistických metod, protože pro určité oblasti záběru bychom neměli informace z celé sekvence snímků. Tuto možnost tedy nebudeme uvažovat.

Základní koncept navrhovaného algoritmu je znázorněn ve schématu na obrázku 3.1. Algoritmus očekává na vstupu nějaký uložený videosoubor. V první části jsou snímky zprůměrovány a je získán referenční snímek. Následně jsou snímky postupně za sebou zpracovány v hlavním cyklu algoritmu. Pro body referenční mřížky jsou optickým tokem získány odpovídající pozice v aktuálním snímku. Poté je z těchto nových pozic získána zpětným optickým tokem jejich původní pozice. Tímto jsou získány dva navzájem inverzní deformační vektory, z nichž je poté interpolací vytvořena mapa posunů. V posledním kroku je s využitím této mapy provedena nerigidní transformace snímku a ten je zbaven geometrických deformací způsobených atmosférickou turbulencí. Na konec jsou snímky poskládány zpět za sebe a uloženy. Výstupem algoritmu je videosekvence bez atmosférických turbulencí.

3.1 Potlačení atmosférické turbulence

Potlačení atmosférické turbulence je hlavním problémem této práce. Jelikož předpokládáme, že na začátku algoritmu máme k dispozici celou sekvenci snímků a snímaná scéna je statická, můžeme celkem snadno získat na začátku algoritmu referenční snímek. Hledání „šťastných“ snímků nebo oblastí by bylo oproti tomu náročnější.

Použijeme tedy metodu obnovy obrazu z více snímků. Tyto metody využívají pro odstranění geometrické deformace nějakou metodu nerigidní obrazové registrace. Nerigidní registrace spočívá v porovnání aktuálně zpracovávaného snímku s referenčním snímkem. Vznikne odhad deformační matice a na jejím základě se deformovaný snímek rekonstruuje. Referenční snímek je snímek bez geometrické deformace. Takový snímek je předem neznámý a získává se zprůměrováním celé sekvence [5]. I když je takovýto snímek značně rozmazaný, je



Obrázek 3.1: Schéma navrhovaného algoritmu

zněj možné poměrně dobře odhadnout hrubé tvary původních objektů v záběru. Běžně se k tomuto zprůměrování používají stovky snímků.

Jak uvádí Mao [7], počet použitých snímků určuje kvalitu rekonstrukce, tedy čím více snímků, tím lepší výsledek. Mao ve své práci dále ukázal, že uspokojující výsledky lze získat i při menším počtu snímků. Určité zlepšení kvality je znát už při použití pouhých 10 snímků a při použití 20 snímků je výsledná rekonstrukce docela úspěšná. Výsledek je ovšem závislý na stupni deformace, pokud je turbulence hodně divoká, výsledky rekonstrukce jsou horší.

Jak bylo uvedeno v druhé kapitole, turbulentní pohyb má charakter normálního rozložení. Na tomto základě je tento způsob získání referenčního snímku pro registraci postaven. Oproti tomu, pokud se v obraze vyskytuje nějaký pohyblivý objekt, tento pohyb má ve většině případů odlišný charakter a neovlivňuje výsledek registrace. Prakticky je toto dokázáno opět v [7], kde je registrací obnoven záběr pohybujícího se chodce. Kvalita výsledku závisí na rozdílu rychlosti pohybu chodce a rychlosti turbulentního pohybu. S tímto poznatkem by se nám mohlo podařit vytvořit algoritmus robustní vzhledem k pohyblivým objektům.

3.1.1 Výpočet referenčního snímku

Prvním krokem algoritmu je tedy vypočítání průměrného snímku pro jeho použití jako referenčního snímku pro obrazovou registraci.

Po načtení sekvence převedeme jednotlivé snímky do šedotónového spektra (výpočet s jedním kanálem intenzity je jednodušší než když je třeba počítat s třemi barevnými kanály, barevná informace obsažená ve snímku navíc není pro tento výpočet vůbec potřebná). Tyto šedotónové snímky jsou poté sečteny do jednoho snímku a hodnoty v jednotlivých pixelech jsou poté vyděleny celkovým počtem snímků v sekvenci. Tímto je získán prostý aritmetický průměr ze snímků sekvence.

V získaném referenčním snímku jsou poté rozmístěny v pevné mřížce kontrolní body. V těchto bodech jsou poté zjišťovány posuny obrazu a deformační vektory. Takto je algoritmus navžen podle článku Zhua a Milanfara [3]. Tento přístup má však jeden velký nedostatek. Body

v pevné mřížce mohou být umístěny do oblasti snímku s malou nebo žádnou texturou (jednolité plochy). Následné hledání posunů v těchto bodech je obtížné, nebo může úplně selhat.

Tento problém může být eliminován detekcí takzvaných význačných bodů obrazu. Je pouze potřeba otočit postup hledání deformačních vektorů. Referenční smímek totiž trpí nedostatkem ostrosti, který vyplývá ze způsobu jakým byl získán. V takovém snímku může být detekce význačných bodů obtížná. Tyto body je tedy třeba detekovat v aktuálním snímku a jejich posuny hledat v referenčním (a poté nazpět). V tomto se může zdát výhodnější použití bodů pevné mřížky, které stačí rozmístit na referenční snímek jednou na začátku algoritmu. Avšak pravděpodobnost, že body pevné mřížky budou dobře trasovatelné, nemusí být velká (záleží na hustotě mřížky a samotném obrazovém obsahu).

3.1.2 Nalezení význačných bodů

Aby bylo možné hledat poposuny v obraze, je nutné aby obraz obsahoval rozlišitelné objekty, jejich části, nebo struktury s jednoduchými opakujícími se vzory [8]. Naštěstí, ve většině reálných záběrů se dá toto předpokládat. Některé části snímku jsou pro rozpoznávání pohybu vhodnější než jiné. Jsou to ty části, které se výrazně liší od svého okolí, jako například rohy objektů nebo různé výstupky. Tyto body se nazývají význačné body a jejich hledáním se zabývají takzvané rohové detektory. Správným výběrem význačných bodů lze zlepšit efektivitu následných algoritmů [8].

Algoritmy pro nalezení význačných bodů vychází z myšlenky hledání rohů v obrázku a následné fitraci pro vyloučení málo významných bodů. Jedním ze základních algoritmů je Harrisův-Stephensův detektor rohů, publikovaný v roce 1988. Tento algoritmus zkoumá okolí bodu pomocí okénkové funkce a sleduje změny v intenzitě světla při posunu ve všech směrech. Dále využívá hodnotící funkci, z jejíhož výsledku se dá určit, zda se v okolí nachází roh [9].

V roce 1994 publikovali J. Shi a C. Tomasi algoritmus nazvaný „Good features to track“, který vychází z Harrisova-Stephensova detektoru. Tento algoritmus hledá dostatečný počet dostatečně významných bodů vhodných k trasování pohybu mezi dvěma snímky. Oproti Harrisovu-Stephensovu detektoru výrazně zjednodušuje výpočet hodnotící funkce. Rohy jsou poté vybrány na základě kvalitativního prahu počítaného vzhledem k nejvýznamějšímu rohu. Dále jsou rohy porovnávány podle minimální eukleidovské vzdálenosti. Pokud jsou některé rohy příliš blízko sebe je zachován pouze ten významější [10].

Když máme detekovány význačné body v aktuálním snímku, je možné začít hledat posuny těchto bodů oproti referenčnímu snímku. Význačné body obrazu a jejich okolí jsou analyzovány a jsou hledány odpovídající pozice těchto bodů v referenčním snímku. Výsledkem tohoto kroku je deformační vektor, neboli zjištěné pozice pro každý význačný bod, ke kterému se našel odpovídající bod z referenčního snímku [8].

3.1.3 Optický tok

Hledání pohybu mezi nějakými dvěma snímky je typický problém, který řeší algoritmy optického toku. Tyto algoritmy se dělí na dvě kategorie: hustý optický tok a řídký optický tok. Hustý optický tok se počítá pro všechny body v obraze. Zde je však problém s počítáním toku na jednolitých plochách, což se dá řešit interpolací s pomocí bodů, pro které lze tok vypočítat snadněji. I tak ale řešení nemusí být uspokojivé a výpočet je složitý a časově náročný. Příkladem tohoto algoritmu je metoda Horn-Schunck [8].

Naproti tomu řídký optický tok se počítá pouze pro určité body, které by měly být poměrně snadno sledovatelné. Postup nalezení těchto bodů jsme si představili v předcházející podkapitole. Důležitým faktem je, že řídký optický tok je většinou výrazně méně časově náročný oproti hustému optickému toku. Nejznámějším algoritmem tohoto typu je metoda Lucas-Kanade [8].

Algoritmus Lucas-Kanade využívá při výpočtu pouze malé okolí význačného bodu, a proto dokáže vypočítat pouze malé změny optického toku v mezích daného okolí. Tento nedostatek byl později vyřešen použitím pyramidové implementace, která dokáže zachytit mnohem větší změny optického toku. Algoritmus dále předpokládá, že intenzita bodů se skokově nemění a sousední pixely se pohybují stejným směrem [8,11].

Pyramidová implementace spočívá ve zmenšení snímku na poloviční rozměry a dále. Tímto vznikne několik vrstev obrázku o různé velikosti, na které se postupně aplikuje algoritmus Lucas-Kanade. Při zmenšení rozlišení obrazu a použití stejně velkého okolí je možné trasovat pohyby větší než velikost okolí v původním obrázku s plným rozlišením [8].

Takto vypočítáme optický tok jednou pro význačné body z aktuálního snímku a získáme pozice těchto bodů v referenčním snímku. Tyto nově získané body opět přivedeme na vstup algoritmu optického toku a pokusíme se je trasovat zpět do aktuálního snímku a měli bychom získat zpátky pozice význačných bodů. Tyto body poté porovnáme s původními význačnými body, a pokud se neliší, budeme s nimi dále pracovat, jinak je nebudeme uvažovat. Z nalezených bodů zjistíme odečtením původních pozic vzájemné posuny mezi nimi a z těchto posunů budeme konstruovat mapu posunů.

3.1.4 Nerigidní transformace

Pro nerigidní transformaci obrazu je třeba sestavit mapu posunů, která popisuje jak se má který pixel aktuálního snímku přesunout. Pokud máme počáteční body v pevné mřižce, je vytvoření mapy jednoduché. Stačí vzít posuny v bodech mřížky a body mezi nimi dopočítat lineární nebo bikubickou interpolací. Pokud však máme body rozmístěné nerovnoměrně po celém snímku, je tato interpolace složitější.

Možným řešením je rozdělit interpolaci na dva kroky. V prvním kroku rozmístíme nad obrázkem opět pevnou mřížku bodů a v těchto bodech vypočítámé průměrné hodnoty

posunů z význačných bodů v okolí. V druhém kroku už jen dopočteme body mimo mřížku opět lineární nebo bikubickou interpolací.

Otázkou zůstává jak budeme počítat průměrné hodnoty posunů z okolních bodů. Prostý aritmetický průměr není příliš vhodný. Lepší je použít vážený aritmetický průměr, kde váha jednotlivých průměrovaných bodů je definována jako jeho vzdálenost od daného bodu mřížky.

Když máme sestavenou mapu posunů jednotlivých pixelů stačí už jen provést samotnou transformaci. Po provedení obrazové registrace získáme snímek pokud možno bez geometrických deformací. Tyto deformace však nemusí být odstraněny úplně, protože obraz může obsahovat špatně trasovatelné oblasti jako jednolité plochy, gradientní plochy nebo dlouhé rovné linie. Pokud jsou tyto oblasti dostatečně malé, nemusí způsobovat díky použité interpolaci problém. Jestliže tyto oblasti budou větší, nemusí se podařit je správně transformovat a turbulentní zkreslení v obraze zůstane. Vzhledem k tomuto faktu je další zpracování snímku, za účelem zlepšení obrazové kvality, obtížné a výsledný efekt nemusí být znatelný. Registrovaný smímek navíc stále obsahuje určité rozostření. Toto rozostření se můžeme případně pokusit odstranit pomocí nějakého algoritmu „deblurringu“.

3.2 Shrnutí

V této kapitole jsme navrhli algoritmus pro odstranění atmosférické turbulence. Tento algoritmus je navržen s ohledem na robustnost vůči lokálnímu pohybu objektů v záběru. Implementací tohoto algoritmu, s pomocí knihovny OpenCV, se budeme zabývat v následující kapitole, a potom v páté kapitole se pokusíme algoritmus optimalizovat pro zrychlení jeho výpočtu, aby běžel v reálném čase.

4 Implementace

Algoritmus bude implementován v jazyce C++ s pomocí knihovny OpenCV (Open Source Computer Vision Library – otevřená knihovna pro počítačové vidění) ve verzi 2.x. Tato knihovna obsahuje velké množství optimalizovaných algoritmů z oblasti počítačového viděni, které je možné přímo použít ve vyvíjených aplikacích. Těchto algoritmů se tedy budeme snažit využít i v naší aplikaci. Implementace bude probíhat na platformě Windows a bude využito vývojové prostředí Microsoft Visual Studio 2012.

4.1 Knihovna OpenCV

Knihovna OpenCV je šířena pod BSD licencí a tudíž je její použití zdarma jak pro akademické tak i komerční účely. Obsahuje rozhraní pro jazyky C/C++, Python a Java, a podporuje platformy Windows, Linux, Mac OS, iOS i Android. Tato knihovna byla navržena pro efektivitu výpočtu se silným zaměřením na použití v aplikacích běžících v reálném čase. Knihovna je napsána v optimalizovaném C/C++ a díky tomu může využívat výhod vícejádrového zpracování. S povoleným OpenCL frameworkem může také využít dalších výhod hardwarové akcelerace na nižších vrstvách heterogeních výpočetních platforem. Knihovna je využívána v mnoha různých odvětvích od interaktivního umění, inspekce dolů, spojování webových map až k pokročilé robotice [16].

4.2 Struktura programu

Programová struktura je poměrně jednoduchá. Program obsahuje pouze jednu třídu, která zprostředkovává hlavní funkcionalitu navrženého algoritmu a další pomocné funkce, například pro načítání a ukládání videosouborů.

`class` **`TurbulenceRemove`**

Třída implementující načítání, zpracování a ukládání videosekvencí deformovaných atmosférickou turbulencí.

Privátní atributy třídy **`TurbulenceRemove`**:

`int imgCount` – počet snímků v sekvenci,
`double fps` – počet snímků za vteřinu,
`vector<Mat> originalSequence` – vektor jednotlivých snímků sekvence,
`vector<Mat> restoredSequence` – vektor upravených snímků,
`Mat averageImage` – průměrný snímek.

Veřejné metody třídy **TurbulenceRemove**:

TurbulenceRemove()

Konstruktor.

bool **loadSequence**(const string **fileName**)

Načte zadanou videosekvenci. Pokud se nepovede otevřít soubor, tak vrací chybu.

filename – název vstupního souboru.

bool **saveSequence**(const string **fileName**)

Uloží zpracovanou videosekvenci do určeného souboru.

filename – název výstupního souboru.

void **processSequence**(size_t **step**, double **e** = 0.5)

Zpracuje načtenou videosekvenci podle [3] s použitím pevné mřížky bodů.

step – rozteč bodů mřížky; je stejná v obou osách,

e – maximální povolená odchylka při počitáni zpětného optického toku.

void **processSequenceF**(size_t **step**, double **e** = 0.5)

Zpracuje načtenou videosekvenci podle navrženého vylepšení s využitím detekce význačných bodů a jejich interpolace do mřížky.

step – rozteč bodů mřížky; je stejná v obou osách,

e – maximální povolená odchylka při počitáni zpětného optického toku.

void **clear**()

Uvolní paměť alokovanou pro uložení videosekvencí.

4.3 Popis implementace

V hlavní funkci programu je jasně zřetelný tok programu. Na začátku vytvoříme instanci třídy `TurbulenceRemove` a zavoláme její metodu `loadSequence()` s názvem souboru který chceme zpracovat.

Načtenou sekvenci zpracujeme zavoláním metody `processSequenceF()` s patřičnými parametry. Písmeno F v jejím názvu značí, že metoda využívá detekce význačných bodů – anglicky „features“. Toto volání způsobí zpracování podle námi navržené metody. Pro zpracování podle originální metody Zhua a Milanfara [3] je možné zavolat metodu `loadSequence()`.

Když máme sekvenci zpracovanou zbývá ji jen uložit. K tomuto slouží metoda `saveSequence()`, jejíž jediný parametr je název pod jakým chceme opravenou sekvenci uložit.

4.3.1 Načtení videosekvence

Pro načítání videosekvence použijeme třídu `VideoCapture`. Tato třída získává snímky ze souboru po jednom, tudíž je nutné soubor načítat v cyklu. Cyklus ukončíme, když metoda `read()` vrátí hodnotu `false` jako signalizaci konce videosouboru. Jednotlivé snímky jsou postupně vkládány metodou `push_back()` do vektoru `originalSequence` třídy `TurbulenceRemove`. Ve stejném cyklu je přímo možné počítat referenční snímek.

Pro výpočet referenčního snímku si vytvoříme ještě před započetím cyklu novou strukturu `Mat` typu `CV_32S` o stejné velikosti jako mají snímky videa a naplníme ji nulovými hodnotami. Do této matice budeme postupně přičítat snimky sekvence. Načítané snímky mohou být barevné a tak je musíme převést funkcí `cvtColor()` do šedotónového spektra. Pixely ve snímku jsou uložené jako osmibitové hodnoty intenzity světla. Aby se nám součet snímků vešel do rozsahu datového typu, je třeba ještě snímek převést funkcí `convertTo()` na větší datový typ `CV_32S`. Nyní můžeme snímek přičíst k předchozím.

Po skončení cyklu načítání zjistíme počet prvků ve vektoru `originalSequence` a uložíme jej jako počet snímků sekvence. Touto hodnotou podělíme pixely akumulovaného snímku a získáme referenční (průměrný) snímek. Referenční snímek na závěr převedeme zpět na původní datový typ `CV_8U`.

Při načítání snímku je možné uložit kromě originální sekvence snímků taktéž sekvenci šedotónovýh snímků, ale pro úsporu paměti je ukládána pouze originální sekvence. Při zpracovávání jsou poté snímky převáděny znovu do šedotónového spektra. Tento převod však není příliš náročný a tak nám toto nevadí.

4.3.2 Odstranění turbulence ze sekvence

Odstranění turbulence ze sekvence probíhá v cyklu skze všechny snímky. Pro každý snímek detekujeme význačné body a tyto body poté trasujeme optickým tokem do referenčního snímku. Trasování provedeme pro kontrolu ještě zpětně. Získané pozice bodů porovnáme s původními pozicemi význačných bodů. Pro body, které jsou shodné, vypočteme posuny oproti referenčnímu snímku. Při použití pevné mřížky bychom z posunů vytvořili malou mapu posunů a tu poté jednoduše zvětšili a funkce `resize()` by provedla potřebnou interpolaci. V našem případě však máme body rozmístěny nerovnoměrně ve snímku. Posuny těchto bodů musíme nejprve „ručně“ interpolovat do bodů pevné mřížky. Poté teprve použijeme funkci `resize()` a posuny v bodech mřížky bikubicky interpolujeme a získáme mapu posunů. Pomocí ní poté transformujeme originální deformovaný snímek.

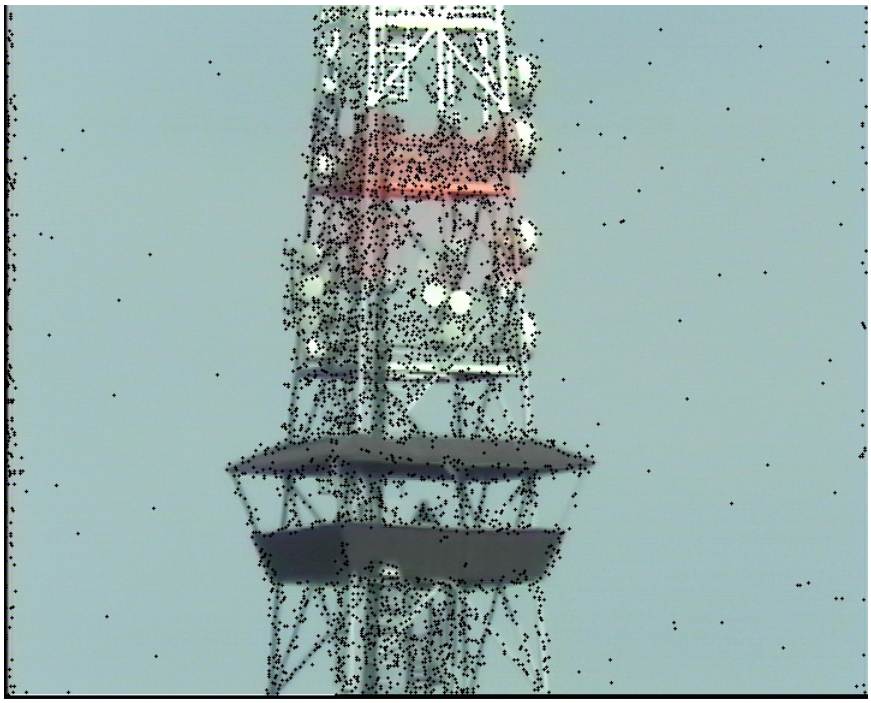
Před samotným cyklem je třeba definovat struktury, které budeme během výpočtů používat. Definujeme tedy několik vektorů pro práci s detekovanými a trasovanými body. Jsou to vektory bodů typu `Point2f` pro uložení detekovaných význačných bodů, bodů trasovaných do referenčního snímku a bodů zpětně trasovaných do aktuálního snímku. Pro uchování stavu

trasování optickým tokem potřebujeme dále dva vektory s prvky typu `uchar` a vektor prvků `float` pro uložení chyby trasování.

Dále potřebujeme mřížku bodů v kterých budeme interpolovat posuny obrazu. Tyto body získáme průchodem rozměrů snímku se zadaným krokem a uložíme si je do vektoru. Pro interpolaci v těchto bodech budeme dále potřebovat jádro s vahami bodů okolí. Okolí bude mít velikost zadaného kroku ve všech směrech. Vytvoříme proto čtvercovou matici o velikosti strany `(step*2)+1` pixelů. Tuto matici naplníme vahami. Váhy jednotlivých pixelů spočítáme jako převrácenou hodnotu vzdálenosti od středu jádra. Váhu ještě znormujeme vydělením hodnotou maximální vzdálenosti.

Jako poslední věc budeme potřebovat mapu souřadnic. Mapa posunů, kterou později během výpočtu získáme, totiž obsahuje posuny relativní k jednotlivým pixelům, ale funkce `remap()` počítá s posuny relativními k bodu `(0, 0)`. Zkostruujeme tedy mapu souřadnic, kterou poté sečteme s mapou posunů a měli bychom získat potřebný výsledek.

Když máme vše potřebné připraveno, můžeme spustit cyklus zpracováni jednotlivých snímků. V cyklu nejdříve získáme aktuální snímek sekvence a převedeme ho do odstínů šedi. V takovém snímku detekujeme význačné body funkci `GoodFeaturesToTrack()`. Funkci předáme také vektor do kterého má uložit detekované body a nastavíme parametry detekce. Vhodné parametry byly vybrány experimentálně. Postup jejich hledání je popsán v následující kapitole. Výsledek detekce pro snímek z videa „věž“ je možné vidět na obrázku 4.1.



Obrázek 4.1: Význačné body

Nyní přistoupíme k trasování bodů optickým tokem. Funkci `calcOpticalFlowPyrLK()` předáme aktuální a referenční snímek, vektor detekovaných význačných bodů, vektor pro uložení nalezených posunutých bodů a stavový a chybový vektor. Pro výpočet zpětného optického toku prohodíme pořadí snímků a vektor nalezených bodů předáme jako vstupní body a nově nalezené body uložíme do jiného vektoru. Stejně tak uložíme stavové hodnoty do jiného vektoru. Informace z chybového vektoru nepotřebujeme a tak můžeme použít opět stejný vektor.

Dále je třeba vybrat správné význačné body pro interpolaci. Bod musí být nalezen dopředným i zpětným optickým tokem a pozice bodu získaná zpětným tokem se nesmí příliš lišit od původní pozice význačného bodu. Pro vybrané body vypočteme posun jako rozdíl původní pozice a pozice získané dopředným tokem. Tyto posuny si uložíme do řídké mapy posunů. Vybrané body pro snímek z videa „věž“ je možné vidět na obrázku 4.2.

Obrázek 4.2: Optický tok, posuny bodů jsou vyznačeny modře.

Z řídké mapy posunů budeme interpolovat posuny v bodech mřížky. Pro interpolaci použijeme vážený aritmetický průměr. Metoda je inspirovaná Gaussovým filterem, ale výpočet neprovádíme na celém snímku, pouze v daných bodech pevné mřížky. Pro každý bod mřížky vynásobíme pixely jeho okolí s předem vytvořeným jádrem vah a dílčí součiny sečteme. Výsledný součet pak podělíme sumou vah nenulových pixelů. Tímto získáme vážený průměr posunů v okolí bodu mřížky a jeho hodnotu uložíme jako posun v bodě mřížky. Z posunů

v bodech mřížky sestavíme malou mapu posunů, která obsahuje pouze body této mřížky. Abychom redukovali možné nedostatky vzniklé malým počtem význačných bodů v okolí, tak mapu lehce vyhladíme mediánovým filterem. Výslednou mřížku posunů pro video „věž“ lze vidět na obrázku 4.3.

Tuto malou mapu posunů zvětšíme funkcí `resize()` na plnou velikost aktuálního snímku. Funkce `resize()` se postará o interpolaci chybějících bodů. Tím získáme velkou mapu posunů. Tuto mapu sečteme s předem vytvořenou mapou souřadnic a výsledkem bude mapa pozic pro posun odpovídajících pixelů.

Jiný způsob výpočtu mapy posunů by mohl využívat triangulaci význačných bodů a interpolovat hodnoty ve vzniklých trojúhelnících pomocí barycentrických souřadnic. Problém je, že tyto operace jsou výpočetně náročné a v případě barycentrické interpolace tato není v OpenCV vůbec implementována, takže její použití by bylo obtížné.

Na závěr provedeme nerigidní transformaci originálního barevného snímku podle vytvořené mapy. Tím získáme registrovaný snímek, který uložíme do vektoru `restoredSequence` třídy `TurbulenceRemove`.

Obrázek 4.3: Mřížka posunů, posuny jsou vyznačeny modře.

4.3.3 Nastavení detekce význačných bodů

V algoritmu je použita funkce `goodFeaturesToTrack()` pro detekci význačných bodů. U této funkce je důležité správně zvolit její parametry, protože na tom, jaké body získáme, závisí výsledná kvalita rekonstrukce. Povinnými parametry funkce jsou: maximální počet detekovaných bodů, úroveň kvality bodů a minimální vzdálenost bodů. Funkce má také další parametry, ale ty nejsou povinné a můžeme jim ponechat standardní hodnoty.

Maximální počet detekovaných není nastavován explicitně, ale raději jako určitá část celkového počtu pixelů ve snímku. Pokud bychom nastavili tuto hranici příliš nízko, detekovali bychom sice kvalitní body v krátkém čase, ale pro zjištění turbulentní deformace v rámci celého snímku by jich nebyl dostatečný počet. Naopak příliš vysoká hranice způsobí dlouhý výpočet a detekování i málo kvalitních bodů, které nebude schopen optický tok trasovat. Po chvíli experimentování jsme zvolili hodnotu 1/100 celkového počtu pixelů v obraze.

Druhým důležitým parametrem je úroveň kvality bodů. Body jsou totiž algoritmem vybírány podle jejich kvality vzhledem k bodu s nejvyšší kvalitou a úroveň kvality určuje spodní hranici pro tento výběr. Úroveň kvality je třeba nastavit tak, aby byl dostatečně využit limit maximálního počtu bodů, jinak bychom mohli získat příliš nízký počet bodů. Po sérii dalších experimentů jsme zvolili hodnotu 0,0001. Pro reálná videa by sice stačilo nastavit vyšší kvalitu, ale pro umělá videa poté nastával problém s nedostatkem detekovaných bodů.

Posledním povinným parametrem je minimální eukleidovská vzdálenost mezi body. Tento prametr pro nás není až tak důležitý a tak byla jeho hodnota nastavena na 1, pouze pro zajištění minimálního rozestupu bodů. Později během testtování na GPU však bylo zjištěno, že tato funkce se téměř vůbec nezrychlí. Způsobovalo to právě nastavení minimální vzdálenosti mezi body na hodnotu 1. Toto vyžadovalo počítání vzdálenosti mezi detekovanými body a GPU nedokázala tuto operaci dostatečně urychlit. Protože nám jde o rychlost algoritmu, nastavili jsme minimální vzdálenost na 0 abychom zamezili počítání vzdálenosti a celý algoritmus se zrychlil. Toto může mít za následek detekci bodů těsně vedle sebe. V reálných záběrech je však výskyt takovýchto blízkých bodů nepravděpodobný a nebude nám vadit.

4.3.4 Uložení sekvence

K ukládání videosekvence použijeme třídu `VideoWriter`, která poskytuje veškeré potřebné rozhraní k zápisu videosouboru. Podobně jako načítání sekvence i ukládání probíhá v cyklu po jednotlivých snímcích.

Nejdříve otevřeme soubor pro zápis. K tomu slouží funkce `open()`, které předáme název souboru, určíme kodek pro kompresi snímků, frekvenci snímků a jejich velikost. V cyklu poté projdeme vektor obnovených snímků `restoredSequence` a jednotlivé snímky zapíšeme pomocí funkce `write()` do souboru.

4.4 Shrnutí

V této kapitole jsme si představili knihovnu OpenCV, kterou jsme využili při implementaci programu. Poté jsme popsali samotnou implementaci algoritmu a jeho jednotlivých kroků. V následující kapitole se pokusíme algoritmus optimalizovat pro běh v reálném čase.

5 Optimalizace

Optimalizaci algoritmu provedeme jeho implementací na GPU grafické karty. GPU v dnešních grafických kartách totiž v mnoha ohledech předčí výkon klasických procesorů. Výkonem GPU nelze plně nahradit operace, které provádí CPU, lze však některé operace naprogramovat tak, že využívají právě potenciálu GPU. Pro výpočty na GPU existuje několik různých technologií. Nejznámějšími jsou technologie CUDA a OpenCL. CUDA je proprietární technologie společnosti NVidia a je podporována pouze na jejich grafických kartách [20]. OpenCL je naproti tomu otevřený standard spravovaný konsorciem Khronos [21], je implementován na různých typech zařízení (CPU, GPU, FPGA nebo ARM), a je podporován všemi hlavními výrobci GPU i CPU.

Knihovna OpenCV již v základu obsahuje modul jak pro technologii CUDA (modul gpu) tak i pro technologii OpenCL (modul ocl) [17]. Protože testování našeho programu bude probíhat na platformě AMD využijeme pro výpočty modul ocl.

5.1 Modul ocl knihovny OpenCV

OpenCV modul ocl obsahuje soubor tříd a funkcí, které implementují a akcelerují funkcionalitu OpenCV na zařízeních kompatibilních se standardem OpenCL. Modul je vyvíjen a testován převážně na grafických kartách od výrobců Nvidia a AMD, ale také na integrovaných grafických čipech AMD APU a Intel HD. Výkon implementovaných algoritmů záleží na daném zařízení a jeho schopnostech. Avšak jejich přesnost a matematická správnost byla na všech testovaných GPU verifikována jako identická s CPU implementací [22].

Modul ocl obsahuje užitkové funkce a nízkoúrovňová primitiva, které poskytují výkonou infrastrukturu pro vývoj vizuálních algoritmů. Modul dále obsahuje algoritmy vyšší úrovně, poskytující funkcinalitu připravenou pro přímé použití vývojáři (například optický tok nebo detekce obličejů).

Modul ocl je navržen pro jednoduché používání a nevyžaduje žádnou znalost standardu OpenCL. Na základní úrovni může být modul ocl vnímán jako soubor urychlovačů, které dokážou využít vysokého výpočetního výkonu, který GPU zařízení nabízejí. Tohoto konceptu využijeme při optimalizaci našeho algoritmu.

5.2 Optimalizace algoritmu

Aby bylo možné algoritmus implementovat na grafické kartě, musí nainstalovaná grafická karta podporovat technologii OpenCL. Pokud ji podporuje, je možné další kroky algoritmu jednoduše přepsat pomocí funkcí z modulu ocl knihovny OpenCV. Vstupní snímky je třeba nahrát funkcí `oclMat::upload()` do paměti GPU a na konci algoritmu získat obnovené snímky zase zpět do operační paměti funkcí `oclMat::download()`.

K ověření podpory OpenCL implementujeme ve třídě `TurbulenceRemove` metodu `testOCL()`. Tato metoda interně volá funkci `ocl::getOpenCLDevices()`, která nalezne všechna instalovaná zařízení GPU podporující OpenCL. Pokud je nějaké zažízení nalezeno nastavíme na něj výpočetní kontext funkcí `ocl::setDevice()` a vrátíme `true`. Pokud není žádné zařízení nalezeno tak vrátíme `false`.

Metody pro načítání a ukládání videosekvencí zůstávají nezměněny. Úprav se dočká pouze metoda pro zpracování sekvence `processSequenceF()`. Pro účely testování nebudeme upravovat přímo tuto metodu, ale vyvoříme její kopii nazvanou `processSequenceG()` a v ní budeme provádět úpravy. Písmeno G v jejím názvu značí, že k výpočtům používá GPU.

V modulu ocl jsou detekce význačných bodů a výpočet optického toku implementovány jako třídy, tudíž je třeba před započetím cyklu vytvořit jejich instance. Pro detektor `ocl::GoodFeaturesToTrackDetector_OCL` navíc nastavíme potřebné parametry detekce stejně jako u funkce `goodFeaturesToTrack()`. Funkce z modulu ocl vyžadují vstupní snímky jako struktury oclMat uložené v paměti GPU a stejné struktury také vracejí, tudíž je nutné definovat potřebné struktury oclMat pro uložení snímků a detekovaných bodů. Funkcí `ocl::upload()` nahrajeme referenční snímek do paměti GPU a můžeme spustit cyklus.

Na začátku cyklu vezmeme aktuální snímek a nahrajeme ho do GPU, a funkcí `ocl::cvtColor()` jej převedeme do odstínů šedi. Poté zavoláme operátor kulatých závorek `()` detektoru význačných bodů a předáme mu aktuální snímek a odkaz na strukturu `oclMat`, do které má uložit detekované body. Tyto body předáme metodě `ocl::PyrLKOpticalFlow::sparse()` spolu s aktuálním snímkem, referenčním snímkem a strukturami pro uložení trasovaných bodů a stavu. Tímto máme zjištěný deformační vektor pro aktuální snímek. Ještě jednou zavoláme tuto funkci s prohozenými snímky, abychom získali zpětný deformační vektor.

Další operace budeme provádět zpět na CPU a tak potřebujeme výstupní struktury uložit zpátky do operační paměti. Tyto struktury jsou v paměti GPU uloženy jako jednořádkové matice. Po uložení do operační paměti potřebujeme jejich jednotlivé prvky (pro další použití) naskládat do kontejneru `std::vector`. Potřebujeme takto uložit vektor význačných bodů, vektor posunutých bodů, vektor zpětně posunutých bodů a oba stavové vektory z optického

toku. Naštěstí mají všechny tyto struktury stejnou velikost a můžeme je projít jedním společným cyklem.

Zbývající část algoritmu už zůstává nezměněna. Interpolaci posunů do mřížky by bylo výhodné akcelerovat, ale bohužel v knihovně OpenCV neexistuje funkce, která by náš problém řešila, a tak není možné tento „ruční“ výpočet akcelerovat na GPU. Bylo by možné například konvolvovat mapu posunů s definovaným jádrem vah pomocí funkce `Filter2D()`, ale protože mapa obsahuje pouze malý počet nenulových prvků, není dělitel váženého průměru konstantní hodnotou a získání korektního výsledku by bylo následně poměrně obtížné. Další použité funkce `resize()` a `remap()` sice na GPU akcelerovat jdou, ale jak bude popsáno v kapitole o testování, tyto operace nejsou příliš výpočetně náročné a dosažené zrychlení není, v rámci doby potřebné ke zpracovani jednoho snímku, příliš podstatné. Navíc by bylo nutné opět nahrávat snímky do paměti GPU a zpět, a tyto operace stojí také nějaký čas, který by vyvážil dosažené zrychlení.

5.3 Shrnutí

V této kapitole jsme si představili možnosti zrychlení algoritmu výpočtem na grafické kartě. Také jsme si představili modul ocl knihovny OpenCV, který obsahuje funkce pro akceleraci výpočtů na OpenCL kompatibilním hardwaru. Dále jsme popsali provedené optimalizace našeho algoritmu. V další kapitole o testování se podíváme jak se nám povedlo nebo nepovedlo algoritmus zrychlit.

6 Testování a experimenty

Cílem testování je zjistit, jak námi implementovaný algoritmus funguje a zda je po provedené optimalizaci schopen běhu v reálném čase. Implementovaný algoritmus byl testován na sadě reálných i uměle vygenerovaných videí. Testovacím systémem je stolní počítač s čtyřjádrovým procesorem AMD Athlon II X4 640 taktovaným na frekvenci 3000 MHz a grafickou kartou AMD Radeon R7 240 s taktem jádra 900 Mhz, podporou OpenCL 1.2 a 320 unifikovanými shadery.

Pro testování byla použita různá reálná i uměle vygenerovaná videa. Ukázky videí je možné vidět na obrázcích 6.1 až 6.6, technické parametry videí jsou uvedeny v tabulce 6.1. Celá videa jsou uložena na přiloženém DVD. Videa „věž“ a „auto“ poskytnul k použití pan Ing. David Herman. Záběry měsíce „moon1“ a „moon2 byly nalezeny na serveru youtube.com [23, 24]. Videa „tvary_1“ a „tvary_2“ byly vytvořeny uměle v programu Adobe AfterEffects použitím efektu „Turbulent displace“.

6.1 Průběh testování

Testování probíhalo postupně již během implementace. Během testování byly sledovány výsledky jednotlivých kroků algoritmu a vizuálně bylo kontrolováno, zda se algorimus chová korektně. Protože algoritmus má dva vstupní parametry: rozteč mřížky a tolerance chyby, bylo testováno jak se bude chovat při změně těchto parametrů. Tímto testováním bylo zjištěno, že pro toleranci chyby je vhodná hodnota 0,5. Vhodná hodnota rozteče mřížky zůstává předmětem experimentování.

Po dokončení implementace byly provedeny testovací experimenty. Jednotlivá videa byla zpracována nejdříve algoritmem podle Zhua a Milanfara [3] a následně námi navrženým algoritmem. Poté byl algoritmus optimalizován a byly provedeny nové experimenty. Algoritmus byl vždy spuštěn několikrát s různými hodnotami rozteče mřížky.

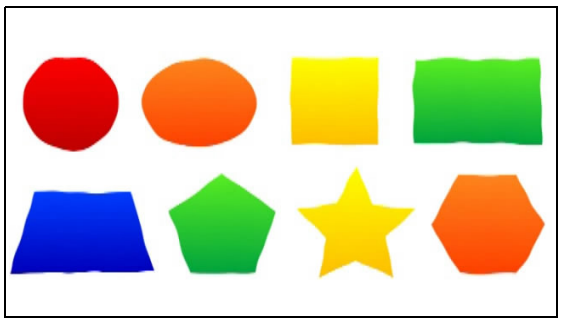
Během těchto experimentů byly měřeny doby trvání jednotlivých kroků algoritmu, doba zpracování jednoho snímku a počet použitých význačných bodů. Naměřené hodnoty byly poté průměrované pro celé video. Pro zhodnocení kvality provedených změn v obraze byla implementována jednoduchá metrika. Tato metrika počítá celkovou sumu rozdílných pixelů v obnoveném snímku oproti nedeformovanému snímku. Takovýto snímek bohužel nemáme k dispozici u reálných videí. Metrika je tedy počítána pouze u uměle vytvořených videí.

Obrázek 6.1: vez.avi

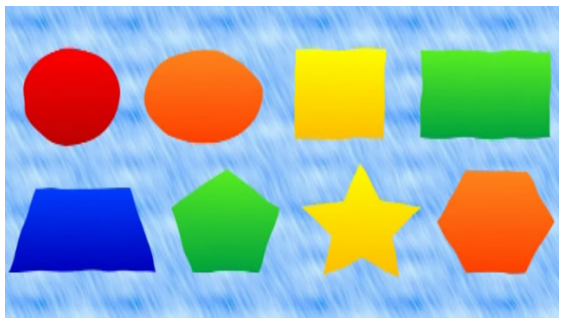
Obrázek 6.2: auto.avi

Obrázek 6.3: moon1.mp4

Obrázek 6.4: moon2.mp4



Obrázek 6.5: tvary_1.mp4



Obrázek 6.6: tvary_2.mp4

video	šířka	výška	doba trvání [s]	počet snímků	fps
vez.avi	720	576	0:12	156	15
auto.avi	768	576	0:18	272	15
moon1.mp4	640	360	0:27	683	25
moon2.mp4	640	360	0:20	510	25
tvary_1.mp4	640	360	0:30	750	25
tvary_2.mp4	640	360	0:15	375	25

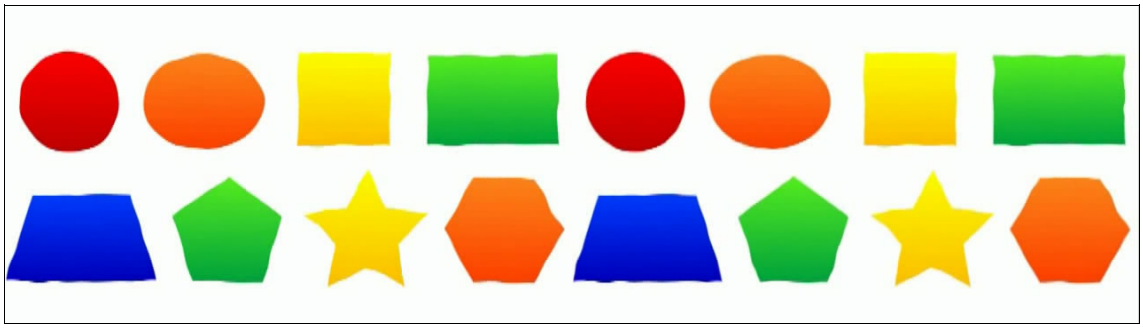
Tabulka 6.1: Technické parametry videí

6.2 Rozbor výsledků

Výsledky experimentů rozebereme postupně pro jednotlivá videa. Podíváme se na specifika jednotlivých videí a porovnáme výkon a výsledky tří verzí algoritmu: základní verze dle Zhua [3], námi navžená verze a optimalizovaná verze.

Umělá videa byla tvořena za účelem názornosti výsledného obrazu. Proto byly do prvního umělého videa umístěny jednoduché tvary. Do druhého videa bylo navíc přidáno texturové pozadí pro lepší detekci význačných bodů. Videa se záběry měsíce byla vybrána, aby bylo možné otestovat chování algoritmu na záběrech pořízených na velkou vzdálenost. Reálná videa „věž“ a „auto“ byla vybrána jako typické záběry zkreslené atmosférickou turbulencí. Obě videa navíc obsahují drobné pohybující se objekty a bude zajímavé sledovat zda a jak tyto objekty ovlivní výstup algoritmu.

Ukázky registrovaných videí je možné vidět na obrázcích 6.7 až 6.12,. Obrázky jsou složeny z registrovaného snímku a originálního snímku pro porovnání. Ukázky jsou však pouze ilustrativní, protože na takovýchto statických snímcích není možné jasně poznat provedené změny. Všechna registrovaná videa jsou však dostupné k prohlédnutí na přiloženém disku DVD.



Obrázek 6.7: Registrované video „tvary_1“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.

6.2.1 Kvalita výstupu

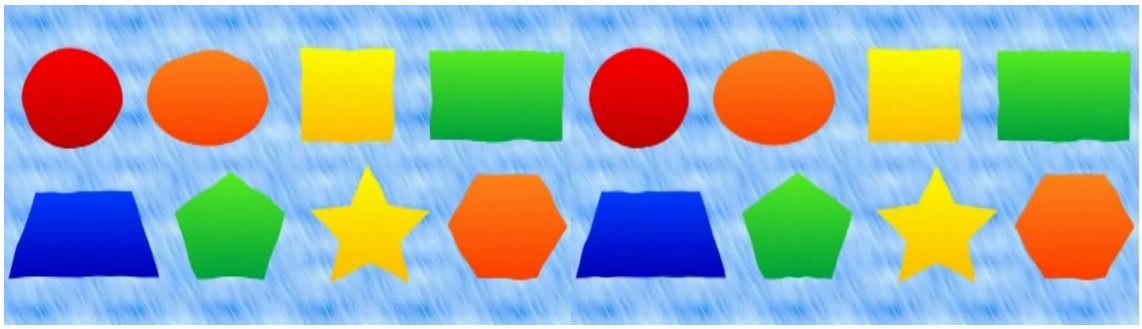
U hodnocení kvality výstupu nás bude hlavě zajímat rozdíl oproti ideálnímu snímku, tedy výsledek metriky. Metriky pro videa „tvary_1“ a „tvary_2“ jsou uvedeny v tabulkách 6.2 a 6.3. U videa „tvary_1“ je problém s nedostatkem bodů k trasování. Video totiž obsahuje příliš jednolitých ploch nebo rovných linií a málo rohů. Výsledkem je pouze malý počet význačných bodů, což má za následek pouze nepatrné zlepšení metriky. Ve videu je přesto možné vidět určitá zlepšení. Například u kruhu nebo oválu je jejich původní tvar téměř obnoven a rohy ostatních objektů jsou stabilizovány. Ve videu „tvary_2“ je tento problém eliminován díky strukturovanému pozadí. Rovné linie objektů však stále nejsou úplně obnoveny.

Výsledná metrika je počítána jako průměr hodnot ze všech snímků, proto může být toto číslo mírně zkreslené, ale přesto je z tabulek možné vyčíst několik informací. Pro základní algoritmus je jasná tendence zlepšování kvality se zhušťující se mřížkou, neboli čím je rozteč mřížky větší, tím je ve výsledném obraze obsaženo více chyb. Pro rozteč mřížky 14 bodů je dokonce algoritmus, již tak nepřesný, že je v něm více chyb než v originálu.

Když se podíváme na výsledky pro navržený algoritmus, je vidět, že na rozteči mřížky až tak nezáleží, dokud je menší než určitá mez. Pro rozteč mřížky 14 bodů je metrika totiž opět horší než originál, což může naznačovat, že správnou rozteč mřížky je třeba hledat v závislosti na úrovni turbulence. Celkově je ale metrika u druhého videa navrženým algoritmem zlepšena asi o 30-40 % což je poměrně slušný výsledek.

rozteč mřížky	orig	4	6	8	10	12	14
základní alg.	7447	7336	7343	7357	7361	7423	7870
navržený alg.	7447	7297	7271	7264	7260	7270	7834

Tabulka 6.2: Výsledky metriky pro video „tvary_1“. Sloupec orig udává výsledek metriky pro originální deformované video.



Obrázek 6.8: Registrované video „tvary_2“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.

rozteč mřížky	orig	4	6	8	10	12	14
základní alg.	10962	4759	4879	4866	5011	5361	13062
navržený alg.	10962	7027	7271	6278	6070	6509	11611

Tabulka 6.3: Výsledky metriky pro video „tvary_2“. Sloupec orig udává výsledek metriky pro originální deformované video.

Pro ostatní videa bohužel nemáme dostupný ideální snímek a není možné pro ně vypočítat metriku, ale z výstupních videí lze vyvodit stejné závěry jako pro videa umělá. Video „věž“ například obsahuje hodně rovných linií, tyto jsou však většinou různě přerušeny a nejsou tak zřetelné jako u umělých videí, což umožňuje detekci rohů a správnou registraci. Pouze linie u horní a spodní části věže nebo na bocích ochozu jsou výrazné a ve výsledném videu je u nich možné vidět občasné chvění.

U videí „auto“ a záběrů měsíce je možné vidět chvění způsobené nízkým počtem detekovaných bodů v oblastech s nízkou texturou. Ve videu auto je tento jev vidět zejména u stromů a keřů v pozadí. U záběrů měsíce je tento nepříznivý jev vidět hlavně v místech mimo krátery.

Jak již bylo zmíněno, ve videích „auto“ a „věž“ se vyskytují pohybující se objekty. Ve videu „věž“ jsou to lidé na ochozu. Jeden člověk pochází sem a tam, a druhý pouze stojí a mává rukou. Jelikož jsou oba pohyby poměrně rychlé, nejsou zaměnitelné s turbulentním pohybem a algoritmus nijak negativně neovlivňuje. Naproti tomu ve druhém videu „auto“ je zachycený člověk, vystupující z auta, poměrně výrazně deformován. Jeho pohyb je totiž pomalý a je tak snadno zaměnitelný s turbulentním pohybem a algoritmus se jej snaží redukovat.

Obrázek 6.9: Registrované video „věž“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.

6.2.2 Výkon algoritmu

Výkonem algoritmu rozumíme rychlost jeho výpočtu. Nejdříve se podíváme, jak rychle jsou prováděny jednotlivé kroky navrženého algoritmu a porovnáme tyto rychlosti s výsledky optimalizovaného algoritmu. Poté se podíváme na rychlost algoritmu jako celku, tedy budeme zkoumat dobu zpracování jednoho snímku. Abchom dosáhli zpracování v reálném čase, musel by algoritmus zpracovat jeden snímek pod 40 ms pro videa s frekvencí 25 snímků za vteřinu nebo pod 67 ms pro videa s frekvencí 15 snímků za vteřinu.

Podrobnější rozbor rychlostí provedeme jen pro videa „tvary_1“ a „věž“. Pro ostaní videa jsou výsledky obdobné. Doby trvání jednotlivých kroků jsou zapsány v tabulkách 6.4 a 6.5. Časy v tabulkách jsou zprůměrované z hodnot naměřených při spuštění s různými roztečemi mřížky. Doby zpracování jednoho snímku jsou znázorněny v grafech 6.1 a 6.2.

Z tabulek 6.4 a 6.5 vyplývá, že operacemi, které nejvíce zaměstnávají navržený algoritmus jsou: detekce význačných bodů (funkce `goodFeaturesToTrack()`), výpočet optického toku (funkce `calcOpticalFlowPyrLK()`) a dále ještě interpolace mřížky posunů. Implementací na GPU se nám podařilo u videa „tvary_1“ zrychlit detekci význačných bodů o 93 % a výpočet optického toku o 91 %. Ve videu „věž“ činila tato zrychlení 89 % u detekce bodů a dokonce 96 % u optického toku.

	detekce bodů	optický tok	výběr bodů	interpolace mřížky	konstrukce mapy posunů	transformace
navržený alg.	0,044	0,195	0,001	0,179	0,004	0,005
optimalizovaný alg.	0,003	0,017	0,002	0,184	0,004	0,005

Tabulka 6.4: Doby trvání jednotlivých kroků algoritmu pro video „tvary_1“.

Obrázek 6.10: Registrované video „auto“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.

	detekce bodů	optický tok	výběr bodů	interpolace mřížky	konstrukce mapy posunů	transformace
navržený alg.	0,090	0,459	0,003	0,325	0,009	0,015
optimalizovaný alg.	0,010	0,018	0,004	0,334	0,009	0,015

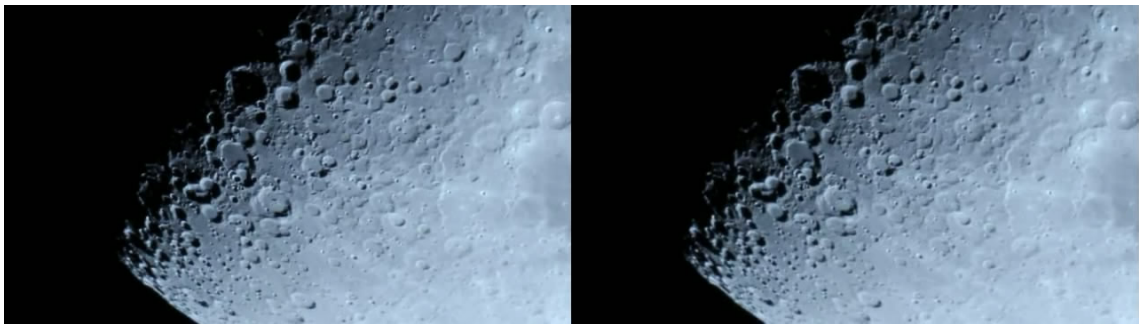
Tabulka 6.5: Doby trvání jednotlivých kroků algoritmu pro video „věž“.

Výpočet interpolace mřížky se nám bohužel nepodařilo zrychlit a zůstává tak jedinou brzdou celého algoritmu. Celkové zrychlení je poté viditelné v grafech 6.1 a 6.2. U obou videí je toto zrychlení přibližně stejné, přibližně 50 %. Je to zrychlení znatelné avšak nedostatečné pro běh v reálném čase. Pokud by se nám nějak podařilo interpolaci zrychlit alespoň tak jako detekci bodů a optický tok, výpočet jednoho snímku by se pohyboval někde kolem hranice nutné pro běh v reálném čase.

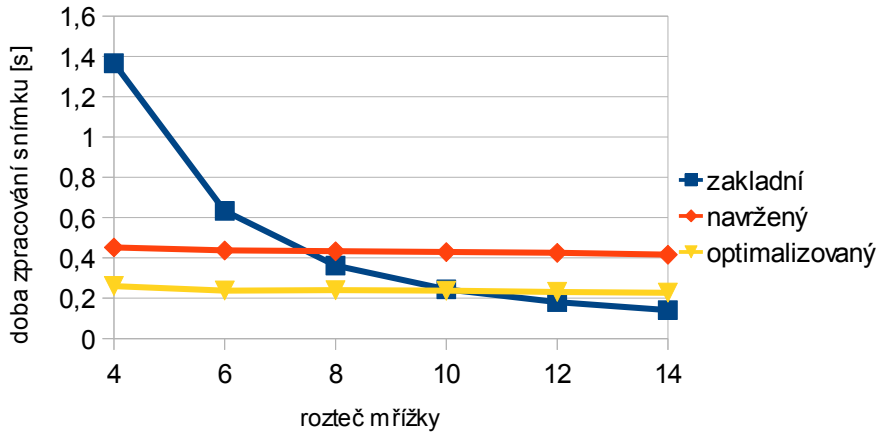
Z grafů je dále možné vidět, že rychlost zpracování jednoho snímku u navrženého algorimu není závislá na roztečí mřížky, jako je tomu u základního algoritmu. Navržený algoritmus je nyní závislý pouze a jenom na rozměrech snímků vstupního videa. Kvůli tomuto poznatku byly provedeny dodatečné testy. Videa „auto“ a „věž“ byla zmenšena na poloviční rozměry a tato zmenšená videa byla znovu zpracována algoritmem. Výsledná rychlost optimalizovaného algoritmu se u těchto videí pohybovala v rozmezí 100-120 ms. Není to stále rychlost dostatečná , ale je to výrazný skok směrem k běhu v reálném čase.

6.3 Shrnutí

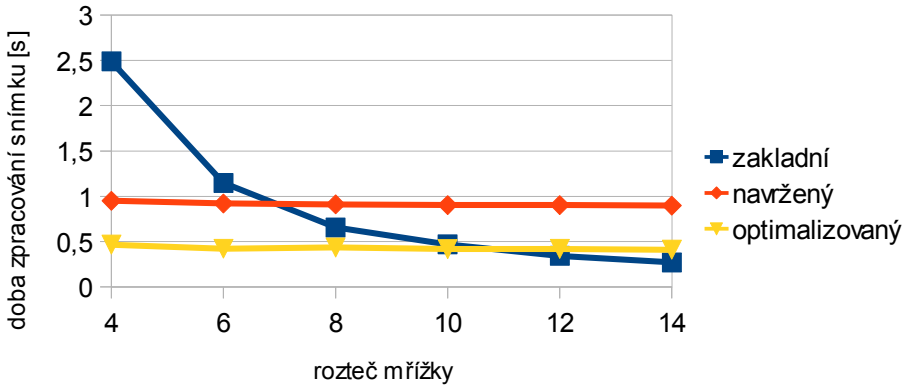
V této kapitole jsme testovali navržený algorimus, zkoumali jsme kvalitu jeho výsledků a rychlost jeho zpracování. Zjistili jsme některé nedostatky a nedokonalosti tohoto algoritmu. Nabyté poznatky shrneme v poslední kapitole a budeme diskutovat možná vylepšení.



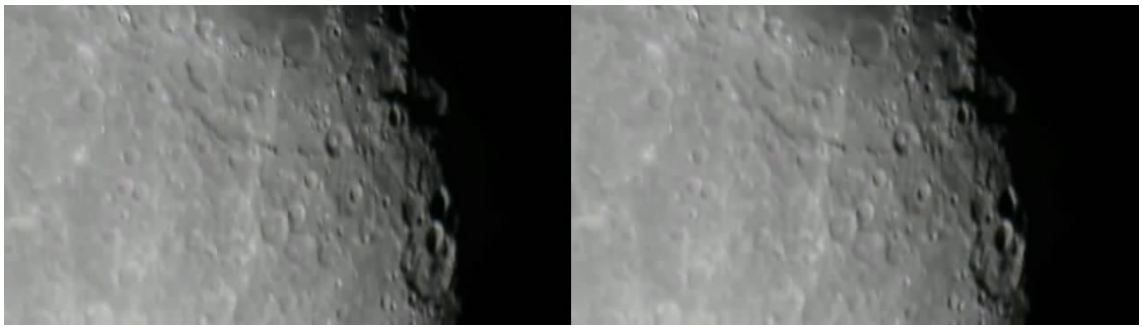
Obrázek 6.11: Registrované video „moon1“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.



Graf 6.1: Srovnání rychlosti ve videu „tvary_1“



Graf 6.2: Srovnání rychlosti ve videu „věž“



Obrázek 6.12: Registrované video „moon2“. Vlevo je snímek originálního videa a vedle něj je registrovaný snímek.

7 Závěr

Cílem této práce bylo navrhnout a implementovat algoritmus pro potlačení atmosférické turbulence z videa. Nejříve jsme zkoumali možnosti, jak tento problém řešit. Poznali jsme některé současné přístupy k řešení tohoto problému. Na základě nabytých znalostí jsme navrhli vlastní algoritmus. Tento algorimus jsme poté implementovali s pomocí knihovny OpenCV a následně jsme jej optimalizovali pro zvýšení rychlosti výpočtu. Algoritmus jsme otestovali a provedli jsme experimenty na sadě testovacích videí.

Z experimentů jsme zjistili, že kvalita výsledků algoritmu je závislá na úrovni turbulence a na samotném obrazovém obsahu. Algoritmus může mít prolémy, pokud jsou ve videu velké plochy s malou strukturou nebo dlouhé linie. Pro optimální výsledek je třeba najít správné nastavení rozteče bodů mřížky, která je při výpočtu používána.

Na testovacích videích jsme ověřili trvrzení Y. Maa [7], že objekty pohybující se v obraze nejsou algoritmem ovlivněny, pokud je jejich pohyb dostatečně odlišný od turbulentního pohybu. Na druhou stranu jsme zjistili, že pokud se tyto dva pohyby příliš neliší, tak může být pohybující se objekt algoritmem poměrně výrazně deformován.

Měřením jsme zjistili, že implementací algoritmu na grafické kartě dokážeme mnohé výpočty výrazně zrychlit. Bohužel se nám některé výpočty nepodařilo na grafické kartě implementvat a nemohly být tím pádem urychleny. Toto má za následek, že algoritmus, alespoň na testovacím počítači, neběží přímo v reálném čase.

O implementovaném programu je možné říct, že pokud najdeme vhodný parametr mřížky, algoritmus zvládne atmosférickou turbulenci ze vstupního videa poměrně dobře odstranit. A s dostatečně výkonným hardwarem by to měl zvládnout i v reálném čase.

7.1 Možná vylepšení

Algoritmus je možné vylepšit několika způsoby. Co se týká rychlosti, bylo by možné implementovat interpolaci bodů do mřížky ručně přímo v OpenCL, nebo najít, či vymyslet jiný způsob interpolace. Některé nápady byly již zmíněny dříve v textu práce.

Kvalitu rekonstruovaného videa je možné zvýšit jiným způsobem hledání význačných bodů. Pro vylepšení rekonstrukce v oblastech s nízkou texturou by bylo možné zkombinovat hledání význačných bodů s pevnou mřížkou bodů. Dále by mohly být nějakým hranovým detektorem detekovány hrany a tyto poté rekonstruovány. Místo řídkého optického toku by mohl být pro hledání posunu mezi snímky použit nějaký algoritmus výpočtu hustého optického toku.

Pro uživatelský komfort by také bylo dobré zautomatizovat hledání parametru mřížky, aby toto nemusel uživatel ručně zadávat.. K tomuto by bylo potřeba vymyslet výpočet nějaké lepší metriky, která nepotřebuje znát ideální snímek.

Literatura

[1] VAŠÍČEK, Jiří. *Vysvětlení některých meteorologických pojmů a jevů: Atmosférická turbulence*. Český hydrometeorologický ústav [online]. 2008 [cit. 2014-01-26]. Dostupné z: http://old.chmi.cz/meteo/olm/Let_met/Pojmy_2.htm#turbulence

[2] *Atmospheric turbulence*. In: Encyclopedia Britannica [online]. ©2014 [cit. 2014-02-13]. Dostupné z: http://www.britannica.com/EBchecked/topic/41528/atmospheric-turbulence

[3] ZHU, X. a P. MILANFAR. *Removing Atmospheric Turbulence via Space-Invariant Deconvolution*. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. 2013, roč. 35, č. 1. DOI:10.1109/TPAMI.2012.82

[4] SHIMIZU, M., S. YOSHIMURA, M. TANAKA a M. OKUTOMI. *Super-resolution from image sequence under influence of hot-air optical turbulence*. IEEE Confference on Computer Vision and Pattern Recognition, 2008. DOI: 10.1109/CVPR.2008.4587525

[5] ZHU, X. a P. MILANFAR. *Image Reconstruction from Videos Distorted by Atmospheric Turbulence*. SPIE 7543, Visual Information Processing and Communication, 2010. DOI: 10.1117/12.840127

[6] ZHU, X. a P. MILANFAR. *Stabilizing and deblurring atmospheric turbulence.* IEEE International Conference on Computational Photography, 2011. DOI: 10.1109/ICCPHOT.2011.5753122

[7] MAO, Y. a J. GILLES. *Non rigid geometric distortions correction: Application to Atmospheric Turbulence Stabilization*. In: Inverse Problems and Imaging Journal. Missouri, USA: American Institute of Mathematical Sciences, duben 2012, s. 531-546. 6, 3. ISSN 1930-8337.

[8] DUŠEK, S. *Určení parametrů pohybu ze snímků kamery.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 76 s. Vedoucí diplomové práce Ing. Ilona Kalová, Ph.D.

[9] *Harris Corner Detection*. In: OpenCV documentation [online]. © 2011-2014 [cit. 2014-02-13]. Dostupné z: http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

[10] *Shi-Tomasi Corner Detector & Good Features to Track*. In: OpenCV documentation [online]. © 2011-2014 [cit. 2014-02-13]. Dostupné z: http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html

[11] *Optical Flow*. In: OpenCV documentation [online]. © 2011-2014 [cit. 2014-02-13]. Dostupné z: http://docs.opencv.org/trunk/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html

[12] XU, Z., X. SU a Z. ZHANG. *Multi-frame Image Super-resolution by Total-variation Regularization*. In: Journal of Information and Computational Science. Hong Kong: BINARY INFORMATION PRESS, 2012, s. 945-953. 9: 4. ISSN 1548–7741. Dostupné z: http://www.joics.com

[13] AUBAILLY, M., M. A. VORONTSOV, G. W. CARHART a M. T. VALLEY. *Automated video enhancement from a stream of atmospherically-distorted images: the lucky-region fusion approach*. SPIE Proceedings. 12. 4. 2009, č. 7463. DOI: 10.1117/12.828332.

[14] ANANTRASIRICHAI, N., A. ACHIM, N.G. KINGSBURY a D.R. BULL. *Atmospheric Turbulence Mitigation using Complex Wavelet-based Fusion*. In: IEEE Transactions on Image Processing. New Jersey, USA: IEEE Signal Processing Society, 26. 2. 2013, 2398 - 2408. 22, 6. ISSN 1057-7149. DOI: 10.1109/TIP.2013.2249078.

[15] LI, D., R.M. MERSEREAU a S. SIMSKE. *Atmospheric Turbulence-Degraded Image Restoration Using Principal Components Analysis*. In: IEEE Geoscience and Remote Sensing Letters. New Jersey, USA: IEEE Geoscience and Remote Sensing Society, 16. 7. 2007, 340 - 344. 4, 3. ISSN 1545-598X. DOI: 10.1109/LGRS.2007.895691.

[16] *OpenCV* [online]. © 2014 [cit. 2014-04-18]. Dostupné z : http://opencv.org/

[17] *OpenCV documentation* [online]. © 2011-2014 [cit. 2014-04-18]. Dostupné z: http://docs.opencv.org/

[18] BRADSKI, Gary R a Adrian KAEHLER. *Learning OpenCV*. Sebastopol: O'Reilly, 2008, xvii, 555 s. ISBN 978-0-596-51613-0.

[19] LAGANIÉRE, Robert. *OpenCV 2 computer vision application programming cookbook: over 50 recepies to master this library of programming functions for real-time computer vision*. 1st ed. Brimmingham: Packt Publishing, 2011, iii, 287 s. Quick Answers to Common Problems. ISBN 978-1-84951-324-1.

[20] *CUDA: Parallel Programming and Computing Platform. NVIDIA CORPORATION.* NVIDIA - World Leader in Visual Computing Technologies *[online]. ©2014 [cit. 2014-04-19]. Dostupné z: http://www.nvidia.com/object/cuda_home_new.html*

[21] *OpenCL: The open standard for parallel programming of heterogeneous systems.* KHRONOS GROUP. The Khronos Group Inc. [online]. ©2014 [cit. 2014-04-19]. Dostupné z: http://www.khronos.org/opencl/

[22] OpenCL Module Introduction. *OpenCV documentation* [online]. © 2011-2014 [cit. 2014-05-19]. Dostupné z: http://docs.opencv.org/modules/ocl/doc/introduction.html

[23] Atmospheric turbulence illustrated. In: *Youtube.com* [online]. 2014 [cit. 2014-04-19]. Dostupné z: https://www.youtube.com/watch?v=GQbm8r4ukg0

[24] Moon - Seeing - Atmospheric turbulence. In: *Youtube.com* [online]. 2011 [cit. 2014-04-19]. Dostupné z: https://www.youtube.com/watch?v=WUJWoPiDs1s

Seznam příloh

Příloha A. Datový disk DVD

Součástí práce je příloha ve formě datového disku DVD. Přiložený datový disk obsahuje nasledující adresáře:

thesis – Adresář obsahuje tento dokument v elektronické podobě ve formátu pdf a zdrojovou podobu tohoto dokumentu ve formátu odf.

application – V podadresáři **source** se nacházejí zdrojové soubory navrženého programu ve formě solution balíčku pro Microsoft Visual Studio 2012. V podadresáři **bin** se nachází přeložená a spustitelná verze tohoto programu ve formě exe souboru pro 32-bitovou verzi systému Microsoft Windows a dynamické knihovny potřebné pro běh programu ve formátu dll. Podadresář **doc** obsahuje dokumenatci ve fotmátu html vygenerovanou programem doxygen.

videos – Adresář obsahuje soubory originálních testovacích videí.

examples – Adresář obsahuje ukázky testovacích videí zpracovaných implementovaným algoritmem s nastavením různých hodnot parametrů. Adresář dále obsahuje kontrolní výpis programu s informacemi o průběhu zpracování videí.

Kořenový adresář DVD navíc obsahuje soubor readme.txt s popisem obsahu DVD a jednoduchým návodem pro překlad a používání programu.