UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Naknadna stabilizacija videoposnetkov

(Subsequent video stabilization)

Ime in priimek: Kevin Sedevcic
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, september 2014

Ključna dokumentacijska informacija

Ime in PRIIMEK: Kevin SEDEVCIC

Naslov zaključne naloge: Naknadna stabilizacija videoposnetkov

Kraj: Koper

Leto: 2014

Število listov: 48 Število slik: 15 Število tabel: 2

Število prilog: 1 Število strani prilog: 5 Število referenc: 27

Mentor: doc dr. Peter Rogelj

Ključne besede: ocena gibanja, popravljanje gibanja, stabilizacija videoposnetkov, algoritmi za iskanje po blokih, fazna korelacija, optični tok

Izvleček:
Zaključna naloga se prične s predstavitvijo zahtev o vedno širši uporabi tehnik za stabilizacijo videoposnetkov in se nadaljuje s poglavjem o oceni gibanja slike. V prvem poglavju so naštete in razložene teoretične podrobnosti algoritmov za iskanje po blokih (BMA). Obravnavani algoritmi so: popolno iskanje (ES), dve različici trikoračnega iskanja (TSS in NTSS), enostavno in učinkovito iskanje (SES), štiri-koračno iskanje (FSS), iskanje z vzorcem diamanta (DS) in algoritem s prilagodljivim vzor-cem (ARPS). Poleg navedenih algoritmov je obravnavana še tehnika za prepoznavanje fazne korelacije (PC) in metodi za prepoznavanje optičnega toka (OF). Temu sledi teoretična primerjava tehnik in izpostavljanje njihovih dobrih in slabih lastnosti. Opi- sane so tudi uporabljene statistične funkcije pri oceni pogreška primerjave: povprečje kvadriranih pogreškov (MSE), vsota absolutnih razlik (SAD), povprečje absolutnih raz- lik (MAD), vsota kvadriranih razlik (SSE) in vsota absolutnih transformiranih razlik (SATD). Končni rezultat ocene gibanja, ki jo dobimo z uporabo omenjenih postopkov, je vektor premika (MV). V drugem delu zaključne naloge je razloženo, kako je lahko vektor premika uporabljen za popravo gibanja videoposnetka. Obravnavane so poprave translacijskega gibanja in rotacije slike. V zadnjem delu zaključne naloge je razložena lastno izdelana programska oprema. Le-ta je sprogtamirana v C++ programskem jeziku z uporabo odprtokodne knjižnice OpenCV za obdelavo slik. V zaključku so izpostavljene dobre in slabe lastnosti implementiranih postopkov.

Key words documentation

Name and SURNAME: Kevin SEDEVCIC

Title of final project paper: Subsequent video stabilization

Place: Koper

Year: 2014

Number of pages: 48 Number of figures: 15 Number of tables: 2

Number of appendices: 1 Number of appendix pages: 5 Number of references: 27

Mentor: Assoc. Prof. Peter Rogelj, PhD

Keywords: motion estimation, motion compensation, videostabilization, block matching algorithms, phase correlation, optical flow

Abstract:
The following thesis is a review of 3 main techniques of video stabilization that compute motion estimation and motion compensation of two consecutive frames of a video. In the first chapter the author has listed and explained in detail the theoretical parts of block matching algorithms (BMA), phase correlation (PC) technique and optical flow (OF) methods. The discussed BMA are: exhaustive search (ES), two versions of the three step search (TSS and NTSS), the simple and efficient search (SES), the four step search (FSS or 4SS), the diamond search (DS) and adaptive rood pattern search (ARPS). This is followed by description of the PC method and the Lukas & Kanade (LK) and Horn & Schunck (HS) approach for OF estimation. In addition to the explained techniques the author has also listed the possible statistical cost functions for best motion estimation results. The functions are the mean squared error (MSE), the sum of absolute differences (SAD), the mean of absolute differences (MAD), the sum of squared error (SSE) and the sum of absolute transformed differences (SATD). The result of the previously mentioned techniques is a global motion vector (MV) that represents the global motion of two frames. In the second part of the thesis the author has explained how the motion can be compensated using that MV in a translative or in a rotational movement. In the last chapter the author has described the programmed software, written in C++, using the open source library OpenCV for image processing. The conclusive notes expose the good and bad features of the implemented methods discovered through some different test videos.

Zahvala

Zahvaljujem se družini, ki me je vedno podpirala in ki mi je omogočila študij na FAMNIT-u. Zahvaljujem se tudi vsem profesorjem in asistentom, ki so se s profesionalnostjo odzvali na marsikakšno situacijo. Nenazadnje se zahvaljujem sošolcem, s katerimi smo skupaj premostili največje težave. Posebna zahvala pa gre prof. doc. dr. Petru Roglju, ki mi je pomagal in me s hitro odzivnostjo usmerjal pri pisanju zaključne naloge.

Kazalo vsebine

Seznam tabel

Seznam slik

Seznam prilog

Seznam kratic

ipd.	In podobno
npr.	Na primer
BMA	Block matching algorithms - algoritmi za iskanje po blokih
ES	Exhaustive search - popolno iskanje
TSS	Three step search - iskanje v treh korakih
$NTSS$	New three step search - novo iskanje v treh korakih
SES	Simple and efficient search - enostavno in učinkovito iskanje
FSS	Four step search - štiri koračno iskanje
DS	Diamond search - iskanje z vzorcem diamanta
$LDSP$	Large diamond search pattern - veliki iskalni vzorec v obliki diamanta
$SDSP$	Small diamond search pattern - majhen iskalni vzorec v obliki diamanta
$ARPS$	Adaptive rood pattern search - algoritem s prilagodljivim vzorcem
MSE	Mean squared error - povprečje kvadriranih pogreškov
SAD	Sum of absolute differences - vsota absolutnih razlik
MAD	Mean of absolute differences - povprečje absolutnih razlik
SSE	Sum of squared errors - vsota kvadriranih pogreškov
$SATD$	Sum of absolute transformed differences - vsota absolutnih transformiranih razlik
$PSNR$	Peak signal to noise ratio - razmerje ekstremnih vrednosti signala in šuma
MV	Motion vector - vektor premika
FPS	Frame per second - posnetki na sekundo
PC	Phase correlation - fazna korelacija
FT	Fourier transform - fourierova transformacija
DFT	Discrete fourier transform - diskretna fourierova transformacija
FFT	Fast fourier transform - hitra fourierova transformacija
DC	Direct current - enosmerni tok
OF	Optical flow - optični tok
LK	Lukas & Kanade
HS	Horn & Schunk
MA	Moving average - povprečje gibanja
$RANSAC$	RANdom SAmple Consensus - soglasje o naključnem izbranem vzorcu

1 Uvod

Videokamere se uporabljajo vse pogosteje, tako v profesionalni industriji kot v domači uporabi. Cilj pa je skupen pri obeh, in sicer čim višja kakovost videoposnetkov. Poleg nenehne težnje po boljši kvaliteti slike z višjo resolucijo, boljšim kontrastom barv ipd., se je pokazala tudi potreba po stabilizaciji tresljajev videoposnetkov. Samo pomislimo namreč na uporabo kamere z visoko stopnjo povečave in na trenutke, ko kamero uporabljamo med vožnjo v avtomobilu ali jo držimo v rokah. V profesionalni industriji za stabilizacijo tresljajev uporabljajo strojno opremo in optično stabilizacijo, s čimer preprečijo oz. omilijo tresljaje. Vendar je tovrstna oprema draga in večkrat okorna in nerodna. Zato se je uveljavila računalniška obdelava slike. Deli se v tri večje faze:

- ocena gibanja,
- popravek gibanja in
- rekompozicija slike.

V zaključni nalogi sta obravnavani prvi dve fazi, pri čemer je prva razvrščena v tri skupine:

- algoritmi za iskanje po blokih (BMA): ES, TSS, NTSS, SES, 4SS, DS in ARPS,
- tehnika za fazno korelacijo in
- metodi za optični tok (Lucas & Kanade in Horn & Schunck).

Za primerjanje najboljšega ujemanja so uporabljene statistične funkcije, kot so MSE, SAD, MAD, SSE in SATD. V drugem delu pa je predstavljen način popravljanja gibanja pri vsaki od navedenih skupin. S pomočjo kakovostnih algoritmov lahko slike videoposnetkov enostavno obdelamo. V sledečem poglavju sledita opis in primerjava glavnih postopkov in algoritmov pri (naknadni) računalniški videostabilizaciji.

2 Algoritmi in metode za ocenjevanje gibanja

Ocenjevanje gibanja slike ni enostavno. Med procesom lahko namreč pride do nepredvidljivega šuma, ki spreminja vsebino posnetka, lahko se pripetijo hitre nenavadne spremembe svetlobe ali pa posnet izdelek sam daje dvoumne informacije (glej problem odprtine v [10]). Ločevati je potrebno tudi nenamerno gibanje od zaželjenega. Sledi opis glavnih algoritmov za ocenjevanje gibanja z iskanjem po blokih.

2.1 Algoritmi za iskanje po blokih (Block matching algorithms - BMA)

V obdelavi slik se ti algoritmi uporabljajo za primerjavo blokov med dvema zaporednima slikama videoposnetka. Z iskanjem najboljšega ujemanja dveh blokov lahko ocenimo stopnjo gibanja slike, izraženo z vektorjem. Iskanje poteka tako, da je trenutna slika razdeljena na bloke velikosti NxN pikslov. Pomemben prarameter algoritmov je koeficient iskanja p, s katerim se vsak blok primerja z blokom v prejšnji sliki. Najboljše ujemanje blokov se išče v intervalu $(x \pm p, y \pm p)$. Rezultat $x + i, y + j$ kjer i in j predstavljata premik oziroma gibanje bloka v x in y smeri (angl. "motion vector", glej sliko 1). V literaturi se velikokrat zasledi uporaba blokov velikosti 16x16 pikslov in koeficient iskanja enak 7. [2, 6, 12, 13] V primerih, ko se te vrednosti spreminjajo, se lahko spremeni tudi število korakov naslednjih algoritmov, in sicer po kriterijih, navedenih spodaj. [16]
Za število korakov X in koračni razmik S:

$$X = \log_2 (p + 1) \qquad S = 2^{X-1}$$

Iskanje najprimernejšega bloka je odvisno od vrednosti izbrane statistične funkcije. Bolj, kot sta si bloka v primerjavi podobna, nižja je vrednost. Če je vrednost enaka 0, pomeni, da sta si bloka popolnoma enaka. V tem primeru algoritem poišče blok z najmanjšo vrednostjo statistične funkcije (angl. "the least weight"). Pri teh algoritmih se ponavadi uporablja sledeče ocenjevalne statistične funkcije:

povprečje kvadriranih pogreškov (Mean Squared Error - MSE) [2, 12];

$$MSE = \frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}(F1_{ij} - F2_{ij})^2$$

vsota absolutnih razlik (Sum of Absolute Difference - SAD), ki je najlažja in najmanj računsko potratna;

$$SAD = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1}|F1_{ij} - F2_{ij}|$$

povprečje absolutnih razlik (Mean Absolute DIfference - MAD);

$$MAD = \frac{1}{N^2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}|F1_{ij} - F2_{ij}|$$

vsota kvadriranih pogreškov (Sum of Squared Errors - SSE) in

$$SSE = \frac{1}{2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}(F1_{ij} - F2_{ij})^2$$

vsoto absolutnih transformiranih razlik (Sum of absolute transformed differences - SATD).

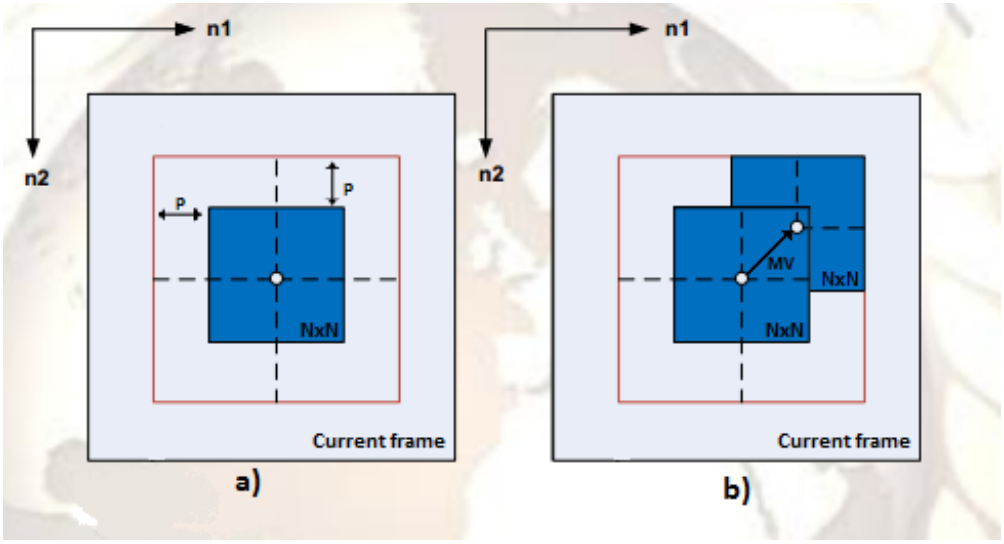
$$SATD = \frac{1}{2}\sum_{j=0}^{N-1}\sum_{i=0}^{N-1}|H * (F1_{ij} - F2_{ij}) * H|$$

$$H = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{vmatrix}$$

SATD deluje na transformaciji razlike pikslov dveh blokov v primerjavi. Dve uporabljeni transformaciji pri tej tehniki sta Hadamardova in LaPlacova. V navedenih formulah $F1_{ij}$ in $F2_{ij}$ predstavljajo piksle na trenutnem bloku in piksle na iskanem bloku, medtem ko H predstavlja Hadamardovo matrično transformacijo. Poleg statističnih funkcij za ocenjevanje pogreška se uporablja še funkcijo za razmerje med jakostjo signala in šuma (angl. "peak signal to noise ratio - PSNR"), ki predstavlja, koliko vpliva povprečni pogrešek na realno vrednost slike. [12]:

$$PSNR = 10 * (log_{10}\frac{PP^2}{MSE})$$

V navedeni enačbi PP predstavlja najvišjo jakost piksla, MSE pa povprečni šum. Sledita opis in razlaga delovanja izbranih algoritmov za iskanje po blokih.



Slika 1: Primer bloka velikosti NxN v iskalnem območju in vektor premika (MV). Slika je pridobljena iz [12].

2.1.1 Popolno iskanje (Exhaustive search - ES)

Popolno iskanje je najbolj točno, a tudi računsko najbolj zahtevno iskanje. ES pregleda vse možne lokacije blokov v iskanem intervalu in primerja vrednost statistične funkcije za najboljše ujemanje z najvišjim PSNR-jem. Zaradi računske zahtevnosti ni primeren za obdelavo slik v realnem času. V nadaljevanju sledi opis hitrih BMA, rezultati katerih se po točnosti najbolj približujejo točnosti ES-ja, a z manjšo računsko zahtevnostjo. [2, 12, 13, 19]

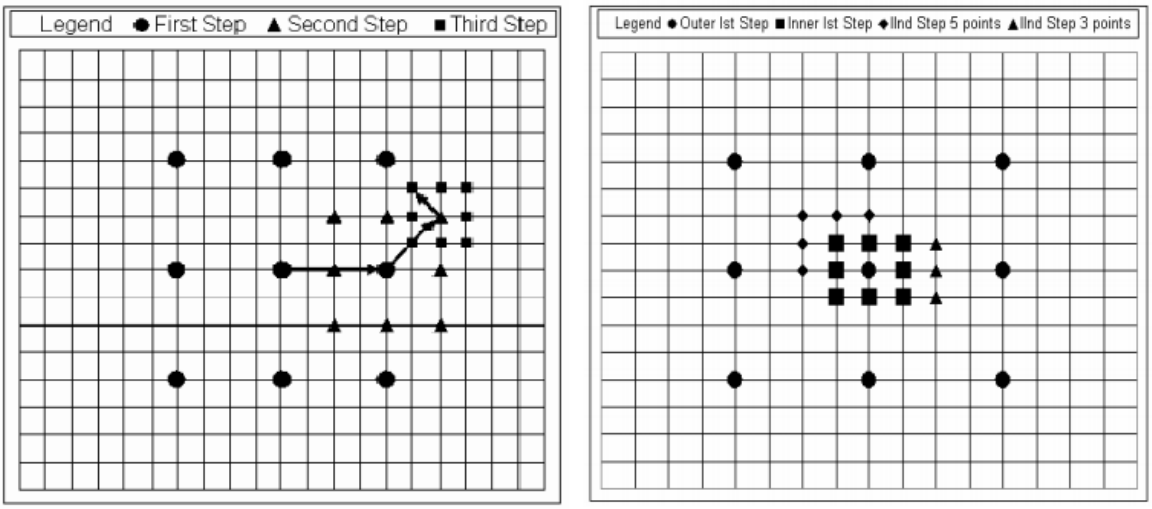
2.1.2 Iskanje v treh korakih (Three step search - TSS)

Zapisan že v 80ih letih 20. stoletja je prvi algoritem, ki znatno izboljša hitrost iskanja. Glavno idejo delovanja si lahko ogledamo na sliki 2 (levo). Začetna lokacija je sredina bloka, denimo (0,0). Definira se tudi koračni razmik, S = 4. Prvi korak algoritma je primerjava začetne točke in osmih točk v njeni okolici z razdaljo S. Po prvem krogu primerjav si izberemo tisto točko, ki ima najmanjšo vrednost izbrane statistične funkcije in jo upoštevamo kot novo začetno točko, S pa razpolovimo. Ta korak ponovimo še dvakrat, dokler ne dobimo S = 1. Zadnja izbrana točka bo rezultat najboljšega ujemanja. [2, 12, 16, 19]

2.1.3 Iskanje v treh korakih 2 (New three step search - NTSS)

Z namenom še dodatnega krajšanja časa iskanja se je kasneje uveljavila nova verzija TSS-ja. NTSS je hitrejši v primeru, ko je izbrana začetna točka že minimum. Prvi

korak je enak prvemu koraku prejšnjega algoritma: določimo začetno točko (0,0) in S = 4. Nato pregledamo vseh 9 točk; če je ena od osmih zunanjih točk minimum, postopek nadaljujemo po prejšnjem algoritmu z razpolovljenim S-jem; če pa je minimum sredinska točka, takoj določimo S = 1 in pregledamo okolico 8 točk. V primeru, da je minimum spet ena od osmih točk, se središče premakne na pozicijo minimuma in ponovi se zadnji korak z enakim S-jem. Razliko v primerjavi s staro različico TSS-ja vidimo v zadnjem koraku, kjer pregledamo samo 3 ali 5 točk (glej sliko 2 (desno)) . [2, 19]



Slika 2: Primer vseh treh korakov algoritma TSS (levo) in vseh možnosti NTSS-ja (desno). Sliki sta pridobljeni iz [2]

2.1.4 Enostavno in učinkovito iskanje (Simple and efficient search - SES)

SES je ponovno izboljšanje prejšnjega algoritma. Deluje na predpostavki, da zaradi nizke možnosti prisotnosti dveh minimumov v dveh nasprotnih smereh istočasno, ni potrebno preveriti vseh osem smeri izbrane točke (kot na sliki 3 a). V prvem koraku si poleg sredinske točke izberemo dve ortogonalni smeri na razdalji S in ju preverimo (3 b). Če ju preverjamo z uporabo MAD-a ali druge ocenjevalne statistične funkcije, izberemo definirani kvadrant. V primeru, ko uporabimo MAD, kvadrant izberemo po sledečih pravilih:

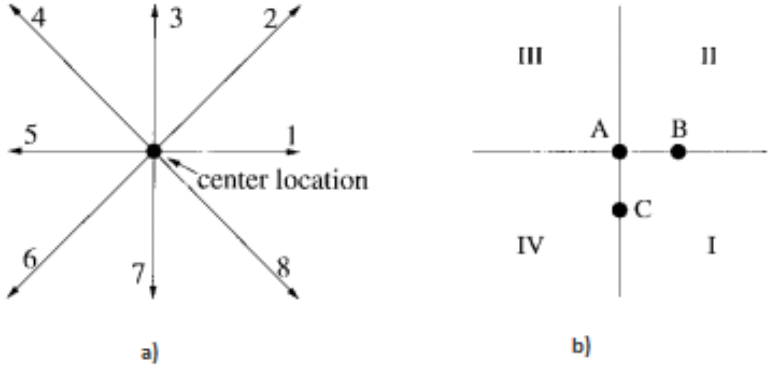
$$if \quad MAD(A) \geq MAD(B) \wedge MAD(A) \geq MAD(C) \quad then \quad select \quad I.$$

$$if \quad MAD(A) \geq MAD(B) \wedge MAD(A) < MAD(C) \quad then \quad select \quad II.$$

$$if \quad MAD(A) < MAD(B) \wedge MAD(A) < MAD(C) \quad then \quad select \quad III.$$

$$if \quad MAD(A) < MAD(B) \wedge MAD(A) \geq MAD(C) \quad then \quad select \quad IV.$$

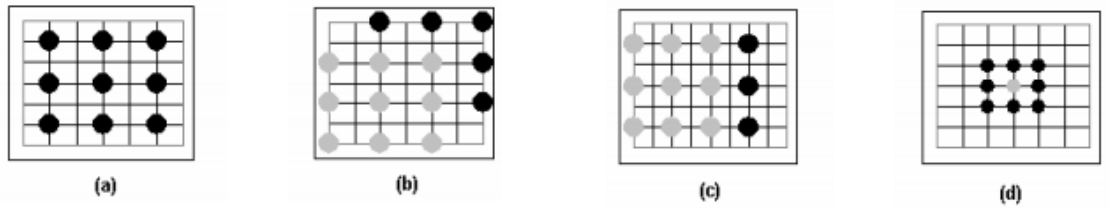
Izbranemu kvadrantu pregledamo kotno točko in dve ortogonalni točki. Med temi izberemo minimum, premaknemo začetno točko na nov minimum in ponovimo postopek. V vsakem koraku razpolovimo S, dokler je S večji od 1. Čeprav SES zelo skrajša čas iskanja najboljšega ujemanja, uradno ni bil sprejet, in sicer zaradi dveh razlogov. Prvič, zaradi pogreškov, saj lahko v praksi dobimo dva minimuma na nasprotnih smereh, in drugič, zaradi uveljavitve FSS-ja, ki je imel isto dobre rezultate, a z višjim PSNR-jem. [2, 16]



Slika 3: Smeri in kvadranti SES-ja (levo) in prvi izbran kvadrant (desno). Sliki sta pridobljeni iz [16].

2.1.5 Štiri koračno iskanje (Four step search - FSS)

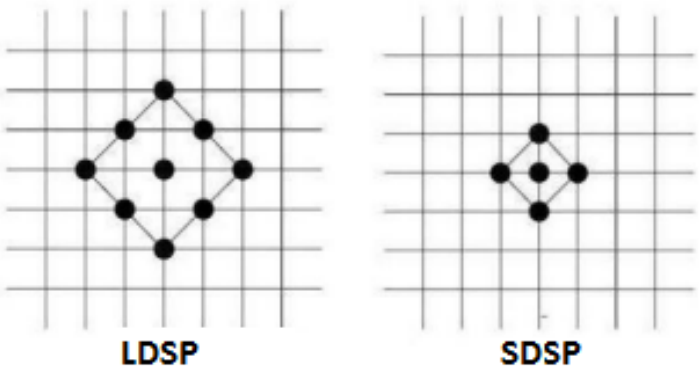
Tudi FSS je ena od različic prejšnjega algoritma TSS. Koračni razmik S bo fiksne dolžine 2 ne glede na iskalni faktor p. Prvi korak je enak kot pri algoritmu TSS, to je primerjanje 8 točk na dolžini S od začetne točke (0,0). Če je minimum sredinska točka, algoritem takoj preskoči na četrti korak. V nasprotnem primeru pa premakne sredinsko točko na izbran minimum in preveri 3 ali 5 novih sosedov (glej sliko 4 b in c).Če je sredinska točka minimum, potem algoritem nadaljuje s četrtim korakom, sicer se izvede tretji korak, ki je enak drugemu. Pri četrtem koraku zmanjšamo koračni razmik S na 1, preverimo sosednje točke in izberemo minimum (vzorec na sliki 4 d), s katerim v nadaljevanju določimo vektor premika z začetne pozicije (0,0) do minimuma. [2, 12, 19]



Slika 4: Primer štirih korakov v FSS-ju. Slike so pridobljene iz [2]

2.1.6 Iskanje z vzorcem diamanta (Diamond search - DS)

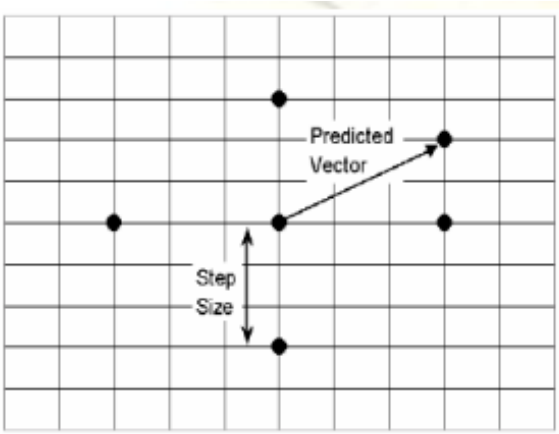
Ta algoritem se od zgoraj opisanih razlikuje zaradi vzorca v obliki diamanta. Uporabljena sta vzorca LDSP in SDSP (glej sliko 5). Algoritem deluje po principu FSS in se razdeli v 3 večje faze. V prvi uporabimo LDSP s centrom na izbrani začetni točki. Pregledamo vseh 9 točk in izberemo minimum. Če izbrana točka ni sredinska, premaknemo središče na izbrano točko in ponovimo korak. Korak ponavljamo, dokler kot sredinske točke ne dobimo minimuma. V tretji fazi zamenjamo obstoječi vzorec s SDSP-jem in primerjamo točke. Rešitev iskanja vektorja premika je dobljen minimum. PSNR tega algoritma je zelo podoben PSNR-ju ES-a, računska zahtevnost pa znatno manjša. DS se klasificira kot eden izmed najboljših algoritmov za iskanje po blokih. [2, 12, 19, 27]



Slika 5: Primer vzorcev iskanja DS. Na levi je prikazan LDSP in na desni manjši SDSP. Sliki sta pridobljeni iz [27].

2.1.7 Algoritem s prilagodljivim vzorcem (Adaptive rood pattern search - ARPS)

Vsak videoposnetek je sestavljen iz več zaporednih slik. Če je število slik v sekundi (FPS) dovolj visoko, smo lahko prepričani, da se v večini videoposnetkov vektorji premika spreminjajo le za majhno vrednost. Na isti način lahko predpostavljamo, da so vektorji premika med sosednjimi bloki podobni. S to lastnostjo lahko izboljšamo hitrost iskanja blokov, tako da za koračni razmik S vzamemo $Max(|X|, |Y|)$, kjer sta X in Y koordinati abscise oz. ordinate vektorja premika sosednjega levega bloka, ki je že bil preiskan (zgled na sliki 6, ki prikazuje napovedan vektor in izbran S). Obstaja veliko različic tega algoritma, npr. napovedan vektor premika je lahko definiran ne samo z bližnjim levim blokom, ampak z večjim številom le-teh. Prvi korak takega algoritma je primerjanje točke, na katero kaže napovedan vektor premika, in štirih točk na razdalji S v navpično in vodoravno smer. V naslednjih korakih pa spremenimo središče na trenutni minimum in iskanje nadaljujemo s SDSP-jem tako dolgo, dokler ne bo najmanjši minimum tudi središče vzorca.
Pri ARPS-ju opazimo dve prednosti v primerjavi z DS-jem. Prvič, v primeru, da je izbrana začetna točka minimum, algoritem pregleda okolico s SDSP-jem (za razliko od DS-ja, ki uporabi LDSP). In drugič, v nasprotnem primeru, če je minimum daleč stran od središča, se ARPS takoj premakne v tisto okolico. [2, 12, 13, 20]



Slika 6: Primer začetnega koraka ARPS-ja. S predvidenim vektorjem premika ("predicted vector") in koračnim razmikom (angl. "step size"). Slika je pridobljena iz [2].

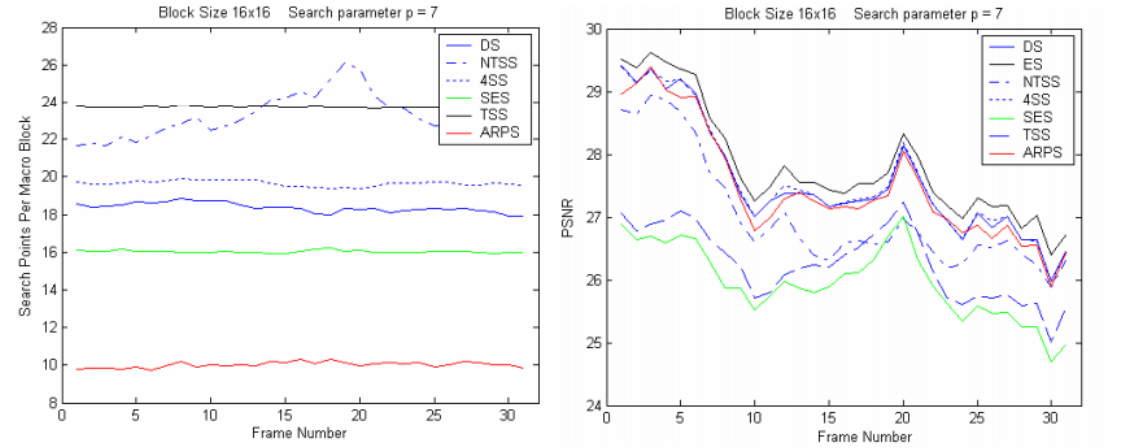
2.1.8 Primerjava algoritmov za iskanje po blokih

V spodnji tabeli so navedena teoretična števila primerjav na blok prejšnjih obravnavanih algoritmov v najboljšem (B) in najslabšem (W) primeru. Na sliki 7, sta pri-

Algoritem	ES	TSS	NTSS		SES		FSS		DS		ARPS	
			B	W	B	W	B	W	B	W	B	W
blok 16x16, p=7	256	25	20	25	10	16	17	27	13	23	10	22

Tabela 2.1: Tabela prikazuje število primerjav prejšnjih 7 opisanih algoritmov v najboljšem (B) in najslabšem (W) primeru, pri velikosti bloka $16x16$ pikslov in koeficientom iskanja $p = 7$. Nižje število predstavlja hitrejši algoritem.

kazana dva grafa, ki prikazujeta povprečno število primerjav na blok (levo) in PSNR vrednosti (desno) v realnem poskusu videoposnetka. Za popolnejšo primerjavo navedenih algoritmov so rezultati pridobljeni iz [2]. Testiranje je bilo izvedeno na "Caltrain"videoposnetku. Makrobloki so velikosti $16x16$ pikslov in koeficient iskanja $p = 7$. Takoj ob pogledu na levi graf je opazno, da se algoritmi ločijo v tri skupine. TSS in NTSS algoritma sta poleg ES-a (v grafu ni upoštevan) najpočasnejša. Lepo se odzovejo FSS (ali 4SS), DS in SES, ki se približujejo najhitrejšemu, to je ARPS. Desni graf pa nam prikaže veliko slabost teh algoritmov. Daljše, kot je zaporedje videoposnetkov, nižja bo vrednost PSNR-ja ob koncu videoposnetka. To pomeni, da zgoraj navedeni algoritmi žrtvujejo kakovost iskanja za večjo hitrost. Primerjava števila primerjav algoritmov s sovpadajočim PSNR-jem pokaže, da čeprav imata SES in FSS v najboljšem primeru podobno število primerjav kot DS in ARPS, imata nižjo vrednost PSNR (slika 7). Več informacij o PSNR-ju najdemo v [2, 12, 19]; ker se PSNR nanaša na kakovost slike pri algoritmih za stiskanje slik z izgubo kakovosti (angl. "lossy compression codecs"), to pa je snov, ki ni obravnavana v zaključni nalogi, na tem mestu PSNR ni podrobneje razložen.



Slika 7: Število primerjav na blok z vsemi zgoraj opisanimi algoritmi (levo) in vrednosti PSNR-ja (desno) na "Caltrain"videoposnetku. Grafa sta pridobljena iz [2].

V naslednjem poglavju bomo obravnavali tehniko za ocenjevanje gibanja celotne slike

s primerjanjem faz pripadajočega signalnega spektra dveh zaporednih slik.

2.2 Fazna korelacija (Phase correlation)

Za razliko od prejšnjih algoritmov z algoritmom fazne korelacije (angl. "phase correlation- PC) prepoznamo premik med dvema slikama preko fazne razlike pripadajočih signalov. Vsak videoposnetek vsebuje enolične informacije o sebi v obliki signalov, ki se razlikujejo po frekvenci, jakosti in fazi. Vsaka slika pa ima točno določene vrednosti le-teh. Za primerjavo dveh slik moramo poznati njun frekvenčni spektrum. Fourierjeva transformacija (FT) je uporabljena pri dekomponiranju slike na njene sinuse in kosinuse (harmonske komponente). Ker uporabljamo digitalne signale, se uporabi diskretno Fourierjevo transformacijo ali DFT, ki opiše digitalni signal s končno množico frekvenčnih (harmonskih) komponent, ki zadostujejo za rekompozicijo slike. Formula je zapisana spodaj: [17]

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)e^{-i2\pi(\frac{ki}{N}+\frac{lj}{N})} \qquad e^{ix} = \cos x + i\sin x$$

F(k,l) je slika v Fourierjevi ali frekvenčni domeni, f(i,j) pa slika v prostorski domeni. Eksponentna funkcija se nanaša na zapis glavnih harmonskih komponent v kompleksnem prostoru. Rezultat 2D DFT je vsota zmnožkov med točkami slike in pripadajočimi frekvencami. Vsaka točka v Fourierjevi domeni označuje določeno frekvenco s slike v časovni domeni, število frekvenc pa je enako številu pikslov slike in se razteza od $F(-f_s/2, -f_s/2)$ do $F(f_s/2, f_s/2)$. f_s je vzorčna frekvenca, ki je v praksi enaka resoluciji slike v smeri ene koordinate. $F(0,0)$ torej predstavlja enosmerno vrednost (DC) in ustreza povprečni svetlosti slike. Nahaja se na sredini slike (primer na sliki 8 c in f). V primeru slike z resolucijo NxN, bi torej bila $F(N/2, N/2)$, najvišja frekvenca signala. [8, 14, 25] Spodnja napisana formula 2.1, opiše razmerje med Hadamardovim množenjem realnega dela FT prve slike (F_a) s konjugiranim kompleksnim delom FT druge slike ($\overline{F_b}$) (angl. "cross power spectrum"). Če rezultatu R izvedemo inverzno Fourierovo transformacijo (2.2) dobimo normalizirano funkcijo z eno najvišjo vrednostjo (angl. "cross correlation"). Koordinati najvišje vrednosti, predstavljata vektor premika med dvema slikama. [17]

$$R = \frac{F_a * \overline{F_b}}{|F_a * \overline{F_b}|} \tag{2.1}$$

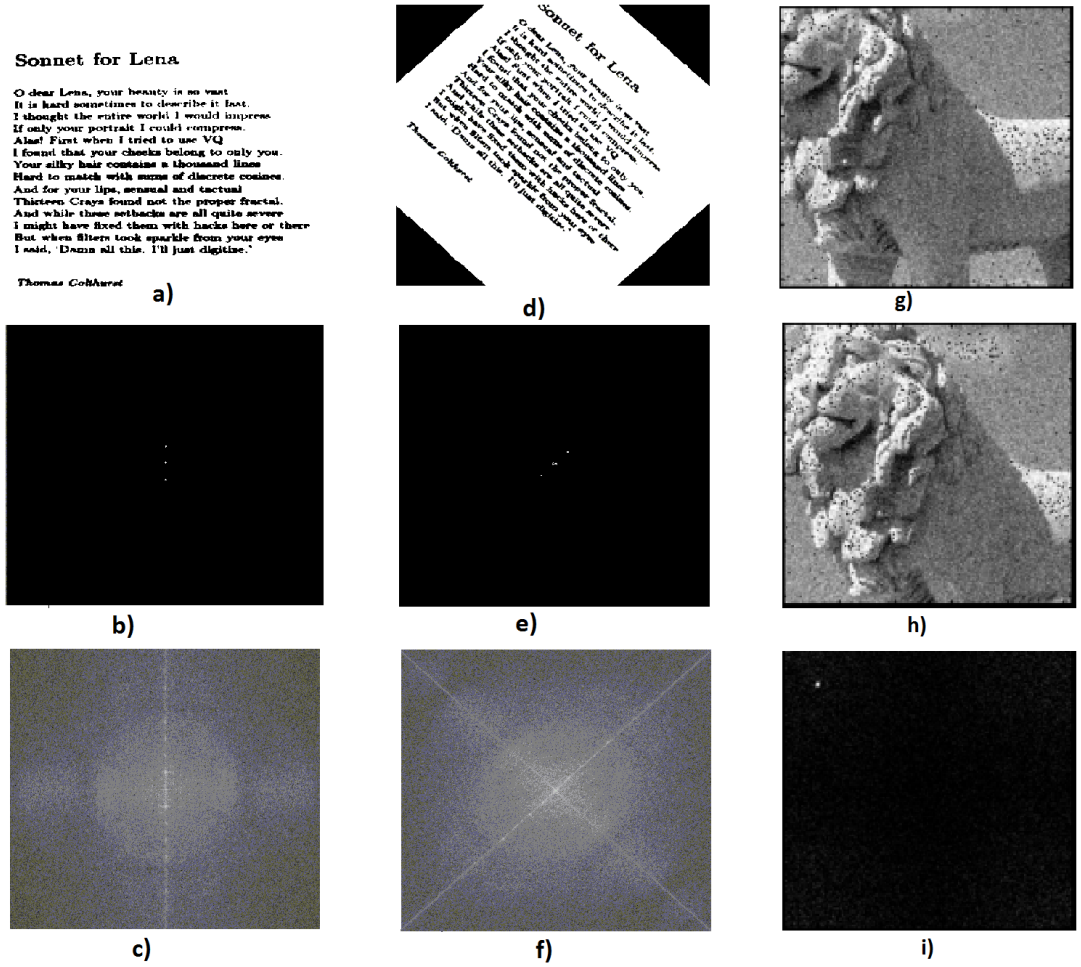
$$f(a,b) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(k,l)e^{i2\pi(\frac{ka}{N}+\frac{lb}{N})} \tag{2.2}$$

Rezultat je slika, ki ima svetlejšo točko na črni podlagi. Lokacija točke predstavlja vektor translacijskega premika. Primer je viden na sliki 8 g, h in i. Na sliki 8 sicer primerjamo delovanje DFT na dveh slikah; prva je navadna slika teksta (a), druga pa prikazuje enak, a obrnjen tekst (d). Sledita sliki, ki prikazujeta vsoto vseh frekvenc v kompleksni obliki (b in e). Na njiju lahko razločimo samo svetlejše pike na sredini; to se zgodi zaradi prevelike razlike med vsotami. Za boljši prikaz lahko vsote logaritmiramo in opazimo večjo raznolikost v frekvencah (sliki c in f). Na sliki c ravna svetlejša navpična črta označuje, da so objekti na sliki usmerjeni vodoravno. Slika f pa ima dve črti, pravokotni ena na drugo, ki prikazujeta, da je slika obrnjena za 45 stopinj v smeri urinega kazalca.Ob pričetku poglavja je bilo omenjeno, da se premik med dvema slikama zazna preko spremembe faze in svetlejša črta na slikah c in f predstavlja skupino najvišjih vsot, ki se zgodijo zaradi premika faze (angl. "peak signal"). Ob premiku slike se bo faza signala spremenila. Razlika v fazi je enaka premiku svetlejše črte, ki sovpada s premikom slike v primerjavi z drugo sliko. Implementacija DFT za obdelavo slike v realnem času se izvede s hitro Fourierjevo transformacijo (FFT), za katero so znane različne vrste imple- mentacije, pri katerih je kompleksnost celo $N \log N$ (Cooley-Tukey [18], Bruun's [26], Bluestein [23]...). Pri algoritmih za iskanje po blokih smo kot rezultat vsakega uje- manja dobili vektor premika, ki je opisal, koliko se je tisti blok premaknil glede na istoležni blok prejšnje slike. Pri fazni razliki lahko za translacijsko gibanje uporabimo FFT na celotni sliki. Slika se razdeli na bloke in na vsakem bloku se uporabi FFT. Rezultat je vektor premika za vsak blok, s katerim lahko popravimo tudi rotacijo slike (več o popravi slike sledi v 3. poglavju).
Omenjena tehnika se zelo lepo odlikuje pri zanesljivosti s faznim in frekvenčnimi motnjami. Rezultat ostane nespremenjen tudi pri neenakomernih spremembah svetlosti.

Naslednje poglavje obravnava tehniki za optični tok; razloženi bosta dve glavni metodi, ki predstavljata temelj današnjih algoritmov za to področje.

2.3 Optični tok (Optical Flow)

Optični tok predstavlja premik objektov na sceni, povzročen zaradi premika med gledalcem (angl. "observer") in sceno s poljem vektorjev. Vektorji predstavljajo hitrost, s katero se objekti premikajo po sliki. Premik je določen na dva različna načina. Prvi način je preko preverjanja razlike v koordinatah dveh točk v dveh zaporednih slikah. Ta način je uporabljen samo za določen predel slike ali objekta na sliki. Drugi način pa je uporabljen za definiranje optičnega toka za vse točke na sliki (gosti optični tok). Metodi, ki sta opisani v nadaljevanju, delujeta po načelu drugega opisanega načina. Pri gostem optičnem toku predpostavljamo, da imamo konstantno jakost na vsaki točki



Slika 8: Primer uporabe Fourierove transformacije v obdelavi slik. V prvih dveh stolpcih vidimo po vrsticah razporejene: originalne slike, rezultat DFT in logaritmirane DFT vrednosti slik. V tretjem stolpcu pa slika g prikazuje originalno sliko, h premaknjeno sliko in i točko, ki je rezultat fazne korelacije za predstavo premika med slikama. Slike so pridobljene iz [7].

na vsakem objektu (npr: svetlost slike), glej formulo 2.3: [4, 8]

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) \tag{2.3}$$

Ker pričujoča zaključna naloga razrešuje problem tresljajev v videoposnetku, ki so v večini sne- mani z visokim številom FPS, lahko predpostavljamo, da bo premik med zaporednima slikama majhen (majhen Δx in Δy). Pri tem se lahko uporabi prva stopnja Taylorjeve vrste, ki prinaša omejitev za izračun optičnega toka, in sicer: [4, 8]

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t$$

Zgornje navedena je enačba za optični tok. Iz nje sledi:

$$\frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t = 0$$

V enačbi so $\frac{\delta I}{\delta x} \equiv I_x$, $\frac{\delta I}{\delta y} \equiv I_y$, odvodi slike (x, y, t) v smeri x, y in $\frac{\delta I}{\delta t} \equiv I_t$ sprememba časa. $\Delta x \equiv \frac{\Delta x}{\Delta t} = u$ in $\Delta y \equiv \frac{\Delta y}{\Delta t} = v$ pa komponenti hitrosti u in v, ki predstavljajo spremembe osi x in y v času t od jakosti slike $I(x, y, t)$. Dobimo nerešljivo enačbo v dveh neznankah, poznano tudi kot problem odprtine (angl. ščini "aperture problem"): [8]

$$I_x u + I_y v = -I_t \tag{2.4}$$

Dodatno enačbo za rešitev optičnega toka dobimo z raznoliko vrsto dodatnih omejitev. Obstajajo različne metode, ki definirajo način pridobitve optičnega toka slike. V nadaljevanju bosta razloženi Lucas & Kanade in Horn & Schunck metodi.

2.3.1 Lucas & Kanade

V LK metodi se problem dveh neznank rešuje s predpostavko, da je iskani vektor optičnega toka konstanten v neki okolici velikosti p za vsak piksel v sliki. V takem primeru metoda razrešuje optični tok z iskanjem najmanjše kvadratne primerjave za iskan piksel med dvema slikama. Za poudarek pikslov, ki se nahajajo v centru iskanega okna, se uporablja Gaussova porazdelitev Kp (poglobljena obrazložitev parametrov enačb v poglavju 2.3 optični tok in v referencah tega poglavja) [1, 4, 15]:

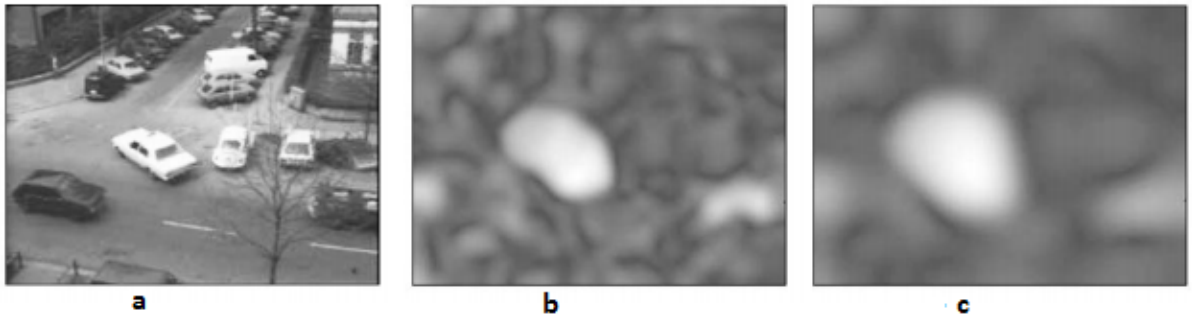
$$E(u, v) = K_p * ((I_x u + I_y v + I_t)^2)$$

Minimalna vrednost, ki sovpada z najboljšim ujemanjem, ima odvod vektorja premika X osi enak nič ($d_x E(u, v) = 0$) in odvod vektorja premika Y osi enak nič ($d_y E(u, v) = 0$). Tako dobimo linearni sistem med posebno matriko (angl. "structure tensor") in iskanima spremenljivkama:

$$\begin{pmatrix} K_p * (I_x^2) & K_p * (I_x I_y) \\ K_p * (I_x I_y) & K_p * (I_y^2) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -K_p * (I_x I_t) \\ -K_p * (I_y I_t) \end{pmatrix}$$

Če je matrični sistem obrnljiv, potem lahko definiramo dve neničelni ($\neq 0$) lastni vrednosti matrike in tako ocenimo hitrost optičnega toka. Če ni obrnljiv in lahko definiramo samo eno neničelno ($\neq 0$) lastno vrednost, lahko definiramo normalni tok, sicer ne moremo oceniti hitrosti. Na sliki 9 lahko opazimo končno rešitev optičnega toka z LK metodo. Slika a predstavlja deseto sliko iz zaporedja videoposnetka "Hamburg taxi"z resolucijo 256 x 191, kjer bel taxi zavija desno, črn avto vozi naravnost v smeri desno, rdeč kombi pa vozi naravnost. Slika b prikazuje skalarje, ki predstavljajo hitrost spreminjanja lege objektov (optični tok) LK metode s parametri $\sigma = 0$, $p = 7.5$. Slika

c pa predstavlja LK metodo s parametri $\sigma = 0$, $p = 15$. Na obeh slikah LK metode so opazne oblike svetlejše barve. Te predstavljajo premikajoče se objekte. Svetlejša, kot je barva, hitreje se objekti premikajo. Zaradi predpostavke konstantne jakosti v okolici izbrane točke opažamo razdelitev slike na manjše dele, kar ne prepušča metodi prepoznavanja majhnih premikanj. Čeprav je sprememba p parametra med b in c sliko velika, opazimo, da so glavni premikajoči se objekti še vedno vidni. [4, 15]



Slika 9: Primer LK metode na Hamburg taxi zaporedju. Prva slika levo je originalna slika, vmesna slika je slika z LK metodo z parametri $\sigma = 0$ in $p = 7.5$. Tretja slika pa ima parametre $\sigma = 0$, $p = 15$. Slike so pridobljene iz [4].

2.3.2 Horn & Schunck

Medtem ko pri LK metodi iščemo vektor premika za vsak piksel, $V : \Omega \to \Re$, se pri HS išče celotno polje vektorjev premika, $V : \Omega \to \Re^2$. HS je prva metoda, ki deluje po variacijskem računu. Na začetku poglavja je bil omenjen problem odprtine (glej enačbo 2.4), ki nam da nerešljivo enačbo z dvema neznankama. Dodatna omejitev enačbi, kot je predpostavka konstantne jakosti v neki okolici pri LK, je ena izmed rešitev za enačbo. HS metoda upošteva, da se lahko sosedni piksli minimalno razlikujejo med seboj. Sledi formula HS metode:

$$E(V) = \int_{\Omega} ((I_x u + I_y v + I_t)^2 + \alpha(|\nabla u|^2 + |\nabla v|^2))dxdy$$

Prvi del definira konstantno jakost na sliki, drugi del pa definira spremembo med sosednimi piksli z določenim faktorjem α (angl. "smoothness constraint"). [3, 4, 8, 11, 15, 24] Večji kot je ta, večja je lahko razlika sosednih pikslov. Ker je to variacijska metoda, lahko rešimo funkcional z ustreznima Euler-Lagrangevima enačbama: [8]

$$0 = \Delta u - \frac{1}{\alpha}(I_x^2 u + (I_x I_y)v + I_x I_t)$$

$$0 = \Delta v - \frac{1}{\alpha}((I_x I_y)u + I_y^2 v + I_y I_t)$$

Δ je LaPlacov operator, $\Delta = \delta_{xx} + \delta_{yy}$, ki se ga v uporabi aproksimira na $\Delta u(x,y) = \bar{u}(x,y) - u(x,y)$. $\bar{u}(x,y)$ je uteženo povprečje od $u(x,y)$ izračunano s sosednimi piksli, kot je prikazano na sliki 10. V primeru, da je jakost vektorja premika v točki $(x,y) \approx 0$, se bo zaradi dodanih omejitev upoštevala jakost sosednih pikslov. Enačbi sta linearni in se lahko rešita z Gauss-Seidelovo metodo. Ker se med računanjem vsakega piksla uporablja tudi sosede, je potrebno po prvi iteraciji ponoviti računanje, tako da bodo nove hitrosti optičnega toka ocenjene z odvodom in povprečjem prejšnjih hitrosti: [3]

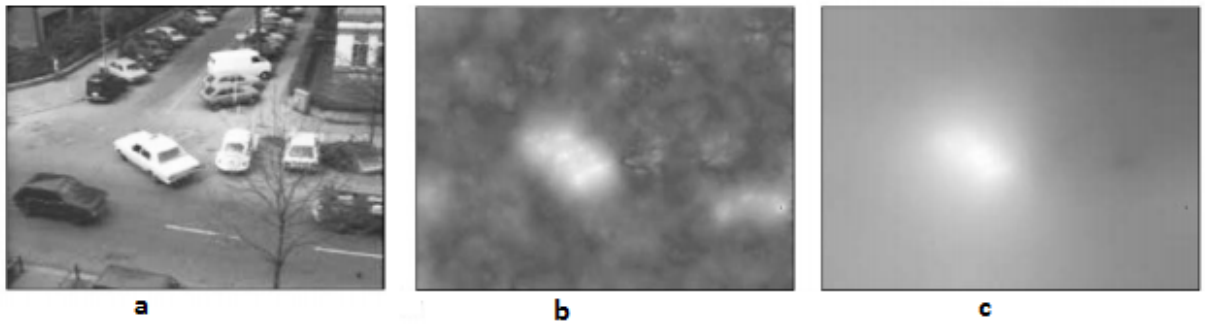
$$u^{n+1} = \bar{u}^n - \frac{I_x[I_x\bar{u}^n + I_y\bar{v}^n + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v^{n+1} = \bar{v}^n - \frac{I_x[I_x\bar{u}^n + I_y\bar{v}^n + I_t]}{\alpha^2 + I_x^2 + I_y^2}$$

Z zapisanima enačbama dobimo optični tok, ki predstavlja hitrosti premika komponent

1/12	1/6	1/12
1/6	-1	1/6
1/12	1/6	1/12

Slika 10: Primer uteženih bližnjih sosedov LaPlaceove enačbe. Slika je pridobljena iz [11].



Slika 11: Primer HS metode na "Hamburg taxi"zaporedju. Slike so pridobljene iz [4].

v zaporedju dveh slik (glej primer na sliki 11). Slika a predstavlja deseto sliko iz zaporedja videoposnetka "Hamburg taxi"(opisana že v LK metodi). Slika b predstavlja

optični tok HS metode s parametri $\sigma = 0$, $\alpha = 10^5$, medtem ko slika c predstavlja HS metoda s parametri $\sigma = 0$, $\alpha = 10^6$. Iz tega optičnega polja lahko razberemo, kako se je druga slika premaknila glede prve. Opis poprave premika sledi v tretjem poglavju.

2.3.3 Lastnosti Lucas & Kanade in Horn & Schunck metod

Metodi sta popolnoma različni: LK išče vektor premika za vsak piksel posebej, medtem ko HS išče celotno vektorsko polje slike naenkrat.

Dobre lastnosti Lucas & Kanade:

- hitro in enostavno računanje in
- pogosto dobri rezultati optičnega toka.

Dobre lastnosti Horn & Schunck:

- gost optični tok;
- deluje tudi za prepoznavanje rotacije slike, ne samo za translacijsko gibanje;
- α parameter je lahko spremenjen za upravljanje ocenjevanja dopolnitve toka v primeru, da je tok v točki $(x, y) \approx 0$ in
- z leti se je v primerjavi z LK jbolj razširil in ima veliko različnih razširitev.

Slabe lastnosti LK:

- ne prepoznava obračanja slike in
- ni bil naknadno izboljšan v primerjavi z HS.

Slabe lastnosti HS:

- manj zanesljiv pri veliki količini šuma v primerjavi z LK in
- računsko zahtevnejši

3 Popravljanje gibanja

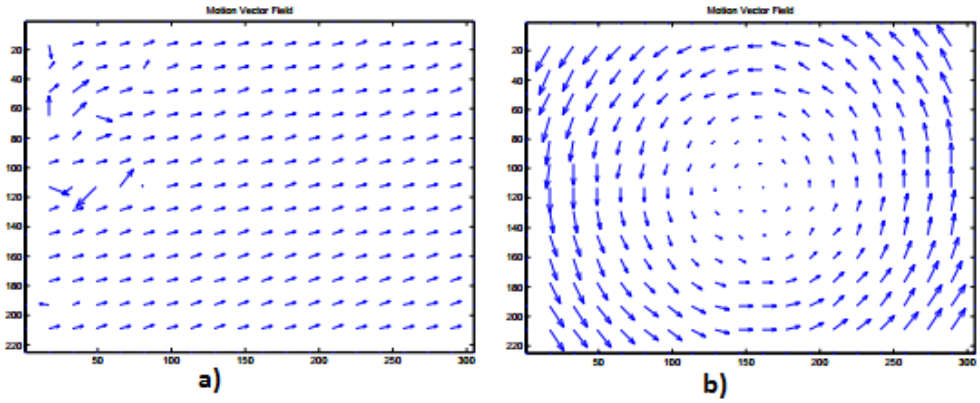
V prejšnjem poglavju smo razložili tehnike prepoznavanja gibanja med dvema slikama v videoposnetku. Te nam poiščejo vektorje premika, ki bodo uporabljeni v zadnji fazi za odpravljanje gibanja slike; tematika, obravnavana v sledečem poglavju. Gibanje lahko ločimo v dve skupini: lokalno in globalno gibanje. Pod pojmom globalno gibanje so povzeti nezaželjeni premiki/tresljaji kamere. Poznamo več vrst globalnih premikov:

- "dolly"- premik kamere naprej in nazaj;
- "track"- premik kamere levo in desno;
- "boom"- premik kamere gor pa dol;
- "pan"- rotacija kamere po Y osi;
- "tilt"- rotacija kamere po X osi in
- "roll"- rotacija kamere po osi snemanja.

Lokalni premiki pa predstavljajo spreminjanje lege nekega objekta na sliki v neki manjši okolici. Pri stabilizaciji tresljajev se upošteva globalne premike. Preden se lotimo poprave, pa je nujno potrebno razumeti, kako se giblje slika. Pri tem si pomagamo z algoritmom za prepoznavo gibanja, ki primerja varianco distribucije vektorjev premika. Če varianca ostaja med določeno mejno vrednostjo, potem je gibanje translacijsko, sicer je gibanje rotacija. Postopek je podrobneje razložen v četrtem poglavju članka v [5]. Sledi razlaga poprave gibanja po X in Y osi (translacijsko gibanje).

3.1 Popravljanje translacijskega gibanja

Sem spadajo vse slike, ki se premikajo po X in Y osi ("dolly", "track", "boom"in skromni "pan"in "tilt"premiki).
Predpostaviti moramo, da večina vektorjev premika (MV) kaže v isto smer (primer na sliki 12 a). Nato moramo izmed vseh vektorjev premika oceniti glavnega, ki predstavlja premik slike. Vektor lahko ocenimo z navadnim povprečjem ali modusom vektorjev vseh točk ali vseh blokov, da bomo izničili negativen učinek tistih vektorjev, ki nimajo



Slika 12: Primer distribucije vektorjev premika za: a translacijsko gibanje slike in b za rotacijo slike. Slike so pridobljene iz [5].

podobne smeri. [5] Poleg vektorjev, ki ne kažejo v isto smer, moramo upoštevati še, če je premik nameren ali ne. Med sabo ju lahko ločimo po njuni hitrosti. Namerni premiki so počasni in enakomerni, medtem ko imajo nezaželjeni hitre spremembe smeri skozi kratek čas. Izločevanje hitrih premikov lahko opravimo s formulo povprečja premikov, glej 3.1 (angl. "moving average"- MA). Ta deluje tako, da za izločevanje nezaželjenih hitrih in velikih premikov upošteva povprečje prejšnjih n-slik.

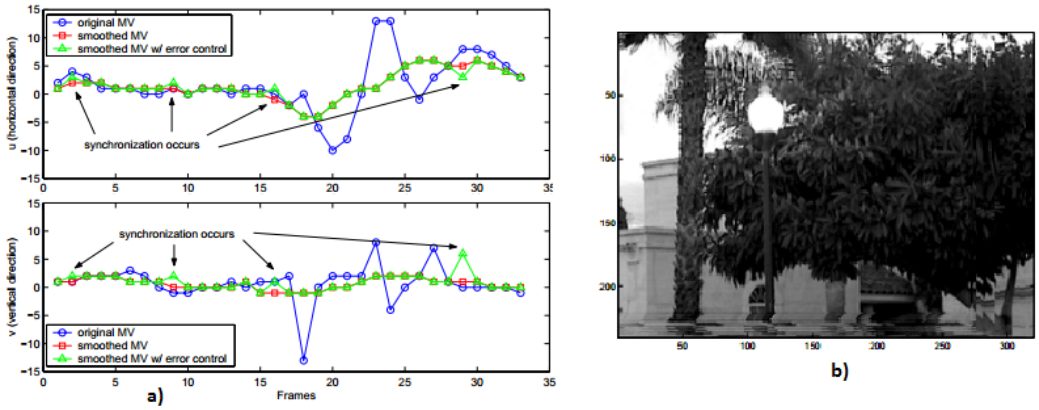
$$MA = \frac{F_N + F_{N-1} + \ldots + F_{N-n}}{n} \tag{3.1}$$

Na tak način dobimo popravljen vektor, ki naj bi predstavljal globalni premik slike. Vektor bomo uporabili za popravo naslednje slike v videoposnetku. S takim pristopom je vsaka nova slika dodana na zaporedje prejšnjih že popravljenih slik. To pa pomeni, da če se pripeti kakšna napaka, se bo le-ta prenesla naprej po zaporedju in ogrožala kakovost videoposnetka (glej sliko 13 b). Kot je razloženo v [5], se lahko poslužujemo formule, ki stalno preverja, če sta originalno in spremenjeno zaporedje preveč različna. Poglejmo si formulo 3.2:

$$|\sum_{j=1}^{i-1} u_{j,originalno} - \sum_{j=1}^{i-1} u_{j,popravljeno}|^2 + |\sum_{j=1}^{i-1} v_{j,originalno} - \sum_{j=1}^{i-1} v_{j,popravljeno}|^2 \leq \epsilon^2 \tag{3.2}$$

V formuli se upoštevajo kvadrirane razlike med u in v vektorji slik in mejno vrednostjo ϵ (angl. "threshold"). Če je mejna vrednost dovolj majhna in ni presežena, bomo upoštevali originalno sliko videoposnetka s popravo vektorja premika kot novo sliko v zaporedju. V nasprotnem primeru se bo zaporedju dodala popravljena slika, upoštevajoč formulo 3.1. Ko pride do uresničenja pogoja s formule 3.2, se popravljeno zaporedje slik sinhronizira z originalno sliko. To služi za odpravo napak, prikazanih na sliki 13 b. Vprimeru, da se sinhronizacija ne zgodi, je zelo priporočeno, da se sinhronizacijo uporabi vsakih n-slik kot preventivo slučajnim napakam.

Z zgoraj razloženim pristopom lahko popravimo hitre premike kamere po X in Y osi. Na sliki 13 vidimo popravek velikih razmikov modre barve. Tehnika deluje dobro, razen v primeru, ko je mejna vrednost ϵ prevelika. V tem primeru bi se v končnem videoposnetku opazilo nezaželene tresljaje.



Slika 13: Primer a prikazuje vrednosti u in v vektorjev pri originalnem in popravljenem zaporedju slik ter zaporedju z nadzorom nad napakami (modra, rdeča in zelena linija), b pa primer kumulative napak preko zaporedja popravljenih slik. Napaka je vidna na spodnjem delu slike in desnem robu v obliki daljših črt. Sliki sta pridobljeni iz [5].

3.2 Popravljanje rotacije slike

V tem poglavju obravnavamo stabilizacijo rotacije slike ("roll"). Postopek je enak prejšnjemu. Predpostavljamo, da smo z algoritmom za prepoznavo gibanja ugotovili, da je slika obrnjena. Najprej s pomočjo algoritmov za iskanje po blokih, fazne korelacije ali metode za optični tok dobimo vektorje premika. Obrat slike lahko definiramo z matriko vrtenja, ki opisuje vrtenje v Evklidskem prostoru, in dvema dodatnima vrednostma (angl. "offset"). Formula je opisana v 3.3.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_{offset} \\ y_{offset} \end{bmatrix} = \sqrt{\alpha^2 + \beta^2} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_{offset} \\ y_{offset} \end{bmatrix} \quad (3.3)$$

Iskani X in Y koordinati nove slike so poiskani z formulo 3.3, ker ima formula 4 nepoznane parametre (α, β, x_{offset} in y_{offset}), lahko enačbo spremenimo v matrični sistem (glej enačbo 3.4) in iz njega poiščemo nepoznane parametre in posledično tudi

koordinate X in Y za vsako točko ali blok v sliki.

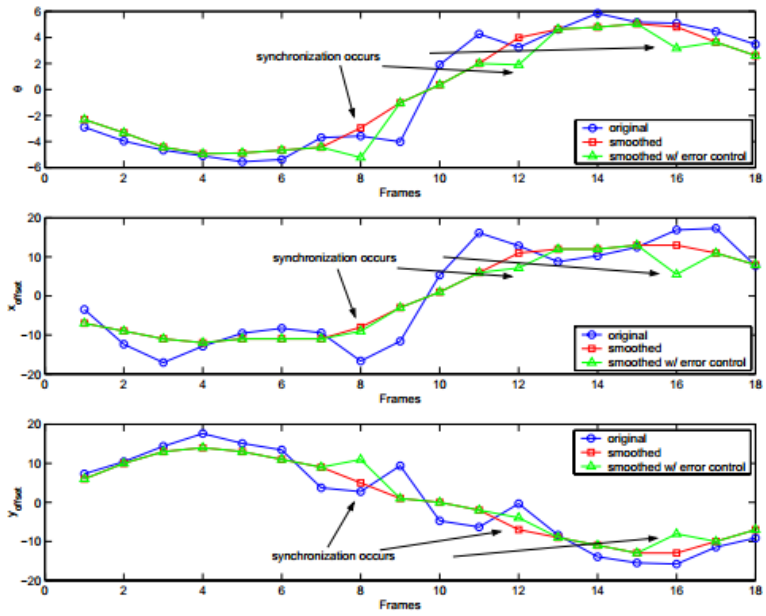
$$\begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ \vdots \\ X_N \\ Y_N \end{bmatrix} = \begin{bmatrix} x_1 & -y_1 & 1 & 0 \\ y_1 & x_1 & 0 & 1 \\ x_2 & -y_2 & 1 & 0 \\ y_2 & x_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & -y_N & 1 & 0 \\ y_N & x_N & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ x_{offset} \\ y_{offset} \end{bmatrix} \tag{3.4}$$

Naslednji korak pa je popravljanje vektorja (angl. "smooth"), to storimo s spremembo kota θ med originalno sliko ("o") in popravljeno sliko ("p"), ki bo vplival na celoten premik slike (enačba 3.5).

$$\begin{bmatrix} \alpha_p \\ \beta_p \\ x_{offset,p} \\ y_{offset,p} \end{bmatrix} = \begin{bmatrix} \sqrt{\alpha_o^2 + \beta_o^2} \cos\theta_p \\ \sqrt{\alpha_o^2 + \beta_o^2} \sin\theta_p \\ x_{offset,p} \\ y_{offset,p} \end{bmatrix} \tag{3.5}$$

S pridobljenimi parametri premika in poprave lahko končno stabiliziramo videoposnetek. Postopek je enak prejšnjemu; vsaka slika bo popravljena glede na prejšnjo sliko v zaporedju. Zaradi morebitnih napak, ki se lahko prenesejo v zaporedju (glej sliko 13 b za primer napak za translacijsko gibanje), uporabimo kontrolno enačbo 3.2. Rezultate poprave gibanja med originalnim in popravljenim zaporedjem slik lahko opazimo v sliki 14.

Pri prejšnjih dveh tehnikah poprave gibanja, ne moremo zagotovit robustnosti v primeru velike količine premikajočih se objektov (angl. "outliers"). Varianca distribucije vektorjev premika bi v tem primeru dajala nerazumljive ali dvoumne rezultate. Algoritem, ki se lepo odzove v teh primerih je RANSAC (angl. "random sample consensus"). Deluje na matematičnih funkcijah. V primeru točk v ravnini, bo algoritem razdelil vse obstoječe točke na dve skupini (angl. "inlier in outliers"). Povezal bo vse tiste premikajoče se točke, ki predpostavlja da spadajo v isto skupino. Pri visokem številu iteracij, je kakovost izbire algoritma zelo visoka. Več o algoritmu tu [21].



Slika 14: Na prvi sliki je prikazano popravljanje kota θ, druga in tretja slika pa prikazujeta popravo x in y koordinat, pri čemer modra črta prikazuje originalne vrednosti in rdeča popravljene vrednosti, medtem ko zelena predstavlja zaporedje s sinhronizacijo slik. Slika je pridobljena iz [5].

4 Implementacija

V prejšnjih poglavjih zaključne naloge je razložena teorija različnih tehnik za oceno gibanja slik videoposnetka in popravo ocenjenega gibanja. V sledečem poglavju pa sledi implementacija lastno izdelane programske opreme. Program je napisan v C++ programskem jeziku z uporabo OpenCV knjižnice. Ta knjižnjica je bila izbrana zato, ker je nastala z namenom hitre obdelave slik, poleg tega je odprtokodna in široko uporabljena.

Sledi razlaga glavnih uporabljenih funkcij in izbranih parametrov programske opreme.

4.1 Programska oprema

Odločil sem se za implementacijo BMA algoritma pri popravi translacijskega gibanja statične slike z gibajoči se objekti v sredini slike. Da lahko algoritem dostopa do posameznih slik je treba vzpostavit nemoteno povezavo do videoposnetka. S tem namenom je objekt $CvCapture*$ zgrajen, ki dekodira izbran videoposnetek v ".avi"formatu. Nato sledi ustvarjanje štirih slik $IplImage*$, dveh črno-belih $source[0]$ in $source[1]$ za uporabo pri BMA ter dveh barvanih $image[0]$ in $image[1]$, v katerih sta shranjeni originalni sliki v zaporedju. Sledi kreacija objekta $CvVideoWriter*$, ki shrani stabilizirane slike v nov videoposnetek. Prebrani sliki sta nato spremenjeni v črno-beli, in uporabljeni kot vhod za $cvCalcOpticalFlowBM$ funkcijo, ki vrne vektorje premika za vsak blok. Vektorji so ločeni v polje vektorjev Y-osi in polje vektorjev X-osi, dostopni na slikah $velocityY$ in $velocityX$. Iz vektorjev premika se oceni glavni vektor premika, ki določa globalni premik slike. X koordinato tega vektorja dobimo s povprečjem vseh vektorjev X-osi, Y koordinato pa s povprečjem vektorjev Y-osi. Z izračunanjem globalnega vektorja se lahko slika popravi. To se izvede tako, da je v vsak piksel stabilizirane slike prepisan piksel popravljenih koordinat. Zadnja popravljena slika je dodana na zaporedje za nov videoposnetek. Nato sledi branje nove slike videoposnetka, ki je preverjena z zadnjo popravljeno sliko. Isti postopek se ponovi za vse ostale slike. Glavne točke omenjenega algoritma dobimo v algoritmu 1. V spodnjem algoritmu so predstavljene samo glavne metode, uporabljene za razumevanje programske opreme, zato je lahko končna različica algoritma, uporabljena pri testiranju videoposnetkov, nekoliko različna od navedenega (glej prilogo A za programsko kodo). Programska oprema je podana v

obliki izvršilne datoteke, ki se izvede iz ukazne vrstice in ji je potrebno podati 4 glavne parametre: pot za videoposnetek, pot za izhodni stabiliziran videoposnetek, velikost bloka in koefient iskanja (glej priložen CD za primer uporabe in delovanja). Popolnejšo razlago naštetih metod in njihovih parametrov najdemo v spletni dokumentaciji OpenCV-ja.

Algoritem 1: Glavni postopek - več v prilogi A

```
1  CvCapture * input_video=cvCaptureFromFile("VideoIN.avi");
2  IplImage * source[ 2 ], * image[ 2 ];
3  source[ 0 ] = cvCreateImage (frame_size, IPL_DEPTH_8U, 1);
4  source[ 1 ] = cvCreateImage (frame_size, IPL_DEPTH_8U, 1);
5  image[ 0 ] = cvCreateImage(frame_size, IPL_DEPTH_8U, 3);
6  image[ 1 ] = cvCreateImage(frame_size, IPL_DEPTH_8U, 3);
7  CvVideoWriter *writer = cvCreateVideoWriter("D://VideoOut.mpeg",
   CV_FOURCC('P','I','M','1'), fps, frame_size,1);
8  cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, 0);
9  image[0] = cvQueryFrame( input_video );
10 while (prebranaSlika < zadnjaSlika) do
11     cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES,
       current_frame);
12     image[1] = cvQueryFrame( input_video );
13     cvCvtColor (image[ 0 ], source[ 0 ], CV_BGR2GRAY);
14     cvCvtColor (image[ 1 ], source[ 1 ], CV_BGR2GRAY);
15     IplImage * velocityX = cvCreateImage(size, IPL_DEPTH_32F, 1);
16     IplImage * velocityY = cvCreateImage(size, IPL_DEPTH_32F, 1);
17     cvCalcOpticalFlowBM (source[ 0 ], source[ 1 ],cvSize(BS,BS), cvSize(SS,
       SS), cvSize(MR, MR), 0,velocityX, velocityY);
18     image[0].at<Vec3b>(i,j) = m.at<Vec3b>(i+avrY, j+avrX);
19     cvWriteFrame( writer, image[0]);
20 releaseAll();
```

4.1.1 Testiranje

Testiranje in rezultati predstavljajo zadnji del zaključne naloge. Najprimernejše parametre je bilo težko določiti, zato končna ocena temelji na mojih subjektivnih opažanjih in občutkih. Testiranja so potekale na računalniku z Windows 7 operacijskim sistemom, štiri jedrno CPE i7 z 2,2Ghz in 6GB RAM pomnilnika.
Testirani videoposnetki se ločujejo po tipologiji premika: vodoravni in navpični premik,

rotacija in nagnjenost na stran. Vsi posnetki so posneti z 2-mi različnimi resolucijami, $320x240$ in $720x480$ z 25 FPS in višjo resolucijo $1280x720$. Spreminjal sem tudi parametre, in sicer velikost blokov $16x16$ s parametrom iskanja enakim 7, $64x64$ s parametrom iskanja 16 in velikosti $128x128$ s parametrom iskanja 16. Scena videoposnetka, je bila miza z okvirom. V okvir so bile spuščene žogice, ki simulirajo naključno premikajoče objekte. Testiranje je potekalo tudi z mirujočimi žogicami. (glej sliko 15 pridobljeno iz videoposnetka). Ker v primeru enotne barve algoritem ne bi zaznal premika, je bila

Slika 15: Primer scene iz testnega videoposnetka.

prisotna težnja po čim bolj raznoliki sliki. Prvi rezultati so pokazali, da je parameter iskanja $p = 7$ premajhen za kakovostno oceno gibanja, saj ne uspe popolnoma popraviti vseh premikov. Tudi velikost bloka se ni izkazala kot optimalna; premajhni bloki niso zaznali večjih objektov, ki so imeli večje površine z enako barvo. Testiranje z bloki velikosti $16x16$ in koeficientom 7 niso prinesli zaželjenih rezultatov. Ocena se niha med 1 in 3 na lestvici od 1 do 10. Ni vidnih razlik pri spremembi resolucije. Niti pri testiranju med mirujočimi in premikajočimi objekti. Druga skupina, z bloki velikosti $64x64$ in koeficientom 16 je prinesla boljše rezultate. Na stabiliziranemu videoposnetku so se opažale večje črne črte na robovih. Znak poprave gibanja. Opažajo se tudi prve vidne razlike med tipologijami gibanja. Ocene se gibljejo med 2 in 5. Najboljše rezultate dobimo spet pri stabiliziranju translacijskega premika. Opozorim spet, da je algoritem uporabljen za stabilizacijo vodoravnih in navpičnih tresljajev, ne pa rotacije in nagnjenosti zato tudi pričakujemo boljše rezultate za tako tipologijo premika. Kljub temu, iz radovednosti, se je k testiranju dodalo primerjavo še teh dveh dodatnih tipologij. Najboljše rezultate prinese tretja skupina. Velikosti blokov $128x128$ pikslov, opomore algoritmu prepoznavanje premika večjih objektov. Kakovost stabiliziranega posnetka je precej boljša od ostalih, čeprav ni popolna. Ocena je enaka oz. malo višja od osta-

lih. Zadnja testna skupina vsebuje videoposnetke z višjo resolucijo, uporabljena za preverjanje hitrosti izvajanja programske opreme pri večjem številu podatkov. Hitrost izvršitve ni prekomerna za uporabo implementirane metode za višje resolucije, v primeru da se uporabi bloke velikosti $128x128$ ali več. V spodnji tabeli 4.1.1 povzamemo vsa izvedena testiranja z izbrano oceno izpeljave.

Kljub testiranju z različnimi parametri, nismo dobili pričakovanih rezultatov. Stabilizacijo lahko znatno izboljšamo, če postopek ponovimo večkrat. Po dveh oz. treh ponovitvah je stabilizacija pri translacijskem gibanju popolna.

V testiranju ni vključenih videoposnetkov s premikajočim ozadjem. Zaradi radovednosti konkretnih možnosti uporabe implementirane metode, smo testirali videoposnetek resolucije $720x480$ s premikajočim ozadjem s parametri: velikost blokov 128x128 in koeficient iskanja 16. Po tretji iteraciji stabilizacije, je implementirana metoda dala povprečno dobre rezultate. Kakovosti ne moremo primerjat z rezultati poprave translacijskih premikov z mirujočem ozadjem.

Velikost bloka	Koef. iskanja	Resolucija	XY	Rotacija	Nagnjenost	Premikajoči
16x16	7	320x240	1	1	1	N
64x64	16	320x240	3	2	2	N
128x128	16	320x240	7	2	2	N
16x16	7	720x480	1	1	1	N
64x64	16	720x480	3	1	1	N
128x128	16	720x480	4	2	2	N
16x16	7	320x240	3	1	1	D
64x64	16	320x240	5	2	1	D
128x128	16	320x240	6	2	1	D
16x16	7	720x480	2	1	1	D
64x64	16	720x480	6	1	2	D
128x128	16	720x480	6	2	3	D
128x128	16	1280x720	6	ni na voljo	ni na voljo	N
128x128	16	1280x720	6	ni na voljo	ni na voljo	D

Tabela 4.1: Povzetek izvedenih testov z definiranimi velikosti blokov (NxN), koeficienti iskanja (p), izbrano resolucijo, prisotnost premikajočih se objektov (DA-NE) in oceno rezultata stabilizacije pri tipologiji gibanja (XY, rotacija, nagnjenost) z razmerjem od 1 - najslabše do 10 - popolna stabilizacija.

5 Zaključek

V prvem delu zaključne naloge so bile teoretično obravnavane 3 glavne tehnike za oceno gibanja slike. Vendar pa nobena tehnika ni popolna za vse vrste videoposnetkov. V drugem delu zaključne naloge je sledila razlaga implementirane vrste BMA z enostavno implementacijsko zahtevnostjo, ki se za rezultat poprave translacijskega gibanja odzove precej dobro. Kot je bilo omenjeno v poglavju BMA, je težava pri uporabi take tehnike računska zahtevnost. Naveden algoritem ni primeren za uporabo v realnem času. Rezultat pa je vseeno dovolj dober pri domači uporabi. Druga slaba lastnost tega algoritma je zahtevnejša implementacija poprave rotacije in nagnjenosti. To bi lahko bila možna nadgradnja navedenega algoritma, ki bi algoritem povzdignila na višji nivo uporabe.

Med programiranjem in branjem dokumentacije knjižnjice OpenCV, je oko zašlo na implementacijo funkcije BMA algoritma. Uporabljena tehnika je ES v spiralni obliki, kar še dodatno upočasni iskanje najboljšega ujemanja blokov. Izboljšava implementacije BMA je lahko tudi možna nadgradnja navedenega algoritma.

Pri tem pa priložim izdelano programsko opremo (priloga A) in izvedena testiranja (priložen CD).

Literatura

[1] S. Baker in I. Matthews Lucas-Kanade 20 Years On: A Unifying Framework. V *The Robotics Institute, Carnegie Mellon University*, 2004, 221-255. *(Citirano na strani 13.)*

[2] A. Barjatya, *Block Matching Algorithms For Motion Estimation*, Final Project Paper, IEEE, 2004. *(Citirano na straneh VII, 2, 3, 4, 5, 6, 7, 8 in 9.)*

[3] J.L. Barron, D.J. Fleet in S.S. Beauchemin System and experiments: Performance of optical flow techniques. V *International journal of computer vision*, 1994, 43-77. *(Citirano na straneh 14 in 15.)*

[4] A. Bruhn, J. Weickert in C. Schnorr Lucas & Kanade Meets Horn & Schunck: Combining Local and Global Optic Flow Methods. V *International Journal of Computer Vision*, 2005, 211-231. *(Citirano na straneh VII, 12, 13, 14 in 15.)*

[5] T. Chen, *Video Stabilization Algorithm Using a Block-Based Parametric Motion Model*, Project report, Information Systems Laboratory Department of Electrical Engineering, Stanford University, 2000. *(Citirano na straneh VII, VIII, 17, 18, 19 in 21.)*

[6] B. Dudek, *Development of software anti-shake filter for video stream*, Master of science thesis, AGH University of Science and Technology, 2011. *(Citirano na strani 2.)*

[7] R. Fisher, S. Perkins in A. Walker, *Fourier transform*, Hypermedia image processing reference, 2003. *(Citirano na straneh VII in 12.)*

[8] D.J. Fleet in A.D. Jepson, *Linear filters, Sampling & Fourier analysis*, 2009. *(Citirano na straneh 10, 12, 13 in 14.)*

[9] H. Foroosh, J.B. Zerubia in M. Berthod Extension of Phase Correlation to Subpixel Registration. V *IEEE*, 2002, 188-200. *(Ni citirano.)*

[10] E. Hildreth, S. Ullman. V *The measurement of visual motion.*, Massachusetts institute of technology, 1982. 1-25 *(Citirano na strani 2.)*

[11] K.P. Horn in B.G. Schunck Determining optical flow. V *Artificial intelligence laboratory, Massachusetts institute of technology*, 1981, 185-203. *(Citirano na straneh VII, 14 in 15.)*

[12] D. Jagiwala in S.N. Shah, Analysis of Block Matching Algorithms for Motion Estimation in H.264 Video CODEC, *IJERA - International Journal of Engineering Research and Applications* 2 (2012), 1396-1401. *(Citirano na straneh VII, 2, 3, 4, 6, 7, 8 in 9.)*

[13] M. Kaushik, Comparative Analysis of Exhaustive Search Algorithm with ARPS Algorithm for Motion Estimation, *IJAIS - International Journal of Applied Information Systems* 1 (2012), 16-19. *(Citirano na straneh 2, 4 in 8.)*

[14] Y. Liang, *Phase-Correlation motion estimation*, Final Project Paper, Stanford university, 1992. *(Citirano na strani 10.)*

[15] S. Liu, *Object trajectory estimation using optical flow* , Master of science thesis, Utah state university, 2009. *(Citirano na straneh 13 in 14.)*

[16] J. Lu in M.L. Liou, A Simple and Efficient Search Algorithm for Block-Matching Motion Estimation, *Express Letters, IEEE* 2 (1997), 429-433. *(Citirano na straneh VII, 2, 4 in 6.)*

[17] P. Lu, *Rotation Invariant Registration of 2D Aerial Images Using Local Phase Correlation*, Thesis in computer science, Department of Information Technology, Uppsala universitet, 2013. *(Citirano na strani 10.)*

[18] D.K. Maslen in D.N.. Rockmore, The Cooley-Tukey FFT and Group Theory, *Notices of the AMS* 11 (2001), 1151-1161. *(Citirano na strani 11.)*

[19] I. Nahhas in M. Drahansky, Analysis of Block Matching Algorithms with Fast Computational and Winner-update Strategies, *International Journal of Signal Processing, Image Processing and Pattern Recognition* 6 (2013), 129-138. *(Citirano na straneh 4, 5, 6, 7 in 9.)*

[20] Y. Nie in K-K. Ma, Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation, *IEEE transactions on imega processing* 11 (2002), 1442-1449. *(Citirano na strani 8.)*

[21] D. Nister, Preemptive RANSAC for Live Structure and Motion Estimation. V *IEEE International Conference on Computer Vision (ICCV 2003)* , 2003, . *(Citirano na strani 20.)*

[22] N. Paragios,Y. Chen in O. Faugeras, *Handbook of mathematical models in computer vision*, 2005. *(Ni citirano.)*

[23] L.R. Rabiner, R.W. Schafer in C.M. Rader, The Chirp z-transform Algorithm, *IEEE transactions on audio and electroacoustics* 2 (1969), 86-92. *(Citirano na strani 11.)*

[24] F. Steinbrücker, T. Pock in D. Cremers, *Large displacement optical flow computation without warping*, IEEE International Conference on Computer Vision (ICCV), 2009. *(Citirano na strani 14.)*

[25] J. Watkinson, *The engineer's guide to motion compensation*, Snell & Wilcox, Durford Mill, Petersfield, Hampshire, Handbook series, 1994. *(Citirano na strani 10.)*

[26] Y. Wu, New FFT structures Based on the Bruun Algorithm, *IEEE transactions on acoustics, speech and signal processing* 1 (1990), 188-191. *(Citirano na strani 11.)*

[27] Shan. Zhu in K-K. Ma, A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation, *IEEE transactions on imega processing* 9 (2000), 287-290. *(Citirano na straneh VII in 7.)*

Priloge

Izvorna koda implementiranega postopka

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/legacy/legacy.hpp"
#include "opencv2/video/video.hpp"
#include <iostream>
#include <math.h>

using namespace cv;
using namespace std;

int main(int argc, char ** argv){
char * input,output;
int BS=0,MR=0,SS=0;

if(argc==5){
        input = argv[1];
        BS = atoi(argv[2]);
        MR = atoi(argv[3]);
        output = argv[4];
        SS= (BS/2)-1;
        cout << "\ninput: "<< input;
        cout << "\nBS: "<< BS;
        cout << "\nMR: "<< MR ;
        cout << "\noutput: " << output ;
}else{
        cout << "Vnesi vse parametre!!!";
        return 0;
```

```
}

/* Kreacija objekta za branje videoposnetka */
CvCapture * input_video = cvCaptureFromFile(input);
if (input_video == NULL){
        cout << "Error: Can't open video.\n";
        return -1;
}

CvSize frame_size;
frame_size.height = (int) cvGetCaptureProperty(
                input_video, CV_CAP_PROP_FRAME_HEIGHT );
frame_size.width = (int) cvGetCaptureProperty( input_video,
                CV_CAP_PROP_FRAME_WIDTH );
/* Dobimo stevilo slik v posnetku: posnetki = st.zadnjega posnetka */
long number_of_frames;
cvSetCaptureProperty(input_video, CV_CAP_PROP_POS_AVI_RATIO, 1.);
number_of_frames = ( int )cvGetCaptureProperty
 (input_video, CV_CAP_PROP_POS_FRAMES);
/* Postavimo se spet na prvo mesto */
cvSetCaptureProperty(input_video, CV_CAP_PROP_POS_FRAMES, 0.);

/* kreacija slik source=crno-bela, image-barvana */
IplImage * source[ 2 ], * image[ 2 ];
source[ 0 ] = cvCreateImage (frame_size, IPL_DEPTH_8U, 1);
source[ 1 ] = cvCreateImage (frame_size, IPL_DEPTH_8U, 1);
image[ 0 ] = cvCreateImage(frame_size, IPL_DEPTH_8U, 3);
image[ 1 ] = cvCreateImage(frame_size, IPL_DEPTH_8U, 3);

/* nastavitve za writer */
double fps = cvGetCaptureProperty(input_video,CV_CAP_PROP_FPS);
CvVideoWriter *writer = cvCreateVideoWriter(output,
 CV_FOURCC('P','I','M','1'), fps, frame_size,1);

/* beremo prvo sliko */
cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, 0);
IplImage * frame = cvQueryFrame( input_video );
if (frame == NULL) {
```

```
        cout << "\nFinish!";
        return -1;
}
cvConvertImage(frame, image[ 0 ]);

int current_frame = 0;

/* while zanka do konca posnetka */
while(current_frame < number_of_frames){
        ++current_frame;

        /*beremo naslednjo sliko*/
        cvSetCaptureProperty( input_video,
                CV_CAP_PROP_POS_FRAMES, current_frame);
        frame = cvQueryFrame( input_video );
        if (frame == NULL) {
                cout << "\nFinish!\n";
                return -1;
        }
        cvConvertImage(frame, image[ 1 ]);
        cvCvtColor (image[ 0 ], source[ 0 ], CV_BGR2GRAY);
        cvCvtColor (image[ 1 ], source[ 1 ], CV_BGR2GRAY);

        /* definiramo parametre zafunkcijo BMA */
        CvSize blockSize = {BS,BS};
        CvSize shiftSize = {SS,SS};
        CvSize size = {(source[0]->width - blockSize.width +
         shiftSize.width)/shiftSize.width, (source[0]->height -
          blockSize.height + shiftSize.height)/shiftSize.height};

        IplImage * velocityX = cvCreateImage(size, IPL_DEPTH_32F, 1),
                 * velocityY = cvCreateImage(size, IPL_DEPTH_32F, 1);

        int width = image[1]->width;
        int height= image[1]->height;

        /* izracunamo Opticni tok z BMA */
        cvCalcOpticalFlowBM (source[ 0 ], source[ 1 ],cvSize(BS,BS),
```

```
 cvSize(SS, SS), cvSize(MR, MR), 0,velocityX, velocityY);

Mat imgX(velocityX,true);
Mat imgY(velocityY,true);
Mat m = cvarrToMat(image[1]);
Mat stabImg=m.clone();
stabImg.setTo(0);
float avrX = 0, avrY=0;

/* izracunamo avrX in avrY koordinati vektorjev premika*/
for (int i=0; i<imgX.rows; i++) {
        for (int j=0; j<imgX.cols; j++) {
                avrX = avrX + imgX.at<float>(i,j);
                avrY = avrY + imgY.at<float>(i,j);
        }
}

/* dobimo koordinati globalnega vektorja premika */
avrX = ceil(avrX/(imgX.rows*imgX.cols));
avrY = ceil(avrY/(imgY.rows*imgY.cols));

/* popravimo gibanje slike */
for (int i=0; i<height; i++) {
        for (int j=0; j<width; j++) {
           if(!((i+avrY) < 0 || (i+avrY) > height ||
                 (j+avrX) < 0 || (j+avrX) > width)){
                stabImg.at<Vec3b>(i,j) =
                        m.at<Vec3b>(i+avrY,j+avrX);
                }
        }
}
/* popravimo originalno sliko */
CvScalar s;
for (int i=0; i<height; i++) {
        for (int j=0; j<width; j++) {
                s.val[0]=stabImg.at<Vec3b>(i,j)[0];
                s.val[1]=stabImg.at<Vec3b>(i,j)[1];
                s.val[2]=stabImg.at<Vec3b>(i,j)[2];
```

```
                        cvSet2D(image[0],i,j,s);
                    }
                }
                /* pisemo stabilizirano sliko na nov videoposnetek*/
                cvWriteFrame( writer, image[0]);
                stabImg.release();
                imgX.release();
                imgY.release();
                m.release();
        }
        /* sprostimo resurze*/
        cvReleaseVideoWriter( &writer );
        cvReleaseImage( &source[ 1 ] );
        cvReleaseImage( &source[ 0 ] );
        cvReleaseImage( &image[ 1 ] );
        cvReleaseImage( &image[ 0 ] );
        }
```