UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

SIMON TERAN

STOHASTIČNI ALGORITEM ZA BLOČNO KOMPENZACIJO GIBANJA PRI KODIRANJU VIDEA

MAGISTRSKO DELO IZ
RAČUNALNIŠTVA IN INFORMATIKE

MENTOR: DR. BRANKO ŠTER

KAMNIK, 2008

Zahvala

Za podporo pri študiju in strokovno vodstvo pri izdelavi magistrske naloge se iskreno zahvaljujem mentorju dr. Branku Šteru.

Za dostop do vseh potrebnih orodij in strojne opreme se zahvaljujem podjetju Insilica d.o.o.

Za stalno vzpodbujanje in podporo v času študija se zahvaljujem ženi Tadeji.

In nenazadnje se zahvaljujem vsem prijateljem, znancem in drugim, ki jih v zahvali nisem izrecno omenil, čeprav so tudi oni posredno ali neposredno pripomogli k nastaku tega dela.

Kazalo

Slike

Tabele

Slovar

angleški izraz	**prevod**
(run, level, last)	(število, nivo, zadnji)
arithmetic coding	aritmetično kodiranje
artefact, artifact	nepravilnost
baseline profile	osnovni profil
bi-prediction	dvojno napovedovanje
bitrate	bitna hitrost
block	blok
blocking distortion	nepravilnosti na robu bloka
compressed	stisnjen
compression ratio	kompresijsko razmerje
context-adaptive arithmetic coding	kontekstno adaptivno aritmetično kodiranje
context-adaptive variable-length codes	kontekstno adaptivno kode spremenljive dolžine
context-based arithmetic coding	kontekstno osnovano aritmetično kodiranje
chroma	barva
data partitioning	delitev podatkov
deblocking filter	deblokirni filter
decoder	dekodirnik
discrete cosine transform (DCT)	diskretna kosinusna transformacija
discrete wavelet transform (DWT)	diskretna valčna transformacija
dynamic range	dinamični razpon
early termination	predhodno zaključevanje
encoder	kodirnik
entropy encoder	kodirnik na osnovi entropije
exponential Golomb VLC	eksponentni Golombov VLC
extended profile	razširjeni profil

angleški izraz	**prevod**
field	polje
fixed-length	nespremenljiva dolžina
frame	okvir
frame number	številka okvira
header	glava
inter prediction	zunanje napovedovanje
interlaced video	prepleteni video
intra prediction	notranje napovedovanje
Karhunen-Loeve transform	transformacija Karhunen-Loeve
list	seznam
long term	dolgotrajen
look-up table	preslikovalna tabela
luma	intenziteta
macroblock	makroblok
main profile	glavni profil
marker	oznaka
motion compensation	kompenzacija gibanja
motion estimation	ocenjevanje gibanja
motion vector	gibalni vektor
picture order count	številka slike v vrsti
predicted frame	referenčni okvir
predictive coding	napovedno kodiranje
predictor	referenčni blok
quantization	kvantizacija
reference frame buffer	pomnilnik za referenčne okvire
reorder	prerazporeditev
residual frame	okvir ostanka
resolution	ločljivost
run of zeros	niz ničel
scaling factor	skalirni faktor
scaling multiplication	skalirno množenje
short term	kratkotrajne
singular value decomposition (SVD)	razcep s singularnimi vrednostmi
skipped macroblock	preskočen makroblok
slice	rezina
spatial prediction	prostorsko napovedovanje
stream	tok
streaming media applications	multimedijske aplikacije s stalnim pretokom

angleški izraz	**prevod**
threshold	prag
trailing ones	enice na koncu sekvence
transformation	transformacija
variable-length code (VLC)	koda spremenljive dolžine
weighted prediction	uteženo napovedovanje
zero encoding	ničelno kodiranje

Poglavje 1

Povzetek

Stohastični algoritem za bločno kompenzacijo gibanja pri kodiranju videa

Glavni cilj naloge je razvoj novega algoritma za ocenjevanje gibanja pri kompresiji videa. Nov algoritem, ki je osnovan na učečih avtomatih, bo razložen in ocenjen na podlagi primerjave z algoritmi, ki se najpogosteje uporabljajo za ocenjevanje gibanja.

V prvem delu naloge je najprej predstavljen splošen pristop h kompresiranju digitalnega videa, nato pa je razložen dejanski primer standarda za kompresijo videa (*H.264* ali *MPEG-4 part 10*). Proces kompresije videa lahko v grobem razdelimo v dva koraka. V prvem koraku se odstranjuje časovna redundanca. S tem imamo v mislih predvsem podobnost med okviri, ki si sledijo v sekvenci. Drugi korak kompresije zmanjšuje prostorsko redundanco okvirov. Kot je razvidno iz nadaljevanja, je ta del zelo podoben stiskanju statične slike.

Iz predstavitve metod kompresije je razvidno, kakšni postopki se uporabljajo pri kompresiji videa. Celoten postopek odstranjevanja časovne redundance bazira na ocenjevanju gibanja. S tem izbira algoritma za ocenjevanje gibanja postane ena ključnih odločitev pri načrtovanju kodirnika videa.

Drugi del naloge je posvečen algoritmom za ocenjevanje gibanja. Najprej bodo predstavljeni najpogosteje uporabljeni algoritmi za ocenjevanje gibanja. Za tem je opisan nov način reševanja problema ocenjevanja gibanja, ki je osnovan na učečih avtomatih. Razložen je princip delovanja učečih avtomatov in uporaba le-tega pri problemu ocenjevanja gibanja. Predstavljene so tudi pomanjkljivosti osnovnega algoritma in možne izboljšave.

Na koncu naloge se primerja nov algoritem z ostalimi. Algoritme se uporabi na

treh različnih sekvencah videa. Parametra primerjave sta porabljen čas in kakovost rezultatov. Ugotovljeno je, da vsi algoritmi ponujajo približno enako razmerje med hitrostjo in kakovostjo. Na novo razvit algoritem spada v sredino lestvice algoritmov, če jih razporedimo po času izvajanja. Tu je glavni konkurent popularno *Iskanje v treh korakih*. Izkazalo se je, da z algoritmom na osnovi učečih avtomatov dobimo nekoliko boljše rezultate kot z *Iskanjem v treh korakih*.

Ključne besede: ocenjevanje gibanja, kompresija videa, MPEG, H.264, učeči avtomati.

Abstract

A stochastic algorithm for block-based motion compensation in video coding

The main purpose of this work is development of a new algorithm for motion estimation. The new algorithm, which is based on learning automata, is explained and evaluated based on comparison with frequently used algorithms for motion estimation.

In the first part a general approach to video compression is presented, later on specific example of standard (*H.264* or *MPEG-4 part 10*) for video compression is explained. The video compression process in general consists of two steps. The first step is removing of the temporal redundancy. This is possible due to similarities between neighboring frames in the video sequence. In the second step spatial redundancy is removed. As it will be seen, this part is very similar to static image compression.

From the first part the complete process of video compression is clearly presented. The concept of removing temporal redundancy is based on motion estimation. These facts make the decision, which algorithm to use, very important in video encoder development.

Second part of this work is dedicated to motion estimation algorithms. First, the most frequently used algorithms are presented. After that, a new approach to solving motion estimation problem, which is based on learning automata, is introduced. The basic ideas of learning automata are shown and also their application in motion estimation problem. Some disadvantages of this algorithm are also shown as well as possible workarounds.

At the end of this work the new algorithm is compared to other algorithms for motion estimation. All the algorithms are used on three different video sequences. They are compared regarding processing time and quality of results. It is shown that all algorithms offer approximately equal ratio between processing time and quality. The new algorithm falls in the middle if algorithms are ordered based on

processing time. The main competitor in this field is popular *Tree Step Search*. The new algorithm produces better results than his competitor *Tree Step Search*.

Keywords: motion estimation, video compression, MPEG, H.264, learning automata.

Poglavje 2

Uvod

2.1 Vizija

Zazvoni video telefon. Javiš se. Na zaslonu se pojavi obraz prijatelja. Pozdravita se ... Oba vidita malo, jasno sliko drugega na prenosnem telefonu. Po končanem klicu si prikličeš živo sliko z nogometne tekme. Ker osnovna slika ni najboljša, hitro preklopiš v dražji, a bolj kvaliteten prenos. Uživaš v zadnjih minutah napete tekme.

To je eden od scenarijev, ki so ga imeli v mislih snovalci standarda *H.264 / MPEG 4 part 10*: zelo učinkovita in zanesljiva dvosmerna video komunikacija, odporna na probleme pri prenosu podatkov.

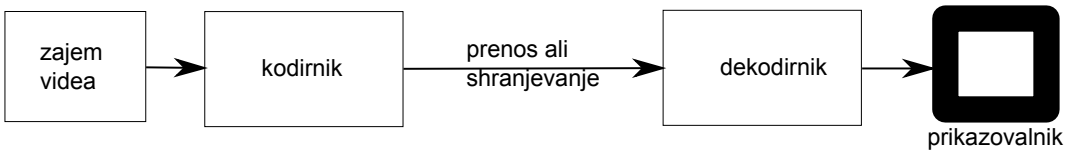
Po uspehu digitalne video industrije, predvsem digitalne televizije in DVD videa, ki sloni na standardu *MPEG 2*, so se začele pojavljati potrebe po boljšem orodju za kompresijo videa. To je vodilo v razvoj novega standarda *ITU-T recommendation H.264/ISO/IEC 14496 part 10*, znanega tudi pod imenom *MPEG 4 part 10* ali *MPEG 4 AVC* (AVC za *advance video coding*).

Hitrosti omrežij vztrajno naraščajo, prav tako velikosti medijev za hranjenje podatkov. Glede na to, da cena na prenesen bit po omrežju ali shranjen bit pada, na prvi pogled ni očitno, zakaj je kompresija videa potrebna. Kompresija videa prinaša dve pomembni prednosti. Prva je omogočanje prenosa in shranjevanja digitalnega videa, kjer ni možno uporabljati surovih (*raw*) podatkov. Na primer, na DVD disk lahko shranimo le nekaj sekund nekompresiranega videa v televizijski ločljivosti. Druga prednost je bolj učinkovita raba medijev. Na primer, če je na voljo kanal z visoko bitno hitrostjo, je bolj privlačno pošiljanje kompresiranega videa visoke ločljivosti ali več video kanalov, kot en nekompresiran video nizke ločljivosti.

2.2 Kompresija videa

Naloga kompresije videa je zmanjševanje količine podatkov, potrebnih za predstavitev digitalnega videa.

Algoritem za kompresijo videa si predstavljamo kot par kodirnik (*encoder*) in dekodirnik (*decoder*), kar je pogosto opisano z besedo *CODEC*. Na sliki 2.1 vidimo potek procesa kodiranja in dekodiranja.



Slika 2.1: Kodirnik/dekodirnik.

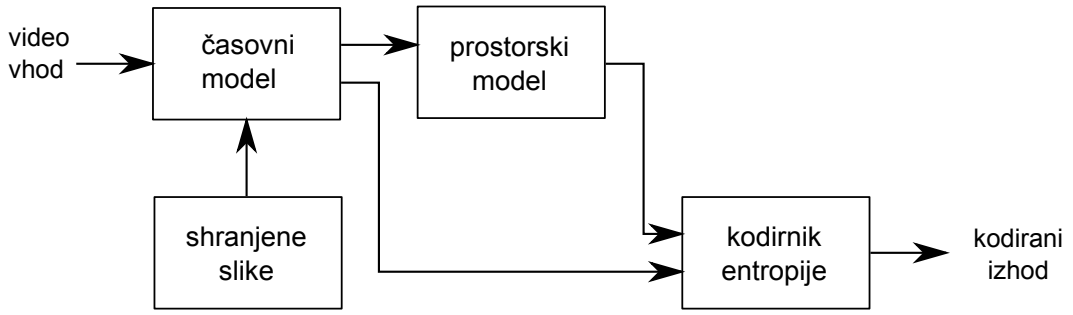
Večina CODEC-ov je izgubnih, torej kodiran in dekodiran video ni enak originalnemu. Obstajajo tudi brezizgubni CODEC-i, ki se uporabljajo le v primeru, da je ohranjanje kvalitete originalnega videa bolj pomembno kot velikost podatkov.

Digitalni video si lahko predstavljamo kot zaporedje okvirov. Algoritmi za kompresijo videa izkoriščajo tako časovno kot prostorsko redundanco v sekvenci. V časovni domeni so si zaporedni okviri pogosto zelo podobni, torej je precej redundantnih podatkov. Enako velja tudi za prostorsko domeno. V posameznem okviru so si sosednje pike pogosto zelo podobne. Pod prostorsko kompresijo si lahko predstavljamo tudi kompresijo slike.

Na sliki 2.2 vidimo blokovni diagram tipičnega kodirnika. V vhodni sekvenci se najprej odpravi časovna redundanca s pomočjo shranjenih okvirov. Na okviru ostanka (*residual frame*) se odpravi še prostorska redundanca, nato pa se generira končni video s pomočjo kodirnika na osnovi entropije (*entropy encoder*).

2.3 Ocenjevanje gibanja

Ocenjevanje gibanja (*motion estimation*) in kompenzacija gibanja (*motion compensation*) sta metodi za zmanjševanje časovne redundance v video sekvenci. Rezultat ocenjevanja gibanja je referenčni okvir, ki se v kompenzaciji odšteje od trenutnega. S tem se dobi okvir ostanka. Operacija ocenjevanja gibanja se izvaja le v kodirniku in ima velik vpliv na učinkovitost kodiranja. Dobra ocena (napoved) zmanjša količino informacije v okviru ostanka in s tem poveča učinkovitost kodiranja. Vendar pa je iskanje referenčnega okvira procesorsko zelo zahtevna naloga.



Slika 2.2: Diagram kodirnika.

Najbolj razširjen način ocenjevanja gibanja je *bločno ocenjevanje gibanja*. V tem primeru se okvir najprej razdeli na manjše bloke, nato pa se za vsak blok poišče najprimernejši referenčni blok v referenčnem okviru. Zaradi poenostavljanja iskanja območje iskanja v referenčnem okviru ni celoten okvir, temveč le njegov del okrog pozicije trenutnega bloka. Podrobnosti vidimo na sliki 2.3.

Razlika v poziciji med trenutnim in referenčnim blokom (gibalni vektor) je lahko omejena glede na način kodiranja. Tipična je omejitev, da leži referenčni blok znotraj omejenega področja okoli trenutnega bloka. Gibalni vektor lahko kaže na kateri koli blok znotraj tega področja (področja iskanja).

Cilj algoritma za ocenjevanje gibanja je poiskati vsakemu bloku gibalni vektor, ki minimizira energijo v okviru ostanka po kompenzaciji gibanja. Iskanje takega vektorja ponavadi vključuje računanje energije v ostanku na več različnih pozicijah. Iz tega sledi, da ima izbira načina merjenja energije v ostanku velik vpliv na procesorsko zahtevnost celotnega algoritma.

S sledečimi enačbami so opisani trije različni pristopi k računanju energije v ostanku.

Povprečna kvadratna napaka (*Mean Squared Error, MSE*):

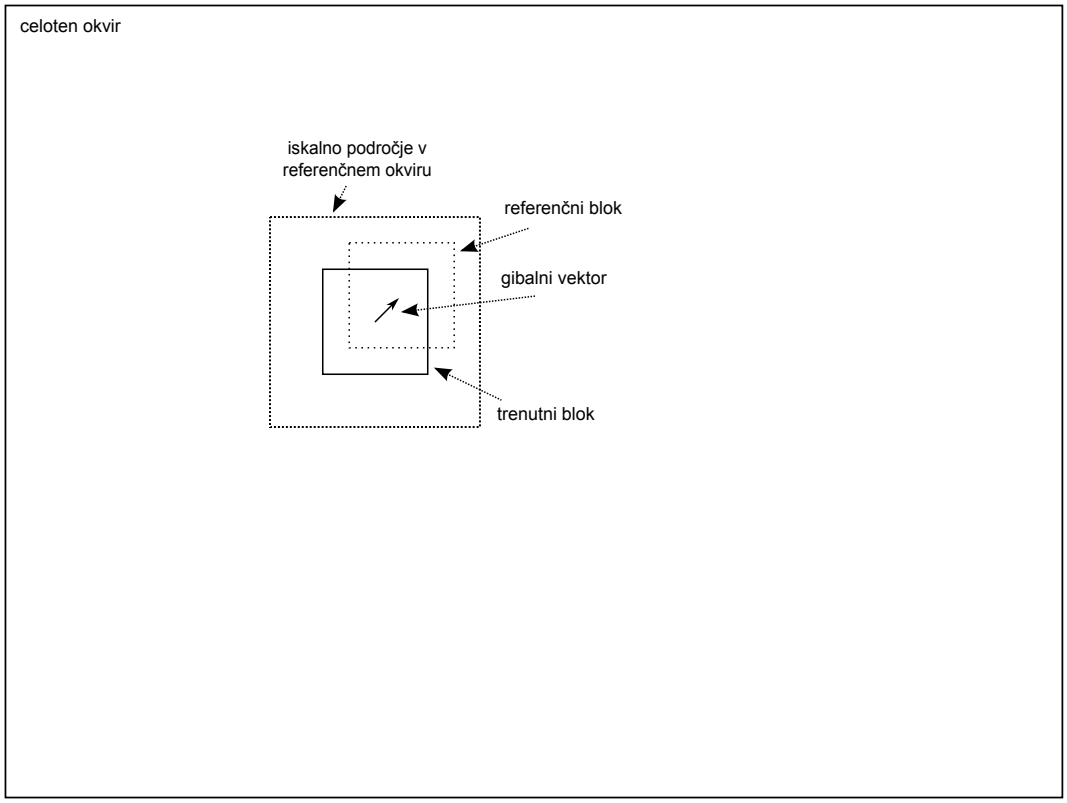
$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - R_{ij})^2. \tag{2.1}$$

Povprečna absolutna napaka (*Mean Absolute Error, MAE*):

$$MAE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \mid C_{ij} - R_{ij} \mid . \tag{2.2}$$

Vsota absolutnih napak (*Sum of Absolute Errors, SAE*):

$$SAE = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \mid C_{ij} - R_{ij} \mid . \tag{2.3}$$



Slika 2.3: Bločno ocenjevanje gibanja.

Napake se računajo na blokih velikosti $N \times N$, C_{ij} je vzorec trenutnega bloka, R_{ij} pa vzorec referenčnega bloka.

Zaradi svoje preprostosti je v praksi najpogosteje uporabljena vsota absolutnih napak.

Pri sami izbiri referenčnega področja se mora upoštevati tudi dejstvo, da je poleg bloka ostanka v dekodirnik treba poslati tudi gibalni vektor. Krajše vektorje se da opisati z manj biti (glej poglavje 4.3.10) in zato ni vedno optimalno tisto referenčno področje, ki pusti v bloku ostanka najmanj energije.

Računanje cele napake na novi lokaciji ni nujno potrebno. V praksi se računanje pogosto zaključi, ko vsota preseže trenutni minimum. Na primer, po vsakem izračunu notranje vsote ($\sum_{j=0}^{N-1} \mid C_{ij} - R_{ij} \mid$) se rezultat primerja s trenutnim najmanjšim. Če je ta manjši od trenutnega, se računanje zaključi. Ta postopek se imenuje *predhodno zaključevanje* (*early termination*).

Poglavje 3

Kompresija videa

V tem poglavju si bomo podrobneje ogledali vsakega od treh korakov, ki sestavljajo tipičen kodirnik (glej sliko 2.2).

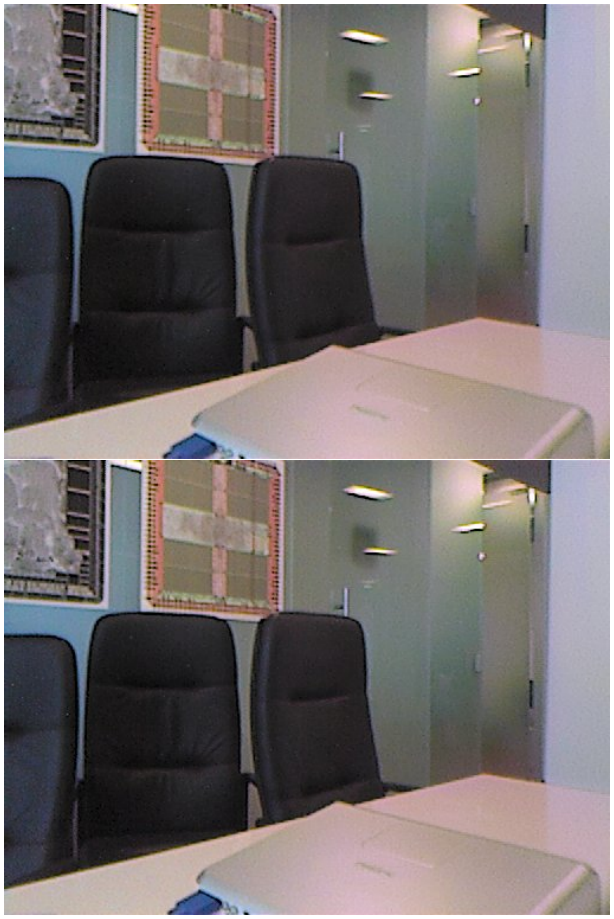
3.1 Časovni model

Cilj časovnega modela je odstraniti redundantne podatke v sekvenci okvirov s tem, da ustvari referenčni okvir in ga odšteje od trenutnega. Rezultat tega procesa je okvir ostanka, ki vsebuje manj informacije kot originalen okvir, in postopek, kako smo do tega okvira prišli. Okvir ostanka je nato kodiran in poslan v dekodirnik, kjer se s pomočjo opisa postopka generira referenčni okvir in se prišteje okviru ostanka. S tem dobimo originalen okvir. Referenčni okvir je generiran iz enega ali več prejšnjih oz. sledečih okvirov.

Najpreprostejša metoda odstranjevanja časovne redundance je uporaba predhodnega okvira kot referenčnega. Na sliki 3.1 vidimo dva zaporedna okvira v video sekvenci. Prvi okvir je uporabljen kot referenčni za drugega. Okvir ostanka lahko vidimo na sliki 3.2. Na tej sliki srednja siva pomeni, da med okviroma ni razlike, svetlejša in temnejša področja pa označujejo razliko med okviroma. Najbolj očiten problem takega napovedovanja je, da okvir ostanka še vedno vsebuje veliko informacije.

Razlika med zaporednima okviroma lahko nastane zaradi štirih razlogov:

- premikanje objektov na sceni (npr.: premikajoč se avto),
- premikanje kamere,



Slika 3.1: Zaporedna okvira v sekvenci videa.

Slika 3.2: Okvir ostanka.

- neodkrita področja (npr.: ko se avto premakne drugam, se odkrije del ozadja) in
- spremembe v osvetlitvi.

V prvem in drugem primeru gre za premikanje pik med okviroma. Tu je možno natančno predvideti smer premikanja pik med okviroma. Tako lahko dobimo pričakovano vrednost za vsako piko. S tem močno zmanjšamo količino informacije, ki jo vsebuje okvir ostanka. Vendar se ta metoda v praksi ne obnese najbolje. Natančno predvidevanje gibanja vsake pike je procesorsko zelo zahteven proces in po koncu procesa bi morali za vsako piko poslati v dekodirnik gibalni vektor, kar bi naneslo več podatkov, kot bi jih pridobili pri okviru ostanka.

Najbolj praktična in trenutno tudi najbolj razširjena je kompenzacija premikanja s pomočjo kvadratnih delov oz. blokov trenutnega okvira (*bločna kompenzacija*). Za vsak blok velikosti $M \times N$ trenutnega okvira moramo poiskati najboljšega kandidata (tudi blok velikosti $M \times N$) iz referenčnega okvira. To storimo s primerjanjem trenutnega bloka z nekaj ali vsemi bloki referenčnega okvira znotraj določenega iskalnega področja. Kriterij za izbiro je ponavadi, koliko energije ostane v bloku ostanka, ki ga dobimo z odštevanjem trenutnega bloka od referenčnega. Ta proces imenujemo *ocenjevanje gibanja* (*motion estimation*). Izbrani blok postane referenčni blok za trenutni blok. Z odštevanjem obeh blokov dobimo blok ostanka. Ta je kodiran in poslan (shranjen), prav tako pa tudi razlika med pozicijo trenutnega bloka in referenčnega bloka: gibalni vektor (*motion vector*).

Dekodirnik uporabi gibalni vektor za rekonstrukcijo referenčnega področja, dekodira blok ostanka in ga prišteje referenčnemu bloku. Tako dobi originalni blok.

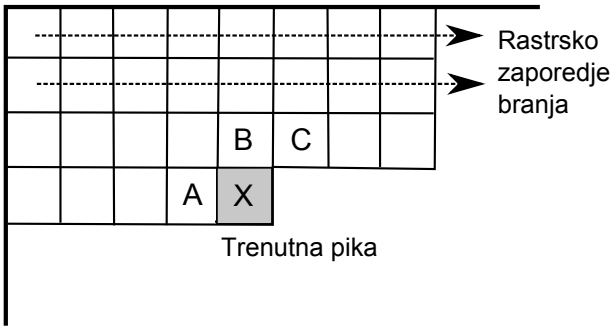
Bločna kompenzacija je priljubljena zaradi naslednjih razlogov. Je zelo preprosta, lepo se ujema s pravokotno obliko okvira in s prostorskim kodiranjem, ki je ponavadi tudi osnovano nad bloki (diskretna kosinusna transformacija), in daje zadovoljive rezultate. Metoda ima tudi nekaj pomanjkljivosti. Naravni objekti so le redko pravokotnih oblik in tudi gibanje ni poravnano s širino pike. Poleg tega metoda ni sposobna kompenzacije bolj zapletenih gibanj, kot sta na primer rotacija, 'zoom' itd.

3.2 Prostorski model

V prostorskem modelu se izvajajo operacije na trenutnem okviru. Postopki so zelo podobni kodiranju statične slike (npr. kompresija JPEG). Model je sestavljen iz treh osnovnih operacij: transformacija, kvantizacija in prerazporeditev (*reorder*).

Pri kodiranju statične slike imamo še kodiranje na osnovi entropije, ki v primeru kodiranja videa uporablja tudi nekatere podatke časovnega modela (gibalni vektor) in je zato obravnavano posebej.

Poleg naštetih postopkov pri kodiranju slike pogosto uporabimo tudi napovedovanje iz že kodiranih pik. Kot pri kompenzaciji gibanja za napovedovanje uporabimo prejšnje (ali sledeče) okvire. Tako lahko uporabimo pike, ki smo jih že kodirali. V večini primerov okvir kodiramo v rastrskem zaporedju. V tem primeru lahko pike nad in levo od trenutne uporabimo za napoved (pike A, B in C na sliki 3.3). Kodirnik izbere najboljšo od sosednjih pik in jo odšteje od trenutne pike. Dobljeno zakodira, vključno z informacijo o izbiri referenčne pike. Dekodirnik mora ustrezno sosednjo piko prišteti trenutni. Tu nastane problem, ker je kodiranje ponavadi izgubno in izbrana referenčna pika nima enake vrednosti pred in po kodiranju. Iz tega razloga kodirnik ne sme uporabljati originalnih vrednosti, temveč rekonstruirane.



Slika 3.3: Prostorsko napovedovanje.

3.2.1 Transformacija

Namen transformacije pri kodiranju slike ali videa je preslikati podatke v drugo domeno. Izbrana transformacija mora izpolnjevati nekaj kriterijev:

- Podatki v novi domeni ne smejo biti odvisni eden od drugega in večina informacije mora biti v čim manjšem številu koeficientov.
- Transformacija mora biti obrnljiva.
- Postopek transformacije mora biti čim bolj preprost (malo zahtev po pomnilniku, malo aritmetičnih operacij, lahko se izvede v aritmetiki z omejeno natančnostjo itd.).

Večino transformacij lahko razvrstimo v dve kategoriji. V prvi najdemo transformacije, ki operirajo nad bloki. To so transformacija Karhunen-Loeve, razcep s singularnimi vrednostmi (SVD) in diskretna kosinusna transformacija (DCT). Te transformacije uporabljajo bloke velikosti $N \times N$ izvornega okvira, kar ima za posledico malo zahtev po pomnilniku, in se dobro ujamejo s kompenzacijo gibanja, ki je prav tako osnovana nad bloki. Edina slaba stran je možno pojavljanje nepravilnosti na robovih blokov. V drugo kategorijo sodijo transformacije, ki se izvajajo nad celim okvirom (oz. nad večjim delom okvira) hkrati. Najbolj znan predstavnik te kategorije je diskretna valčna (wavelet) transformacija. Dokazano je bilo, da se te vrste transformacij bolje obnesejo kot transformacije, ki so osnovane nad bloki, vendar zaradi visokih zahtev po pomnilniku v praksi niso najbolj priljubljene.

3.2.2 Kvantizacija

S procesom kvantizacije se preslika signal X z določenim razponom v signal Y z reduciranim razponom, ki ga lahko predstavimo z manj biti kot original.

Proces lahko izvajamo nad skalarji. Preprost primer je zaokroževanje k najbližjemu celemu številu. Ta proces je izguben, ker v dekodirniku ni mogoče izračunati natančne originalne vrednosti. Splošen primer kvantizacije je:

$$FQ = \text{round}(\frac{X}{QP}),$$

$$Y = FQ \cdot QP$$

kjer je QP korak kvantizacije. V tabeli 3.1 vidimo primer kvantizacije v odvisnosti od koraka kvantizacije.

Kot je razvidno iz primera, je izbira kvantizacijskega koraka zelo pomembna pri kodiranju. Izbira velikega koraka bi pomenila zelo učinkovito kompresijo, po drugi strani pa bi bil dekompresiran okvir le grob približek originala. V primeru majhnega kvantizacijskega koraka bi se dekompresiran okvir zelo dobro ujemal z originalom, vendar kompresijsko razmerje ne bi bilo najboljše.

Kvantizacija se ponavadi uporablja nad transformiranimi podatki. S tem se odstranijo manj pomembni podatki (npr.: koeficienti diskretne kosinusne transformacije, ki so blizu nič).

Drug način kvantizacije je vektorska kvantizacija. V tem primeru kodirnik preslika množico vhodnih podatkov v eno samo vrednost (kodo) in dekodirnik preslika dobljeno kodo v približek originalne množice podatkov. Množica vektorjev je shranjena v tabeli kod, ki jo imata tako kodirnik kot dekodirnik. Naloga kodirnika je poiskati najbližji približek trenutnega vektorja v tabeli kod in poslati dekodirniku

Tabela 3.1: Primer kvantizacije.

	Y			
X	QP=1	QP=2	QP=3	QP=5
-4	-4	-4	-3	-5
-3	-3	-2	-3	-5
-2	-2	-2	-3	0
-1	-1	0	0	0
0	0	0	0	0
1	1	0	0	0
2	2	2	3	0
3	3	2	3	5
4	4	4	3	5
5	5	4	6	5
6	6	6	6	5
7	7	6	6	5
8	8	8	9	10
9	9	8	9	10
10	10	10	9	10
11	11	10	12	10
...				

indeks vektorja v tabeli. Dekodirnik pri dekodiranju uporabi približek originalnega vektorja, ki se nahaja na dobljenem indeksu.

3.2.3 Prerazporeditev in ničelno kodiranje

Po končanih operacijah kvantizacije in transformacije dobimo polje koeficientov, v katerem so redki različni od nič. Zaželeno je, da so ti skupaj v zaporedju. To se opravi v operaciji prerazporeditve. Tu se koeficienti iz kvantiziranega polja preberejo v posebnem vrstnem redu, ki je specifičen za vsako transformacijo. Na sliki 3.4 vidimo primer tako imenovanega *cik-cak* (*zig-zag*) branja iz bloka velikosti 8x8 pri diskretni kosinusni transformaciji. Branje poteka po naraščajočih indeksih od 1 do 64.

Slika 3.4: Cik-cak prerazporeditev za DCT.

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Izhod iz prerazporeditve je polje, kjer je ponavadi na začetku ena ali več skupin neničelnih koeficientov, ki jim sledi zaporedje koeficientov z vrednostjo nič. Večje število koeficientov z vrednostjo nič lahko predstavimo v bolj kompaktni obliki. Na primer s serijo parov (število, nivo), kjer *število* predstavlja število koeficientov enakih nič pred koeficientom z vrednostjo *nivo*.

Ker pa se v večini primerov polje zaključi z velikim številom koeficientov enakih nič (v primeru diskretne kosinusne transformacije so to koeficienti višjih frekvenc), se pogosto namesto sekvence parov (število, nivo) uporabljajo trojčki (število, nivo, zadnji), kjer *zadnji* sporoča, ali je to zadnji neničelni koeficient v bloku.

3.3 Kodirnik na osnovi entropije

Kodirnik na osnovi entropije preslika serijo podatkov, ki predstavljajo elemente v video sekvenci, v stisnjen tok podatkov, ki je primeren za pošiljanje ali shranjevanje. Vhodni simboli so lahko kvantizirani in ničelno kodirani koeficienti, gibalni vektorji, posebne oznake (potrebne za sinhronizacijo toka podatkov), glava (bloka, slike ...) ali druge dodatne informacije (to so informacije, ki niso nujne za dekompresijo).

3.3.1 Napovedno kodiranje

Pri nekaterih simbolih ostaja velika korelacija v lokalnih področjih na okviru. Na primer: povprečna vrednost (enosmerna komponenta oz. *DC* koeficient) sosednjih blokov, sosednji gibalni vektorji itd. Učinkovitost kodiranja se lahko izboljša, če napovemo vrednost trenutnega elementa s pomočjo sosednjih, že kodiranih. V tem primeru se mora dekodirniku dostaviti le razlika med napovednim (referenčnim) in trenutnim simbolom. Ta postopek imenujemo napovedno kodiranje (*predictive coding*).

3.3.2 Kode spremenljivih dolžin

Kodirnik preslika vhodne simbole v zaporedje kod spremenljivih dolžin (*variable-length code*, *VLC*). Bolj pogosti simboli se zakodirajo v krajše kode, manj pogosti pa v daljše kode. V primeru dovolj velike množice simbolov to vodi v zmanjšanje števila podatkov.

Najbolj tipičen predstavnik teh kodirnikov je Huffmanovo kodiranje. To vsakemu simbolu priredi ustrezno kodo, glede na verjetnost pojavljanja. Pri tem kodiranju je nujno izračunati verjetnosti pojavljanja vsakega simbola. Če so te natančne, se ta postopek izkaže za zelo učinkovitega. To kodiranje pa ima dve pomanjkljivosti. Prva je, da mora dekodirnik uporabiti iste kode kot kodirnik, kar v praksi pomeni, da mora kodirnik poleg kodiranih podatkov poslati tudi verjetnostno tabelo pojavljanja simbolov. S tem se učinkovitost kompresije zelo zmanjša. Druga pomanjkljivost je, da tabelo verjetnosti dobimo šele, ko je kodirana že cela sekvenca, kar pomeni dodatno zakasnitev na izhodu.

Da bi se izognili tem pomanjkljivostim, veliko algoritmov za kompresiranje uporablja vnaprej generirane tabele verjetnosti.

Slaba stran vseh metod, ki so osnovane na podlagi Huffmanovega kodiranja, je tudi, da so zelo občutljive na napake pri prenosu. Če se pojavi napaka, se le-ta razširi na

celo sekvenco v dekodirniku.

3.3.3 Aritmetično kodiranje

Pomanjkljivost vseh kodirnikov, ki preslikajo v kode spremenljivih dolžin, je, da se simbolu vedno dodeli celo število bitov. To ni optimalno, ker optimalna vrednost ponavadi ni celo število. Na primer simbol z verjetnostjo 0,5 je v najboljšem primeru predstavljen z enim bitom.

Aritmetično kodiranje predstavlja alternativo Huffmanovem kodiranju in se lahko bolj približa teoretični maksimalni kompresiji. Aritmetično kodiranje kodira sekvenco simbolov v eno ulomljeno število.

3.4 Model CODEC-a

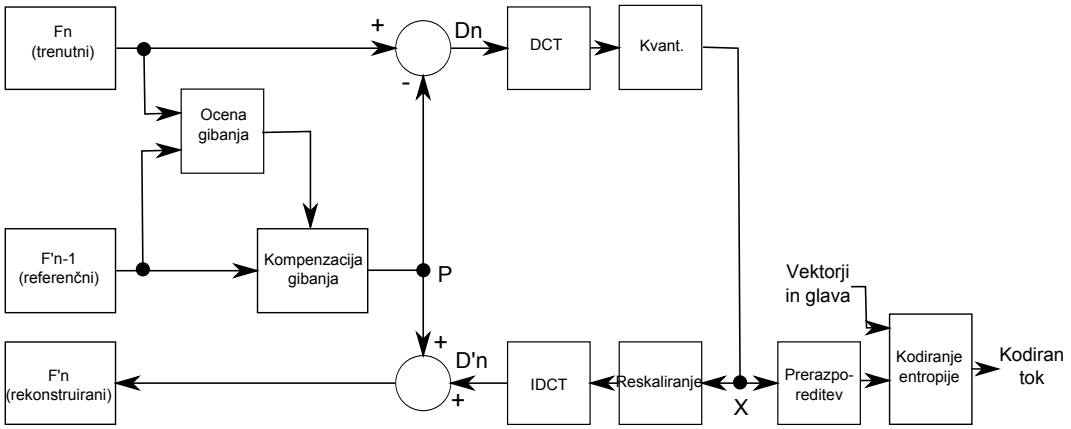
Velika večina video CODEC-ov sloni na istem modelu, ki vsebuje ocenjevanje in kompenzacijo gibanja, transformacijo in kodirnik na osnovi entropije. Ta model je pogosto opisan kot hibridni *DPCM/DCT* (*Differential Pulse Code Modulation/Discrete Cosine Transform*) model CODEC-a. Vsi CODEC-i, ki so združljivi s *H.261*, *H.263*, *MPEG 1*, *MPEG 2*, *MPEG 4 Visual* in *H.264*, morajo imeti implementirano podobno množico kodirnih in dekodirnih funkcij. Seveda je veliko razlik v podrobnostih med standardi in različnimi izvedbami.

Sliki 3.5 in 3.6 prikazujeta hibridni DPCM/DCT kodirnik in dekodirnik. V kodirniku je okvir n (F_n) procesiran v kodiran tok, v dekodirniku je kompresiran tok dekompresiran, okvir v video sekvenci pa je rekonstruiran.

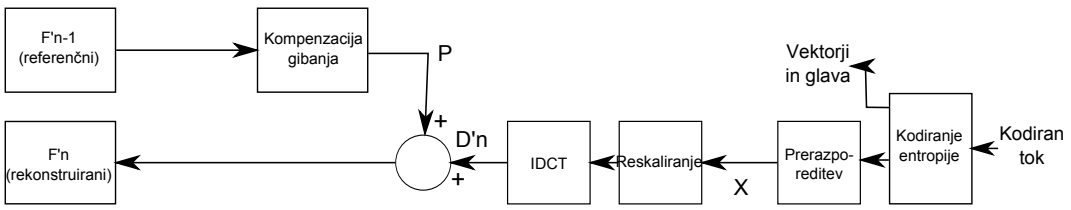
V kodirniku obstajata dve poti. Prva gre od leve proti desni, ta skrbi za kodiranje videa, in druga od desne proti levi, ta je zadolžena za rekonstrukcijo videa. Pot kodiranja je sestavljena iz:

1. Okvir video sekvence (F_n) je prisoten na vhodu, v procesiranje se pošlje vsak blok posebej.

2. F_n primerjamo z referenčnim okvirom. Opravi se ocenjevanje gibanja za vsak blok. S tem dobimo blok iz referenčnega okvira, ki se najbolj ujema s trenutnim. Razlika med pozicijama trenutnega bloka in bloka iz referenčnega okvira je gibalni vektor.



Slika 3.5: Model kodirnika DPCM/DCT.



Slika 3.6: Model dekodirnika DPCM/DCT.

3. Glede na gibalni vektor se generira referenčni blok P, ki se uporabi za kompenzacijo gibanja.

4. P se odšteje od trenutnega bloka, s čimer dobimo blok ostanka D.

5. D je transformiran s pomočjo diskretne kosinusne transformacije. Ponavadi je D razdeljen v podbloke velikosti 8x8 ali 4x4. Vsak od njih je posebej transformiran.

6. Vsak podblok je kvantiziran (X).

7. Koeficienti so prerazporejeni. Ničelno kodiranje.

8. V zadnjem koraku so koeficienti, ustrezni gibalni vektor in glava (*header*) kodirani v kodirniku na osnovi entropije, s čimer dobimo kodirani tok.

Pot rekonstrukcije je sestavljena iz:

1. Vsak kvantiziran blok X je reskaliran in transformiran z inverzno transformacijo, s čimer dobimo dekodiran blok ostanka D', ki pa zaradi kvantizacije ni enak D-ju.

2. Referenčni blok za kompenzacijo gibanja se doda bloku ostanka D'. Tako dobimo rekonstruirani originalni blok in generiramo cel okvir, ki ga kasneje lahko uporabimo za napovedovanje in kompenzacijo gibanja.

V dekodirniku imamo samo eno pot. To je pot od desne proti levi:

1. Kodiran tok podatkov je najprej dekodiran v dekodirniku na osnovi entropije. Rezultat tega so koeficienti, gibalni vektorji in glava.

2. Z inverzno operacijo ničelnega kodiranja in prerazporejanja dobimo transformiran in kvantiziran blok X.

3. X je reskaliran in transformiran z inverzno transformacijo. S tem se generira blok ostanka D'.

4. Dekodiran gibalni vektor je uporabljen za lociranje bloka P v prej shranjenem referenčnem okviru, ki je bil uporabljen za kompenzacijo gibanja.

5. P se prišteje D'-ju, s čimer dobimo rekonstruiran blok. S pomočjo vseh rekonstruiranih blokov dobimo dekodiran okvir F'_n.

Ko je okvir dekodiran, je pripravljen za prikaz in za shranjevanje kot referenčni okvir.

Iz slik in razlage vidimo, da del kodirnika skrbi za dekodiranje okvira. S tem zagotovimo, da imata kodirnik in dekodirnik enake referenčne okvire za napovedovanje in kompenzacijo gibanja.

Poglavje 4

Standard H.264

4.1 Uvod

Skupini, ki skrbita za standarde, *Moving Picture Experts Group (MPEG)* in *Video Coding Experts Group* sta razvili nov standard, ki zagotavlja boljše rezultate kot prejšnja standarda *MPEG 4* in *H.263*. Novi standard se imenuje AVC (*Advance Video Coding*) in je bil objavljen kot 10. del standarda *MPEG 4* (*MPEG 4 Part 10*) in *ITU-T Recommendation H.264*.

4.1.1 Terminologija

Sledi nekaj pomembne terminologije, ki se uporablja v standardu H.264:

Polje ali okvir je kodiran v *kodiran okvir*. Vsak kodiran okvir ima *številko okvira*, ki ni nujno povezana z vrstnim redom dekodiranja, in *številko okvira v vrsti*, ki definira vrsto dekodiranja okvira. Že kodiran okvir (*referenčni okvir*) lahko uporabimo za zunanje napovedovanje pri kodiranju naslednjih okvirov. Referenčni okviri so zbrani v dveh seznamih (seznam 0 in seznam 1).

Kodiran okvir je sestavljen iz *makroblokov*. Vsak od njih vsebuje polje pik velikosti 16x16 (16x16 intenzitet (*luma*) in pripadajoče število barvnih vzorcev (*chroma*): dvakrat po 8x8). Znotraj vsakega okvira so makrobloki razporejeni v rezine (*slice*). *Rezina I* lahko vsebuje le makrobloke tipa I, *Rezina P* lahko vsebuje makrobloke tipa I in P in *rezina B* lahko vsebuje le makrobloke tipa B in I.

V *makroblokih I* se uporablja notranje napovedovanje (*intra prediction*) iz dekodi-

ranih delov trenutne rezine. V *makroblokih P* se uporablja zunanje napovedovanje (*inter prediction*) iz referenčnega okvira. Za napoved vsakega makrobloka se lahko uporabi kateri koli okvir iz seznama 0. V *makroblokih B* se uporablja zunanje napovedovanje iz referenčnega okvira. Za napoved vsakega makrobloka se lahko uporabi en ali dva okvira iz seznama 0 ali 1.

4.2 Struktura H.264

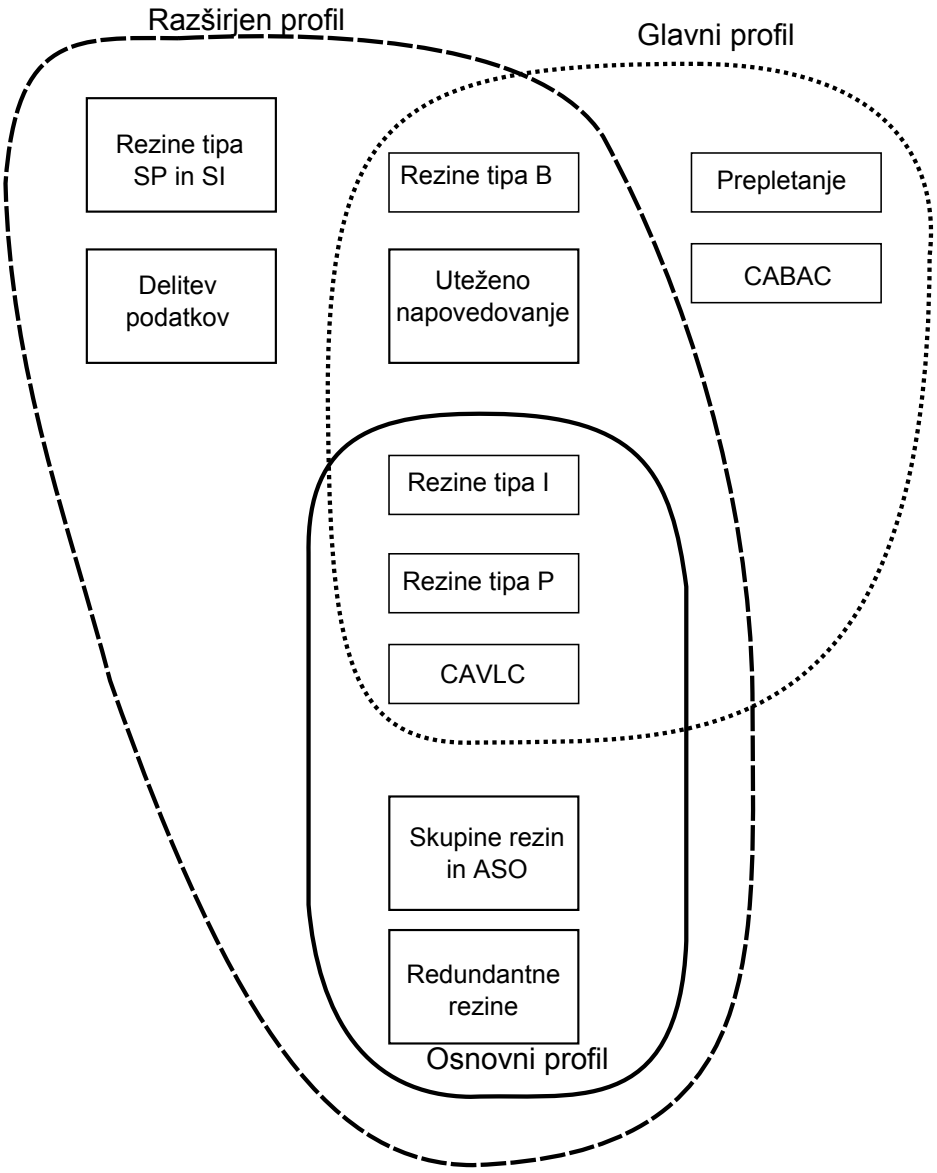
4.2.1 Profili

Standard H.264 definira tri profile. Vsak od njih podpira določeno množico funkcij kodiranja in za vsakega je specificirano, kaj morata vsebovati kodirnik in dekodirnik, ki sta združljiva s tem profilom.

V osnovnem profilu (*baseline profile*) imamo podporo za notranje in zunanje kodiranje (rezine tipa I in P) in kodiranje na osnovi entropije s kontekstno adaptivnimi kodami različnih dolžin (*context-adaptive variable-length codes, CAVLC*). V glavnem profilu (*main profile*) poleg naštetega najdemo podporo za prepleteni (*interlaced*) video, zunanje kodiranje s pomočjo rezine tipa B, zunanje kodiranje z uteženim napovedovanjem in kodiranje na osnovi entropije s kontekstno osnovanim aritmetičnim kodiranjem (*context-based aritmetic coding, CABAC*). V razširjenem profilu (*extended profile*) od naštetega ne najdemo podpore za prepleteni video in kontekstno osnovano aritmetično kodiranje, dodana pa je podpora za učinkovito preklapljanje med kodiranimi tokovi (rezine tipa SP in IP) in izboljšana je odpornost na napake z delitvijo podatkov (*data partitioning*).

Osnovni profil je namenjen predvsem video telefoniji, video konferencam in brezžičnim komunikacijam, glavni profil naj bi se uporabljal pri oddajanju televizije in shranjevanju videa, razširjeni profil pa pri multimedijskih aplikacijah s stalnim pretokom (*streaming media applications*). Seveda se vsak lahko uporabi v veliko različnih aplikacijah in niso namenjeni izključno prej omenjenim.

Slika 4.1 prikazuje odnose med profili in katera orodja podpira standard. Iz slike vidimo, da je osnovni profil podmnožica razširjenega profila, ni pa podmnožica glavnega profila.



Slika 4.1: Profili v H.264.

4.2.2 Referenčni okviri

Kodirnik H.264 lahko uporabi en ali dva okvira iz množice prej kodiranih okvirov kot referenco za napovedovanje in kompenzacijo gibanja pri vsakem zunanje kodiranem makrobloku. To omogoči kodirniku, da poišče najboljši približek v večji množici okvirov.

Kodirnik in dekodirnik vzdržujeta enega ali dva seznama referenčnih okvirov, ki so bili prej kodirani in dekodirani. Zunanje kodirani makrobloki, ki so del rezine P, lahko uporabljajo referenčni okvir iz seznama 0, makrobloki, ki so del rezine B, pa lahko izbirajo med referenčnimi okviri iz seznama 0 in 1.

4.2.3 Rezine

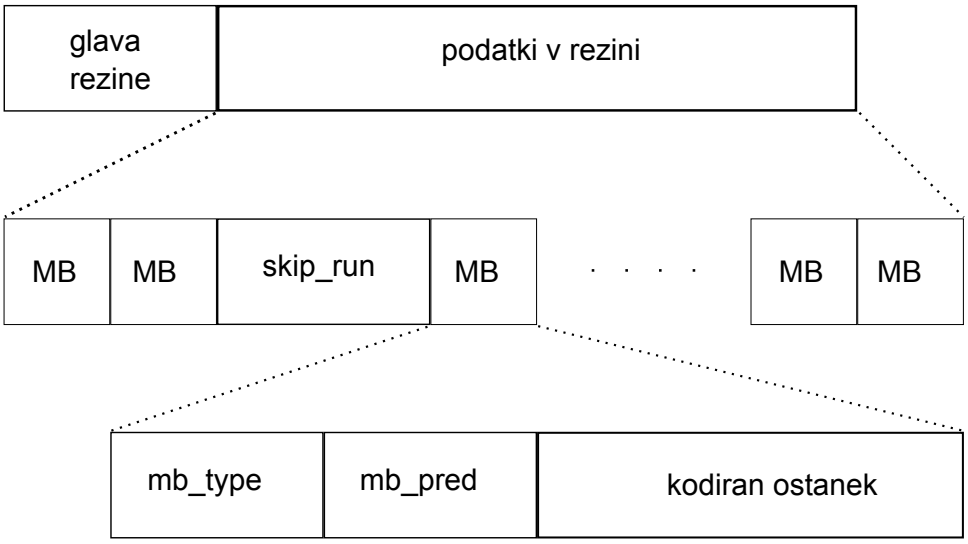
Okvir v video sekvenci je kodiran kot ena ali več rezin, kjer vsaka vsebuje celo število makroblokov (od 1 do vseh v okviru). Število makroblokov v rezini je lahko različno od rezine do rezine. Obstaja pet tipov rezin, ki so opisani v tabeli 4.1. Kodiran okvir lahko vsebuje različne tipe rezin.

Tabela 4.1: Tipi rezin.

tip rezine	**opis**	**profil**
I (*Intra*)	Vsebuje le makrobloke I (vsak blok ali makroblok je napovedan iz prej kodiranih podatkov iste rezine).	vsi
P (*Predicted*)	Vsebuje makrobloke P (vsak blok je napovedan iz enega referenčnega okvira iz seznama 0) in/ali I.	vsi
B (*Bi-predicted*)	Vsebuje makrobloke B (vsak blok je napovedan iz referenčnega okvira iz seznama 0 in/ali seznama 1) in/ali I.	razširjeni in osnovni
SP (*Switching P*)	Omogoča preklope med kodiranimi tokovi. Vsebuje makrobloke P in/ali I.	razširjeni
SI (*Switching I*)	Omogoča preklope med kodiranimi tokovi. Vsebuje makrobloke SI (posebna vrsta notranje kodiranega makrobloka)	razširjeni

Slika 4.2 prikazuje poenostavljeno sintakso kodirane rezine. V rezinini glavi določimo (poleg ostalih stvari) tip rezine in okvir, ki mu pripada. Glava lahko vsebuje tudi

ukaze za upravljanje z referenčnimi okviri. Podatkovni del rezine vsebuje serijo kodiranih makroblokov in označi preskočene (nekodirane) makrobloke. Vsak makroblok je sestavljen iz glave in kodiranih podatkov ostanka.



Slika 4.2: Sintaksa rezine.

4.2.4 Makroblok

Makroblok vsebuje kodirane podatke, ki predstavljajo področje velikosti 16x16 vzorcev video okvira (16x16 vzorcev intenzitet in dve barvni polji 8x8) in elemente, opisane v tabeli 4.2. Makrobloki so oštevilčeni v rastrskem zaporedju znotraj okvira.

4.3 Osnovni profil

4.3.1 Pregled

Osnovni profil (*Baseline Profile*) podpira kodirane sekvence, ki vsebujejo rezine I in P. Rezine I vsebujejo notranje kodirane makrobloke, ki so lahko napovedani iz prej kodiranih makroblokov iste rezine. Rezine P lahko vsebujejo notranje kodirane, zunanje kodirane in preskočene makrobloke. Zunanje kodirani makrobloki v rezinah P so napovedani iz prej kodiranih okvirov z uporabo kompenzacije gibanja z natančnostjo četrtine pike.

Tabela 4.2: Elementi makrobloka.

mb_type	Določi, ali je makroblok kodiran v notranjem ali zunanjem načinu.
mb_pred	V notranje kodiranih makroblokih določi način napovedovanja; v zunanje kodiranih makroblokih določi referenčni okvir iz seznama 0 in/ali 1 in gibalne vektorje za vsak del makrobloka, razen če je izbrana velikost bloka 8x8.
sub_mb_pred	V notranje kodiranih makroblokih pri velikosti bloka 8x8 določi velikost vsakega podbloka, referenčni okvir in vektorje gibanja za vsak podblok.
codded_block_pattern	Označi, kateri blok velikosti 8x8 vsebuje koeficiente transformacije.
mb_qp_delta	Spremeni parameter kvantizacije.
ostanek	Kodirani koeficienti transformacije, ki ustrezajo ostanku okvira po napovedovanju.

Po napovedovanju je makroblok ostanka transformiran s celoštevilčno transformacijo (osnovano na diskretni kosinusni transformaciji) in kvantiziran. Kvantizirani koeficienti so prerazporejeni in obdelani v kodirniku na osnovi entropije. V osnovnem profilu so koeficienti transformacije kodirani z uporabo kontekstno adaptivnih kod spremenljive dolžine in vsi ostali elementi z uporabo kod nespremenljive dolžine ali eksponentne Golombove kode spremenljivih dolžin. Vzporedno s procesom kodiranja so kvantizirani koeficienti tudi skalirani, transformirani z inverzno transformacijo, rekonstruirani (prištet je referenčni blok) in filtrirani z deblokirnim (*de-blocking*) filtrom, preden so shranjeni za uporabo kot referenčni okvir.

4.3.2 Upravljanje z referenčnimi okviri

Okviri, ki so bili dekodirani, so shranjeni v pomnilnik za referenčne okvire (*decoded picture buffer, DPB*), tako v kodirniku, kot tudi v dekodirniku. Oba vzdržujeta seznam prej dekodiranih okvirov (seznam referenčnih okvirov 0) za uporabo pri napovedovanju in kompenziranju gibanja v zunanje kodiranih makroblokih v rezinah P. Za napovedovanje v rezinah P lahko seznam 0 vsebuje okvire, ki so pred ali za trenutnim okvirom v vrstnem redu prikazovanja in so lahko kratkotrajni (*short term*) ali dolgotrajni (*long term*). Privzeto je, da je vsak dekodiran okvir označen

za kratkotrajnega in shranjen za napovedovanje. Kratkotrajni okviri se identificirajo po *številkah okvira*. Dolgotrajni okviri so tipično starejši, lahko se uporabljajo za napovedovanje in se identificirajo po spremenljivki *LongTermPicNum*. V DPB ostanejo, dokler se eksplicitno ne odstranijo ali zamenjajo.

Ko je okvir kodiran in rekonstruiran v kodirniku ali dekodiran v dekodirniku, je spravljen v DPB in je (a) označen kot 'neuporabljen za referenco', (b) označen kot kratkotrajen okvir, (c) označen kot dolgotrajen okvir ali (d) samo poslan na izhod (prikazovalnik). Kratkotrajni okviri so razvrščeni v pomnilniku za referenčne okvire po padajoči vrednosti spremenljivke *PicNum* (številka okvira), dolgotrajni pa prav tako po padajoči vrednosti spremenljivke *LongTermPicNum*. Kodirnik lahko spremeni privzeto ureditev v seznamu. Vsak nov okvir je dodan na seznam kratkotrajnih okvirov na pozicijo 0. Indeksi ostalih kratkotrajnih okvirov se povečajo. Če je število kratkotrajnih in dolgotrajnih okvirov enako maksimalnemu številu referenčnih okvirov, se iz seznama odstrani najstarejši kratkotrajni okvir. Učinek tega procesa je, da tako kodirnik kot dekodirnik vzdržujeta 'okno' N kratkotrajnih referenčnih okvirov, vključno s trenutnim. Ukazi za *adaptiven nadzor pomnilnika* (*adaptive memory control*), ki jih pošilja kodirnik, upravljajo z indeksi kratkotrajnih in dolgotrajnih okvirov. S temi ukazi lahko kratkotrajnega označimo kot dolgotrajnega, katerega koli kratkotrajnega ali dolgotrajnega pa lahko označimo kot 'neuporabljena za referenco'.

Kodirnik izbere referenčni okvir iz seznama 0 za kodiranje vsakega dela zunanje kodiranega makrobloka. Izbiro referenčnega okvira sporoči z indeksom, kjer je indeks 0 prvi kratkotrajni okvir, indeksi prvega dolgotrajnega okvira pa se začnejo po zadnjem kratkotrajnem.

Kodirnik pošlje *IDR* (*Instantaneous Decoder Refresh*) kodiran okvir (vsebuje rezine I ali IP), če želi izprazniti vsebino pomnilnika za referenčne okvire. Ob sprejemu kodiranega okvira *IDR* dekodirnik označi vse okvire v seznamu kot 'neuporabljene za referenco'. Prvi okvir v video sekvenci je vedno okvir *IDR*.

4.3.3 Rezine

Kodiran tok podatkov, ki ustreza osnovnemu profilu, vsebuje rezine I in P. Rezine I vsebujejo samo notranje kodirane makrobloke, rezine P pa vsebujejo zunanje kodirane makrobloke, notranje kodirane makrobloke in preskočene makrobloke. Če je makroblok označen kot preskočen, se o njem ne pošlje nobenih podatkov več. Dekoder izračuna vektor preskočenega makrobloka in ga rekonstruira s pomočjo napovedi za kompenzacijo gibanja.

Kodirnik H.264 lahko vstavi posebno ločevalno podatkovno enoto med dva kodirana

okvira. S tem označi začetek novega okvira in sporoči, kateri tipi rezin so v njem dovoljeni. Če enota *RBSP* ni uporabljena, se od dekodirnika pričakuje, da bo zaznal začetek novega okvira na podlagi glave prve rezine v okviru.

4.3.4 Napovedovanje makroblokov

Vsak kodiran makroblok je napovedan iz prej kodiranih podatkov. Notranje kodirani makrobloki so napovedani iz podatkov v trenutni rezini, ki so bili že kodirani, dekodirani in rekonstruirani. Zunanje kodirani makrobloki pa so napovedani iz prej kodiranih okvirov.

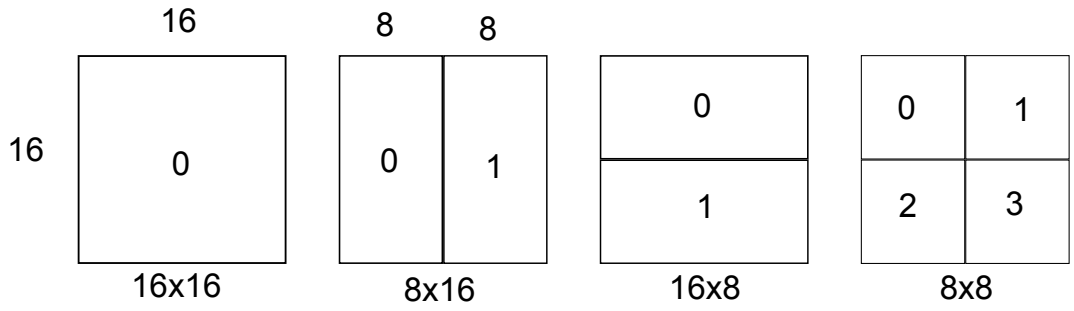
Referenčni blok je odštet od trenutnega, rezultat (ostanek) pa je kompresiran in poslan v dekodirnik skupaj z informacijo, potrebno za ponovitev procesa napovedovanja (gibalni vektor, način napovedovanja itd.). Dekodirnik si naredi identičen referenčni blok in ga prišteje dekodiranemu ostanku. Napovedovanje v kodirniku je osnovano na kodiranih in rekonstruiranih podatkih. S tem se zagotovi, da imata oba, kodirnik in dekodirnik, enake možnosti za napovedovanje.

4.3.5 Zunanje napovedovanje

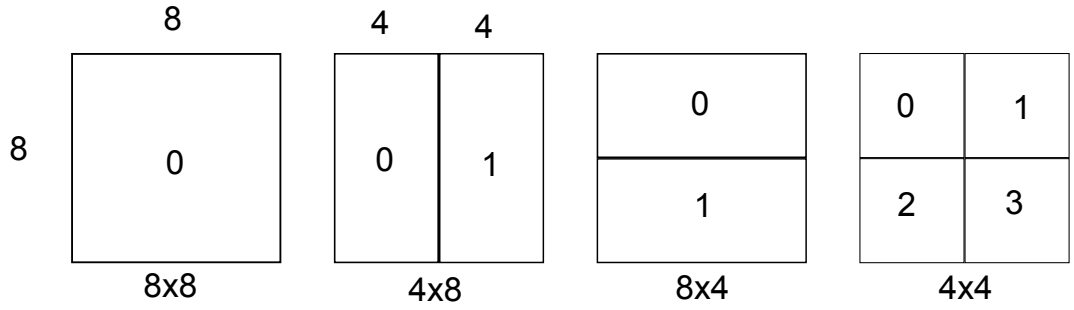
Pri zunanjem napovedovanju gre za napovedovanje iz prej kodiranih okvirov z uporabo kompenzacije, osnovane na blokih. Pomembna razlika glede na prejšnje standarde je podpora različnim velikostim blokov (od 16x16 do 4x4) in večja natančnost gibalnih vektorjev. Tu bodo opisana orodja, ki so na voljo v osnovnem profilu. Razširitve teh orodij v glavnem in razširjenem profilu vsebujejo rezine B in uteženo napovedovanje.

Drevesna struktura kompenzacije gibanja

Vsak makroblok velikosti 16x16 je lahko razdeljen na štiri načine (slika 4.3) in nato uporabljen za kompenzacijo gibanja. Lahko je kot en blok velikosti 16x16, dva 16x8, dva 8x16 ali štirje bloki velikosti 8x8. Če je izbrana razdelitev na bloke velikosti 8x8, se lahko vsak od teh nadalje razdeli na štiri načine (slika 4.4): en blok velikosti 8x8, dva 8x4, dva 4x8 ali štirje bloki velikosti 4x4. Ta razdelitev omogoči veliko število različnih možnih kombinacij znotraj vsakega makrobloka. Metoda razdeljevanja makroblokov v manjše bloke (podbloke) različnih velikosti za kompenzacijo gibanja je poznana pod imenom *drevesna struktura kompenzacije gibanja* (*tree structured motion compensation*).



Slika 4.3: Delitev makrobloka 16x16, 16x8, 8x16, 8x8.

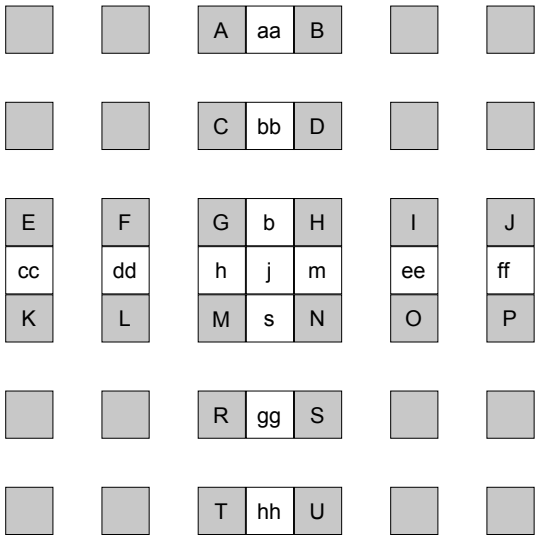


Slika 4.4: Delitev makrobloka 8x8, 8x4, 4x8, 4x4.

Ločeni gibalni vektorji so potrebni za vsak del oziroma podblok. Vsak gibalni vektor mora biti kodiran in poslan skupaj z odločitvijo o velikosti. Iz tega sledi, da izbira velikih delov (16x16, 16x8, 8x16) vodi v majhno število gibalnih vektorjev in majhno število odločitev o velikosti dela. Po drugi strani pa lahko v ostanku makrobloka po kompenzaciji gibanja še vedno ostane veliko informacije, ki jo je treba kodirati. Izbira malih delov makrobloka (8x4, 4x8, 8x8) za kompenzacijo gibanja zmanjša velikost informacije v ostanku, vendar zahteva večje število gibalnih vektorjev in podatkov o izbiri velikosti dela. S tem ima izbira velikosti blokov pri ocenjevanju gibanja velik vpliv na učinkovitost kompresije. V splošnem se večji deli uporabljajo na bolj enovitih (homogenih) delih okvira in manjši na delih, kjer je več podrobnosti in na robovih premikajočih se objektov.

Gibalni vektorji

Vsak del oziroma podblok v zunanje kodiranem makrobloku je napovedan z enako velikega področja v referenčnem okviru. Razlika med področji (gibalni vektor) ima natančnost četrtine pike. Ker vrednosti med pikami v referenčnem okviru ne obstajajo, jih je potrebno izračunati s pomočjo interpolacije sosednjih pik.



Slika 4.5: Interpolacija sredinskih pik.

Računanje interpoliranih vrednosti Najprej se izračunajo vrednosti na sredini med originalnimi pikami. Vsaka vrednost, ki je na sredini med dvema originalnima pikama (na primer b, h, m in s na sliki 4.5), je izračunana iz originalnih pik s pomočjo šestih vzorcev v filtru s končnim impulznim odzivom (FIR) z utežmi (1/32, -5/32, 20/8, 20/8, -5/32, 1/32). Na primer vzorec b je izračunan iz originalnih pik E, F, G, H, I in J:

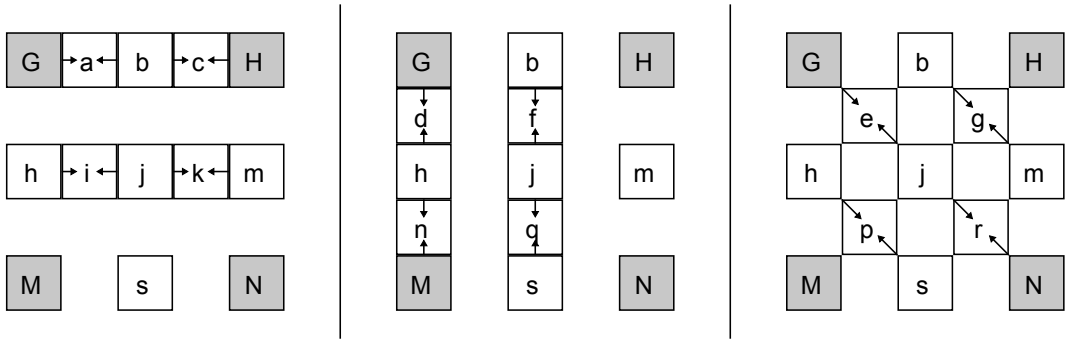
$$b = \text{round}((E - 5F + 20G + 20H - 5I + J)/32)$$

Podobno dobimo h s filtriranjem A, C, G, N, R in T. Ko se izračunajo vse horizontalne in vertikalne vrednosti, se ostale vrednosti, ki so na sredini med pikami, izračunajo z interpolacijo med šestimi vertikalnimi ali horizontalnimi sredinskimi vrednostmi, ki smo jih že izračunali. Na primer j je izračunan s filtriranjem vrednosti cc, dd, h, m, ee in ff. Interpolacijski filter s šestimi vhodi je relativno zapleten, vendar se zelo natančno ujema z originalnimi pikami in zagotavlja dobro osnovo za kompenzacijo gibanja.

Ko so izračunane vse sredinske vrednosti, lahko izračunamo vrednosti na četrtinski natančnosti. Te dobimo z linearno interpolacijo (slika 4.6). Četrtinske vrednosti, ki so vertikalno ali horizontalno poravnane s sredinsko in originalno piko (a, c, i, k, d, f, n in q na sliki), so linearno interpolirane med temi pikami. Na primer:

$$a = \text{round}((G + b)/2)$$

Ostale vrednosti na četrtinskih pozicijah so linearno interpolirane med diagonalnimi pari pik. Na primer, e je interpoliran med G in j.

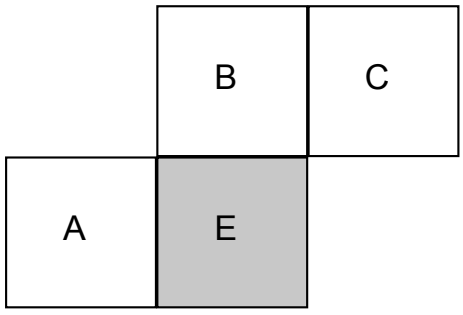


Slika 4.6: Interpolacija četrtinskih pik.

Napovedovanje gibalnih vektorjev

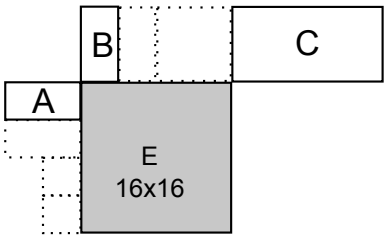
Kodiranje gibalnih vektorjev za vsak del posebej nanese veliko število bitov, ki se morajo prenesti, sploh, če so izbrani majhni deli. Gibalni vektorji sosednjih blokov so si ponavadi zelo podobni, zato se lahko vsak gibalni vektor napove iz sosednjega, že kodiranega dela. Napovedni vektor (*motion vector predicted, MVp*) se dobi na podlagi prej izračunanih gibalnih vektorjev. Kodirana in poslana je razlika (*motion vector difference MVD*) med trenutnim vektorjem in napovednim. Metoda generiranja napovednega gibalnega vektorja je odvisna od velikosti bloka in od razpoložljivosti sosednjih vektorjev.

Naj bo E trenutni makroblok oziroma del makrobloka. Naj bo A blok ali del bloka levo od E-ja, B blok nad E in C nad in levo od E. Če je več kot en blok levo od E, je izbran najvišji za A. Če je več kot en blok nad E, je izbran najbolj levi za B. Na sliki 4.7 vidimo izbiro sosednjih blokov v primeru, da so vsi deli enako veliki. Na sliki 4.8 pa vidimo izbiro sosednjih blokov v primeru, da deli niso enako veliki.



Slika 4.7: Trenuten in sosednji bloki enake velikosti.

- Pri blokih, razen pri velikostih 16x8 in 8x16, je napovedni vektor median delov



Slika 4.8: Trenuten in sosednji bloki različnih velikosti.

A, B in C.

- Za bloke velikosti 16x8 je za zgornjega napovedni vektor vektor dela B, za spodnjega pa vektor dela A.
- Za bloke velikosti 8x16 je za levega napovedni vektor vektor dela A in za desnega vektor dela C.
- Za preskočene makrobloke se napovedni vektor generira kot v prvem primeru.

Če eden ali več prej poslanih blokov ni na voljo (npr. ni v trenutni rezini), je izbira napovednega gibalnega vektorja temu primerno spremenjena. V dekodirniku je napovedni vektor dobljen po enakem postopku kot v kodirniku in prištet prejetemu ostanku gibalnega vektorja.

4.3.6 Notranje napovedovanje

Pri načinu notranjega napovedovanja je referenčni blok P generiran na osnovi prej kodiranih in rekonstruiranih blokov iste rezine in odštet od trenutnega bloka pred kodiranjem. P je lahko generiran za vsako območje velikosti 4x4 ali 16x16 intenzitet. Skupno je devet načinov napovedovanja za bloke intenzitet velikosti 4x4, štirje za bloke intenzitet velikosti 16x16 in štirje za barvne komponente. Kodirnik tipično izbere način napovedovanja, ki daje najmanjšo razliko med referenčnim in trenutnim blokom.

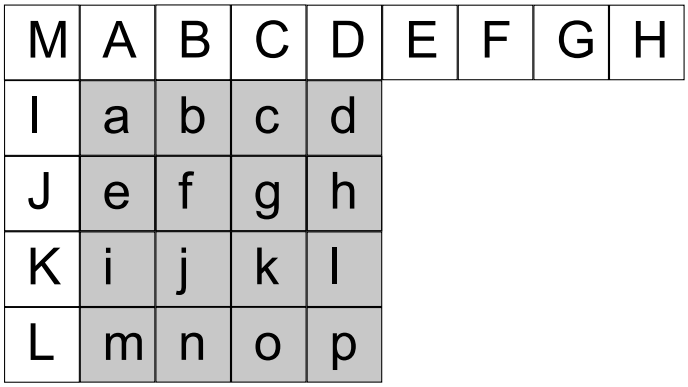
Načini napovedovanja bloka intenzitet velikosti 4x4

Vzorci nad in levo (A–M na sliki 4.9) od trenutnega bloka so že bili kodirani in so zato na voljo za napovedovanje. Vzorci a–p referenčnega bloka so izračunani iz vzorcev A–M. Načini napovedovanja so opisani v tabeli 4.3 in ilustrirani na sliki 4.10.

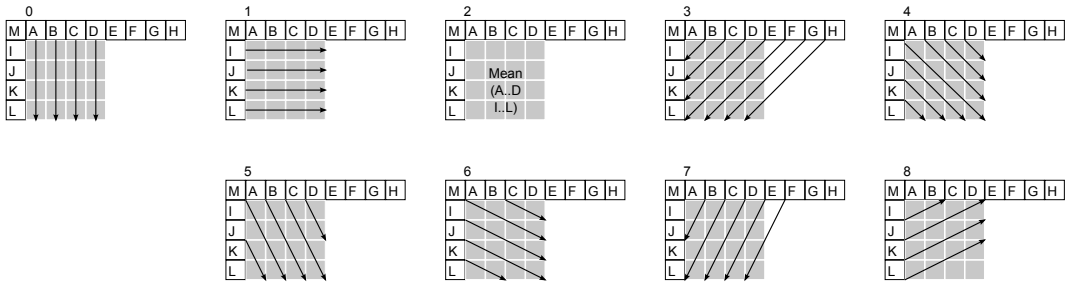
Način 2 je modificiran glede na razpoložljivost vzorcev okoli trenutnega bloka, vsi ostali načini so lahko uporabljeni le, če so vsi potrebni vzorci za napovedovanje na voljo.

Pri načinih 3–8 je referenčni blok izračunan kot uteženo povprečje. Na primer, če je izbran način 4, je vzorec d napovedan z:

$$\text{round}(B/4 + C/2 + D/4).$$



Slika 4.9: Označevanje pri 4x4 napovedovanju.



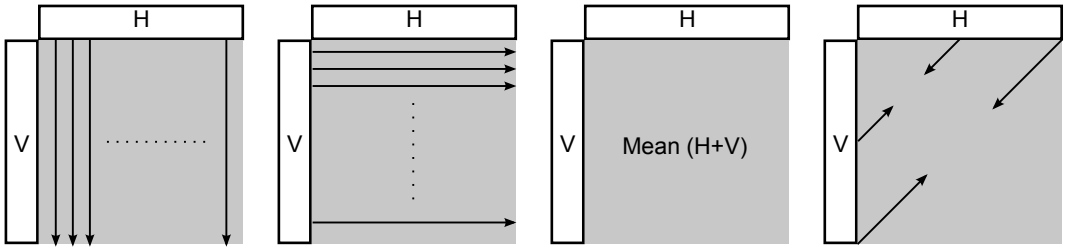
Slika 4.10: Načini napovedovanja območja intenzitet velikosti 4x4.

Načini napovedovanja bloka intenzitet velikosti 16x16

Drug način notranjega napovedovanja je napovedovanje celega makrobloka velikosti 16x16 z eno operacijo. Na voljo so štirje načini, ki so opisani v tabeli 4.4 in ilustrirani na sliki 4.11.

Tabela 4.3: Načini napovedovanja 4x4 regije intenzitet.

način 0 (vertikalno)	Zgornji vzorci A, B, C in D se ekstrapolirajo vertikalno.
način 1 (horizontalno)	Levi vzorci I, J, K in L se ekstrapolirajo horizontalno.
način 2 (DC)	Vsi vzorci se napovejo s povprečjem vzorcev od A do D in od I do L.
način 3 (navzdol-levo)	Vzorci so interpolirani med zgornjimi desnimi in spodnjimi levimi vzorci pod kotom 45 °.
način 4 (navzdol-desno)	Vzorci so ekstrapolirani od zgoraj proti desni pod kotom 45 °.
način 5 (vertikalno-desno)	Ekstrapoliranje približno pod kotom 26.6 ° desno od vertikale.
način 6 (horizontalno-navzdol)	Ekstrapoliranje približno pod kotom 26.6 ° navzdol od horizontale.
način 7 (vertikalno-levo)	Ekstrapoliranje približno pod kotom 26.6 ° levo od vertikale.
način 8 (horizontalno-navzgor)	Ekstrapoliranje približno pod kotom 26.6 ° navzgor od horizontale.



Slika 4.11: Načini napovedovanja regije intenzitet 16x16.

Tabela 4.4: Načini napovedovanja področja intenzitet velikosti 16x16.

način 0 (vertikalno)	Ekstrapolacija zgornjih vzorcev.
način 1 (horizontalno)	Ekstrapolacija levih vzorcev.
način 2 (DC)	Povprečje zgornjih in levih vzorcev.
način 3 (glajenje)	Funkcija je umeščena med zgornje in leve vzorce. Uporabna je na področjih z blagim spreminjanjem intenzitete.

Načini napovedovanja barvnega bloka velikosti 8x8

Vsaka regija barvnih komponent velikosti 8x8 notranje kodiranega makrobloka je napovedana iz prej kodiranih barvnih komponent nad in/ali levo od trenutne regije. Obe barvni komponenti uporabljata isti način napovedovanja. Štirje načini napovedovanja so zelo podobni napovedovanju blokov intenzitet velikosti 16x16, opisanem v poglavju 4.3.6 in ilustriranem na sliki 4.11. Edina razlika je številčenje načinov. Način 0 je tu povprečje (DC), način 1 je horizontalni, način 2 vertikalni in način 3 glajenje.

Sporočanje načina notranjega napovedovanja

Izbira načina notranjega napovedovanja za vsak blok velikosti 4x4 se mora posredovati dekoderju, kar lahko povzroči veliko število bitov. Vendar so načini napovedovanja med sosedi pogosto zelo podobni. Na primer, naj bodo A, B in E levi, zgornji in trenutni blok. Če sta prej kodirana bloka uporabila način 1, je velika verjetnost, da bo tudi za blok E najboljša izbira način 1. Da bi izkoristili to lastnost, se uporablja napovedovanje za sporočanje notranjega načina 4x4. Za vsak trenutni blok kodirnik in dekodirnik izračunata najbolj verjetni način napovedovanja: minimum sosednjih načinov. Če kateri od teh sosednjih blokov ni na voljo (ni v isti rezini ali ni kodiran v notranjem načinu 4x4), se za ta blok privzame vrednost 2 (DC).

Kodirnik za vsak blok velikosti 4x4 pošlje zastavico *prev_intra4x4_pred_mode*. Če je zastavica enaka '1', je uporabljena najbolj verjetna napoved. Če je zastavica '0', se pošlje še en parameter, *rem_intra4x4_pred_mode*. Če je ta parameter manjši kot napovedan, se uporabi način, kot ga sporoči parameter, sicer se parametru prišteje 1. S tem se doseže, da je različnih vrednosti parametra samo 8.

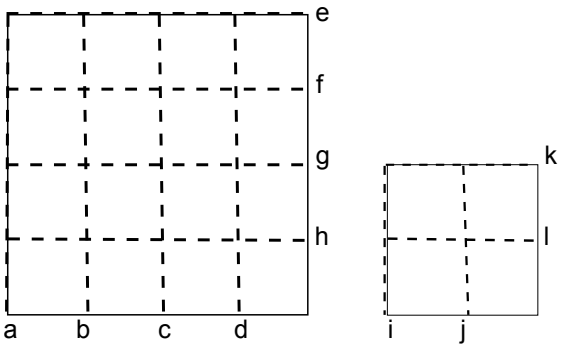
4.3.7 Deblokirni filter

Deblokirni filter je uporabljen v vsakem dekodiranem makrobloku za zmanjševanje nepravilnosti na robu bloka. Uporabljen je po inverzni transformaciji v kodirniku (pred rekonstrukcijo in shranjevanjem makrobloka za bodoče napovedi) in v dekodirniku (pred rekonstrukcijo in prikazovanjem makrobloka). Filter zgladi robove blokov in izboljša izgled dekodiranega okvira.

Filtriranje se uporabi na vertikalnih in horizontalnih robovih v blokih velikosti 4x4 v makrobloku (razen na robovih na meji rezine) v naslednjem vrstnem redu:

1. Filtriranje 4 vertikalnih robov intenzitet (po vrsti a, b, c in d na sliki 4.12).

2. Filtriranje 4 horizontalnih robov intenzitet (po vrsti *e*, *f*, *g* in *h* na sliki 4.12).
3. Filtriranje 2 vertikalnih robov obeh barvnih komponent (po vrsti *i*, *j* na sliki 4.12).
4. Filtriranje 2 horizontalnih robov obeh barvnih komponent (po vrsti *k*, *l* na sliki 4.12).



Slika 4.12: Vrstni red filtriranja robov v makrobloku.

Vsako filtriranje lahko vpliva na tri vzorce na vsaki strani roba. Slika 4.13 prikazuje štiri vzorce na vsaki strani vertikalnega ali horizontalnega roba bloka *p* in *q*. 'Moč' filtra je odvisna od trenutnega kvantiziranja, načina kodiranja sosednjih blokov in gradienta vzorcev na okviru.

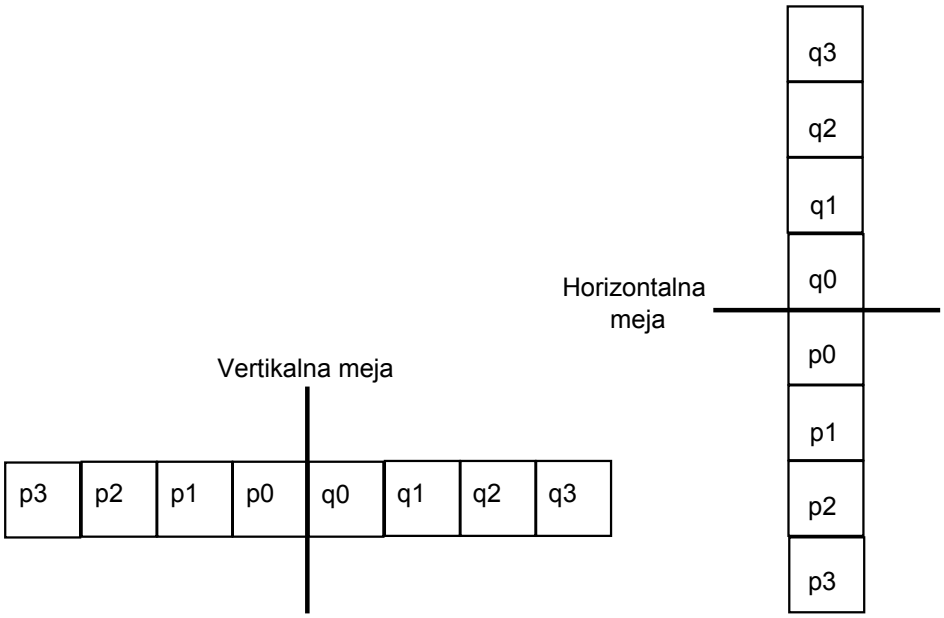
Parameter moči bS je izbran glede na pravila, opisana v tabeli 4.5.

Rezultat uporabe teh pravil je, da je filter najmočnejši na mestih, kjer bi se najverjetneje pojavile nepravilnosti na robu bloka, kot je npr. meja med notranje kodiranimi makrobloki ali meja med bloki, ki vsebujejo kodirane koeficiente.

Skupina vzorcev iz množice (*p0*, *p1*, *p2*, *q0*, *q1*, *q2*) je filtrirana, če:

- $bS > 0$ in
- $|p0 - q0| < \alpha$ in $|p1 - p0| < \beta$ in $|q1 - q0| < \beta$.

Parametra α in β sta praga, definirana v standardu. Povečujeta se s povprečnim kvantizacijskim parametrom QP blokov *p* in *q*. Posledica te filtrove odločitve je, da se izklopi, ko zazna večjo spremembo (gradient) na meji bloka v originalnem okviru. Če je QP majhen, je vse razen zelo majhne spremembe verjetno del objekta na okviru, ki se mora ohraniti, in sta zato parametra α in β majhna. Ko je QP velik, je več verjetnosti za nepravilnosti na robu bloka in sta zato parametra α in β večja.



Slika 4.13: Vzorci ob vertikalni in horizontalni meji.

Tabela 4.5: Pravila za določanje parametra moči.

p in/ali q sta notranje kodirana, meja je meja makrobloka	$bS = 4$ (najmočnejši filter)
p in q sta notranje kodirana, meja ni meja makrobloka	$bS = 3$
niti p niti q nista notranje kodirana; p in q vsebujeta kodirane koeficiente	$bS = 2$
niti p niti q nista notranje kodirana; blok ne vsebuje kodiranih koeficientov; p in q uporabljata različno referenčno sliko ali različno število referenčnih slik ali imata gibalna vektorja različna za več kot en vzorec intenzitete	bS =1
sicer	$bS = 0$ (brez filtriranja)

4.3.8 Transformacija in kvantizacija

Standard H.264 uporablja tri transformacije glede na tip ostanka podatkov, ki mora biti kodiran: Hadamardovo transformacijo za polja *DC* koeficientov intenzitet velikosti 4x4 v notranje kodiranih makroblokih, napovedanih v načinu 16x16, Hadamardovo transformacijo za polja barvnih *DC* koeficientov velikosti 2x2 (v vseh makroblokih) in transformacijo, osnovano na diskretni kosinusni transformaciji za vse ostale bloke ostankov podatkov velikosti 4x4.

Podatki v bloku so poslani v vrstnem redu, prikazanem na sliki 4.14. Če je makroblok kodiran v notranjem načinu 16x16, potem je blok, ki vsebuje transformirane *DC* koeficiente za vsak blok intenzitet velikosti 4x4, označen z -1 in poslan najprej. Za tem so poslani ostanki bloka intenzitet 0–15 v prikazanem vrstnem redu (*DC* koeficienti v makroblokih, kodiranih v notranjem načinu 16x16, niso poslani). Bloka 16 in 17, ki vsebujeta polja *DC* koeficientov barvnih komponent velikosti 2x2, sta na vrsti naslednja. Na koncu pridejo na vrsto bloki 18–25 z ostanki barvne komponente brez *DC* koeficientov.

Transformacija in kvantizacija ostankov velikosti 4x4 (bloki 0–15, 18–25)

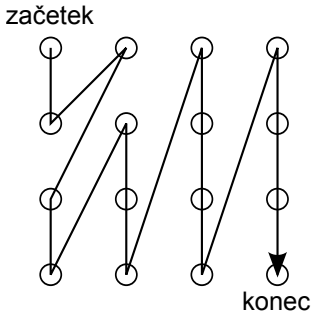
Transformacija operira nad bloki ostanka podatkov velikosti 4x4 (označeni z 0–15 in 18–25 na sliki 4.15) po kompenzaciji gibanja ali notranji kompenzaciji. Osnovana je na diskretni kosinusni transformaciji z nekaj temeljnimi razlikami:

1. Transformacija operira s celimi števili.
2. Lahko zagotovimo, da se dobljeni rezultati med kodirnikom in dekodirnikom ne bodo razlikovali.
3. Jedro transformacije se lahko izvrši z uporabo samo seštevanja in pomika (*shift*).
4. Skalirno množenje (del transformacije) je del kvantizacije, s čimer reduciramo število množenj.

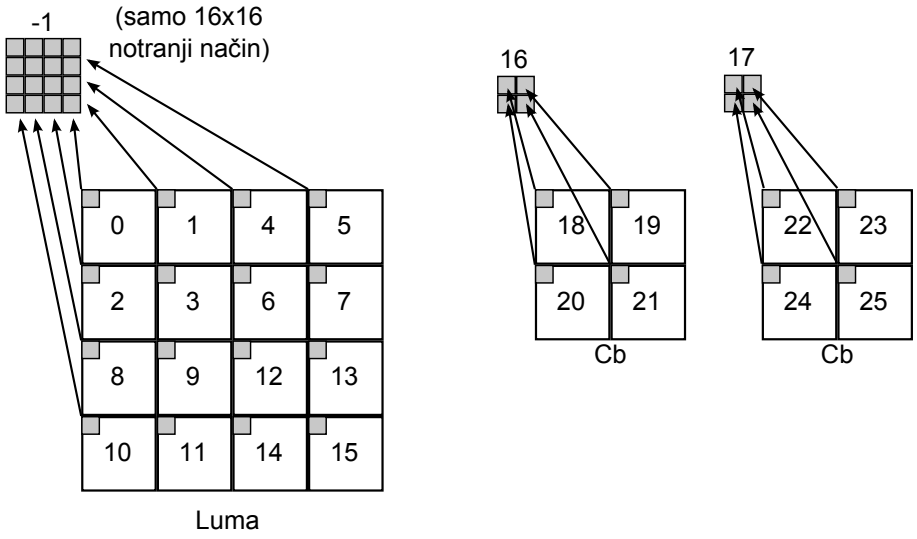
Inverzna kvantizacija (skaliranje) in inverzna transformacija sta lahko narejeni s pomočjo 16-bitne celoštevilske aritmetike s samo enim množenjem na koeficient brez izgube natančnosti.

Transformacija je podana z enačbo:

$$Y = \left(C_f X C_f^T\right) \otimes E_f \tag{4.1}$$



Slika 4.14: Vrstni red prerazporeditve v 4x4 blokih intenzitet.



Slika 4.15: Vrstni red pošiljanja ostankov blokov znotraj makrobloka.

$$= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

kjer je:

$$a = \frac{1}{2} \text{ in } b = \sqrt{\frac{2}{5}}. \tag{4.2}$$

CXC^T je jedro transformacije, E je matrika skalirnih faktorjev, simbol $\otimes$ pa pomeni, da je vsak element CXC^T množen s soležnim elementom v matriki E (skalarno množenje namesto vektorskega). Jedro transformacije se lahko izvede v celoštevilski aritmetiki samo s seštevanjem, odštevanjem in pomikom. Postskalirna operacija $\otimes E$ zahteva eno množenje vsakega koeficienta in se lahko premakne v proces kvantizacije.

Inverzna transformacija je definirana s sekvenco aritmetičnih operacij:

$$\begin{aligned} X &= C_i^t \left(Y \otimes E_i\right) C_i \\ &= \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left([Y] \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \end{aligned} \tag{4.3}$$

Standard H.264 uporablja skalarno kvantizacijo (glej poglavje 3.2.2). Mehanizem za navadno in inverzno kvantizacijo je nekoliko bolj zapleten, ker ne sme uporabljati deljenja in operacij v plavajoči vejici in mora imeti vgrajeni post in preskalirni matriki E_f in E_i.

Osnovna operacija kvantizacije je:

$$Z_{ij} = \text{round}\left(\frac{Y_{ij}}{Qstep}\right),$$

kjer je Y_{ij} koeficient prej opisane transformacije, *Qstep* določa velikost kvantizacijskega koraka in Z_{ij} je kvantiziran koeficient.

V standardu je podprtih 52 možnih vrednosti parametra *Qstep* in oštevilčeni so s parametrom QP (tabela 4.6). Vrednost parametra *Qstep* se podvoji vsak šesti QP. S široko paleto parametrov je kodirniku omogočena izbira med velikostjo kodiranega toka in kvaliteto. Vrednost QP je lahko različna za intenzitete in barvne komponente.

Postskalirni faktor a^2, $ab/2$ ali $b^2/4$ je dodan operaciji kvantiziranja. Vhodni blok X je najprej transformiran, kar da koeficiente $W = CXC^T$. Za tem je vsak koeficient

Tabela 4.6: Velikosti kvantizacijskega koraka.

QP	**Qstep**	**QP**	**Qstep**
0	0.625	18	5
1	0.6875	...	...
2	0.8125	24	10
3	0.875	...	...
4	1	30	20
5	1.125	...	...
6	1.25	36	40
7	1.375	...	...
8	1.625	42	80
9	1.75	...	...
10	2	48	160
11	2.25	...	...
12	2.5	51	224

W_{ij} kvantiziran in skaliran v isti operaciji:

$$Z_{ij} = \text{round}\left(W_{ij} * \frac{PF}{Qstep}\right)$$

FP je a^2, $ab/2$ ali $b^2/2$ glede na pozicijo (i,j).

Da bi poenostavili aritmetiko, je faktor $(PF/Qstep)$ v referenčnem modelu uporabljen kot množenje s faktorjem MF in premik desno. S tem se lahko izognemo množenju:

$$Z_{ij} = \text{round}\left(W_{ij} * \frac{MF}{2^{qbits}}\right),$$

kjer je

$$\frac{MF}{2^{qbits}} = \frac{FP}{Qstep}$$

in

$$qbits = 15 + \text{floor}(\frac{QP}{6}).$$

Celoštevilska aritmetika iz prejšnje enačbe se lahko izvede z

$$|Z_{ij}| = (|W_{ij}| * MF + f) >> qbits$$

$$\text{sign}(Z_{ij}) = \text{sign}(W_{ij}),$$

kjer je >> binarni pomik v desno.

Osnovna operacija skaliranja (inverzna operacija kvantiziranja) je

$$Y'_{ij} = Z_{ij} Qstep.$$

Preskalirni faktor inverzne transformacije je priključen tej operaciji skupaj s konstantnim skalirnim faktorjem 64:

$$W'_{ij} = Z_{ij} Qstep * PF * 64.$$

W'_{ij} je skalirni koeficient, ki je transformiran z jedrom inverzne transformacije $C_i^T W C_i$. Vrednost se na izhodu inverzne transformacije deli s 64. S tem se odstrani skalirni faktor.

Transformacija in kvantizacija *DC* koeficientov intenzitet velikosti 4x4 (samo v 16x16 notranjem načinu)

Če je makroblok kodiran v notranjem načinu 16x16, je vsak blok ostanka velikosti 4x4 najprej transformiran z uporabo jedra prej opisane transformacije ($C_f X C_f^T$). Za tem so DC koeficienti še enkrat kodirani s Hadamardovo transformacijo 4x4:

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} [W_d] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2. \tag{4.4}$$

W_d je blok 4x4 DC koeficientov in Y_d je isti blok po transformaciji. Izhodni koeficienti so kvantizirani, s čimer dobimo blok kvantiziranih DC koeficientov:

$$|Z_{D(ij)}| = (|Y_{D(ij)}| MF_{(0,0)} + 2f) >> (qbits + 1)$$

$$\text{sign}(Z_{D(ij)}) = \text{sign}(Y_{D(ij)}).$$

$MF_{(0,0)}$ je faktor na poziciji (0,0), f in $qbits$ sta definirana kot prej.

V dekodirniku je najprej izvršena inverzna Hadamardova transformacija, ki ji sledi reskaliranje (vrstni red mogoče ni pričakovan):

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} [Z_d] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right). \tag{4.5}$$

Skaliranje v dekoderju je izvedeno z:

$$W'_{D(i,j)} = W_{QD(ij)} V_{(0,0)} 2^{\text{floor}(QP/6)-2}; QP \geq 12$$

$$W'_{D(i,j)} = \left[W_{QD(i,j)}V_{(0,0)} + 2^(1 - \text{floor}(QP/6))\right] >> (2 - \text{floor}(QP/6)); QP < 12,$$

$V_{(0,0)}$ je skalirni faktor na poziciji (0,0). Ker je ta konstanten v celem bloku, se lahko izvedeta operaciji reskaliranja in inverzne transformacije v katerem koli vrstnem redu. Specificiran vrstni red je izbran, da bi se povečal dinamični razpon inverzne transformacije.

Reskalirani koeficienti W'_D so vstavljeni v prave bloke velikosti 4x4 in vsak je transformiran z inverzno transformacijo z uporabo jedra transformacije ($C_i^T W' C_i$). Pri notranje kodiranih makroblokih velikosti 16x16 je veliko informacije v DC koeficientih blokov velikosti 4x4, ki so si zelo podobni. Po dodatni transformaciji je informacija koncentrirana v majhno število pomembnih koeficientov.

Transformacija in kvantizacija barvnih *DC* koeficientov velikosti 2x2

Vsak blok barvnih komponent veliksti 4x4 je transformiran, kot je opisano v poglavju 4.3.8. DC koeficienti iz vsakega bloka barvnih koeficientov velikosti 4x4 so razdeljeni v bloke (W_D) velikosti 2x2 in še enkrat transformirani pred kvantizacijo:

$$Y_D = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} [W_d] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{4.6}$$

Kvantizacija blokov velikosti 2x2 je opravljena z:

$$|Z_{D(ij)}| = (|Y_{D(ij)}| * MF_{(0,0)} + 2f) >> (qbits + 1)$$

$$\text{sign}(Z_{D(ij)}) = \text{sign}(Y_{D(ij)}).$$

$MF_{(0,0)}$ je faktor na poziciji (0,0), f in $qbits$ sta definirana enako kot prej.

Med dekodiranjem je inverzna transformacija izvedena pred skaliranjem:

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} [Z_d] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{4.7}$$

Skaliranje je izvršeno z:

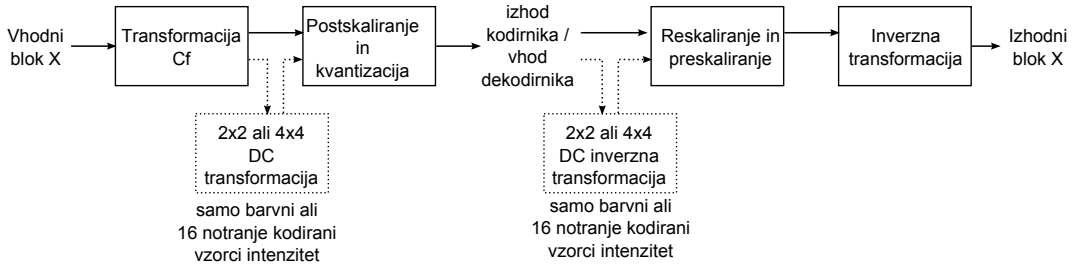
$$W'_{D(i,j)} = W_{QD(ij)}V_{(0,0)}2^{\text{floor}(QP/6)-1};\ QP \geq 6$$

$$W'_{D(i,j)} = \left[W_{QD(i,j)}V_{(0,0)}\right] >> 1;\ QP < 6.$$

Reskalirani koeficienti so dodani v ustrezne 4x4 bloke barvnih koeficientov, ki so transformirani kot v prejšnjem poglavju ($C_i^T W' C_i$).

Celoten proces transformacije, kvantizacije, reskaliranja in inverzne transformacije

Celoten proces od vhodnega bloka ostanka X do izhodnega bloka ostanka X' je opisan spodaj in ilustriran na sliki 4.16. Kodiranje:



Slika 4.16: Diagram transformacije, kvantizacije, reskaliranja in inverzne transformacije.

1. Vhod: 4x4 blok ostankov: X.
2. Jedro transformacije: $W = C_f X C_f^T$ (ki mu sledi transformacija barvnih DC koeficientov ali notranjih 16x16 DC koeficientov intenzitet).
3. Postskaliranje in kvantizacija: $Z = W * \text{round}(PF/Qstep)$.

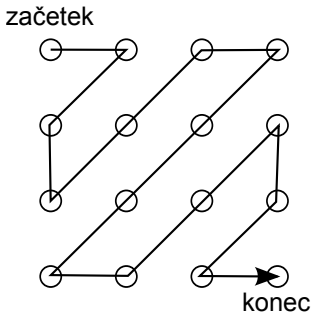
Dekodiranje (inverzna transformacija za barvne DC in notranje 16x16 DC koeficiente intenzitet):

1. Skaliranje v dekoderju (z inverzno transformacijo preskaliranje): $W' = A * Qstep * FP * 64$ (različno za barvne DC koeficiente in notranje 16x16 DC koeficiente intenzitet).
2. Jedro inverzne transformacije: $X' = C_i^T W' C_i$.
3. Postskaliranje: $X'' = \text{round}(x'/64)$.
4. Izhod: blok ostanka velikosti 4x4: X''.

4.3.9 Prerazporeditev

V kodirniku je vsak blok kvantiziranih koeficientov velikosti 4x4 preslikan v vrsto šestnajstih elementov v *cik-cak* zaporedju (slika 4.17). Enako so prerazporejeni tudi

bloki velikosti 4x4, ki jih tvorijo *DC* koeficienti v makroblokih, ki so kodirani v notranjem načinu 16x16. Tako ostane 15 koeficientov, ki se razporedijo v istem vrstnem redu, s tem da se razporejanje začne pri drugem koeficientu. Podobno se zgodi z barvnimi komponentami: najprej se preberejo in razporedijo *DC* koeficienti, ki tvorijo bloke velikosti 2x2, nato pa ostali koeficienti, začenši pri drugem v bloku.



Slika 4.17: *Cik-cak* zaporedje pri blokih intenzitet velikosti 4x4.

4.3.10 Kodiranje na osnovi entropije

Nad nivojem rezin so elementi kodirani kot binarne kode fiksne ali spremenljive dolžine. Elementi so kodirani z uporabo kod spremenljivih dolžin ali kontekstno adaptivnega aritmetičnega kodiranja (*CABAC*), odvisno od načina kodiranja na osnovi entropije. Če je parameter *entropy_coding_mode* enak 0, je blok ostanka kodiran s kontekstno adaptivnim kodiranjem spremenljivih dolžin. Pri ostalih enotah, ki so kodirane s spremenljivimi dolžinami, za to uporabimo eksponentni Golombov kodek. Primere parametrov, ki morajo biti kodirani in poslani, najdemo v tabeli 4.7.

Eksponentno Golombovo kodiranje na osnovi entropije

Eksponentne Golombove kode so spremenljive dolžine z regularno strukturo. S pregledom prvih nekaj kod (tabela 4.8) vidimo, da so konstruirane po ključu:

$$[Mzeros][1][INFO].$$

INFO je M-bitno polje, ki vsebuje informacijo. Prva koda nima ničel na začetku in polja *INFO* na koncu. Kodi za števili 1 in 2 imata 1-bitno polje *INFO*, kode za števila od 3 do 6 imajo 2-bitno *INFO* polje in tako dalje. Dolžina vsake eksponentne Golombove kode je $(2M + 1)$ bitov in je za število *num* generirana z:

$$M = \text{floor}(\log_2[num + 1])$$

Tabela 4.7: Primeri parametrov.

parameter	**opis**
Sintaktični elementi na nivoju sekvence, slike ali rezine	Glava in parametri.
Tip makrobloka (*mb_type*)	Metoda napovedovanja za vsak kodiran makroblok.
Vzorec kodiranega makrobloka	Sporoča, kateri blok v makrobloku vsebuje kodirane koeficiente.
Parameter kvantizacije	Poslan kot razlika vrednosti med prejšnjim in trenutnim parametrom *QP*.
Indeks referenčnega okvira	Sporoča referenčni okvir pri zunanjem napovedovanju.
Gibalni vektor	Poslan kot razlika od napovedanega gibalnega vektorja.
Ostanek podatkov	Podatkovni koeficienti za vsak blok velikosti 4x4 ali 2x2.

Tabela 4.8: Eksponentne Golombove kode.

Št. kode	**koda**
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

$$INFO = num + 1 - 2^M.$$

Kode se dekodirajo na sledeč način:

1. Preberi M ničel do prve 1.
2. Preberi M-bitno polje *INFO*.
3. $num = 2^M + INFO - 1$.

Parametri so preslikani v število *num* na enega izmed načinov, prikazanih v tabeli 4.9.

Tabela 4.9: Načini preslikav.

Tip preslikave	**opis**
ue	Nepredznačena direktna preslikava. Uporabljeno za tipe makroblokov, indekse referenčnega okvira in ostalo.
te	Verzija eksponentne Golombove tabele kod, kjer so kratke kode zaokrožene.
se	Preslikava predznačenih števil, uporabljena za razliko med gibalnimi vektorji, razliko med QP in podobno. K je preslikan v *num* na sledeč način (tabela 4.10): $num = 2\|k\|,\ (k \leq 0)$ $num = 2\|k\| - 1,\ (k > 0)$.
me	Preslikani simboli. Parametri so preslikani glede na tabelo specificirano v standardu. V tabeli 4.11 je del tabele *coded_block_pattern*, ki za zunanje napovedane makrobloke pokaže kateri blok velikosti 8x8 v makrobloku vsebuje neničelne koeficiente.

Vsaka od teh preslikav je namenjena izdelavi manjših kod za vrednosti, ki se pogosteje pojavljajo, in daljših kod za manj pogoste vrednosti.

Tabela 4.10: Preslikava predznačenih števil *se*.

k	**num**
0	0
1	1
-1	2
2	3
-2	4
3	5
...	...

Tabela 4.11: Del tabele *coded_block_pattern*.

coded_block_pattern	**num**
0 (ni neničelnih blokov)	0
16 (barvni DC blok neničelen)	1
1 (zgornji levi blok intenzitet velikosti 8x8 neničelen)	2
2 (zgornji desni blok intenzitet velikosti 8x8 neničelen)	3
3 (spodnji levi blok intenzitet velikosti 8x8 neničelen)	4
4 (spodnji desni blok intenzitet velikosti 8x8 neničelen)	5
32 (barvni DC in AC bloki neničelni)	6
...	...

Kontekstno osnovano kodiranje spremenljivih dolžin (*CAVLC*)

To metodo uporabimo za kodiranje ostanka prerazporejenih (*cik-cak*) blokov koeficientov velikosti 4x4 (in 2x2). CAVLC je zasnovan, da izkoristi nekaj lastnosti kvantiziranega bloka velikosti 4x4:

1. Po napovedovanju, transformaciji in kvantizaciji so bloki tipično redki (vsebujejo večinoma ničle). CAVLC uporabi kodiranje (število-nivo) za kompaktno predstavitev niza ničel.
2. Najvišji neničelni koeficienti po *cik-cak* prerazporeditvi so tipično ±1 in CAVLC kodira število visoko frekvenčnih ±1 koeficientov na zelo kompakten način.
3. Število neničelnih koeficientov v sosednjih blokih je podobno. Število koeficientov je kodirano z uporabo preslikovalne (*look-up*) tabele, njen izbor je odvisen od števila neničelnih koeficientov v sosednjih blokih.
4. Nivo neničelnih koeficientov je tipično višje na začetku prerazporejene vrste in nižje pri visoko frekvenčnih koeficientih. CAVLC to izkoristi s prilagajanjem izbire preslikovalne tabele *VLC* za nivo parametrov v odvisnosti od nivoja pravkar kodiranih koeficientov.

CAVLC kodira blok koeficientov na sledeč način:

coeff_token	Kodira število neničelnih koeficientov (*TotalCoeff*) in *TrailingOnes*.
trailing_ones_sign_flag	Predznak vrednosti *TrailingOnes*.
level_prefix	Prvi del kode neničelnih koeficientov.
level_sufix	Drugi del kode neničelnih koeficientov.
total_zeros	Kodira število ničel, ki se pojavijo po prvem neničelnem koeficientu.
run_before	Število ničel pred vsakim neničelnim koeficientom v obratni *cik-cak* ureditvi.

Kodiranje števila koeficientov in niza enic na koncu sekvence
Prvi *VLC*, *coeff_token*, kodira število neničelnih koeficientov (*TotalCoeff*) in število enic na koncu sekvence ±1 (*TrailingOnes*). *TotalCoeff* je lahko od 0 (nič neničelnih koeficientov v bloku velikosti 4x4) do 16 (vsi neničelni) in *TrailingOnes* je lahko od 0 do 3. Če so na koncu več kot trije ±1 koeficienti, so le zadnji trije tretirani kot 'poseben primer', ostali pa kot navadni koeficienti. Na izbiro so štiri možnosti preslikovalne tabele za kodiranje *coeff_token* v blokih velikosti 4x4. Izbira tabele je odvisna od števila neničelnih koeficientov v levem in zgornjem prej kodiranem bloku. Parameter *nC* je kodiran na sledeč način:

- Če sta na voljo levi in zgornji blok: $nC = \text{round}((nA + nB)/2)$.
- Če je na voljo samo zgornji blok: $nC = nB$.
- Če je na voljo le levi blok: $nC = nA$.
- Če na voljo ni nobenega: $nC = 0$.

Parameter nC izbere preslikovalno tabelo (tabela 4.12), tako da se izbira *VLC* prilagodi številu kodiranih koeficientov v sosednjih blokih. Tabela 1 je nagnjena k manjšemu številu koeficientov, tako da se nizkim vrednostim *TotalCoeff* dodeli kratke kode in visokim vrednostim *TotalCoeff* dolge kode. Tabela 2 je usmerjena k srednjim številom koeficientov, tabela 3 pa k višjim številom koeficientov. Tabela 4 dodeli fiksno 6-bitno kodo vsakemu paru *TotalCoeff* in vrednosti *TrailingOnes*.

Tabela 4.12: Izbira preslikovalne tabele za *coeff_token*.

N	**tabela za *coeff_token***
0, 1	tabela 1
2, 3	tabela 2
4, 5, 6, 7	tabela 3
8 in več	tabela 4

Kodiranje predznaka za vsak *TrailingOne*

Predznak za vsak *TrailingOne*, posredovan s *coeff_token*, je kodiran z enim samim bitom (0 je +, 1 je –) v obratnem vrstnem redu (začetek pri višjih frekvencah).

Kodiranje nivoja ostalih neničelnih koeficientov

Nivo (predznak in velikost) vsakega od ostalih neničelnih koeficienton v bloku je kodiran v obratnem vrstnem redu: od visokih frekvenc proti *DC* koeficientu. Koda za vsak nivo je narejena iz predpone (*level_prefix*) in pripone (*level_suffix*). Dolžina pripone (*suffixLength*) je lahko med 0 in 6 bitov in je prilagojena na nivo vsakega naslednjega kodiranega nivoja. Majhna vrednost *suffixLength* je primerna za nivoje z malimi vrednostmi, visoka vrednost pa je primerna za nivoje z visokimi vrednostmi. Izbira je opravljena na sledeč način:

1. Inicializiraj *sufixLength* na 0 (razen, če je več kot 10 neničelnih koeficientov in manj kot tri enice na koncu sekvence; v tem primeru je inicializiran na 1).
2. Kodiraj koeficient najvišje frekvence.

3. Če je velikost koeficienta višja kot prej določeni prag, se *suffixLength* poveča.

Tako se izbira pripone ujema z velikostjo pravkar kodiranega koeficienta.

Kodiranje števila ničel pred zadnjim koeficientom

Vsota vseh ničel pred zadnjim neničelnim koeficientom je kodirana z *VLC total_zeros*. Razlog za pošiljanje posebne *VLC* za sporočanje števila ničel je v tem, da veliko blokov vsebuje nekaj neničelnih koeficientov na začetku polja in se s tem pristopom izognemo kodiranju niza ničel na koncu polja.

Kodiranje nizov ničel

Število ničel pred vsakim neničelnim koeficientom je kodirano v obratnem vrstnem redu. Parameter *run_before* je kodiran za vsak neničelni koeficient, začenši z visokimi frekvencami, z dvema izjemama:

1. Če ni več ničel za kodirati.

2. Ni potrebno kodirati *run_before* za zadnji neničelni koeficient (najnižja frekvenca).

VLC vsakega niza ničel je izbrana glede na število ničel, ki še niso bile kodirane in *run_before*. Na primer, če sta ostali samo dve ničli za kodiranje, ima lahko *run_before* le tri vrednosti (0, 1 ali 2), tako *VLC* ne rabi več kot 2 bitov. Če je za kodiranje ostalo še šest ničel, potem *run_before* lahko dobi sedem različnih vrednosti in mora biti tabela *VLC* ustrezno daljša.

4.4 Glavni profil

Glavni profil (*Main profile*) se uporablja za oddajanje medijskih aplikacij, kot je na primer digitalna televizija ali shranjevanje digitalnega videa. Glavni profil je skoraj nadmnožica osnovnega profila, le skupine rezin, ASO in redundantne rezine niso podprte. Dodatna orodja v glavnem profilu so rezine B, uteženo napovedovanje, podpora za prepleteni video in *CABAC* (alternativna metoda kodiranja na osnovi entropije).

4.4.1 Rezine B

Vsak del zunanje kodiranega makrobloka v rezini B se lahko napove iz enega ali dveh referenčnih okvirov, pred ali za trenutnim v časovnem zaporedju. Odvisno od referenčnih okvirov, shranjenih v kodirniku in dekodirniku, to daje veliko možnosti za izbiro referenčnega okvira za napovedovanje v makroblokih B.

Referenčni okviri

Makrobloki B uporabljajo dva seznama prej kodiranih referenčnih okvirov: seznam 0 in seznam 1, ki vsebujeta kratkotrajne in dolgotrajne okvire (glej poglavje 4.3.2). Ta dva seznama lahko vsebujeta okvire, ki so predhodni ali naslednji v zaporedju prikazovanja. Z dolgotrajnimi se rokuje enako kot je to opisan v poglavju 4.3.2. Kratkotrajni okviri, pred in po trenutnem, in privzet vrstni red indeksov so sledeči:

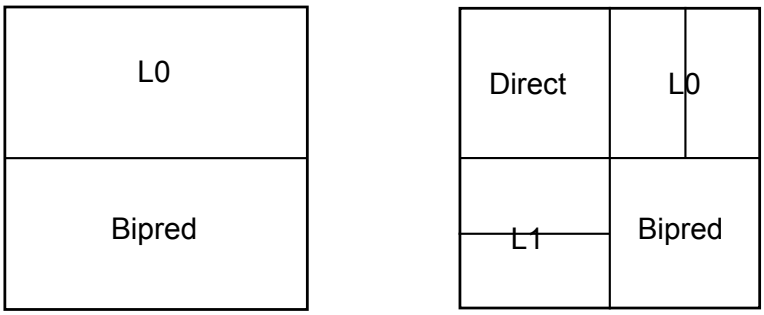
- Seznam 0: Najbližjemu okviru pred trenutnim je dodeljen indeks 0, ki mu sledijo predhodni, za njim pa naslednji.
- Seznam 1: Najbližjem okviru za trenutnim je dodeljen indeks 0, ki mu sledijo naslednji, za njimi pa predhodni.

Možnosti napovedovanja

Deli makrobloka se lahko napovejo na več različnih načinov: direktni način (poglavje 4.4.1), napovedovanje iz seznama 0 referenčnih okvirov, napovedovanje iz seznama 1 referenčnih okvirov ali dvojno napovedovanje (*bi-prediction*) iz seznama 0 in seznama 1 referenčnih okvirov. Za vsak del makrobloka je lahko uporabljeno drugačno napovedovanje. Če je izbran del makrobloka velikosti 8x8, se uporabi enako napovedovanje za vse poddele. Slika 4.18 kaže dva primera veljavnih napovedovanj. Na levi strani imamo dva bloka velikosti 16x8, prvi se napoveduje iz seznama 0, drugi pa z dvojnim napovedovanjem. Na desni strani so štirje bloki velikosti 8x8, ki uporabljajo direktno napovedovanje, napovedovanje iz seznama 0, napovedovanje iz seznama 1 in dvojno napovedovanje.

Dvojno napovedovanje

V načinu dvojnega napovedovanja (*bi-prediction*) je referenčni blok zgrajen iz referenčnih okvirov seznama 0 in 1. Dobita se dve referenčni področji: eno iz seznama



Slika 4.18: Primera načinov napovedovanja v makroblokih rezin B.

0, drugo iz seznama 1 (za to rabimo dva gibalna vektorja). Končno referenčno področje dobimo s pomočjo povprečja obeh. Razen v primeru uteženega napovedovanja (poglavje 4.4.2) se uporabi naslednja enačba:

$$pred(i,j) = (pred0(i,j) + pred1(i,j) + 1) >> 1.$$

pred0(i,j) in *pred1(i,j)* sta vzorca, dobljena iz referenčnih okvirov iz seznamov 0 in 1, *pred(i,j)* pa je dvojno napovedan vzorec. Ko se izračunajo vsi referenčni vzorci, dobimo ostanek bloka z odštevanjem referenčnega bloka od trenutnega.

Gibalni vektor za seznam 0 in 1 v dvojno napovedanem makrobloku ali bloku je napovedan iz sosednjih gibalnih vektorjev iste smeri, gledano časovno. Na primer vektor trenutnega makrobloka, ki kaže na predhodni okvir, je napovedan iz sosednjih vektorjev, ki tudi kažejo na predhodne okvire.

Direktno napovedovanje

V direktnem načinu napovedovanja ni poslan noben vektor za napovedovanje makroblokov ali delov makroblokov rezin B. Namesto tega dekodirnik izračuna vektorje seznama 0 in 1 glede na prejšnje vektorje in uporabi te za dvojno napovedovanje dekodiranih ostankov. Preskočen makroblok v rezini B je rekonstruiran z uporabo direktnega napovedovanja.

Zastavica v glavi rezine sporoči, ali se uporabi prostorska ali časovna metoda računanja vektorjev v direktnem načinu.

V prostorskem direktnem načinu so vektorji za seznam 0 in seznam 1 izračunani na sledeč način. Vektorje izračunamo z postopkom opisanem v poglavju 4.3.5. Če imajo soležni makrobloki ali njihovi deli v prvem referenčnem okviru seznama 1 gibalni vektor, ki je manj kot ± 2 vzorca intenzitete različen od trenutnega, sta eden ali oba napovedna vektorja nastavljena na nič. Drugače so napovedni vektorji iz seznama

0 in 1 uporabljeni za dvojno napovedovanje kompenzacije gibanja. V časovnem modelu dekodirnik opravi sledeče korake:

1. Najdi referenčni okvir iz seznama 0 za soležni makroblok ali njegov del v okviru seznama 1. Ta referenca postane referenca seznama 0 za trenutni makroblok ali njegov del.

2. Poišči vektor seznama 0 za soležni makroblok ali njegov del v okviru seznama 1.

3. Skaliraj gibalni vektor glede na razliko v zaporedni številki okvira med trenutnim in referenčnim okvirom iz seznama 1. Tako se dobi novi vektor seznama 1.

4. Skaliraj gibalni vektor glede na razliko v zaporedni številki okvira med trenutnim in referenčnim okvirom iz seznama 0. Tako se dobi novi vektor seznama 0.

Ti načini se spremenijo, če na primer referenčni makrobloki ali njihovi deli niso na voljo ali so zunanje kodirani.

4.4.2 Uteženo napovedovanje

Uteženo napovedovanje je metoda spreminjanja (skaliranja) vzorcev za kompenzacijo gibanja v makroblokih P in B. V H.264 poznamo tri tipe uteženega napovedovanja:

1. Makroblok rezine P, eksplicitno uteženo napovedovanje.

2. Makroblok rezine B, eksplicitno uteženo napovedovanje.

3. Makroblok rezine B, implicitno uteženo napovedovanje.

Vsak vzorec napovedovanja je skaliran za faktor uteži *w0* ali *w1* pred napovedovanjem za kompenzacijo gibanja. Pri eksplicitnih tipih je faktor uteži določen v kodirniku in poslan v glavi rezine. Če je uporabljeno implicitno napovedovanje, sta faktorja *w0* in *w1* izračunana glede na relativno časovno pozicijo referenčnih okvirov v seznamu 0 in seznamu 1. Večji faktor uteži je uporabljen, če je referenčni okvir časovno bližje trenutnemu, in manjši, če je referenčni okvir časovno dlje od trenutnega.

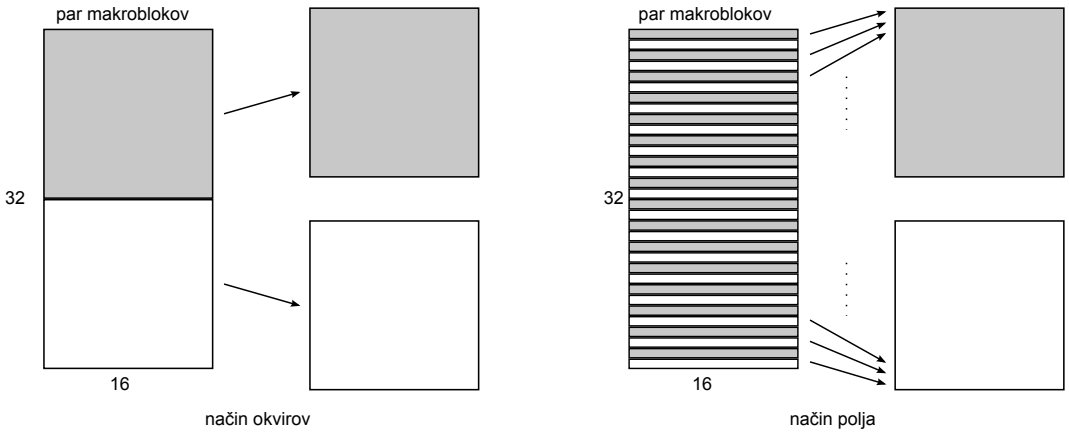
Uteženo napovedovanje se pogosto uporablja za omogočanje eksplicitne ali implicitne kontrole relativnih prispevkov referenčnih okvirov na proces kompenzacije gibanja. Uteženo napovedovanje je na primer lahko zelo učinkovito za kodiranje bledenja (*fade*) ene scene v drugo.

4.4.3 Prepleteni video

Učinkovito kodiranje prepletenega videa zahteva orodja, ki so optimirana za kompresijo makroblokov polja. Če je podprto kodiranje polja, se sporoči tip slike (cel okvir ali polje) v glavi vsake rezine. V prilagodljivem makroblok (*makroblock-adaptive*) načinu kodiranja se izbira okvira ali polja opravi na nivoju makrobloka. V tem načinu je trenutna rezina procesirana v 16 vzorcev širokih in 32 vzorcev visokih enotah intenzitet. Vsaka od teh je kodirana kot par makroblokov (slika 4.19). Kodirnik lahko izbere, ali bo par makroblokov kodiran kot dva bloka okvira ali kot dva makrobloka polja. Prav tako lahko izbere optimalni način kodiranja za vsak del slike.

Kodiranje rezine ali para makroblokov v načinu polja zahteva spremembe nekaterih korakov kodiranja in dekodiranja, opisanih v poglavju 4.3.

Na primer vsako kodirano polje se tretira kot ločen referenčni okvir za napovedovanje v rezinah P in B.



Slika 4.19: Prilagodljivo kodiranje makroblokov.

4.4.4 Kontekstno osnovano adaptivno binarno aritmetično kodiranje (*CABAC*)

Če je parameter okvira *entropy_coding_mode* enak 1, je izbran sistem aritmetičnega kodiranje in dekodiranja H.264 elementov. Kontekstno osnovano binarno aritmetično kodiranje (CABAC) doseže dober učinek kodiranja z izbiro verjetnostnega modela za (a) vsak element sintakse glede na elementovo vsebino, (b) prilagajanje ocen verjetnosti glede na lokalno statistiko in (c) uporabo aritmetičnega kodiranja namesto kodiranja spremenljivih dolžin. Kodiranje podatkov vključuje naslednje stopnje:

1. Binarizacija: CABAC uporablja binarno aritmetično kodiranje, kar pomeni, da so kodirani samo binarni znaki (0 in 1). Nebinarne vrednosti (npr.: koeficienti transformacije, gibalni vektorji, vsi simboli, ki imajo več kot dve možni vrednosti) so pred aritmetičnim kodiranjem binarizirani ali spremenjeni v binarno kodo. Ta proces je podoben procesu spreminjanja podatkovnih simbolov v kode spremenljivih dolžin, vendar je v tem primeru dobljen rezultat pred pošiljanjem še enkrat kodiran.

2. Izbira modela vsebine: Model vsebine je verjetnostni model enega ali več bitov binariziranega simbola in je izbran iz nabora možnih modelov, odvisno od statistike pravkar kodiranih podatkovnih simbolov. Model vsebine shrani podatek, s kolikšno verjetnostjo je vsak bit 0 ali 1.

3. Aritmetično kodiranje: Aritmetični kodirnik kodira vsak bit glede na izbran verjetnostni model (glej poglavje 3.3.3).

4. Posodabljanje verjetnosti: Izbrani model vsebine je posodobljen glede na dejansko kodirano vrednost (če je bil kodiran bit 1, se verjetnost za pojavitev tega bita poveča).

Stopnje 2, 3 in 4 se ponovijo za vsak bit binariziranega simbola.

4.5 Razširjeni profil

Razširjeni profil (*Extended profile*) je v praksi uporaben za aplikacije, kot je pretok videa. Vsebuje vse lastnosti osnovnega profila (nadmnožica osnovnega profila) s podporo za rezine B (poglavje 4.4.1) in uteženo napovedovanje (poglavje 4.4.2). Dodana so orodja za učinkovit pretok videa prek omrežja kot je Internet: rezini SI in SP, ki pospešita preklop med različnimi kodiranimi tokovi, in rezine z deljenimi podatki, ki izboljšajo delovanje v okolju, kjer so pogoste napake.

4.5.1 Rezine SP in SI

Rezine SP in SI so posebno kodirane rezine, ki, med drugim, omogočijo efektivno preklapljanje med video tokovi in naključni dostop v dekodirniku. Skupna zahteva aplikacij video pretoka je, da zna dekodirnik preklapljati med različno kodiranimi tokovi. Na primer, video je kodiran za pošiljanje po Internetu pri različnih bitnih hitrostih. Dekodirnik poizkuša dekodirati tok najvišje bitne hitrosti in če ugotovi, da pretok skozi omrežje ni dovolj velik, avtomatsko preklopi na tok z manjšo bitno hitrostjo.

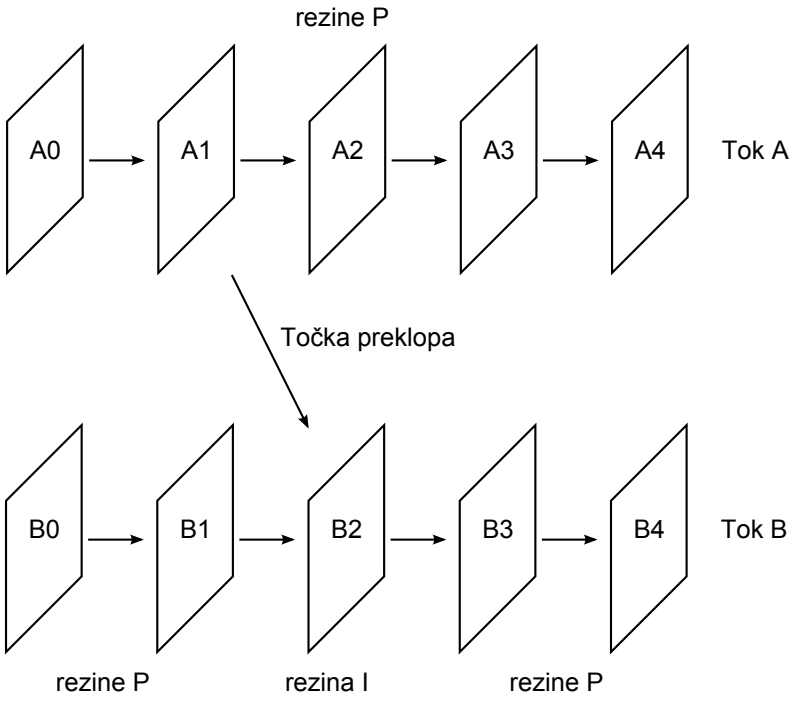
Rezine SP so narejene za podporo preklapljanja med podobnimi sekvencami videa (na primer, isti video, kodiran z različnimi bitnimi hitrostmi) brez kazni povečanja velikosti podatkov za rezino I (slika 4.20). Na točki preklopa (okvir 2 na slikah 4.20 in 4.21) so tri rezine SP, vse kodirane s pomočjo kompenzacije gibanja (s tem je kodiranje bolj učinkovito kot pri rezinah I). Rezina SP A_2 je dekodirana s pomočjo referenčnega okvira A_1 in rezina SR B_2 je dekodirana na osnovi napovedovanja iz okvira B_1. Ključna v procesu preklapljanja je rezina SR AB_2, ki je generirana tako, da za generiranje okvira B_2 uporabi napovedovanje iz referenčnega okvira A_1. Dodatna rezina SR je potrebna na vsaki točki preklopa za preklapljanje v drugo smer (rezina SR BA_2), vendar je vseeno zelo verjetno to bolj učinkovit način kodiranje okvirov A_2 in B_2 kot rezine I. V tabeli 4.13 vidimo korake, ki so potrebni za preklop med video tokoma A in B.

Tabela 4.13: Preklapljanje med video tokovoma A in B z uporabo rezine SP.

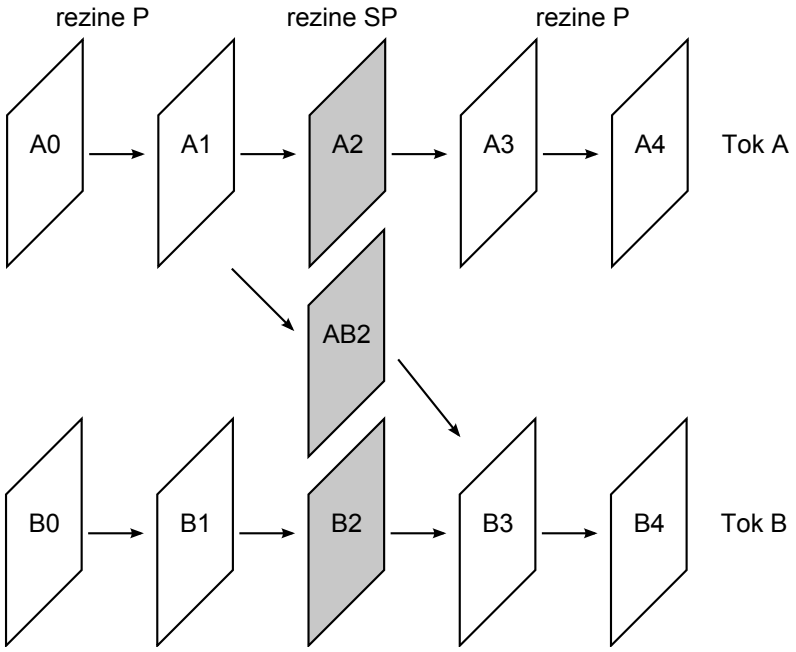
vhod v dekodirnik	**referenca za kompenzacijo gibanja**	**Izhod iz dekodirnika**
rezina P A_0	[prejšnji okviri]	dekodiran okvir A_0
rezina P A_1	dekodiran okvir A_0	dekodiran okvir A_1
rezina SP AB_2	dekodiran okvir A_1	dekodiran okvir B_2
rezina P B_3	dekodiran okvir B_2	dekodiran okvir B_3
...	...	...

Slika 4.22 prikazuje poenostavljen diagram procesa kodiranja rezine SP A_2, ki je dobljena z odštevanjem dekodiranega okvira A_1 od okvira A_2 in nato kodiranjem ostanka. Drugače kot pri navadni rezini P, se tukaj odštevanje zgodi v prostoru transformacije (po transformaciji). Rezina SP B_2 je kodirana na enak način (slika 4.23). Dekodirnik, ki je že dekodiral okvir A_1, lahko dekodira rezino SP A_2, kot je prikazano na sliki 4.24.

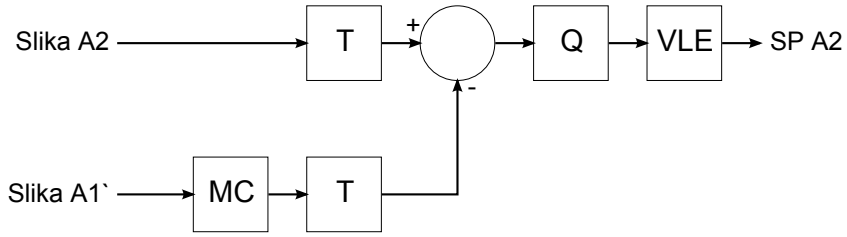
Rezina SP AB_2 je kodirana, kot je prikazano na sliki 4.25. Okvir B_2 (okvir, na katerega želimo preklopiti), ki je transformiran in napovedan za kompenzacijo gibanja,



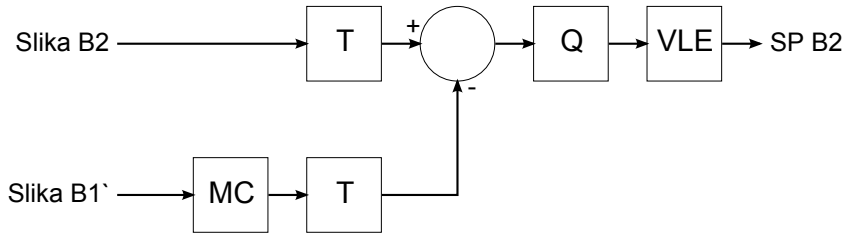
Slika 4.20: Preklop med video tokovi s pomočjo rezine I.



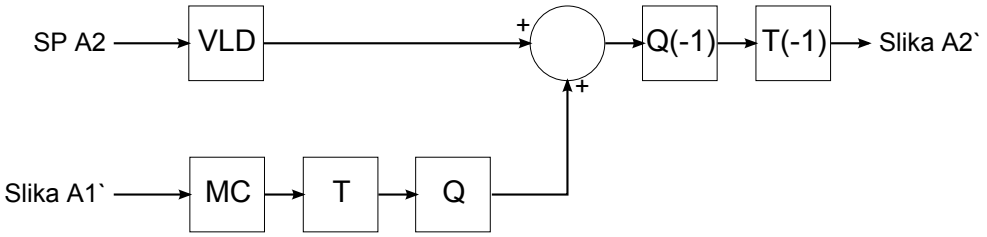
Slika 4.21: Preklop med video tokovi s pomočjo rezine SP.



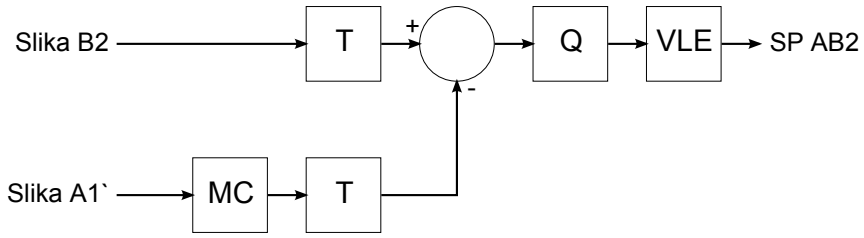
Slika 4.22: Kodiranje rezine SP A_2 (poenostavljeno).



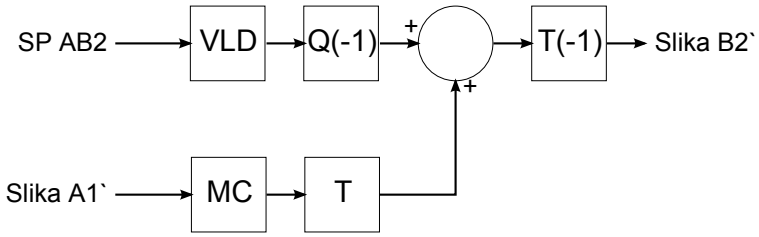
Slika 4.23: Kodiranje rezine SP B_2 (poenostavljeno).



Slika 4.24: Dekodiranje rezine SP A_2 (poenostavljeno).



Slika 4.25: Kodiranje rezine SP AB_2 (poenostavljeno).



Slika 4.26: Dekodiranje rezine SP AB_2 (poenostavljeno).

je bil narejen na podlagi okvira A_1' (dekodiran okvir, iz katerega preklapljamo). Blok MC v tem diagramu poskuša najti najbolj podoben makroblok za vsak makroblok iz okvira B_2, pri čemer uporablja dekodiran okvir A_1 kot referenčni. Okvir, ki je generiran za kompenzacijo gibanja, je transformiran in nato odštet od transformiranega okvira B_2. Ostanek je kvantiziran, kodiran in poslan (shranjen).

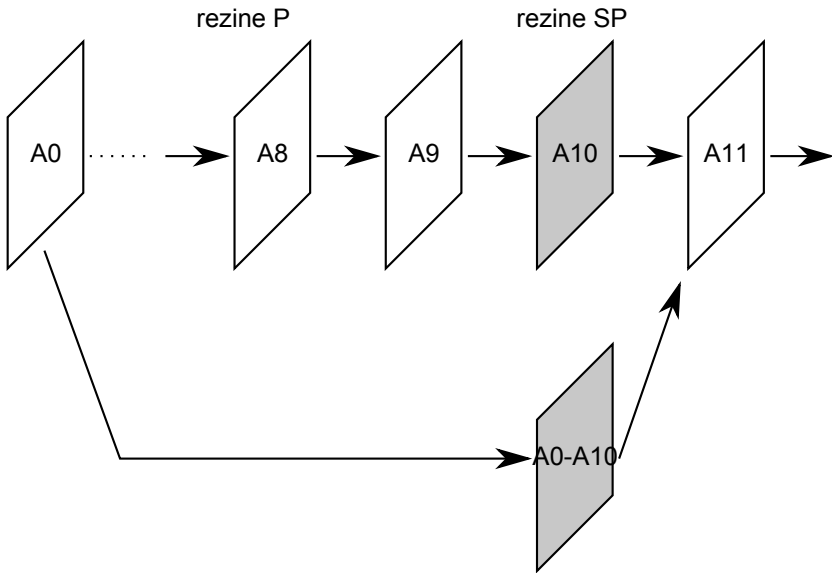
Dekodirnik, ki je prej dekodiral okvir A_1', lahko dekodira rezino SP AB_2 in dobi okvir B_2 (slika 4.26). Iz A_1' s pomočjo poslanih gibalnih vektorjev generiramo okvir za kompenzacijo gibanja, ki ga transformiramo in prištejemo dekodiranemu in skaliranemu ostanku, nato je rezultat inverzno transformiran, s čimer se dobi B_2'.

Če sta tokova A in B različni verziji iste originalne sekvence, kodirani z različnimi bitnimi hitrostmi, bi moralo biti napovedovanje za kompenzacijo gibanja okvira B_2 iz A_1' zelo učinkovito. Rezultati kažejo, da je uporaba rezin SP mnogo bolj učinkovita kot vstavljanje rezin I na mesto preklopa. Drug način uporabe rezin SP je omogočiti naključni dostop. Na primer, kodirnik lahko vstavi rezino SP in preklopno rezino SP na pozicijo okvira 10 (slika 4.27). Dekodirnik lahko preskoči iz okvira A_0 direktno na okvir A_{10} z dekodiranjem okvira A_0 in nato preklopne rezine SP A_{1-10}, s čimer dobimo okvir A_{10}.

V razširjenem profilu imamo še en tip preklopnih rezin. To so rezine SI. Uporabljene so podobno kot preklopne rezine SP, s to razliko, da jih z uporabo notranjega napovedovalnega načina 4x4 (glej poglavje 4.3.6) iz prej dekodiranih delov rekonstruiranega okvira. Ta tip rezine je primeren, na primer, za preklop iz sekvence na povsem drugačno sekvenco (v tem primeru kompenzacija gibanja nima učinka, ker si sekvenci nista dovolj podobni).

4.5.2 Rezine z deljenimi podatki

Kodirani podatki ene rezine so razdeljeni na ločene dele (A, B in C), od katerih vsak vsebuje podmnožico kodirane rezine. Del A vsebuje glavo rezine in glavo vsakega makrobloka v rezini, del B vsebuje kodirane ostanke podatkov za notranje kodirane



Slika 4.27: Hitro previjanje s pomočjo rezin SP.

in makrobloke SI, del C pa vsebuje kodirane ostanke podatkov za zunanje kodirane makrobloke. Vsak del je lahko v ločeni enoti na nivoju omrežja in je zato lahko poslan ločeno.

Če je del A izgubljen, je zelo težko oziroma nemogoče rekonstruirati rezino, zato je ta del zelo občutljiv na napake pri prenosu. Dela B in C lahko dekodiramo neodvisno. Dekodirnik lahko dekodira le dela A in B ali dela A in C, kar vodi v fleksibilen sistem, ki ni občutljiv na napake.

Poglavje 5

Standardne iskalne metode

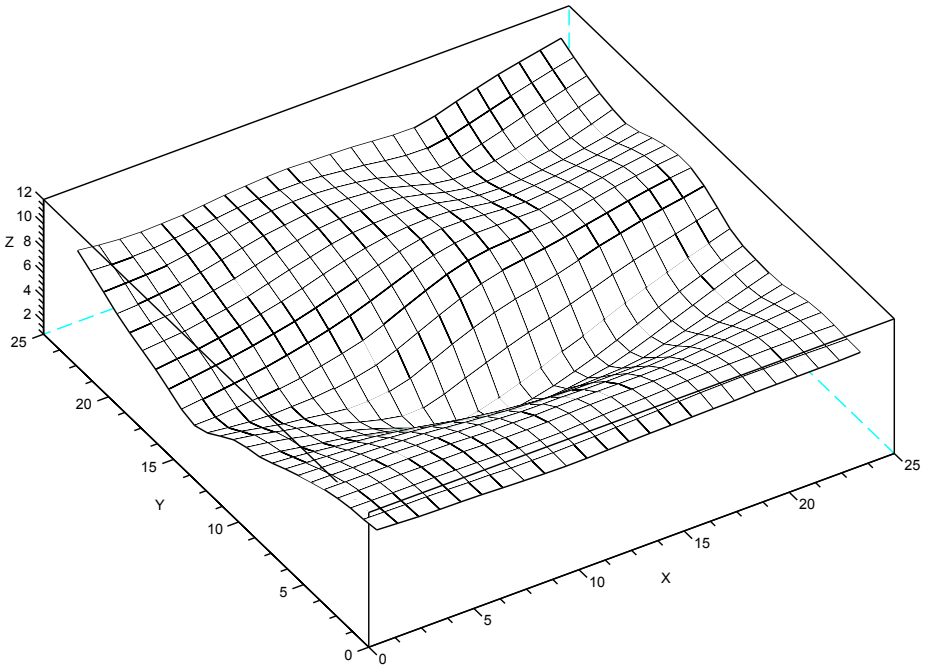
5.1 Uvod

V tem poglavju bodo predstavljeni štirje tipični algoritmi za ocenjevanje gibanja. Najprej bo predstavljeno *Polno iskanje*, to je algoritem, ki zagotavlja najboljši rezultat, vendar za to porabi največ časa. Za tem bodo predstavljeni hitrejši algoritmi *Iskanje v treh korakih* [7], *Eden naenkrat* [8] in *Povprečna piramida* [9]. Ti opravijo iskanje v krajšem času, vendar ne zagotavljajo najboljših rezultatov. Na sliki 5.1 vidimo 3D graf napake, dobljen s *Polnim iskanjem*. Iz grafa sta razvidna dva minimuma, kar predstavlja nevarnost za mnoge hitrejše algoritme.

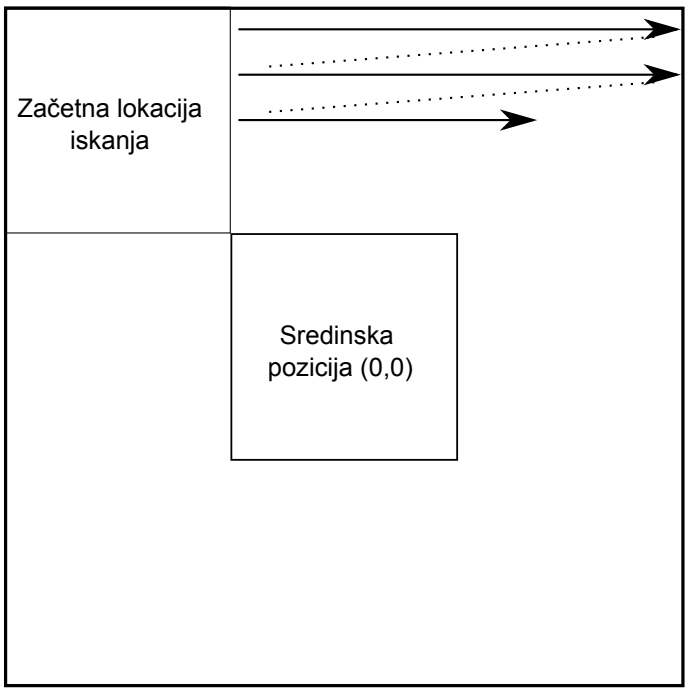
5.2 Polno iskanje

Z algoritmom za ocenjevanje gibanja *Polno iskanje* (*Full Search*) se izračuna vrednost napake za vsa možna referenčna področja. Pri tem dobimo optimalni rezultat, vendar se za to porabi največ časa.

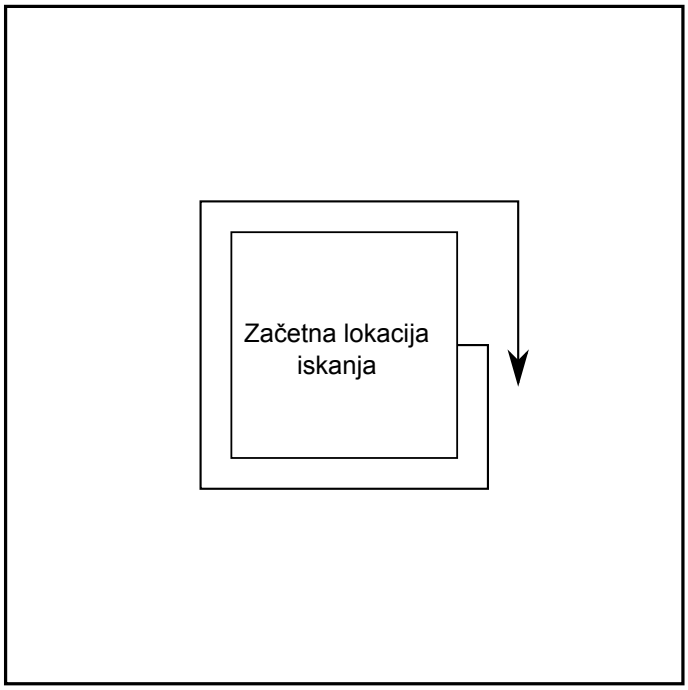
Na sliki 5.2 vidimo rastrsko strategijo iskanja. V tem primeru je prva lokacija, za katero izračunamo napako, zgornja leva, nato se iskanje nadaljuje v rastrskem zaporedju. V video sekvencah je tipično najboljši vektor blizu sredinske lokacije (vektor 0,0). Računanje polnega iskanja je lahko poenostavljeno, tako da se iskanje začne na sredinski lokaciji in nadaljuje v spiralnem vrstnem redu okoli sredine (slika 5.3). Ta način se dobro obnese, če je uporabljeno predhodno zaključevanje (*early termination*) (glej poglavje 2.3).



Slika 5.1: 3D graf napake na vseh lokacija v iskalnem področju.



Slika 5.2: Polno iskanje, rastrski način.

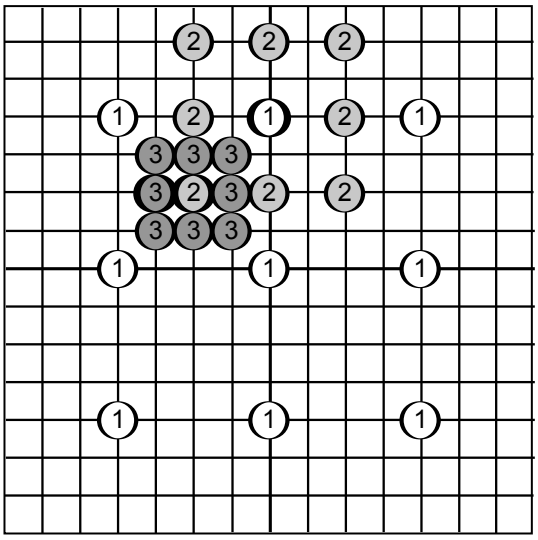


Slika 5.3: Polno iskanje, spiralni način.

5.3 Iskanje v treh korakih

Algoritem *Iskanje v treh korakih* (*Tree Step Search*) je bil predstavljen leta 1981 in je zaradi svoje preprostosti, robustnosti in kljub temu dobrih rezultatov kmalu postal zelo priljubljen.

V prvem koraku preveri napako na sredinski lokaciji in še na osmih lokacijah, oddaljenih za začetno dolžino koraka od sredinske lokacije. Na sliki 5.4 so lokacije, kjer se preverja napaka, označene z '1'. V drugem koraku je lokacija z najmanjšo napako izbrana kot naslednja sredinska lokacija, dolžina koraka pa se razpolovi. Ta dva koraka se ponavljata, dokler ni velikost koraka manjša od 1.



Slika 5.4: Iskanje v treh korakih.

Glavna pomanjkljivost tega iskanja je, da se lahko ujame v lokalni minimum.

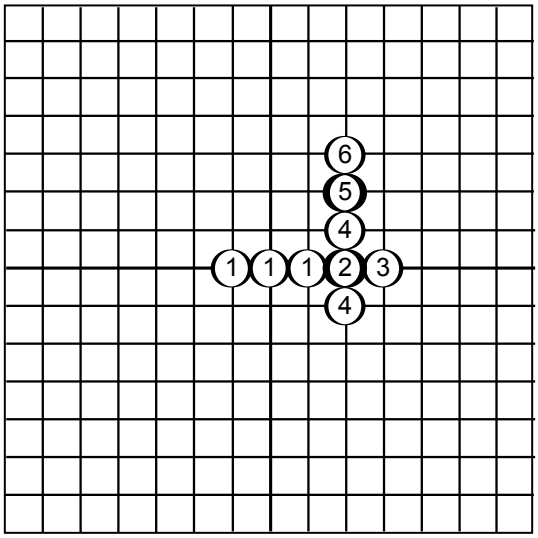
5.4 Eden naenkrat

Algoritem za iskanje gibalnega vektorja *Eden naenkrat* (*One at a Time*) je eden bolj preprostih, a še vedno učinkovit. Uporabljena je preprosta metoda reševanja optimizacijskih problemov z dvema neznankama: poišči najboljšo pozicijo za prvo spremenljivko, medtem ko je druga fiksna, nato fiksiraj prvo spremenljivko in poišči optimalno vrednost druge spremenljivke.

V prvem koraku algoritem izbere tri vrednosti horizontalno okoli centra. Če je najmanjša napaka na sredinski poziciji, se zaključi s horizontalnim iskanjem, sicer

se iskanje nadaljuje v smeri najmanjše napake, dokler vrednost napake pada. Ko končamo s horizontalno fazo, ponovimo enak postopek v vertikalni smeri, s tem da se za sredinsko pozicijo vzame mesto z najmanjšo napako v horizontalni smeri.

Na sliki 5.5 vidimo eno od možnih poti iskanja algoritma *Eden naenkrat*. V tem primeru se najde horizontalni minimum v drugem koraku, vertikalni pa v petem skupno.



Slika 5.5: Eden naenkrat.

Največja prednost tega algoritma je njegova hitrost, saj do končnega rezultata pride z daleč najmanj računanji napake. Posledično so tudi rezultati najslabši. Pogosto se ujame v lokalni minimum blizu središčne pozicije.

5.5 Hierarhični načini iskanja gibalnega vektorja

Za zmanjšanje računske kompleksnosti algoritma so pogosto uporabljene hierarhične metode, ki iskanje minimuma začno na podmnožici celotnih podatkov in nato v več korakih pridejo do celotnih podatkov.

Tipičen predstavnik teh algoritmov je *Povprečna piramida* (*Mean pyramid*). Pri tem algoritmu se zmanjša velikost področja iskanja, tako da se izračuna povprečje določene regije in se to uporabi kot ena pika na višjem nivoju. Tipično se izračuna povprečje 2x2 pik originalnega področja iskanja (nivo 0) in se to preslika v eno piko na nivoju 1. Za piko na nivoju 2 se izračuna 4x4 pik originalnega področja. Splošna

enačba za izračun pike na nivoju L $g_L(p, q)$ je:

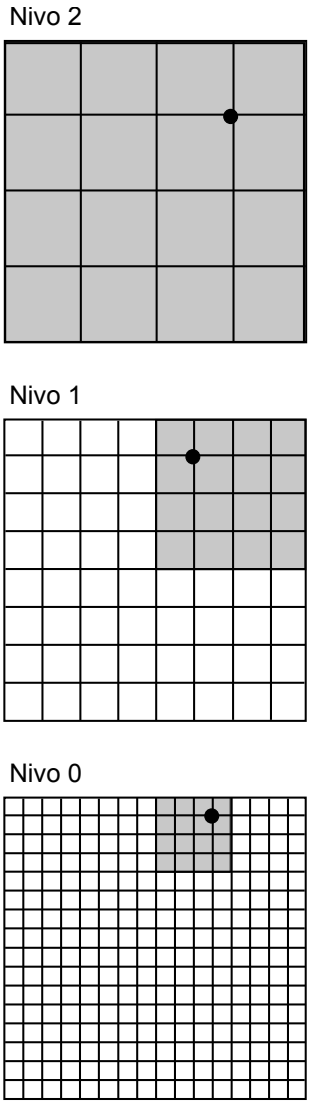
$$g_L(p,q) = \frac{1}{4}\left(\sum_{u=0}^{1}\sum_{v=0}^{1} g_{L-1}(2p+u, 2q+v)\right).$$

Pozitiven stranski učinek povprečenja je večja odpornost na šum v sliki.

Ko je piramida zgrajena, se na najvišjem nivoju uporabi enega od standardnih iskalnih algoritmov (na primer *Iskanje v treh korakih*). Rezultat, dobljen na najvišjem nivoju, se posreduje nivo niže, soležno področje pa se izbere kot sredinsko. Ta postopek se ponavlja, dokler se ne pride do najnižjega nivoja.

Na sliki 5.6 vidimo postopek iskanja z metodo povprečne piramide. S sivo je označeno področje iskanja, pika pa označuje blok z najmanjšo napako.

Iz že povedanega sledi, da je ta algoritem procesorsko bolj zahteven od ostalih, saj moramo enega od teh algoritmov uporabiti na vsakem nivoju, vendar so tudi rezultati temu primerni. S tem pristopom se izognemo nevarnosti, da bi končali v lokalnem minimumu.



Slika 5.6: Povprečna piramida.

Poglavje 6

Stohastični algoritem učečih avtomatov

6.1 Uvod

V tem poglavju bo opisan stohastični algoritem, ki je uporabljen za iskanje danemu makrobloku najbolj podobnega bloka v referenčnem okviru. Algoritem se sicer uporablja za posodabljanje verjetnosti izvajanja akcij pri učečih avtomatih oziroma pri stohastičnih avtomatih s spremenljivo strukturo, ki so podvrsta učečih avtomatov.

Najprej bo opisanih nekaj osnov učečih avtomatov, nato omenjeni algoritem, na koncu pa način, kako ga lahko uporabimo v primeru ocenjevanja gibanja (*motion estimation*), ki služi za bločno kompenzacijo gibanja (*block-based motion compensation*).

6.2 Učeči avtomati

Učeči avtomati [11] so matematični modeli sistemov, ki v nekem okolju izvajajo akcije in sproti dobivajo iz okolja oceno posamezne akcije. V skladu s tem skušajo svoje nadaljnje delovanje, tj. izvajanje akcij, izboljšati, tako da bi dobivali čimbolj ugodne ocene. Akcije se lahko ocenjuje na različne načine, najenostavnejša in tudi najpogosteje uporabljena možnost pa je, da je ocena dvovrednostna oz. binarna. V tem primeru s parametrom β povemo, ali gre za ugoden odziv (nagrada) ali za neugoden odziv okolja (kazen). Vrednost $\beta = 0$ pomeni nagrado, $\beta = 1$ pa kazen. Cilj avtomata je torej izbirati akcije, tako da bo odziv okolja čimvečkrat $\beta = 0$. Za

to obstaja količina, ki jo imenujemo povprečna kazen in označujemo z M.

Pri učečih avtomatih predpostavimo, da je okolje stohastično in da torej posamezne akcije kaznuje (oz. nagrajuje) z določeno verjetnostjo. Zato je okolje podano z množico verjetnosti kaznovanja $c = \{c_1, c_2, ..., c_r\}$, kjer je r število akcij. Če se te verjetnosti s časom ne spreminjajo, je okolje stacionarno, če pa se, je okolje nestacionarno.

V grobem delimo učeče avtomate na

- avtomate s fiksno strukturo in
- avtomate s spremenljivo strukturo.

Avtomati s fiksno strukturo (AFS) [10] so bolj podobni običajnim končnim avtomatom. AFS imajo definirana stanja, med katerimi lahko prehajajo deterministično (deterministični AFS) ali stohastično (stohastični AFS). V vsakem stanju lahko izvedejo določeno akcijo (determinističen izhodni operator), lahko pa tudi katerokoli akcijo z določeno verjetnostjo (stohastičen izhodni operator). Zato so AFS običajno definirani kar z verjetnostnimi matrikami: za operator prehajanja stanj in za izhodni operator. Glede na to, da delujejo vedno na enak način (deterministični AFS) ali vsaj z enakimi verjetnostmi (stohastični AFS), od njih ne pričakujemo izboljšanega delovanja v prihodnosti, zato pravzaprav niso 'učeči' v pravem smislu.

Avtomati s spremenljivo strukturo [12] pa so vedno stohastični, zato jih imenujemo tudi stohastični avtomati s spremenljivo strukturo (SASS). SASS so manj podobni klasičnim končnim avtomatom, ampak bolj Markovskim procesom in Markovskim verigam. Pri SASS imamo definirane verjetnosti izvajanja posameznih akcij oz. vektor verjetnosti akcij $\mathbf{p}$. Verjetnosti akcij se s časom spreminjajo oz. posodabljajo, in sicer tako, da bi avtomat deloval bolje oz. z manjšo povprečno napako M. Načini spreminjanja verjetnosti akcij se imenujejo *korekcijske sheme*.

6.3 Korekcijske sheme

Če izvedemo v času n akcijo α_i, potem je treba v primeru, da je bil učinek te akcije ugoden (tj. nagrada oz. $\beta = 0$), njeno verjetnost povečati, verjetnosti vseh ostalih akcij pa zmanjšati, vendar tako, da vsota verjetnosti vseh akcij še vedno ostane 1. Kar se tiče vrstnega reda, korekcijske sheme uberejo ravno obratno pot: najprej verjetnosti ostalih akcij zmanjšajo (v skladu z določenim pravilom), nato pa verjetnost izvedene akcije povečajo za vsoto vseh redukcij.

Če pa je bil učinek akcije α_i neugoden (tj. kazen oz. $\beta = 1$), je potrebno verjetnost te akcije zmanjšati, verjetnosti vseh ostalih pa povečati. Tudi tukaj korekcijske sheme definirajo, kako je treba verjetnosti vseh ostalih akcij povečati, iz tega pa seveda lahko ugotovimo, kako je treba zmanjšati verjetnost kaznovane akcije.

Korekcijsko shemo lahko zapišemo bolj formalno:

če je $\alpha(n) = \alpha_i$ in $\beta(n) = 0$:

$$\begin{aligned} p_j(n+1) &= p_j(n) - g_j[p(n)], \quad \text{za vse } j \neq i \\ p_i(n+1) &= p_i(n) + \sum_{j=1, j\neq i}^{r} g_j[p(n)] \end{aligned} \tag{6.1}$$

če je $\alpha(n) = \alpha_i$ in $\beta(n) = 1$:

$$\begin{aligned} p_j(n+1) &= p_j(n) + h_j[p(n)], \quad \text{za vse } j \neq i \\ p_i(n+1) &= p_i(n) - \sum_{j=1, j\neq i}^{r} h_j[p(n)]. \end{aligned}$$

Pri tem je n diskreten čas, g_j in h_j pa sta korekcijski funkciji, ki morata zagotavljati, da se verjetnosti tudi po korekcijah ohranjajo znotraj intervala $[0, 1]$. Zato morata izpolnjevati naslednje pogoje (za vse $j = 1, ..., r$): zveznost, nenegativnost, $0 < g_j(p) < p_j$ in

$$0 < \sum_{j=1, j\neq i}^{r} [p_j + h_j(p)] < 1\,, \qquad \text{za vse } i = 1, ..., r\,.$$

6.4 Linearna nagrajevalno-kaznovalna korekcijska shema

Pri linearnih korekcijskih shemah sta funkciji g_j in h_j linearni funkciji verjetnosti izvajanja j-te akcije p_j:

$$\begin{aligned} g_j[p(n)] &= a p_j(n) \\ h_j[p(n)] &= \frac{b}{r-1} - b p_j(n). \end{aligned} \tag{6.2}$$

Pri tem sta a in b parametra za nagrado in kazen, $0 < a < 1$, $0 \leq b < 1$. Ta dva parametra vplivata na velikost spremembe verjetnosti v posameznem koraku. Taka shema se imenuje tudi *linearna nagrajevalno-kaznovalna shema* ali L_{R-P} (*linear reward-penalty*) [13].

Če vstavimo enačbi 6.2 v enačbe 6.1, dobimo shemo L_{R-P}:

$$\begin{aligned}
&\alpha(n) = \alpha_i,\ \beta(n) = 0 : \qquad (6.3)\\
&\qquad p_j(n+1) = (1-a)p_j(n), \quad \text{za vse } j \neq i\\
&\qquad p_i(n+1) = p_i(n) + a[1 - p_i(n)], \quad \text{ker} \sum_{j=1, j\neq i}^{r} p_j(n) = 1 - p_i(n)\\
&\alpha(n) = \alpha_i,\ \beta(n) = 1 :\\
&\qquad p_j(n+1) = \frac{b}{r-1} + (1-b)p_j(n), \quad \text{za vse } j \neq i\\
&\qquad p_i(n+1) = (1-b)p_i(n).
\end{aligned}$$

Na začetku običajno postavimo verjetnosti vseh akcij na enako vrednost:

$$p_j(0) = \frac{1}{r}\,, \qquad j = 1, ..., r\,,$$

kjer je r število akcij.

Znano je [11], da je shema L_{R-P} ergodična. To pomeni, da vektor verjetnosti akcij $\mathbf{p}(n)$ konvergira k neki vrednosti $\mathbf{p}^*$, ki je neodvisna od začetnih verjetnosti $\mathbf{p}(0)$. Možno je pokazati [11], da so asimptotične verjetnosti akcij enake

$$\lim_{n\to\infty} E[p_i(n)] = \frac{1/c_i}{\sum_{j=1}^{r} 1/c_j}\,, \qquad i = 1, ..., r. \qquad (6.4)$$

Dobra lastnost sheme L_{R-P} je tudi ta, da se ob spremembi okolja (sprememba verjetnosti kaznovanja akcij) lahko sčasoma prilagodi novi situaciji. To je zelo pomembno, kajti pri iskalnem postopku lahko predpostavimo, da najboljša smer iskanja ni v vsaki točki enaka, ampak se lahko v iskalnem prostoru spreminja.

Uporabili smo modificiran algoritem L_{R-P}, pri katerem se v primeru kazni ($\beta = 1$) verjetnosti ostalih akcij povečajo za enak prirastek. Torej

$$\begin{aligned}
&\alpha(n) = \alpha_i,\ \beta(n) = 1 : \qquad (6.5)\\
&\qquad p_j(n+1) = p_j(n) + \frac{b}{r-1}p_i(n), \quad \text{za vse } j \neq i\\
&\qquad p_i(n+1) = (1-b)p_i(n).
\end{aligned}$$

Ker se verjetnost i-te akcije zmanjša za $b \cdot p_i(n)$, se verjetnost ostalih poveča za $\frac{b \cdot p_i(n)}{r-1}$. Eksperimenti so pokazali, da ta modificaran algoritem deluje nekoliko bolje od običajnega L_{R-P}

6.5 Uporaba modificiranega algoritma L_{R-P} pri ocenjevanju gibanja

Iskalni algoritem za ocenjevanje gibanja naj bi vsakemu makrobloku našel čimbolj podoben blok v referenčnem okviru. Iskanje v referenčnem okviru je seveda lokalno omejeno na določeno področje. V sredini tega področja je pozicija trenutnega makrobloka, parameter zamika (W_{SIZE}) pa določa, kako daleč od trenutnega makrobloka še iščemo bloke v referenčnem okviru.

Algoritem L_{R-P} lahko uporabimo za spreminjanje verjetnosti premikanja v posameznih smereh glede na uspešnost premika. Pri tem premiki predstavljajo akcije avtomata, uspešnost premika pa oceno okolja. Premik je uspešen, kadar je v ciljni točki kriterijska funkcija manjša kakor v začetni.

Na začetku iskanja najboljšega referenčnega področja je izbrano sredinsko področje v referenčni sliki. Algoritem lahko izbere 4 možne premike: levo, gor, desno in dol. To so 4 akcije, ki imajo na začetku enako verjetnost (v kolikor nimamo druge informacije). S pomočjo generatorja naključnih števil in glede na trenutne verjetnosti izberemo naslednjo akcijo oziroma smer premika. Na novi lokaciji se izračuna napaka in rezultat se primerja s trenutno najboljšim. Če je napaka manjša, potem se verjetnost trenutni akciji poveča (akcija se nagradi), ostalim ustrezno zmanjša, izhodišče za naslednjo akcijo pa se premakne v smeri izbrane akcije. V nasprotnem primeru se verjetnost izbrani akciji zmanjša (kazen), ostalim akcijam se poveča in izhodišče ostane na isti lokaciji. S tem je bil zaključen korak algoritma in na vrsti je zopet izbira akcije s pomočjo generatorja naključnih števil. Koliko korakov bomo izvedli, je odvisno predvsem od tega, kako dobre rezultate želimo in koliko časa smo za to pripravljeni porabiti.

Pričakujemo, da algoritem L_{R-P} dovolj hitro prilagodi verjetnosti tako, da se pomikamo v obetavni smeri, torej taki, v kateri kriterijska funkcija pada. Če se v nekem delu iskalnega prostora razmere spremenijo, mora algoritem dovolj hitro prilagoditi verjetnosti. Paziti je treba, da razdalja premika ni prevelika in da učilni parameter a (pri shemi L_{R-P} sta a in b običajno kar enaka), ki določa velikost spremembe verjetnosti, ni premajhen.

6.6 Preizkušanje osnovnega algoritma

V osnovnem algoritmu (tako bomo imenovali modificiran L_{R-P}, vendar brez dodatkov) se je kot najboljša izkazala vrednost učilnega parametra 0,2. Razdalja, za koliko se premaknemo v izbrani smeri, pa je postavljena na 1.

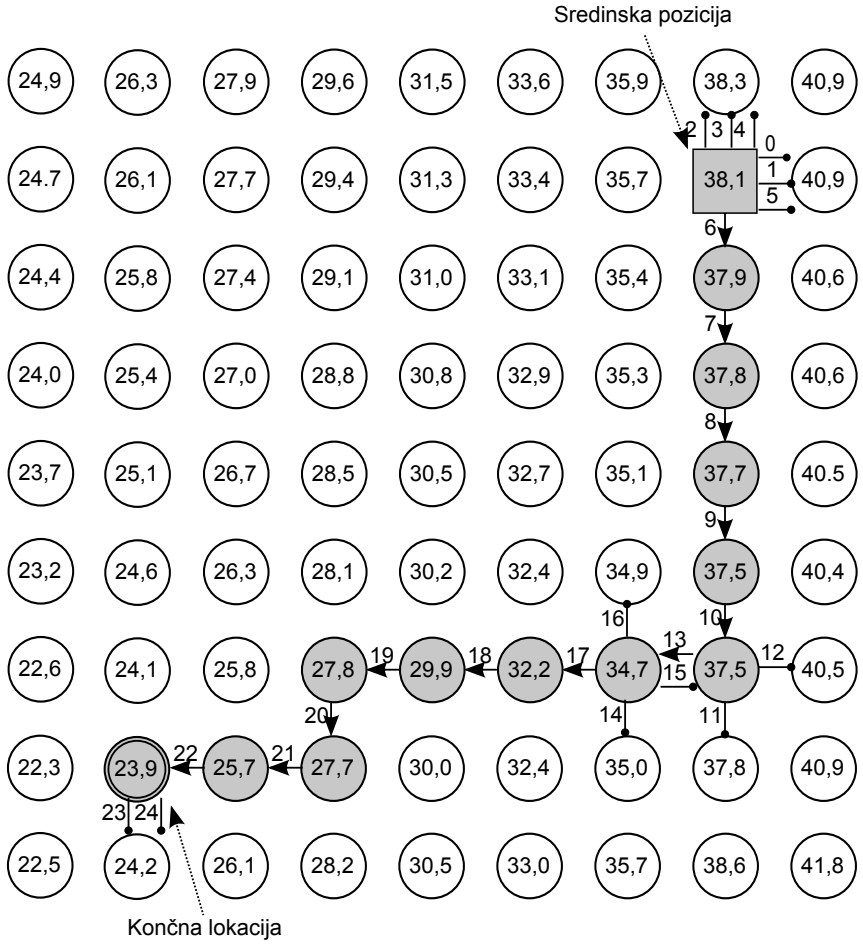
V tabeli 6.1 in na sliki 6.1 vidimo primer poteka algoritma v 25. korakih. Slika 6.2 grafično prikazuje spreminjanje verjetnosti posameznih akcij pri izvajanju algoritma.

Tabela 6.1: Potek osnovnega algoritma.

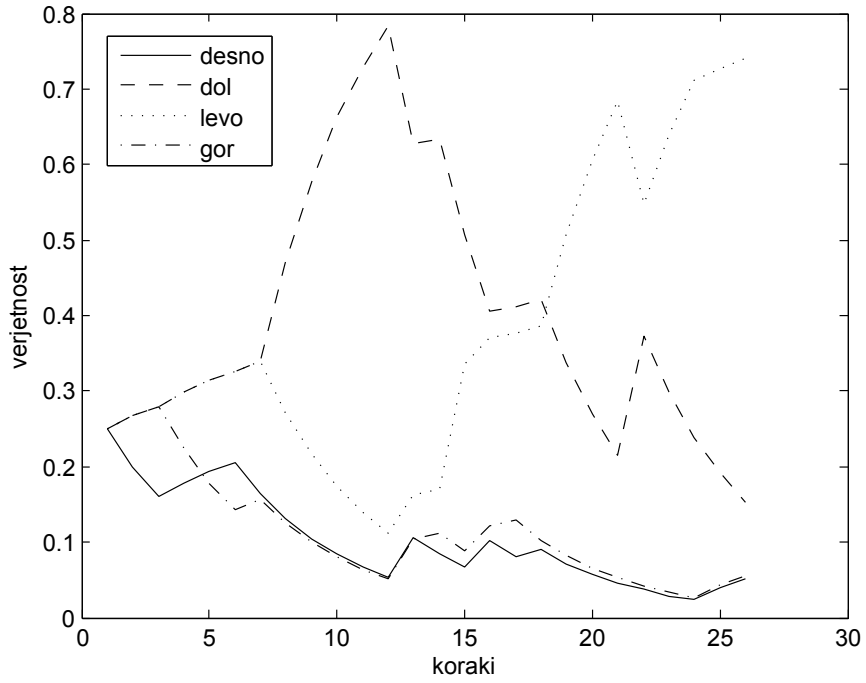
korak	**akcija**	**napaka**	**posledica**	**verjetnosti**			
				desno	**dol**	**levo**	**gor**
ini	-	38.18	-	0.250	0.250	0.250	0.250
0	desno	40.86	kazen	0.200	0.267	0.267	0.267
1	desno	40.86	kazen	0.160	0.280	0.280	0.280
2	gor	38.35	kazen	0.179	0.299	0.299	0.224
3	gor	38.35	kazen	0.194	0.314	0.314	0.179
4	gor	38.35	kazen	0.206	0.326	0.326	0.143
5	desno	40.86	kazen	0.164	0.339	0.339	0.157
6	dol	37.97	nagrada	0.132	0.471	0.271	0.126
7	dol	37.83	nagrada	0.105	0.577	0.217	0.101
8	dol	37.74	nagrada	0.084	0.662	0.174	0.080
9	dol	37.57	nagrada	0.067	0.729	0.139	0.064
10	dol	37.53	nagrada	0.054	0.783	0.111	0.051
11	dol	37.89	kazen	0.106	0.627	0.163	0.104
12	desno	40.52	kazen	0.085	0.634	0.170	0.111
13	levo	34.77	nagrada	0.068	0.507	0.336	0.089
14	dol	35.04	kazen	0.102	0.406	0.370	0.122
15	desno	37.53	kazen	0.081	0.412	0.377	0.129
16	gor	34.91	kazen	0.090	0.421	0.386	0.103
17	levo	32.23	nagrada	0.072	0.337	0.508	0.083
18	levo	29.93	nagrada	0.058	0.269	0.607	0.066
19	levo	27.80	nagrada	0.046	0.216	0.685	0.053
20	dol	27.79	nagrada	0.037	0.372	0.548	0.042
21	levo	25.76	nagrada	0.029	0.298	0.639	0.034
22	levo	23.95	nagrada	0.024	0.238	0.711	0.027
23	dol	24.24	kazen	0.039	0.191	0.727	0.043
24	dol	24.24	kazen	0.052	0.153	0.740	0.056

Zelo uporabna lastnost algoritma je, da ima uporabnik možnost nadzirati čas izvajanja in s tem tudi kakovost dobljenih rezultatov. To se doseže s številom korakov izvajanja algoritma. Večje število korakov pomeni daljši čas izvajanja in boljše rezultate. Na sliki 6.3 vidimo graf porabljenega časa v odvisnosti od števila korakov. Graf na sliki 6.4 prikazuje povprečno absolutno napako (MAE) v odvisnosti od števila korakov.

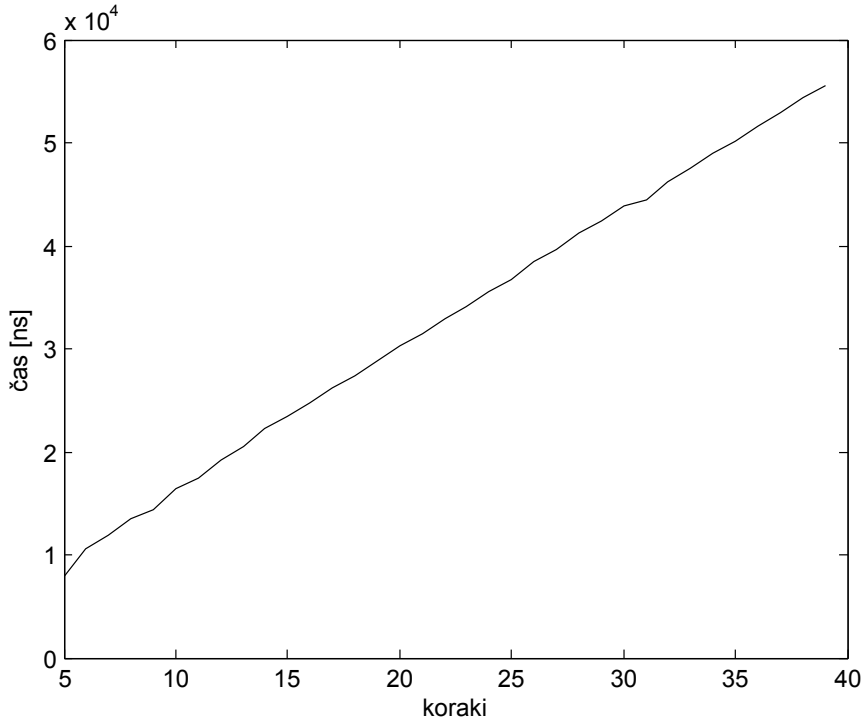
Po pričakovanju je graf na sliki 6.5, kjer je prikazano razmerje med mapako in



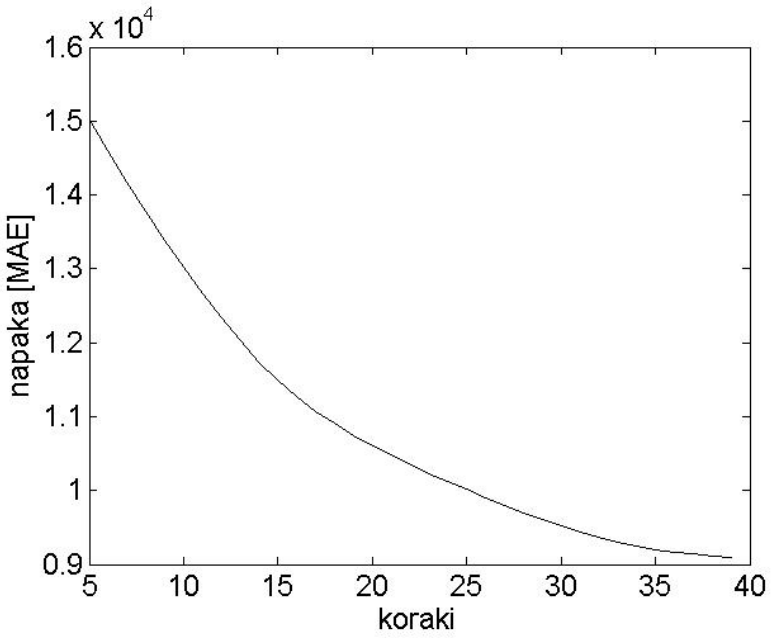
Slika 6.1: Primer poteka osnovnega algoritma.



Slika 6.2: Grafični prikaz spreminjanja verjetnosti posameznih akcij.



Slika 6.3: Graf porabljenega časa v odvisnosti od števila korakov.



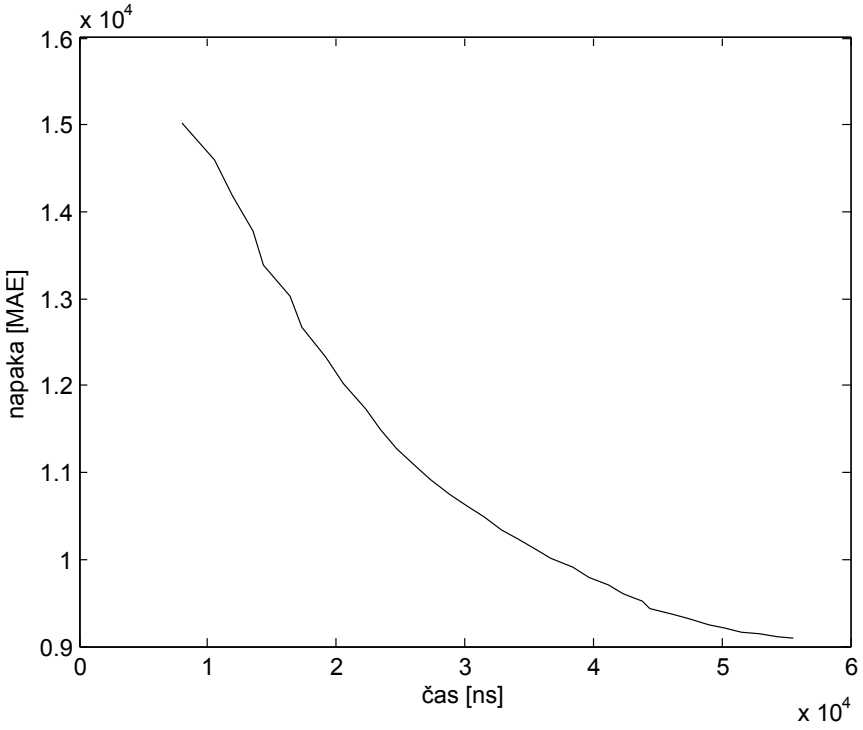
Slika 6.4: Graf napake v odvisnosti od števila korakov.

porabljenim časom, zelo podobem grafu 6.4, kajti razmerje med časom in številom korakov je praktično linearno (graf 6.3).

6.7 Izboljšave algoritma

Osnovni algoritem ima nekaj pomanjkljivosti. Večino izmed bolj očitnih pomanjkljivosti se da odpraviti, vendar samo odpravljanje pomanjkljivosti nekaj stane. Bodisi se to pozna na hitrosti algoritma ali na potrebi po pomnilniku, ki ga algoritem potrebuje pri svojem izvajanju. Najpreprostejša in dokaj učinkovita izboljšava algoritma je *uporaba različnih faktorjev kazni in nagrade, a in b.* Če se pri izvajanju algoritma ista akcija pogosto izkaže za uspešno, je velika verjetnost, da bo ista akcija izbrana tudi v naslednjih korakih. Problem nastane, ko pridemo do pozicije v izbrani smeri, kjer je vrednost napake večja od trenutne. V tem primeru se v osnovnem algoritmu verjetnost izbrane akcije sicer nekoliko zmanjša, vendar je to še vedno najbolj verjetna akcija naslednjega koraka. Po drugi strani je poskušanje v isti smeri nesmiselno, saj je vrednost napake nespremenjena. To hibo se lahko odpravi s tem, da se poveča faktor kazni. V tabeli 6.2 in na sliki 6.6 vidimo potek izboljšanega algoritma, kjer je uporabljen faktor nagrade 0,2, faktor kazni pa 0,5.

Na ta način se, po meritvah, doseže približno 3% boljše rezultate, s tem, da ni razlik



Slika 6.5: Graf napake v odvisnosti od porabljenega časa.

v času izvajanja algoritma.

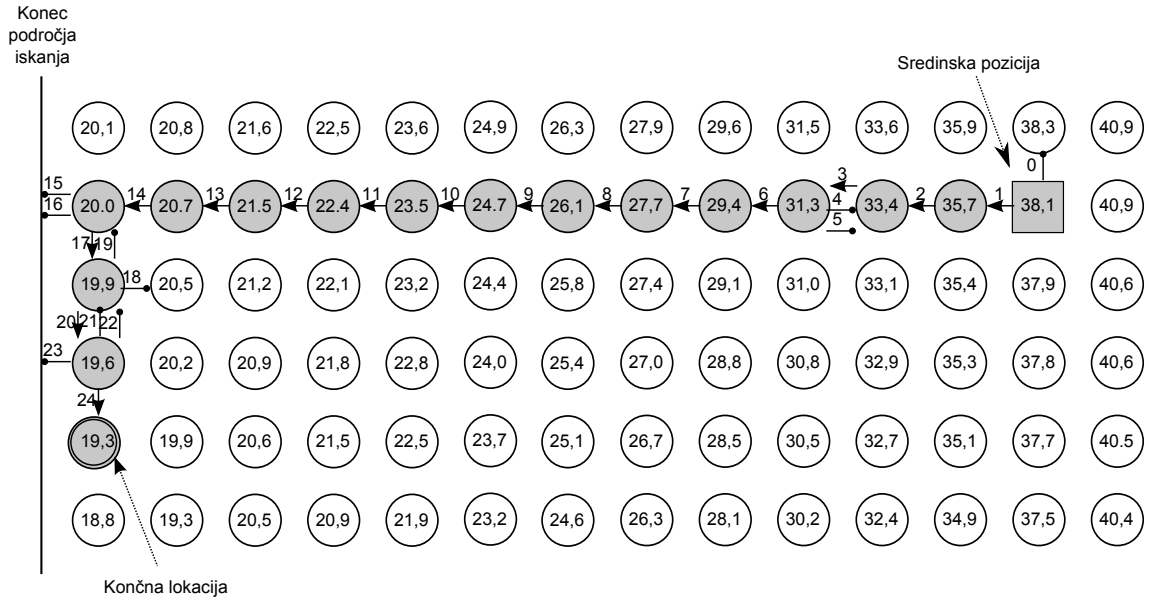
Drug, bolj zapleten, a tudi bolj učinkovit način reševanja istega problema, je *z uporabo učečega avtomata (UA) na vsaki lokaciji* v področju iskanja. Inicializacija verjetnosti se v tem primeru opravi le na začetku okvira, saj pri iskanju referenčnega področja enega makrobloka ni mogoče priti večkrat na isto lokacijo v področju iskanja.

Vrednost napake pri tem pristopu k izboljševanju osnovnega algoritma je približno 5% manjša, vendar porabimo v povprečju odstotek več časa za izvajanje. Pomanjkljivost te rešitve je večja potreba po pomnilniku, saj si mora izboljšani algoritem zapomniti štiri verjetnosti za vsako možno pozicijo v področju iskanja, medtem ko osnovni algoritem hrani le štiri verjetnosti.

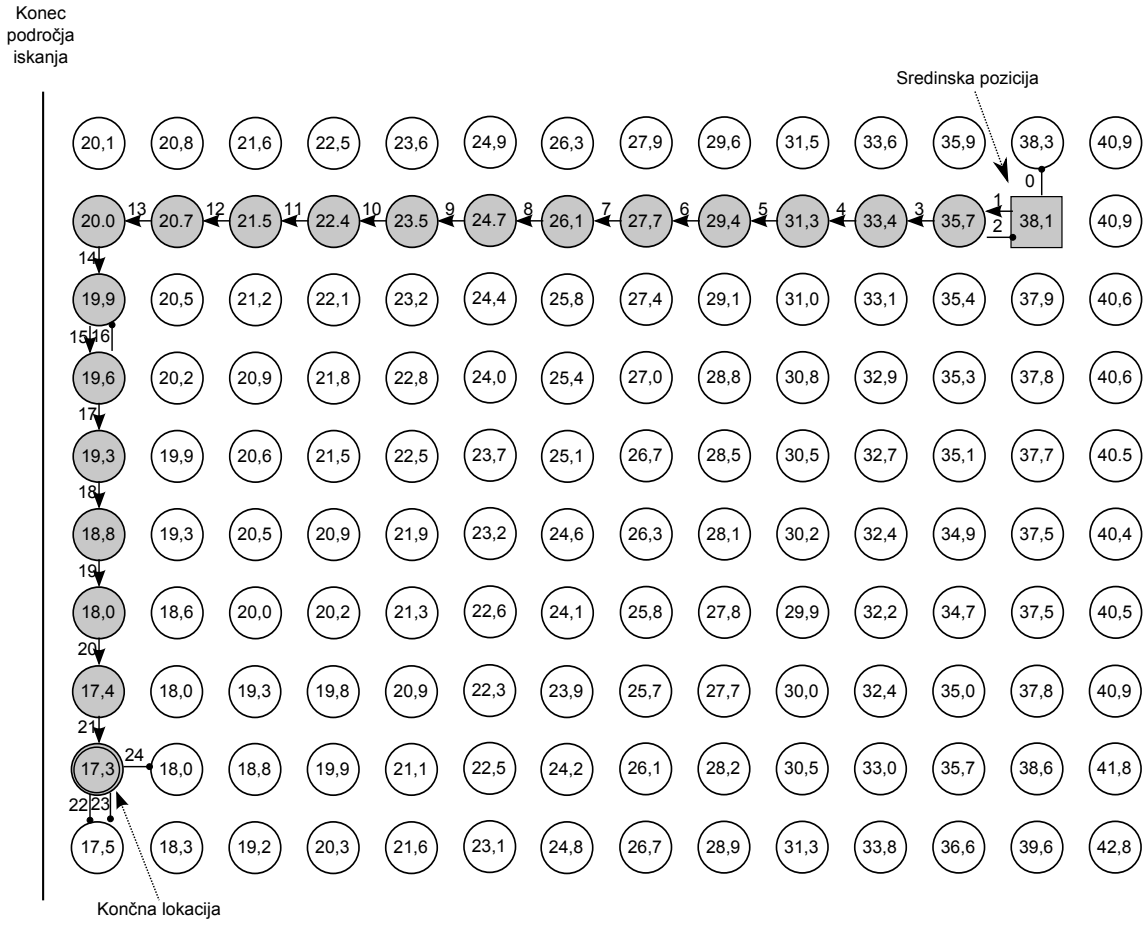
Tabela 6.3 in slika 6.7 prikazujeta potek modificiranega algoritma.

Tabela 6.2: Potek algoritma s povečanim faktorjem kazni.

korak	**akcija**	**napaka**	**posledica**	**verjetnosti**			
				desno	**dol**	**levo**	**gor**
ini	-	38.184	-	0.250	0.250	0.250	0.250
0	gor	38.35	kazen	0.292	0.292	0.292	0.125
1	levo	35.74	nagrada	0.233	0.233	0.433	0.100
2	levo	33.46	nagrada	0.187	0.187	0.547	0.080
3	levo	31.39	nagrada	0.149	0.149	0.637	0.064
4	desno	33.46	kazen	0.075	0.174	0.662	0.089
5	desno	33.46	kazen	0.037	0.187	0.675	0.101
6	levo	29.45	nagrada	0.030	0.149	0.740	0.081
7	levo	27.71	nagrada	0.024	0.119	0.792	0.065
8	levo	26.14	nagrada	0.019	0.096	0.833	0.052
9	levo	24.72	nagrada	0.015	0.076	0.867	0.042
10	levo	23.50	nagrada	0.012	0.061	0.893	0.033
11	levo	22.43	nagrada	0.010	0.049	0.915	0.027
12	levo	21.52	nagrada	0.008	0.039	0.932	0.021
13	levo	20.74	nagrada	0.006	0.031	0.945	0.017
14	levo	20.07	nagrada	0.005	0.025	0.956	0.014
15	levo	rob	kazen	0.164	0.184	0.478	0.173
16	levo	rob	kazen	0.244	0.264	0.239	0.253
17	dol	19.90	nagrada	0.195	0.411	0.191	0.202
18	desno	20.53	kazen	0.098	0.444	0.224	0.235
19	gor	20.07	kazen	0.137	0.483	0.263	0.117
20	dol	19.67	nagrada	0.109	0.586	0.210	0.094
21	gor	19.90	kazen	0.125	0.602	0.226	0.047
22	gor	19.90	kazen	0.133	0.610	0.234	0.023
23	levo	rob	kazen	0.172	0.649	0.117	0.062
24	dol	19.36	nagrada	0.137	0.719	0.094	0.050



Slika 6.6: Primer poteka algoritma z učečim avtomatom s povečanim faktorjem kazni.



Slika 6.7: Primer poteka algoritma z učečim avtomatom na vsaki lokaciji področja iskanja.

Tabela 6.3: Potek algoritma z učečim avtomatom na vsaki lokaciji področju iskanja.

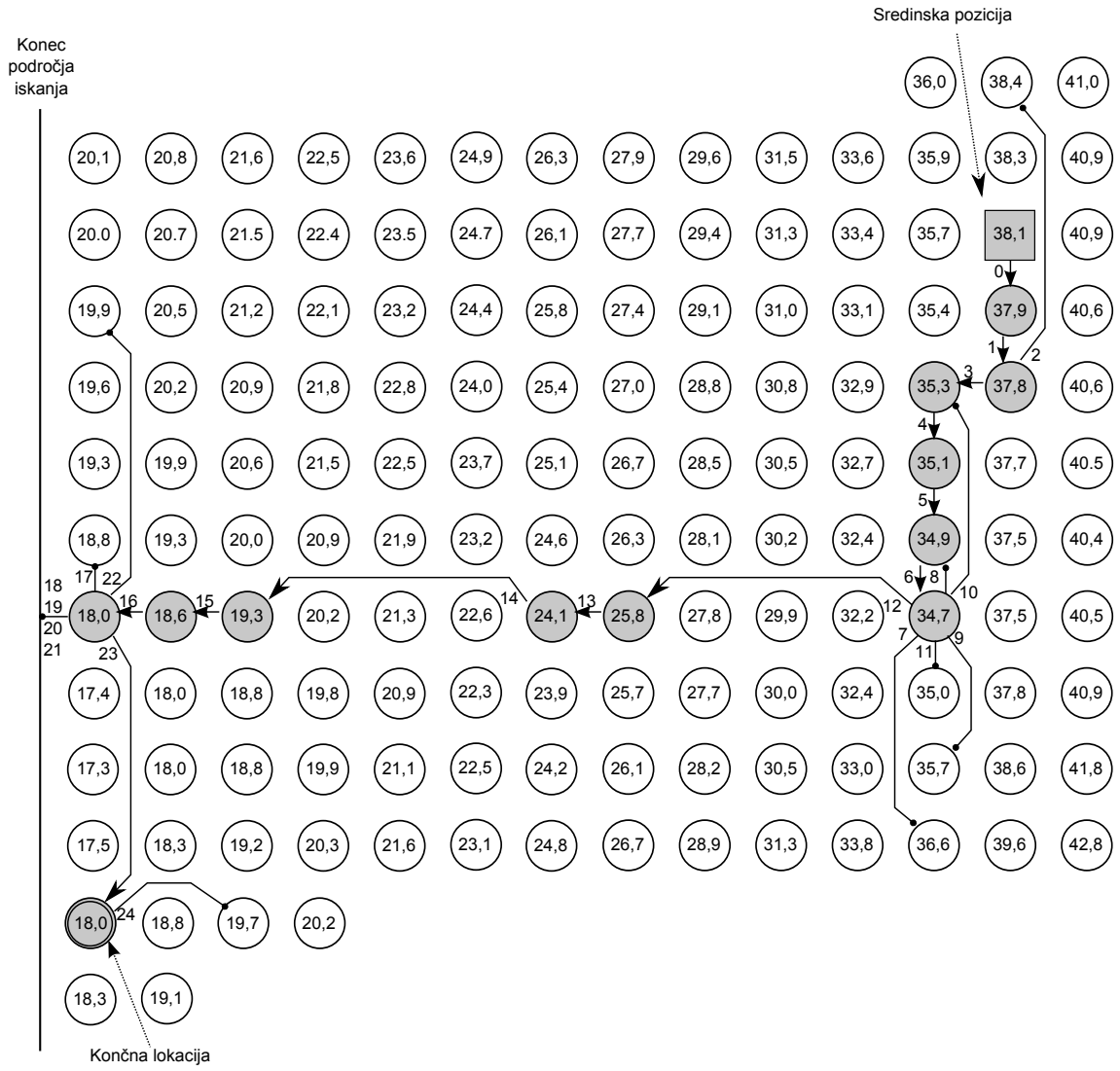
korak	**akcija**	**verjetnosti (pred)**				**napaka**	**posledica**	**verjetnosti (po)**			
		desno	**dol**	**levo**	**gor**			**desno**	**dol**	**levo**	**gor**
ini	-	0.207	0.125	0.097	0.571	-					
0	gor	0.032	0.019	0.026	0.923	38.35	kazen	0.245	0.163	0.135	0.457
1	levo	0.009	0.009	0.936	0.046	35.74	nagrada	0.196	0.130	0.308	0.365
2	desno	0.196	0.130	0.308	0.365	38.18	kazen	0.007	0.009	0.937	0.046
3	levo	0.000	0.000	1.000	0.000	33.46	nagrada	0.006	0.007	0.950	0.037
4	levo	0.000	0.000	1.000	0.000	31.39	nagrada	0.000	0.000	1.000	0.000
5	levo	0.000	0.000	1.000	0.000	29.45	nagrada	0.000	0.000	1.000	0.000
6	levo	0.000	0.006	0.993	0.001	27.71	nagrada	0.000	0.000	1.000	0.000
7	levo	0.000	0.000	0.999	0.001	26.14	nagrada	0.000	0.005	0.995	0.000
8	levo	0.000	0.000	1.000	0.000	24.72	nagrada	0.000	0.000	0.999	0.000
9	levo	0.000	0.000	1.000	0.000	23.50	nagrada	0.000	0.000	1.000	0.000
10	levo	0.000	0.000	1.000	0.000	22.43	nagrada	0.000	0.000	1.000	0.000
11	levo	0.027	0.027	0.895	0.051	21.52	nagrada	0.000	0.000	1.000	0.000
12	levo	0.000	0.000	1.000	0.000	20.74	nagrada	0.021	0.022	0.916	0.041
13	levo	0.032	0.854	0.040	0.074	20.07	nagrada	0.000	0.000	1.000	0.000
14	dol	0.001	0.998	0.001	0.001	19.90	nagrada	0.025	0.883	0.032	0.059
15	dol	0.008	0.978	0.007	0.007	19.67	nagrada	0.001	0.998	0.001	0.001
16	gor	0.001	0.998	0.001	0.001	19.90	kazen	0.008	0.979	0.007	0.006
17	dol	0.001	0.994	0.002	0.002	19.36	nagrada	0.006	0.983	0.006	0.005
18	dol	0.022	0.908	0.031	0.040	18.80	nagrada	0.001	0.996	0.002	0.002
19	dol	0.002	0.993	0.002	0.002	18.02	nagrada	0.017	0.927	0.024	0.032
20	dol	0.001	0.997	0.002	0.001	17.46	nagrada	0.002	0.995	0.002	0.002
21	dol	0.004	0.988	0.004	0.004	17.31	nagrada	0.000	0.998	0.001	0.001
22	dol	0.003	0.987	0.004	0.005	17.59	kazen	0.070	0.790	0.070	0.070
23	dol	0.003	0.987	0.004	0.005	17.59	kazen	0.123	0.632	0.122	0.123
24	desno	0.120	0.640	0.120	0.120	18.01	kazen	0.098	0.641	0.131	0.131

Seveda se takoj postavi vprašanje, kaj se zgodi, če se uporabita oba pristopa hkrati. V tem primeru imamo učeči avtomat na vsaki lokaciji in faktor kazni večji od faktorja nagrade. Po meritvah se v povprečju rezultat izboljša za okrog 6%. Morda bi pričakovali 7% izboljšanje, saj v nobenem primeru ne želimo, da se napačna akcija ponavlja. Vendar si algoritem že brez povečanega faktorja kazni zapomni, kje je konec področja, kjer bi dali prednost določeni akciji, in tega ne ponavlja v naslednjem makrobloku. Zaradi tega izboljšava pri povečanem faktorju kazni ni tako velika kot pri osnovnem algoritmu.

Nobena od zgoraj omenjenih izboljšav pa ne odpravi morda največje pomanjkljivosti. To je dejstvo, da se algoritem lahko ujame v lokalni minimum in v njem ostane.

Logični odgovor na to je uporaba različnih razdalj, za katere se bomo premaknili v izbrano smer. Takoj za tem pa se pojavi vprašanje, kako izbrati pravo razdaljo premika. Najbolj preprosta in tudi zelo učinkovita je *vpeljava dodatnega učečega avtomata, ki bo skrbel za odločitve o razdaljah premikanja*. Doda se torej nov učeči avtomat, ki ima toliko akcij, kolikor različnih razdalj je dovoljenih. Na začetku ima vsaka akcija enako verjetnost. Verjetnosti akcij se posodabljajo hkrati z verjetnostmi akcij izbire strani.

Na ta način se pridobi okoli 8% manjše vrednosti napake kot pri osnovnem algoritmu. Kazen zaradi dvojnega posodabljanja verjetnosti je približno 2% več časa za izvajanje. V tabeli 6.4 in na sliki 6.8 vidimo potek algoritma s spremenljivimi razdaljami.



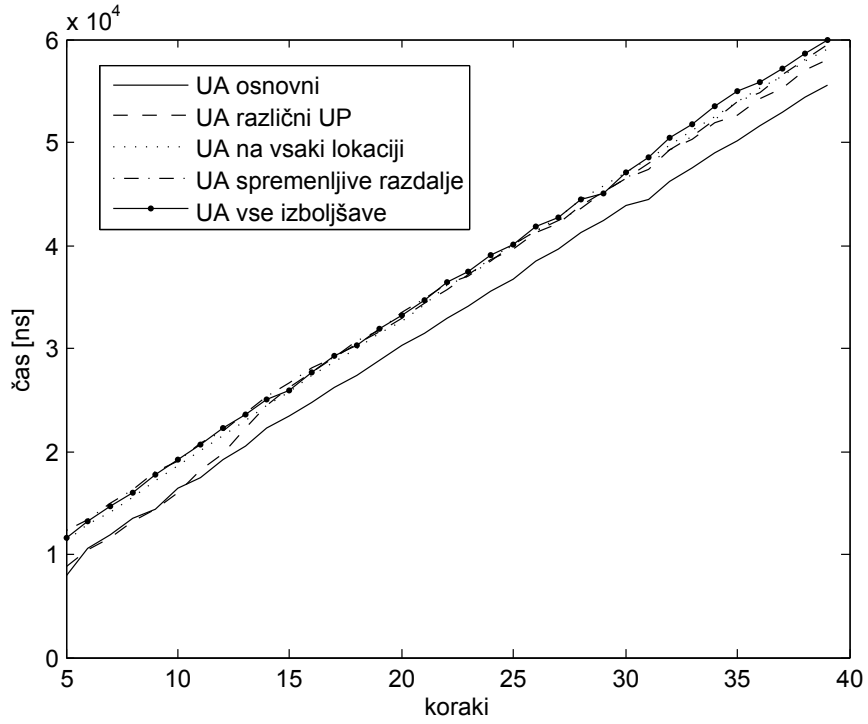
Slika 6.8: Primer poteka algoritma s spremenljivimi razdaljami.

Tabela 6.4: Potek algoritma s spremenljivimi razdaljami.

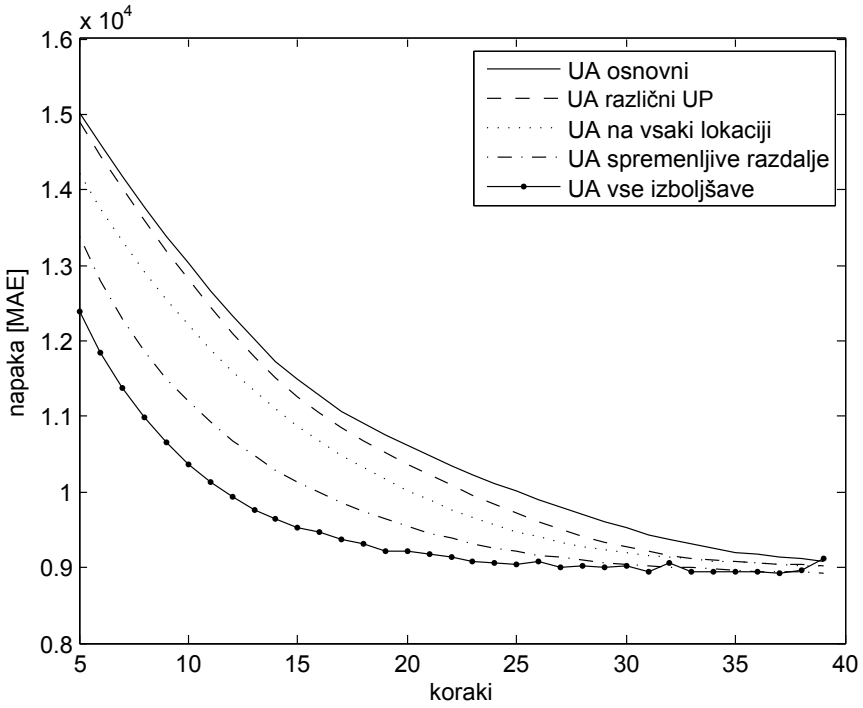
korak	**razdalja**	**akcija**	**napaka**	**posledica**	**verjetnosti akcij**				**verjetnosti korakov**			
					desno	**dol**	**levo**	**gor**	**1**	**2**	**3**	**4**
ini	-	-	38.184	-	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250
0	1	dol	37.97	nagrada	0.200	0.400	0.200	0.200	0.400	0.200	0.200	0.200
1	1	dol	37.83	nagrada	0.160	0.520	0.160	0.160	0.520	0.160	0.160	0.160
2	4	gor	38.42	kazen	0.187	0.547	0.187	0.080	0.547	0.187	0.187	0.080
3	1	levo	35.30	nagrada	0.149	0.437	0.349	0.064	0.637	0.149	0.149	0.064
4	1	dol	35.15	nagrada	0.119	0.550	0.279	0.051	0.710	0.119	0.119	0.051
5	1	dol	34.91	nagrada	0.096	0.640	0.224	0.041	0.768	0.096	0.096	0.041
6	1	dol	34.77	nagrada	0.076	0.712	0.179	0.033	0.814	0.076	0.076	0.033
7	3	dol	36.63	kazen	0.195	0.356	0.298	0.151	0.827	0.089	0.038	0.046
8	1	gor	34.91	kazen	0.220	0.381	0.323	0.076	0.414	0.227	0.176	0.183
9	2	dol	35.74	kazen	0.284	0.191	0.386	0.139	0.451	0.114	0.214	0.221
10	3	gor	35.30	kazen	0.307	0.214	0.409	0.070	0.487	0.149	0.107	0.257
11	1	dol	35.04	kazen	0.343	0.107	0.445	0.105	0.244	0.230	0.188	0.338
12	4	levo	25.87	nagrada	0.274	0.086	0.556	0.084	0.195	0.184	0.151	0.470
13	1	levo	24.14	nagrada	0.219	0.068	0.645	0.067	0.356	0.147	0.120	0.376
14	4	levo	19.34	nagrada	0.175	0.055	0.716	0.054	0.285	0.118	0.096	0.501
15	1	levo	18.61	nagrada	0.140	0.044	0.773	0.043	0.428	0.094	0.077	0.401
16	1	levo	18.02	nagrada	0.112	0.035	0.818	0.034	0.542	0.075	0.062	0.321
17	1	gor	18.80	kazen	0.118	0.041	0.824	0.017	0.271	0.166	0.152	0.411
18	4	levo	rob	kazen	0.255	0.178	0.412	0.155	0.271	0.166	0.152	0.411
19	4	levo	rob	kazen	0.324	0.247	0.206	0.223	0.271	0.166	0.152	0.411
20	2	levo	rob	kazen	0.358	0.281	0.103	0.258	0.271	0.166	0.152	0.411
21	4	levo	rob	kazen	0.376	0.298	0.051	0.275	0.271	0.166	0.152	0.411
22	4	gor	19.90	kazen	0.421	0.344	0.097	0.137	0.340	0.234	0.221	0.206
23	4	dol	18.00	nagrada	0.337	0.475	0.078	0.110	0.272	0.187	0.176	0.364
24	2	desno	19.77	kazen	0.169	0.531	0.134	0.166	0.303	0.094	0.208	0.396

Tudi to izboljšavo lahko kombiniramo s prejšnjima dvema. V najboljši kombinaciji dobimo okrog 10% boljši rezultat, za to pa porabimo 3% več časa.

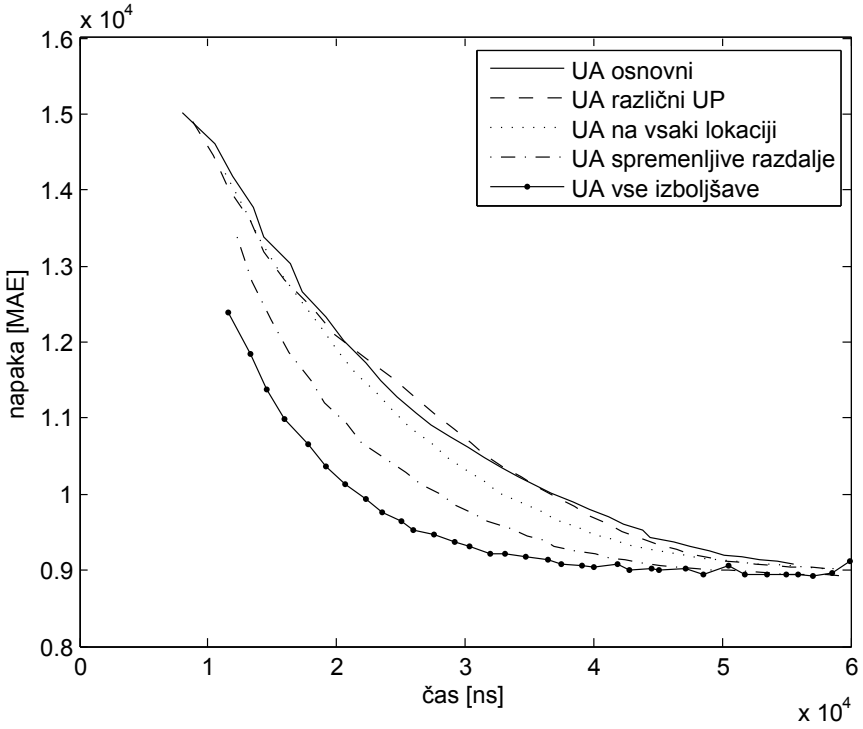
Na slikah 6.9, 6.10 in 6.11 vidimo grafe, ki prikazuje jo zgoraj omenjene izboljšave v odvisnosti od števila korakov. Pri tem *UP* v legendi pomeni učni parameter.



Slika 6.9: Primerjalni graf porabljenega časa v odvisnosti od števila korakov.



Slika 6.10: Primerjalni graf napake v odvisnosti od števila korakov.



Slika 6.11: Primerjalni graf napake v odvisnosti od porabljenega časa.

Poglavje 7

Primerjava algoritmov za ocenjevanje gibanja

Primerjava algoritmov za kompenzacijo gibanja je zelo nehvaležna zadeva. V poštev moramo vzeti zelo veliko različnih parametrov, po katerih lahko razvrstimo algoritme. Ustrezen algoritem lahko izberemo šele, ko vemo, v kakšnem sistemu bo video kodiran in za kakšen namen bo kodiran video uporabljen.

Glavna in najbolj pomembna parametra sta seveda kakovost dobljenih rezultatov in čas izvajanja oziroma procesorska zahtevnost algoritma. Poleg omenjenih dveh je pogosto pomemben tudi podatek, koliko pomnilnika za podatke potrebujemo pri izvajanju algoritma. V mnogo primerih je zaželeno, da se del ali celoten algoritem izvaja na strojni opremi. V tem primeru je seveda bistven podatek, kakšna podpora za ocenjevanje gibanja obstaja v strojni opremi, oziroma kako zahteven bi bil razvoj strojne opreme za pospeševanje algoritma.

Če je edini kriterij za izbiro algoritma kakovost rezultatov, je seveda *Polno iskanje* prava izbira. Ta algoritem nam zagotavlja optimalen rezultat, poleg tega je tudi zelo preprost za implementacijo in ne rabi veliko pomnilnika. Edina, a v večini primerov odločilna pomanjkljivost je čas izvajanja algoritma. Algoritem porabi daleč največ časa za svoje izvajanje, kar je konec koncev tudi povod za iskanje novih, hitrejših algoritmov. Drug ekstremni način iskanja najboljšega referenčnega področja bi bil, da se enostavno vedno vzame sredinsko področje in iskanja sploh ni. Tu seveda ne moremo govoriti o algoritmu za ocenjevanje gibanja, temveč enostavno vzamemo nespremenjen prejšnji okvir kot referenčni. Ta pristop je najhitrejši možen, je tudi zelo preprost in ne porabi veliko pomnilnika, seveda pa so rezultati temu primerno slabi.

Vsi ostali algoritmi se po svoji hitrosti in kakovosti nahajajo nekje med omenjenima

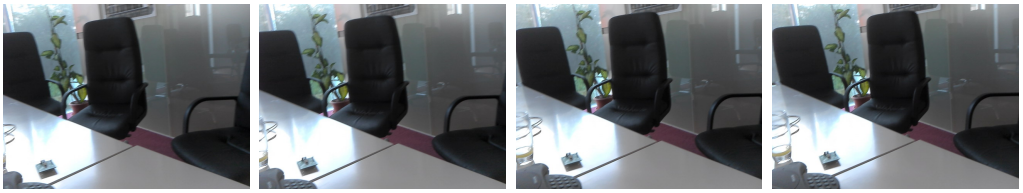
dvema. Po pričakovanju ni noben od njih v vseh pogledih najboljši. Večinoma se razmerje med kakovostjo in hitrostjo ohranja v vseh, s tem da pri nekaterih dobimo boljše rezultate, a za to porabimo več časa.

7.1 Okolje za primerjavo algoritmov

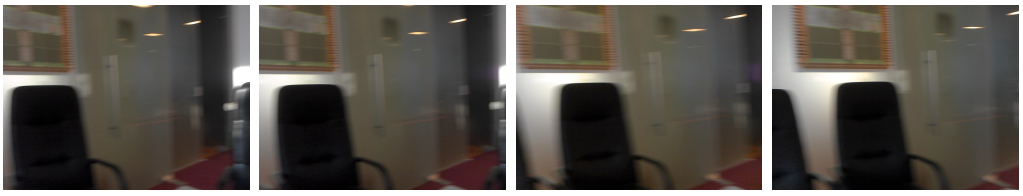
Algoritme bomo primerjali na podlagi treh različnih video sekvenc. V prvi sekvenci gre za počasno premikanje kamere v horizontalni ali vertikalni smeri. Izsek vidimo na sliki 7.1. V tem primeru bo najboljša pozicija referenčnega makrobloka blizu sredinske pozicije in je zato ne bo težko odkriti tudi s slabšimi algoritmi.

Druga sekvenca vsebuje hitrejše premikanje kamere v horizontalni ali vertikalni smeri. Zaporedne okvire iz te sekvence vidimo na sliki 7.2. V tem primeru je najboljša lokacija daleč od sredinske pozicije, zato imajo tudi hitrejši algoritmi, ki po pravilu pričakujejo referenčno področje blizu, večje težave.

Zadnja video sekvenca je najbolj kompleksna. Izsek štirih okvirov vidimo na sliki 7.3. Tu imamo premikanje predmeta na sceni in premikanje kamere v vse smeri. Vse sekvence so sestavljene iz okvirov dimenzije 640x480 pik.



Slika 7.1: Izsek iz sekvence s počasnim premikanjem kamere.



Slika 7.2: Izsek iz sekvence s hitrim premikanjem kamere.

Parameter napake se bo računal s pomočjo povprečne absolutne napake po enačbi 2.2.

Algoritem *Povprečna piramida* bo delal s tremi nivoji. Za iskanje najboljše pozicije na posameznem nivoju bo uporabljeno *Iskanje v treh korakih*.

Slika 7.3: Izsek iz sekvence s kompleksnim premikanjem.

Stohastični algoritem za ocenjevanje gibanja bo delal s 25 koraki.

7.2 Rezultati

V tabelah 7.1, 7.2 in 7.3 vidimo izmerjene rezultate na vseh treh video sekvencah, urejene po naraščajoči vrednosti napake. Primerjava je narejena med vsemi algoritmi, predstavljenimi v poglavju 5.1, in petimi različicami algoritmov na podlagi učečih avtomatov (UA). To so: osnovni algoritem, algoritem z različnimi učilnimi parametri, algoritem z UA na vsaki poziciji v iskalnem področju, algoritem s spremenljivimi razdaljami in algoritem z vsemi omenjenimi izboljšavami. V tabelah vidimo povprečno napako in povprečen čas, porabljen za izvajanje algoritma nad enim okvirom. Poleg tega pa je tu zanimiv tudi podatek, ki pove, kolikokrat je bila klicana funkcija napake.

Tu lahko takoj opazimo, kako tip video sekvence vpliva na uspešnost algoritma. Pri sekvencah s preprostim premikanjem so rezultati zelo podobni, medtem ko se pri kompleksnih premikih pojavijo večje razlike v napaki.

Po pričakovanju je *Polno iskanje* dobilo najmanjšo povprečno napako, a za to porabilo daleč največ časa. Po drugi strani tudi ni presenečenje, da se ne porabi veliko časa, če se vedno izbere sredinsko lokacijo, vendar je napaka pri tem velika. Bolj zanimivo je opazovati ostale algoritme. Med standardnimi algoritmi se je kot najhitrejši izkazal algoritem *Eden naenkrat*, ki je v vrednosti napake pri sekvencah s preprostim gibanjem celo prekosil *Iskanje v treh korakih*. Najboljši rezultati so bili po pričakovanju dobljeni s *Povprečno piramido*, kar pa je porabilo tudi največ časa.

Algoritmi, osnovani na učečih avtomatih, se v izvedbi s 25 koraki tako po hitrosti kot po uspešnosti uvrščajo med *Povprečno piramido* in *Iskanje v treh korakih*. Morda bi pričakovali, da bo vsaj osnovni algoritem deloval hitreje kot iskanje v treh korakih, saj manjkrat računa vrednost napake (25-krat v primeru učečih avtomatov in 27-krat pri iskanju v treh korakih na makroblok), vendar porabi nekaj časa zaradi izbiranja akcije in posodabljanja.

Tabela 7.1: Rezultati meritev na video sekvenci s počasnim premikanjem.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število računanj**
Polno iskanje	9 518.88	688 358	42 450 785
UA z vsemi izboljšavami	10 785.82	36 417	1 838 844
UA s spremenljivimi razdaljami	10 826.02	35 970	1 775 080
UA na vsaki poziciji	11 114.08	35 970	1 980 313
Povprečna piramida	11 206.73	42 985	4 122 211
UA z različnimi učilnimi parametri	11 448.09	35 820	1 948 307
Eden naenkrat	11 809.67	15 671	962 784
UA osnovni	11 823.68	35 671	1 928 806
Iskanje v treh korakih	12 354.41	34 179	2 046 967
brez iskanja	21 054.54	1 194	78 000

Tabela 7.2: Rezultati meritev na video sekvenci s hitrim premikanjem.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število računanj**
Polno iskanje	16 010.10	69 1358	5 159 4031
UA z vsemi izboljšavami	16 884.71	3 5432	211 3821
UA s spremenljivimi razdaljami	17 182.61	3 3580	199 5565
Povprečna piramida	17 250.40	3 9506	455 5643
UA na vsaki poziciji	17 949.32	3 6419	239 2983
UA z različnimi učilnimi parametri	18 197.69	3 5308	232 2655
Eden naenkrat	18 573.26	1 8641	134 9769
UA osnovni	18 638.83	3 4444	228 0873
Iskanje v treh korakih	19 869.43	3 4444	249 0164
brez iskanja	27 894.66	864	9 4800

Tabela 7.3: Rezultati meritev na video sekvenci s kompleksnim premikanjem.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število računanj**
Polno iskanje	8 229.86	708 774	393 159 578
Povprečna piramida	8 834.72	45 877	38 764 840
UA z vsemi izboljšavami	9 048.83	37 996	16 910 717
UA s spremenljivimi razdaljami	9 208.39	37 334	16 415 603
UA na vsaki poziciji	9 478.67	37 665	18 308 900
UA z različnimi učilnimi parametri	9 721.75	37 036	18 001 887
UA osnovni	10 007.28	36 970	17 819 791
Iskanje v treh korakih	10 488.34	35 811	18 946 293
Eden naenkrat	10 926.77	15 894	8 337 948
brez iskanja	17 038.42	1 374	722 400

Glede na to, da imamo možnost kontroliranja časa delovanja algoritma UA, je zanimiva primerjava z ostalimi algoritmi paroma, če je izbrano ustrezno število korakov, tako da se algoritma izvajata približno enako časa.

Rezultate primerjanj vidimo v tabelah 7.4, 7.5 in 7.6. Pri tem je pomembno poudariti, da primerjava z obema ekstremnima algoritmoma (*Polno iskanje* in brez iskanja) ni smiselna, ker oba zagotavljata najboljšo možno rešitev za dani parameter (napako ali čas).

Tabela 7.4: Primerjava algoritmov UA z algoritmom Eden naenkrat.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število korakov**
Eden naenkrat	10 926.77	15 894	-
UA osnovni	13 389.73	14 321	9
UA z različnime učilnimi parametri	12 809.64	15 943	10
UA na vsaki poziciji	12 923.78	15 579	8
UA s spremenljivimi razdaljami	12 295.62	14 933	7
UA vse izboljšave	10 983.29	15 993	7

Iz povedanega in prikazanega je razvidno, da je, če je pomembna hitrost algoritma, najboljša izbira algoritem *Eden naenkrat*. To je preprost algoritem, ki nas dokaj hitro pripelje do sprejemljivih rezultatov. V primerjavi z njim so algoritmi, ki temeljijo na UA, nekoliko počasnejši, vendar dajo nekoliko boljše rezultate. Le v primeru, da jim damo enako (malo) časa, ne presežejo algoritma *Eden naenkrat*.

Tabela 7.5: Primerjava algoritmov UA z Iskanjem v treh korakih.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število korakov**
Iskanje v treh korakih	10 488.34	35 811	-
UA osnovni	10 118.85	35 546	24
UA z različnimi učilnimi parametri	10 087.75	35 629	22
UA na vsaki poziciji	9 770.27	36 076	22
UA s spremenljivimi razdaljami	9 383.58	36 274	22
UA vse izboljšave	9 181.77	34 685	21

Tabela 7.6: Primerjava algoritmov UA s Povprečno piramido.

algoritem	**napaka MAE**	**porabljen čas [ns]**	**število korakov**
povprečna piramida	8 834.72	45 877	-
UA osnovni	9 367.64	46 258	32
UA različni učilni parametri	9 267.65	46 506	30
UA na vsaki poziciji	9 196.75	47 119	30
UA s spremenljivimi razdaljami	9 044.56	46 506	30
UA vse izboljšave	9 003.09	45 082	29

Iskanje v treh korakih je nekako v sredini tako po hitrosti kot po dobljenih rezultatih. V ta razred bi lahko uvrstili tudi algoritme na podlagi učečih avtomatov, s tem, da so dobljeni rezultati tukaj nekoliko boljši.

Med algoritme, ki porabijo največ časa in dajo zato najboljše rezultate, spada *Povprečna piramida*. V teh okoliščinah (čas procesiranja) se obnese bolje kot algoritmi na osnovi učečih avtomatov. Ti se lahko ujamejo v lokalni minimum in povečevanje števila korakov ni več smiselno.

Poglavje 8

Zaključek

Verjetno ni treba posebej poudarjati, kako pomembno vlogo imajo video CODEC-i v današnjem digitalnem svetu. Brez njih si ne bi mogli predstavljati digitalne televizije, gledanja filmov na mobilnikih, brez kompresije videa tudi DVD-ji ne bi obstajali.

Levji delež učinkovitosti kompresije sloni na odstranjevanju časovne redundance, ki temelji na ocenjevanju in kompenzaciji gibanja. Iz tega sledi, da je izbira algoritma za ocenjevanje gibanja ena izmed ključnih odločitev pri načrtovanju kodirnika. Zaradi kompleksnosti problema je to tudi eden izmed procesorsko bolj zahtevnih korakov pri kompresiji videa.

V prejšnjih poglavjih je bil predstavljen in ocenjen nov pristop k ocenjevanju gibanja. To je algoritem za ocenjevanje gibanja, ki je osnovan na učečih avtomatih. Ta nam ponuja nekoliko boljše razmerje med hitrostjo in kakovostjo rezultatov kot popularno *Iskanje v treh korakih*.

Sam algoritem se ponaša z zelo dobro lastnostjo, to je z možnostjo nadzora časa delovanja. Tako se lahko, če kakovost ali velikost kodiranega toka ne odgovarjata, proces kodiranja ponovi z več koraki pri ocenjevanju gibanja in nov video tok bo boljši.

Osnovni algoritem si z večino ostalih preprostih algoritmov za ocenjevanje gibanja deli slabo lastnost, to je možnost, da se ujame v lokalni minimum in v njem obtiči. Ta lastnost je lahko, vsaj delno, odpravljena z dodatnim učečim avtomatom, ki skrbi za različne razdalje korakov. Žal tudi ta izboljšava ni popolnoma imuna na ta problem, saj gre za premikanje le v horizontalni ali vertikalni smeri, torej niso dostopne vse lokacije.

Možna izboljšava bi bila morda v tem, da se algoritem v primeru kazni ne bi vračal na prejšnjo pozicijo, ampak bi izvedel vse akcije. S tem bi se prej ali slej izognili lokalnemu minimumu, vendar za ceno časa.

Stohastična narava algoritma UA namreč omogoča, da algoritem, ki izvaja vse akcije z določeno verjetnostjo preiskuje tudi dele iskalnega področja, ki jih nek deterministični algoritem ne bi. Na ta način bi bilo možno priti do boljših rešitev v primeru minimumov, ki jih je težko najti.

Druga, manj očitna in tudi manj moteča slaba lastnost algoritma, je potreba po generiranju naključnih števil. Verjetno v večini primerov ta lastnost ni moteč dejavnik, problem pa bi lahko nastal, če bi bilo algoritem potrebno implementirati v strojni opremi.

Seveda bi bilo za temeljito oceno algoritma potrebno primerjati še ostale dejavnike, kot na primer potrebo po pomnilniku med delovanjem, možnost uporabe vektorskih instrukcij itd. Kar se kakovosti dobljenih rezultatov tiče, bi bilo potrebno tudi primerjati podobnost sosednjih vektorjev. Poleg kodiranega ostanka podatkov se morajo namreč v dekodirnik dostaviti tudi gibalni vektorji. Ti so prav tako napovedani iz sosednjih makroblokov in je potrebno poslati le razliko. Iz tega sledi, da je lahko končen kodiran tok manjši pri algoritmih, pri katerih se vektorji med sosednjimi makrobloki spreminjajo počasi.

Najboljši test bi bil uporaba algoritma v praksi in hkratno primerjanje z ostalimi podobnimi algoritmi. Zelo težko si je vnaprej zamisliti, kakšne vrste video sekvenc bodo najpogostejše, in na podlagi tega modificirati algoritem.

Za konec je potrebno poudariti, da je algoritem na osnovi učečih avtomatov zelo dobra izbira algoritma za ocenjevanje gibanja. Sama implementacija je dokaj preprosta in nima velikih potreb po pomnilniku. Tudi dobljeni rezultati so v primerjavi z algoritmi, ki porabijo približno enako časa za izvajanje, večinoma boljši.

Literatura

[1] I. E. G. Richardson, "H.264 and MPEG-4 Video Compresion, Video Coding for Next-generation Multimedia", Wiley, 2003.

[2] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (TIU-T Rec. H.264; ISO/IEC 14496-10 AVC)", 2005.

[3] T. Wiegand, G. J. Sullivan, "Overview of the H.264/AVC Video Coding Standard", 2003.

[4] D. Turaga, M. Alkanhal, "Search Algorithms for Block-Matching in Motion Estimation", dostopno na: http:// www.ece.cmu.edu /ẽe899 /project /deepak_mid.htm, 1998.

[5] D. Liu, W. Sun, "Block-Based Fast Motion Estimation Algorithms in Video Compression", 1998.

[6] N.A. Khan, S. Masud, A. Ahmad, "A variable block size motion estimation algorithm for real-time H.264 video encoding", 2005.

[7] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, T. Ishiguro, "Motion-compensated interframe coding for video conferencing", v zborniku NTC'81 (IEEE), pogl 5.3.1 - 5.3.4.

[8] R. Srinivasan, K. R. Rao, "Predictive Coding Based on Efficient Motion Estimation, IEEE Transactions on Communications", št. 33, zv. 8, str. 888-896, 1985.

[9] J. R. Bergen, P. Anandan, K. J. Hanna, R. Hingorani, "Hierarchical Model-Based Motion Estimation", David Sarnoff Research Center.

[10] M. L. Tsetlin, "On the Behaviour of Finite Automata in Random Media, Automation and Remote Control 22", 1210–1219, 1962. Originalno v Avtomatika i Telemekhanika 22, 1345–1354, 1961.

[11] K. Narendra, M. Thathachar, "Learning Automata: An Introduction", Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[12] V. I. Varshavskii, I. P. Vorontsova, "On the Behaviour of Stochastic Automata with a Variable Structure, Automation and Remote Control 24", str. 327–333, 1963.

[13] R. R. Bush, F. Mosteller, "Stochastic Models for Learning", John Wiley & Sons, New York, 1958.

Izjava

Izjavljam, da sem magistrsko delo z naslovom "*Stohastični algoritem za bločno kompenzacijo gibanja pri kodiranju videa*" izdelal samostojno pod vodstvom mentorja dr. Branka Štera. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Simon Teran