UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

SISTEM ZA PRIMERJANJE VIDEO POSNETKOV IN ISKANJE DUPLIKATOV

ALEKSANDER ARUN BAHL

2019

ZAKLJUČNA NALOGA

BAHL

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Sistem za primerjanje video posnetkov in iskanje duplikatov

(A practical approach towards comparing videos and searching for duplicates)

Ime in priimek: Aleksander Arun Bahl
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj
Somentor: doc. dr. Jernej Vičič

Koper, julij 2019

Ključna dokumentacijska informacija

Ime in PRIIMEK: Aleksander Arun BAHL

Naslov zaključne naloge: Sistem za primerjanje video posnetkov in iskanje duplikatov

Kraj: Koper

Leto: 2019

Število listov: 60 Število slik: 27 Število tabel: 1

Število referenc: 22

Mentor: doc. dr. Peter Rogelj

Somentor: doc. dr. Jernej Vičič

Ključne besede: video, primerjava, sličice, križna korelacija, FFMPEG, Octave

Izvleček: V zaključni nalogi raziščemo problem primerjave video posnetkov. Osredotočimo se na iskanje primerjave med množico video posnetkov, za katere smo posumili, da vsebujejo duplikate in izrezke drugih video posnetkov. Skozi proces se uporabi več različnih pristopov za iskanje primerjav. Eno od njih se bo nato še praktično implementiralo v programskem okolju Octave. Osredotočili smo se na primerjavo z uporabo križne korelacije in poravnavo video posnetkov.

Key words documentation

Name and SURNAME: Aleksander Arun BAHL

Title of the final project paper: A practical approach towards comparing videos and searching for duplicates

Place: Koper

Year: 2019

Number of pages: 60 Number of figures: 27 Number of tables: 1

Number of references: 22

Mentor: Assist. Prof. Peter Rogelj, PhD

Comentor: Assist. Prof. Jernej Vičič, PhD

Keywords: Video, Comparison, Frames, Cross correlation, FFMPEG, Octave

Abstract:In the final thesis we explore the problem of comparing videos in a gathered set of videos, which are assumed to contain duplicates and contain related cuts of other videos. Trough the thesis ,we examine several methods to this approach and we choose to implement one. The chosen method will be implemented in the Octave enviroment. The focus will be on comparing videos using cross correlation and alligning the videos.

Zahvala

Zahvaljujem se doc. dr. Petru Roglju za mentorstvo in pomoč pri zaključni nalogi in somentorju, doc. dr. Jerneju Vičiču za dodatno pomoč. Zahvala gre tudi moji družini, ki me je podpirala med raziskovanjem in izdelavo zaključne naloge.

Kazalo vsebine

Kazalo tabel

Kazalo slik

Seznam kratic

FFMPEG	Fast Forward MPEG
MAD	Mean average difference
MSD	Mean squared difference
MAR	Maksimalna absolutna razlika
GNU	GNU's Not Unix
VBR	Variable bit rate
CBR	Constant bit rate

1 Uvod

Od prvega digitalnega zapisa video posnetka dalje količina večpredstavnostnih vsebin z leti narašča. Ena od zmožnosti digitalnih vsebin je kopiranje in transformacija brez omejitev s strani uporabnikov svetovnega spleta. Vsak uporabnik lahko hrani pri sebi svoje kopije, jih spreminja in objavi na spletu. Zaradi takšnih dejanj pride večkrat do podvojenih video posnetkov, pogosto v drugačnih oblikah, kot so podvzorčeni video posnetki, zakodirani v drugačnem formatu ali pa tudi delno vsebovani v drugih video posnetkih. Problem nastane, ko bi radi preverili obstoj duplikatov ali primerjali vsebovanost video posnetkov med seboj.

Povod za raziskavo problema je prišel iz domačega okolja. Na spletu je obstajal vir video posnetkov različnih konferenc in dogodkov. Za vse video posnetke, ki so bili shranjeni na disku, smo ugotovili, da je veliko datotek duplikatov, nekatere so kopije slabše kakovosti ali tudi skrajšane. Iskanje smo izvedli na preprostem primeru dveh video posnetkov, pri čemer en od posnetkov vsebuje del drugega. V predzadnjem poglavju se bomo osredotočili še na nekaj različnih primerov. Za testiranje smo uporabili video posnetek iz www.sample-videos.com [21], ki dovoljuje prosto uporabo video posnetka, kot je omenjeno na spletni strani.

V zaključni nalogi smo se tako posvetili primerjavi video posnetkov in njihove vsebovanosti. S pomočjo programskega okolja Octave in orodja MexopenCV, ki je vmesnik za OpenCV, smo analizirali video posnetke in jih primerjali s pomočjo procesiranja signalov. Cilj zaključne naloge je, da predstavimo eno metodo primerjave vsebovanosti video posnetkov med seboj. Vsa koda je na voljo na githubu [1].

1.1 Opis problema

Če imamo množico video posnetkov, v kateri so duplikati video posnetkov, podvzorčene ali drugače enkodirane kopije, različne vsebovanosti video posnetkov v drugih, je časovno zahtevno ročno preveriti vse video posnetke. V relacijah med video posnetki ne iščemo samo duplikatov, ampak tudi primere, ko je en video posnetek vsebovan v drugem. V začetku razvoja programske kode (julija 2018) ni bila najdena nobena zastonjska alternativa, ki bi zadoščala našim pogojem. Zato smo si zastavili cilj, da z

našim konceptom pokažemo, kako lahko to implementiramo. V obsegu te naloge se bo držalo enakih razmerij dimenzij video posnetkov, širine in višine sličic ter števila sličic na sekundo.

1.2 Kriterij delovanja

Od programskega produkta zaključne naloge želimo, da je zmožen prepoznati vsebovanost enega video posnetka v drugem. Vsak video posnetek je lahko deležen različnih transformacij, kot so podvzorčenje in različne resolucije video posnetka.

Prepoznati mora tudi vsebovanost video posnetka, če je njena dolžina v drugem video posnetku vsaj tretjina časa trajanja izvornega video posnetka, oziroma zahtevamo, da je program zmožen najti tudi delne vsebovanosti.

Razlog za takšno zahtevo izhaja iz resničnostne perspektive, saj v video posnetkih ne iščemo pretirano kratkih vsebovanosti. Vsebovanost enega video posnetka v drugem je lahko tudi celotna, kar pomeni da je celotni video posnetek vstavljen v drugega. Vhodni podatki so lahko tudi video posnetki, ki so v različnih formatih.

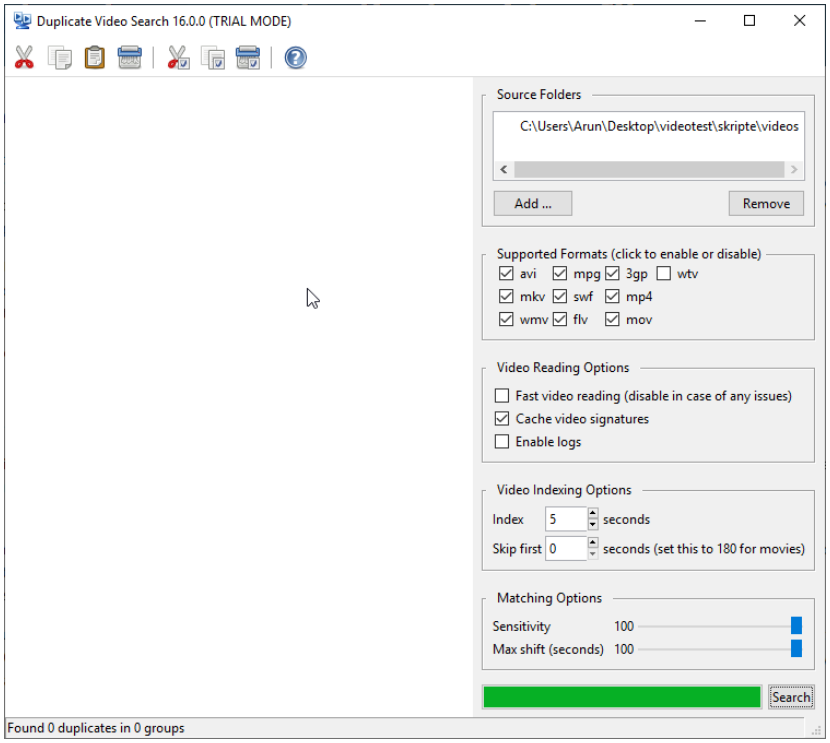
1.3 Pregled področja

V tem poglavju bomo pregledali nekaj obstoječih metod za primerjavo video posnetkov in iskanje duplikatov. Poskusilo se bo tudi analizirati naše obstoječe primere s pomočjo nekaterih teh programov. Nekateri so žal plačniški, zato tudi ponujajo brezplačno preizkusno uporabo z omejitvami.

1.3.1 Duplicatevideosearch

Duplicatevideosearch [3] za primerjavo video posnetkov uporablja *fingerprint-e* oziroma 'prstne' odtise. Ni znano, kako tej odtisi delujejo, saj je program zaprto koden, edina razlaga je objavljena na njihovi spletni strani in pravi, da se odtise uporablja za primerjavo video posnetkov po vsebini in se lahko tudi razlikuje med obrezanimi, različno velikimi v smislu resolucije in podvzorčenimi video posnetki.

Program je plačljiv in ima na voljo brezplačno testno različico za omejen čas. Po prvem zagonu iskanja duplikatov je očitno, da se proces primerjave zelo hitro izvaja. Tako sklepamo, da si program shrani nekakšne zapise vsakega video posnetka, ki jih lahko uporablja za primerjave. Žal program ni bil zmožen razpoznati delne vsebovanosti video posnetkov iz naše testne množice, uporabljene v naslednjih poglavjih, ali duplikatov. Razlog za nedelovanje ni poznan.



Slika 1: Poskus primerjanja vseh naših ustvarjenih video posnetkov z Duplicatevideosearch. V enem direktoriju so bili vsi video posnetki.

1.3.2 pHash

pHash je knjižnica, ki vsebuje nekaj razpršilnih funkcij, katere upoštevajo razlike med vhodnimi podatki. Razpršilna funkcija izračuna edinstven niz za nek vhodni podatek, ponavadi sta to datoteka ali besedilo. Navadne razpršilne funkcije, kot je MD5, so ustvarjene z namenom, da se najmanjša razlika med dvema podobnima datotekama pozna v veliki meri pri izhodnem nizu. pHash upošteva razlike v večpredstavnostnih datotekah, ki se jih lahko izkoristi. Z razpršilno funkcijo pHash se opazi podobnost v nizih razpršenih datotek. Funkcije, ki jih ima na voljo pHash, obdelujejo slike, ampak se to potem tudi lahko uporabi za primerjanje video posnetkov. Vsi argumenti so nastavljivi, saj je to le knjižnica [8].

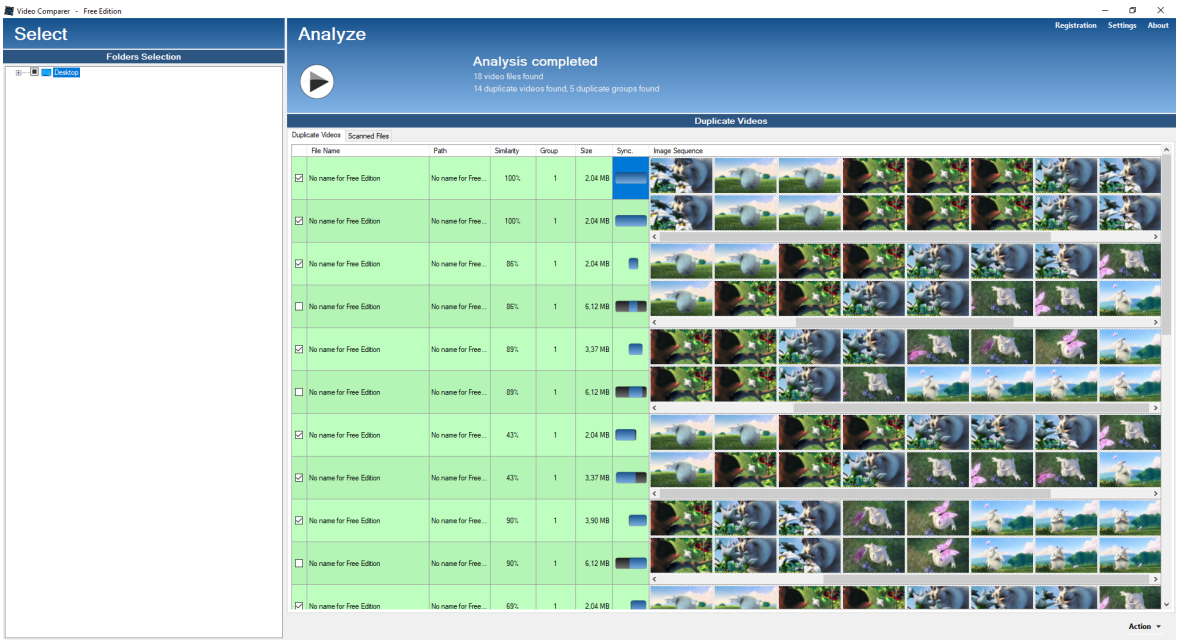
pHash-ova knjižnica vsebuje algoritem, kot je DCT Image Hash [7] oziroma Discrete cosinus transform Image hash, s katerim lahko zaznavamo podobnosti. Z njim lahko iz razpršitev slike dobimo niz in potem s pomočjo Hammingove razdalje primerjamo med izračunanimi nizi [7]. S tem algoritmom lahko primerjamo sličice, ki so bile pod vplivom različnih transformacij, kot so kompresija, rotacija ali zameglitev. Ni pa znano, če ima pHash zmožnost zaznave podobnosti med pari sličic, katera izmed njiju je bila podvzorčena.

Drugi algoritem, DCT Video Hash oziroma Discrete cosinus transform video hash,

izračuna razpršitveni niz za vsak *keyframe* in ga shrani v razpršitveno tabelo oziroma Hash Map za hitri dostop. Ko se primerja video posnetka, šteje ujemanja in če jih imata nad vnaprej določenim pragom, to zazna kot ujemanje. Iz tega sklepamo, da bi bil pHash zmožen zaznati testni par video posnetkov z več različnimi ujemanji, če bi ga uporabili v naši programski kodi.

1.3.3 Video Comparer

Video Comparer ima podobne funkcije kot Duplicatevideosearch. Zraven ponuja še iskanje video posnetkov z različno vsebovanostjo in različnimi zamiki med video posnetki [2].Tudi ta program je plačljiv in zaprto koden z verzijo za preizkušnjo, ki pa ima omejeno uporabo, saj imena video posnetkov v primerjavi niso prikazana. Program je bil zmožen najti nekaj video posnetkov z zamiki in tudi duplikate.



Slika 2: Poskus primerjanja vseh naših video posnetkov s programom Video Comparer, ki prikazuje vsa najdena ujemanja.

Rezultati programa Video Comparer prikazujejo, da je ta zmožen zaznati vsebovanosti s podvzorčenjem, kompresijo in duplikate. Zaradi različice, ki je na voljo za preizkušnjo, ni bilo mogoče ugotoviti, če zazna video posnetke z več ujemanji, saj skrije imena video posnetkov. Med preizkušnjo se je tudi izkazalo, da ne more zaznati ujemanja, če gre le za dva video posnetka v posamezni mapi. Zaznava je delovala, ko smo v analizo vključili celotno množico video posnetkov, ki smo jih uporabljali za testiranje v zaključni nalogi.

1.3.4 Primerjava zmožnosti programov

Duplicatevideosearch ni ustrezal pogojem, ki smo si jih zastavili. Video Comparer se je na drugi strani izkazal na vseh zastavljenih področjih po našem kriteriju delovanja. Za pHash pa sklepamo, da lahko z ustrezno implementacijo deluje za skoraj vse primere, saj v osnovi potrebujemo le nekakšno zaznavo podobnosti med sličicami. V dokumentaciji za pHash nismo našli navedbe, da lahko prepozna podobnost s podvzorčenimi sličicami.

Tabela 1: Tabela primerjav zmožnosti vseh omenjenih programov. Za vsak program smo testirali z testno množico videoposnetkov, ki smo uporabili skozi našo raziskavo.

	Duplicatevideosearch	Video Comparer	pHash
Zaznava duplikatov	✗	✓	✓
Iskanje ujemanj z vsebovanostjo	✗	✓	✓
Iskanje ujemanj z podvzorčenjem	✗	✓	✗
Iskanje ujemanj z kompresijo	✗	✓	✓
Iskanje več ujemanj	✗	✗	✓

1.3.5 Zaključek pregleda

Najprimernejši program za reševanje našega problema je Video Comparer, ampak zaradi preizkusne različice ga ni moč uporabljati brez omejitev. Zadnjega kriterija, to je iskanje več ujemanj, nismo navedli v kriterijih delovanja, saj gre za robni primer v naši testni množici video posnetkov.

Ker je Video Comparer zaprto kodna rešitev in druga orodja ne rešujejo naših zastavljenih nalog v celoti, je smiselno ustvariti nov programski produkt, s katerim želimo poskusiti rešiti te probleme.

1.4 Zgradba video posnetkov

V osnovi je video posnetek sestavljen iz več zaporednih slik, ki jih prikazujemo in predstavimo kot premikajočo se sliko.

Intervali v video posnetkih se označujejo s sličicami na sekundo oziroma s hitrostjo sličic (FPS – frames per second). Velikokrat so hitrosti sličic v razponu od 24 pa tudi do 60 na sekundo, kar je odvisno od namena video posnetka. Filmi so ponavadi hitrosti okoli 24 sličic na sekundo, videi na portalu Youtube pa tudi do 60 sličic na sekundo. Vsak video posnetek ima različne pripadajoče lastnosti. Poleg FPS poznamo tudi:

- ločljivost vsake sličice
- hitrost bitov na sekundo
- kodiranje video posnetka.

Resolucija video posnetka je posledično tudi resolucija vsake sličice v njem. Določa jo število pikslov, ki ga dobimo z množenjem širine in višine. V primeru barvnega videa imamo lahko RGB barvni prostor (pomeni Red, Green, Blue). Z mešanjem teh treh virov svetlob lahko dobimo katerokoli barvo. Vsaka je predstavljena z vrednostmi med 0 in 255, kjer je 255 predstavlja najnižja vrednost najvišjoa intenzitetoa barve in pri 0 barvae ni vmešana. Vsaka barva je določena z osmimi biti. Klasična barvna shema, ki jo imamo v današnjih video posnetkih, je 24 bitna "True color".

Število bitov na sekundo je lastnost videa, ki opisuje količino pretoka informacij v posnetku in določa tudi kakovost video posnetka. Pri CBR kodiranju (Constant Bit Rate) je z večanjem bitov na sekundo višja tudi kakovost video posnetka. To ne velja pri VBR kodiranju, ki se za razliko od konstantne bitne hitrosti spreminja skozi predvajanje večpredstavnostne vsebine. Variabilna bitna hitrost poskuša izboljšati razmerje med velikostjo video posnetka in njegovo kakovostjo, tako da uporablja manj bitov za zapis, kjer ni veliko sprememb med zaporednimi sličicami. VBR se največkrat uporablja za prenos video slike prek spleta. Obstaja tudi nekaj kodirnikov, ki kodirajo video posnetke z višjo kakovostjo z manj biti na sekundo, recimo MPEG.

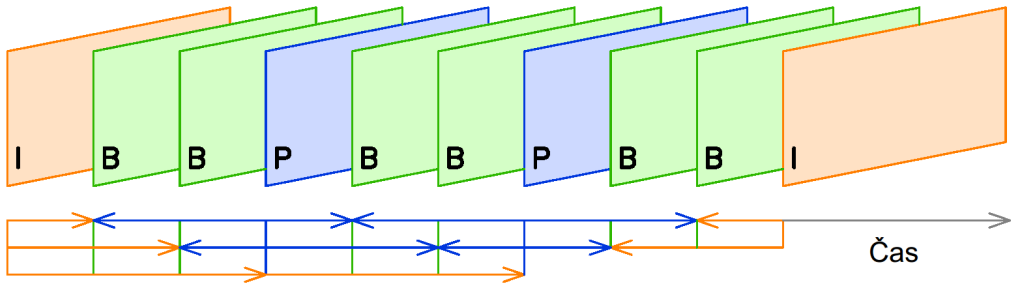
Video datoteke so shranjene oziroma enkodirane v različnih formatih. Najbolj pogosti so:

- MPEG-4 (Moving Picture Experts Group Phase 4),
- AVI (Audio Video Interleave),
- MKV (Matroska Multimedia Container),
- FLV (Flash Video).

MPEG je metoda, ki določa način kompresije video in avdio podatkov. Najbolj znana MPEG-4 se dandanes uporablja skoraj povsod.

V video kompresiji se uporablja intra-okenski kompresijski pretok, ki močno zmanjša velikost video posnetkov. Okvirji video posnetka so razvrščeni po skupinah. V vsaki imamo tri različne okvirje. To so I-okvirji, P-okvirji in B-okvirji. Prvi so referenčni okvirji, ki jih kompresijski algoritem analizira in tako poskuša najti ujemanja v naslednjih okvirjih. I-okvir vsebuje celotno začetno sliko. Kodek nato prevrti nekaj okvirjev naprej in ustvari P-okvir, ki vsebuje razliko od I-okvirja in manj informacij. B-okvirji so vmesni okvirji, ki vsebujejo predvidene razlike med I- in P-okvirji.

Ko se enkodira video, se najprej ustvari I-okvir, nato P-okvir in glede na razlike se nato ustvari še vmesne B-okvirje. Število I-, B- in P-oken je enako številu sličic v video posnetku [5].



Slika 3: I-, P- in B-okvirji video posnetka v kronološkem vrstnem redu, pod sliko so prikazana navezovanja okvirjev med seboj.

2 Metodologija

Odločili smo se za razvoj lastnega postopka, ki deluje drugače od opisanih v prejšnjem poglavju. Za iskanje korelacije med signali, ki predstavljajo podrobnosti posameznega video posnetka, je izhodiščna ideja uporaba križne korelacije. Videoposnetek se bo razdelil na sličice in tako se bo računalo povprečno razliko med posameznimi sličicami, ki razkriva, v kakšnem odnosu so sličice. Na ta način bomo dobili vektor sprememb. Če je razlika majhna, pomeni, da sta si sosednji sličici nekako podobni, velika vrednost pa nakazuje na to, da je razlika velika. Ta tehnika nam dovoli, da bo postopek relativno hitro in tudi zadovoljivo zaznal spremembe med posameznimi sličicami, kljub transformacijam, kot so podvzorčenje in različna enkodiranja, ki jih je video posnetek lahko deležen.

Iz vektorjev sprememb obeh primerjanih video posnetkov se izračuna vektor korelacije med njima. Visoke vrednosti pomenijo možno ujemanje med video posnetki za dano korelacijsko vrednost. Iz vektorja korelacije lahko razberemo enega ali več možnih kandidatov za (delno) ujemanje video posnetkov. V primeru iskanja duplikatov je problem trivialno rešljiv, ampak vseeno spada v domeno našega problema, ker se lahko pojavi.

Proces analize se bo odvijal po naslednjem vrstnem redu:

- Iz video posnetkov se izlušči posamezne sličice, pretvori v sivinske barve in nato podvzorči.
- Za vsak nabor sličic se izračuna vektor sprememb.
- Med dvema vektorjema sprememb se izračuna vektor korelacije s postopkom normirane križne korelacije
- Iz vektorja korelacije se pridobi kandidate za možne zamike.
- Glede na dobljene kandidate se izračuna možen zamik tau - τ, in tudi indekse obsega ujemanja.
- Med seboj se primerja posamezne sličice obeh video posnetkov v obsegu s prej najdenim zamikom in indeksi. V tem koraku dobimo rezultat ujemanja.

Slika 4: Celotni postopek procesa iskanja ujemanja med dvema video posnetkoma, na koncu dobimo seznam obsegov ujemanj.

2.1 Uporabljene metode

Za primerjave video posnetkov je bil uporabljen statistični pristop. Odločali smo se večinoma na povprečnih razlikah med sličicami.

2.1.1 Povprečna absolutna razlika

MAD oziroma *Mean absolute difference* je mera za izračun razlike med dvema spremenljivkama. Če si predstavljamo dve sličici, predstavljeni kot matriki, je povprečna razlika med njima absolutna razlika med enakoležnimi elementi, katere se potem sešteje skupaj in povpreči s številom elementov v matriki, da dobimo eno vrednost.

Enačbo lahko zapišemo na naslednji način; **A** je video posnetek, kjer i predstavlja i-to sličico video posnetka v obliki matrike, x in y ponazarjata indekse elementov v matriki, n pa je število elementov v eni od matrik. Enačba naredi absolutno razliko med i-to in naslednjo matriko ter jo povpreči.

$$\frac{1}{n}\sum_{x,y=1}^{n}|A_i(x,y) - A_{i+1}(x,y)| \tag{2.1}$$

2.1.2 Povprečna kvadratna razlika

Povprečna kvadratna razlika pomeni podobno kot povprečna razlika, ampak ta mera bolj poudari večje razlike, hkrati pa je manj občutljiva na šum, zato pri vektorju sprememb dobimo veliko večje vrednosti pri razlikah. Povprečna kvadratna razlika bi prav prišla, če bi iskali velike razlike, ampak v primeru prekomernega šuma nam to ne pomaga, zato smo izbrali navadno povprečno razliko.

Povprečna kvadratna razlika izračuna razliko med vsemi enakoležnimi elementi v matrikah koordinat x in y. Vsako izračunano razliko kvadrira in nato sešteje. Vsoto nato povpreči s številom elementov v matriki. Oznaka i je indeks sličice v video posnetku, ki je predstavljena kot matrika **A**.

$$\frac{1}{n}\sum_{x,y=1}^{n}(A_i(x,y) - A_{i+1}(x,y))^2 \tag{2.2}$$

Povprečna kvadratna razlika

2.1.3 Maksimalna absolutna razlika

Maksimalna absolutna razlika, oziroma MAR, vrne največjo razliko med dvema matrikama. Metoda vzame dve matriki, ju med seboj odšteje po enakoležnih elementih in vse elemente pretvori v absolutno vrednost ter vzame maksimum. To metodo bomo uporabili v testiranju, saj smo z metodo MAD dosegli dobre rezultate in se je MAR uvedlo šele proti koncu testiranja. A in B sta dve različni matriki z enakimi dimenzijami.

$$max(|A - B|) \tag{2.3}$$

Povprečna kvadratna razlika

2.1.4 Normirana križna korelacija

Križna korelacija je statistična mera za iskanje vzorcev in podobnosti. Pogosto je uporabljena za primerjanje podatkov, v našem primeru so to 1D signali. Uporabljena je tudi v računalniškem vidu [15] [10]. Cilj je poiskati podobnost med dvema slikama.

Korelacijska funkcija primerja vektorje sprememb video posnetkov, tako da jih primerja za vsak možen zamik. Najmanjše prekrivanje med vektorji je lahko za 1 mesto, oziroma v obliki video posnetka je to za 1 sličico. Kot rezultat vrne metoda vektor korelacije. Ker uporabljamo normirano križno korelacijo, nam vrne vrednosti med 1 in -1, kjer vrednost 1 pomeni, da se vektorja popolnoma korelirata v nekem določenem prekrivanju [18].

Funkcija izračuna eno vrednost za kandidata za zamik iz vrednosti korelacije med dvema vektorjema sprememb za eno prekrivanje; **a** in **b** sta vektorja, i je indeks v vektorjih. Celotni zgornji produkt se deli s korenjenim produktom variance obeh vektorjev za eno prekrivanje.

$$c(\tau) = \frac{(a_i - \bar{a}) * (b_i + \tau - \bar{b})}{\sqrt{\sigma_a * \sigma_b}} \tag{2.4}$$

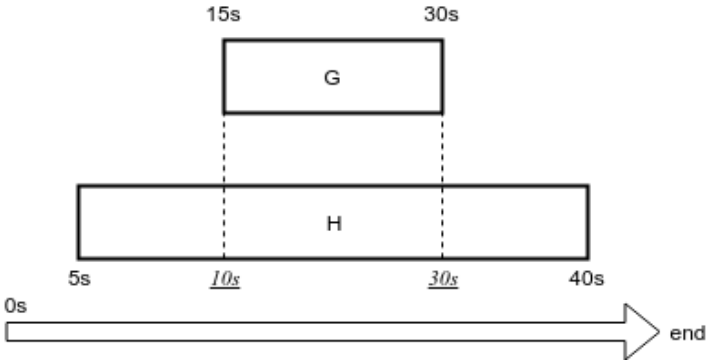
Normirana križna korelacija

2.2 Opis postopka analize

Za postopek analize smo uporabili preprost primer z vsebovanostjo enega video posnetka v drugem. V tem postopku se video posnetka ni podvzorčilo ali kompresiralo. Na začetku procesa smo ustvarili podatkovno strukturo Posnetek, v katero smo shranili vse podatke oziroma parametre video posnetka, ki so potrebni skozi proces. Ključni podatki v strukturi so podvzorčene slike, vektor sprememb in ime video posnetka. Trenutni postopek je namenjen ugotavljanju relacije med dvema video posnetkoma. V naslednjih podpoglavjih si bomo ogledali korake procesa iskanja podobnosti med njima.

2.2.1 Opis primera

Za primer smo iz izvirnega video posnetka, s strani sample-videos.com [21], izrezali dva segmenta: en krajši segment G in daljši segment H, ki je umeščen med 5. in 40. sekundo izvirnega video posnetka. Posnetek G pa je v izvornem video posnetku umeščen med 15. in 30. sekundo. Podčrtane sekunde sta začetek in konec ujemanja video posnetka G. Izvirni video posnetek je dimenzije 1280x720, ima hitrost 25 sličic na sekundo in je dolg 2 minuti in 50 sekund.



Slika 5: Relacija med posnetkoma G in H. Slika prikazuje, kako sta časovno usklajena, ker sta enakega izvora. Prikazuje tudi zamik v primeru, ko najdemo ustrezne vrednosti zanj.

Videoposnetek G je dolg 375 sličic, video H pa 625 sličic. Ker se je podatke pripravilo vnaprej, je že takoj znano, da se video posnetka ujemata, če primerjamo začetek video posnetka G in 250. sličico video posnetka H.

2.2.2 Predprocesiranje

Prvi korak pri primerjavi video posnetkov je predobdelava. Primerjavo smo izvedli v okolju Octave z uporabo knjižnice OpenCV prek vmesnika MexOpenCV. Cilj predprocesiranja je izračun 1D vektorja, ki poskuša dobiti specifične lastnosti video posnetka. V našem primeru je ta lastnost spreminjanje video posnetka skozi čas. Kot vhodni podatek podamo pot do video posnetka, ki ga prek MexOpenCV vmesnika preberemo za vsako sličico. Videoposnetek shranimo kot seznam matrik. Vsaka sličica je shranjena v črno beli barvi. Z odstranitvijo barv se zmanjša tudi velikost sličic, kar pripomore k hitrosti primerjanja. Sličice so shranjene kot matrike, katerih elementi so v razponu med 0 in 255, kjer je 0 črna barva, 255 pa bela.

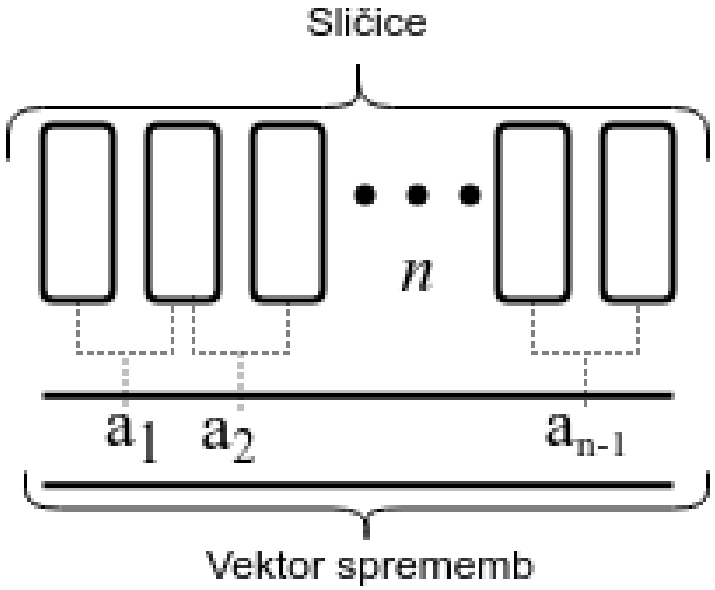
Sličice iz video posnetka se podvzorči na 54 pikslov višine in 96 pikslov širine. To je približno 20-kratno zmanjšanje iz velikosti 1920x1080. Prvotno razmerje med dolžino in širino slike ni pomembno, saj se vse podvzorči na enako velikost. Če bi bila slika v 4:3 razmerju, se jo raztegne na razmerje 16:9. Transformacija se zgodi, ko se podvzorči in spremeni velikost sličice. Ker na koncu samo primerjamo sličice med seboj in iščemo razlike, takšna transformacija ne vpliva na celoten proces.

Vektor sprememb se računa z MAD funkcijo. V vsaki iteraciji se premakne za 1 indeks in primerja trenutno sličico z naslednjo. Rezultat je vektor sprememb, ki je za eno sličico krajši, saj za vsak par sličic izračuna eno vrednost v vektorju sprememb.

Kot rezultat vrne funkcija za predprocesiranje strukturo Posnetek, z vektorjem sprememb in podvzorčenimi sličicami. Proces primerjanja z MAD funkcijo je prikazano

na Sliki 6.

Za video posnetek s številom n sličic, se bo ustvaril vektor sprememb a z $n-1$ vrednosti.



Slika 6: Računanje vektorja sprememb. Vzame se vrednost MAD dveh zaporednih sličic ali katero drugo metodo, ki je primerna za izračun vektorja sprememb. Nato se rezultat vpiše v vektor a.

V vektorju sprememb imamo na x osi indeks spremembe v video posnetku. Visoka vrednost pomeni, da obstaja velika razlika med dvema sličicama. Takšne vrednosti nam veliko pomenijo, saj jih nato križna korelacija v primeru lažje zajame, ker so na voljo izstopajoče vrednosti.

Slika 7: Prikaz vrednosti MAD obeh video posnetkov. Visoke vrednosti nakazujejo velike spremembe med dvema sosednjima sličicama. Ob takšnih spremembah lahko sklepamo, da prihaja do prehodov v video posnetku.

Izračuna se tudi MD5 razpršitev, ki se jo pozneje primerja v funkciji za iskanje kandidatov, a le v primeru, da sta video posnetka popolnoma enaka. Tako se izognemo nepotrebnemu primerjanju dveh video posnetkov. Spodaj sta izračunana niza MD5 razpršitev obeh video posnetkov, katerih razpršitvena niza sta drugačna.

- MD5 razpršitev video posnetka G = 2a1f28800d49717bbf88dc2c704f4390
- MD5 razpršitev video posnetka H = 0f41ba7e5c8e7770c82c0b7a51cef936

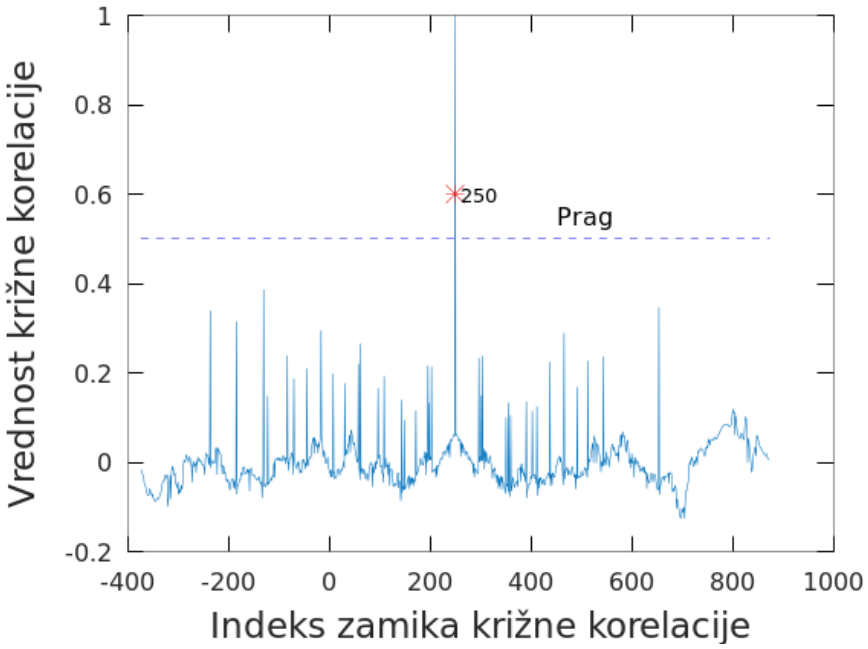
2.2.3 Iskanje kandidatov za zamik iz vektorjev sprememb

V funkciji za iskanje kandidatov se prejme dva video posnetka ter kakšen prag naj bo za zaznavo maksimumov kot argumente. Prag bo določil, s kakšno vrednostjo se lahko korelirata posnetka. V tem delu algoritma se uporablja normirano križno korelacijo, zato bo vektor korelacije imel vrednosti v intervalu od 1 do -1.

Če je i dolžina prvega video posnetka in j dolžina drugega video posnetka, je dolžina vektorja korelacije $i + j - 1$. Če je dobljen kandidat zamika enake vrednosti kot dolžine krajšega video posnetka, je zamik enak 0.

Ko govorimo o prvem in drugem video posnetku, je to povezano s praktično implementacijo funkcije za normirano križno korelacijo, ki bo obrazložena pozneje. V naslednjih poglavjih velja, da je prvi video posnetek v številu sličic krajši ali enak od

drugega. Vektor korelacije dobimo z začetnim indeksom 1, ampak za primernejšo predstavo zamika, se pomakne vektor korelacije za dolžino prvega video posnetka v levo, da lahko z grafa takoj razberemo vrednost zamika.



Slika 8: Prikaz izračunane normirane križne korelacije našega primera. Visoka vrednost pomeni visoko ujemanje na določeni vrednosti korelacije med video posnetki.

S pragom 0.5 je funkcija iskanje_kandidatov.m vrnila vrednost 624, kar se pretvori v 250, 624-374. V primeru več korelacijskih ujemanj bi vrnila več kandidatov. Rdeči križec predstavlja indeks zamika križne korelacije, kjer je presegla naš prag.

2.2.4 Iskanje možnih zamikov

V iskanju zamikov iščemo zamike, pridobljene iz rezultatov križne korelacije. Zamik se dobi tako, da se odšteje dolžina prvega vektorja sprememb od dobljenega indeksa maksimuma iz vektorja korelacije. Temu pravimo *tau*, simbol τ. Zadnji del funkcije še preveri, v kakšnem obsegu se ujema video posnetek z drugim z danim zamikom.

τ = *korelacijska vrednost – dolžina prvega video posnetka*

S pridobljenim zamikom se lahko preveri, če se video posnetka res ujemata. To storimo tako, da med seboj primerjamo sličice v obeh video posnetkih. Funkciji za iskanje ujemanj se poda dve Posnetek strukturi, seznam zamikov in po želji tudi prag. Ta določa, za koliko se lahko sličice med seboj razlikujejo, da se še uveljavi ujema-

nje. Določiti moramo še indekse, ki se jih bo upoštevalo pri primerjavi sličic. Indeks primerjave se računa po naslednjih enačbah:

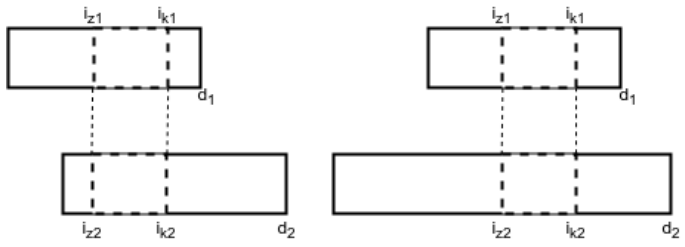
- $i_{z1} = Max(0, -1 * \tau) + 1$ oziroma začetek primerjave prvega video posnetka se izračuna z maksimumom med 0 in $-1 * \tau$. *tau* množimo z -1, ker če je negativen, pomeni, da je treba prvi video posnetek začeti primerjati na drugem indeksu. *tau* je lahko tudi ničeln, zato moramo prišteti 1, saj se indeksiranje v Octave začne pri 1.

- $i_{k1} = Min(d_1, d_2 - \tau)$ oziroma konec primerjave prvega video posnetka se izračuna tako, da se izbira minimum vrednost med dolžino prvega video posnetka d_1 ali dolžino drugega video posnetka d_2 in od tega odštejemo *tau*. Drugi argument v *min* funkciji bo izbran v primeru, če se prednji del prvega video posnetka pojavlja na koncu drugega video posnetka.

- $i_{z2} = Max(\tau, 0) + 1$ oziroma začetek primerjave drugega video posnetka se začne pri začetku posnetka ali pa zamakne za *tau*. Izbere se maksimum, ker v primeru, da je *tau* pozitiven, pomeni, da je treba začeti primerjati drugi video posnetek na drugačnem indeksu. Ker je lahko *tau* ničeln, kar pomeni, da ni zamika, se prišteje 1, saj se v Octave indeksiranje začne z 1.

- $i_{k2} = Min(\tau + d_1, d_2)$ oziroma konec primerjave drugega video posnetka se konča pri minimumu med dolžino drugega video posnetka ali dolžino prvega video posnetka + *tau*.

Indekse grafično predstavimo na naslednji način. Desni del slike prikazuje zamik prvega video posnetka v desno. V tem primeru je *tau* pozitiven. V spodnjem primeru je levi video posnetek zamaknjen v levo, kar pomeni, da je bil *tau* negativen. Če je *tau* enak 0, pomeni, da zamika ni in se začne primerjava video posnetkov brez zamikov; ni poravnave video posnetkov.



Slika 9: Prikaz dveh različnih indeksov primerjanj. V levem primeru je zamik negativen, v desnem pa pozitiven. Zamik je lahko tudi vrednosti 0, kar pomeni, da se ujemata z enakim začetkom video posnetkov.

Za trenutni testni primer video posnetkov G in H so bili izračunani naslednji indeksi:

- $\tau = 250$
- zacetek_prvi (i_{z1}) = 1
- konec_prvi (i_{k1}) = 374
- zacetek_drugi (i_{z2}) = 251
- konec_drugi (i_{k2}) = 624

Funkcija je izračunala zamik za 250 mest. To pomeni, da se video posnetek G 'zamakne' za 250, da pride do ujemanja med video posnetkoma. Prvega se bo začelo primerjati pri prvi sličici in končalo na zadnji sličici. Pri drugem pa se bo začelo primerjati pri 251. in končalo na 624. sličici, saj so indeksi potem prišli do konca dolžine prvega video posnetka.

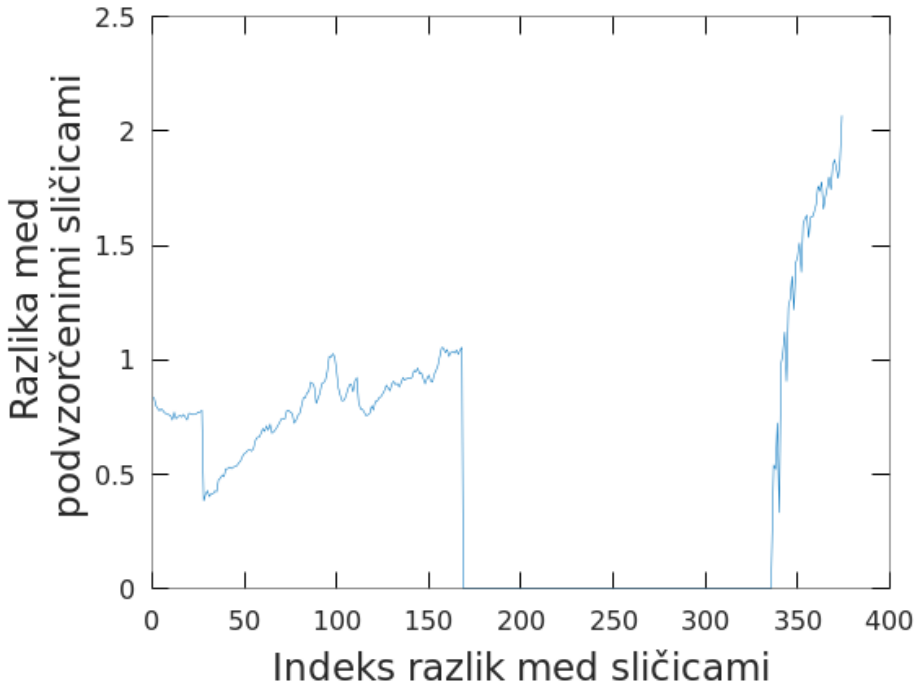
2.2.5 Iskanje intervala ujemanja

Drugi del funkcije iskanje_ujemanj za dani zamik preveri, če se video posnetka res ujemata, in izračuna tudi, na katerih intervalih se ujemata v prekrivanju. *For zanka* uporabi štiri indekse, ki so bili predhodno izračunani, in med njimi primerja podvzorčene sličice obeh video posnetkov. Kot rezultat dobimo vektor ujemanj. Za določitev ujemanja se uporabi prag, pod katerim je lahko vrednost v vektorju, da se to upošteva kot resnično ujemanje. V primeru, da se celotni del ujema, bo funkcija vrnila samo en obseg. V primeru, da je ujemanj več oziroma ujemajočih več delov, bo funkcija vrnila več obsegov. To se lahko zgodi, če pod pragom ni celotnega vektorja ujemanja.

V našem vektorju ujemanj smo uporabili prag 5, ker sta video posnetka, na ravni sličic, skoraj enaka.

- $[1 - 374]$

Vse, kar je pod pragom, se razume kot ujemanje. Na začetku in na koncu vidimo nekakšne razlike, za katere lahko sklepamo, da so nastale pri ustvarjanju testnih podatkov iz izvirnega video posnetka.



Slika 10: Prikaz izračunanih razlik za τ=250. Za prag smo uporabili 5, zato nam vrne kot obseg ujemanj celotno dolžino. Na sliki so vidni naraščaji in padci v samih razlikah, ampak to dojemamo kot razlike pri enkodiranju video posnetkov.

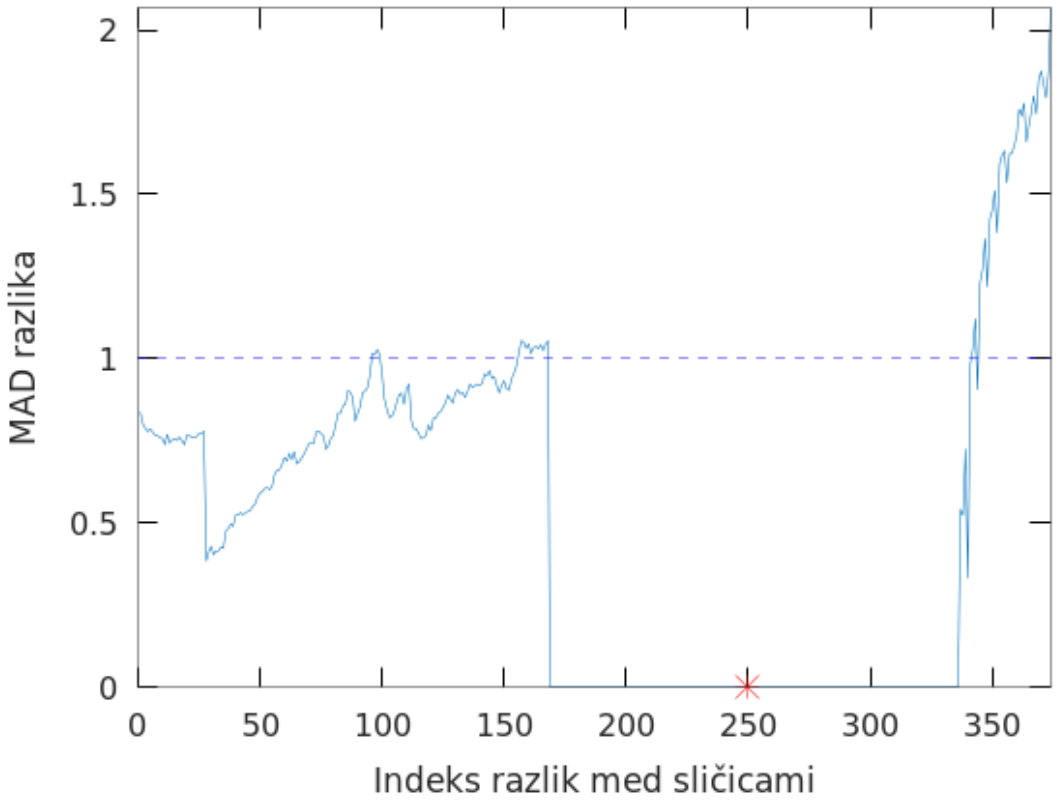
2.2.6 Prikaz rezultatov

Funkcija je podobna prejšnji. Na začetku se izračuna enake indekse, ki se uporabijo za iskanje dolžine ujemanj. Za ujemanje se uporablja enak prag in funkcija MAD, saj spet primerjamo podvzorčene sličice.

Za prikazovanje se ujemajoči sličici 'sešije' skupaj, spodaj pa se pokaže vektor primerjave med množicami podvzorčenih video posnetkov. Vse skupaj se animira in na koncu ustvarjena video animacija prikazuje video posnetka skupaj, ko se ujemata. Črta ponazarja zastavljen prag, rdeča pika pa mesto trenutnega prikazanega ujemanja video posnetkov.

(a) Sličica prvega video posnetka

(b) Sličica drugega video posnetka



(c) Obseg ujemanj

Slika 11: Prikaz ujemanj video posnetkov

3 Praktična implementacija

V poglavju praktične implementacije bomo predstavili programsko orodje, ki je bilo uporabljeno. Razjasnili bomo tudi ključne dele programske kode. Celotna zaključna naloga je bila ustvarjena z naborom odprtokodnih aplikacij in orodij. V uporabi so bili Octave, OpenCV in FFMPEG, ki so nameščeni na operacijskem sistemu XUbuntu Linux verzije 16.04 Xenial Xerus.

3.1 Programska oprema

Zaradi preproste konfiguracije in podpore orodij smo izbrali operacijski sistem Linux. Octave je primeren program, saj ima vgrajena vsa potrebna orodja ter razširitve za obdelavo slik in numerične obdelave podatkov. V naslednjih podpoglavjih si bomo podrobneje ogledali uporabljena programska orodja.

3.1.1 Octave

Octave [14] je odprtokodno programsko orodje, namenjen izvajanju numeričnih operacij. Je tudi alternativa programskemu orodju Matlab [19]. Funkcionalnost lahko razširimo z moduli, ki so spisani v Octave-ovem jeziku, C++, C ali Fortran. Celotno programsko orodje je izdelano v C++ jeziku. Octave se lahko uporablja z grafičnim vmesnikom dodatno ali pa v Linux lupini [9].

Programski jezik v Octave je visokonivojski in njegova sintaksa je podobna kot v Matlab-u. Eden od ciljev razvoja okolja Octave je tudi kompatibilnost z okoljem Matlab. Jezik je podoben C-ju in vsebuje tudi nekaj funkcij iz standardnih knjižnic C-ja in nekatere sistemske klice UNIX-a [13]. Octave-ov jezik podpira tudi različne podatkovne strukture in objektno orientirano programiranje. Nekaj primerov uporabe okolja Octave v njegovi vgrajeni ukazni vrstici.

Vgrajene konstante:

```
>> pi
ans = 3.1416
>> e
ans = 2.7183
```

Operacije z matrikami:

```
>> eye(2)
ans = Diagonal Matrix
1   0
0   1
#zdruzevanje matrik
A = ones (2, 2);
B = zeros (2, 2);
cat (2, A, B)
ans =
 1 1 0 0
 1 1 0 0
```

Nekatere funkcionalnosti iz standardne C knjižnice vgrajene v Octave [13]:

- **fopen()/fclose()**, odpiranje in zapiranje datotek;
- **fprintf/sprintf**, formatiran izhod rezultatov;
- pretvorbe v število/iz števila s plavajočo vejico in celih števil.

Primeri sistemskih klicev, ki jih omogoča Octave [13]:

- **system()** – klicanje zunanjih funkcij iz okolja Octave
- **unix()** – klicanje funkcij v primeru uporabe UNIX operacijskega sistema
- **fork()** – Stvaritev kopije trenutnega procesa
- **exec()** – Zamenjava trenutnega procesa z drugim
- definiranje funkcij v ukazni vrstici, če želimo hitro preizkusiti kakšno funkcijo;
- komentiranje vrstic kode z znakom #, kar je tudi komentar v UNIX skriptah, priročno pri pisanju Octave skript;
- povečevalni/zmanjševalni operatorji kot v C jeziku, ++,–,+=,-=, s tem posledično manj kompleksnosti kode.

Programsko orodje Octave je tudi prosto uporabno pod licenco GNU General Public License (GPL) [14].

Kratek povzetek zgodovine

Octave je bil ustvarjen leta 1988 za pomoč pri delanju kemijskih reakcij v kontekstu učbenika za dodiplomski študij. Leta 1992 se je začel nadaljnji razvoj programske opreme, ki je bil v celoti posvečen temu, da postane Octave pomoč pri znanstvenem računanju. Ime je dobil po profesorju Octave-u Levenspielu, ki je napisal knjigo o kemijskih reaktorjih in njegovih problemih. John W. Eaton, začetni stvaritelj programskega okolja, pa je bil njegov učenec. [17]

Uporabljene razširitve

Octave podpira tudi razne razširitve, ki so ponujene prek njihove izvorne spletne strani, *octave forge* [11]:

- video,
- image,
- signal.

3.1.2 FFMPEG

FFMPEG[1] je programsko orodje, sestavljeno iz mnogih knjižnic in razširitev, ki omogoča podrobno obdelavo video posnetkov, zvoka in drugih multimedijskih datotek. Poznan je po velikem naboru funkcij, ki so zelo prilagodljive. S FFMPEG se lahko reže, enkodira, dekodira, filtrira več različnih video formatov. Program se uporablja izključno v ukazni vrstici in ga lahko drugi programi uporabljajo skozi njegov vmesnik. Programsko orodje je izdano pod GNU Lesser General Public License 2.1+ ali GNU General Public License 2+, odvisno od opcij, ki so izbrane pri uporabi [6].

FFMPEG smo v zaključni nalogi uporabili kot orodje za pripravo testnih podatkov. Tako smo se odločili zaradi začetnih problemov s trenutno implementacijo video razširitve iz Octave Forge. Razširitev je brala samo ključne sličice (key frames) iz video posnetkov, problema pa ni bilo moč najti [12]. FFMPEG smo uporabili tudi za pripravo testnih podatkov, zapisovanje video posnetkov v posamezne sličice pa smo rešili naknadno z MexOpenCV vmesnikom. FFMPEG se bo uporabljal znotraj okolja Octave, s sistemskimi klici oziroma klicem *system* [9].

[1]FFMPEG `https://www.ffmpeg.org/`

3.1.3 OpenCV

OpenCV oziroma *Open source computer vision library* je knjižnica, ki je bila optimizirana za obdelavo video in slikovnih digitalnih vsebin. Podpira vmesnike za več programskih jezikov kot so C++, Java in Python. Podprta je na vseh treh glavnih operacijskih sistemih: Linux, Windows in MacOS. OpenCV je ustvarjen z C/C++, zato vzame v poštev tudi več jedrno zmožnost današnjih računalnikov. [4].

3.1.4 Mexopencv

Mexopencv je vmesnik za uporabo knjižnice OpenCV v okoljih Matlab in Octave. Sposoben je pretvarjati med podatkovnimi tipi OpenCV in Matlab, enako velja za Octave. Mexopencv je odprtokoden projekt, ki je na voljo tudi na github-u [16].

3.2 Priprava okolja

Uporabili smo APT repozitorij [20], iz katerega smo namestili najnovejši Octave, ki je na voljo kot paket za Xubuntu 16.04, tako smo uporabili Octave verzije 4.2.2.

Za uporabo OpenCV v programskem okolju Octave smo uporabili Mexopencv. To je zbirka OpenCV knjižnic, ki so prirejene preko API-jev za uporabo v Matlab ali Octave okolju [22]. Za uporabo Mexopencv smo morali namestiti OpenCV verzije 3.4.1. Ob končani namestitvi smo nato namestili še Mexopencv verzije 3.4.1 in uredili konfiguracijo, da je Octave zaznal inštalacijo Mexopencv. Za namestitev Mexopencv na sistemu Linux smo upoštevali navodila na githubu od Mexopencv [22].

3.3 Duplikati

Za iskanje duplikatov smo uporabili zgoščevalno funkcijo MD5, ki je vgrajena v Octave. Ta ustvari edinstven niz, ki je drugačen za vsak vhodni podatek. Niz se spremeni že za najmanjšo spremembo, zato ga uporabljamo za primerjanje duplikatov.

V predprocesiranju ustvarimo niz iz razpršitev video posnetka z MD5 funkcijo in to nato primerjamo v funkciji za iskanje kandidatov.

3.4 Skripte za pripravo podatkov

Za pripravo video posnetkov se je uporabilo skripte za lepljenje in rezanje video posnetkov, ki kličejo FFMPEG. Za lažjo uporabo se je klice za FFMPEG vneslo v funkcije za uporabo programov zunaj okolja Octave. Za funkcijo *rezanje.m* se poda: video posnetek, začetek, kjer se začne izrez, dolžino izreza in ime izhodne datoteke.

```
1 function rezanje (video,zacetek,dolzina,outputfajl)
2 system(["ffmpeg -i " '' video ' ' "-ss" ' ' int2str(zacetek) ' ' "-strict -2
      -t" ' ' int2str(dolzina) ' ' outputfajl ])
3 endfunction
```

Za združevanje video posnetkov, ki jih uporabimo za testiranje vsebovanosti, uporabimo funkcijo *lepljenje.m*. Tej funkciji podamo *cell arraj* imen video posnetkov in ime izhodnega video posnetka. Funkcija zapiše vsa imena v tekstovno datoteko mylist.txt in kliče na koncu FFMPEG.

```
1 function rezanje (video,zacetek,dolzina,outputfajl)
2 system(["ffmpeg -i " '' video ' ' "-ss" ' ' int2str(zacetek) ' ' "-strict -2
      -t" ' ' int2str(dolzina) ' ' outputfajl ])
3 endfunction
```

3.5 Predprocesiranje

Za predprocesiranje se sklicujemo na funkcije *predprocesiranje.m, mad.m* in *posnetek.m*.

Funkcija *predprocesiranje.m* sprejme kot argument polno pot do video posnetka v obliki niza. Ustvari nekaj začetnih spremenljivk, kot so pot do video posnetka, prazna matrika za shranjevanje podvzorčenih sličic in objekt za procesiranje video posnetkov prek OpenCV-ja.

```
1 function [Posnetek]= predprocesiranje (video_posnetek)
```

Na začetku se je inicializiralo spremenljivko *captureObjekt*, ki vsebuje video posnetek v podatkovnem tipu OpenCV. Z while zanko iteratiramo skozi video posnetek, ga podvzorčimo in shranimo v novo spremenljivko kot seznam matrik. Pri testiranju smo imeli težavo z branjem FLV posnetkov. Zadnjih nekaj sličic je *while zanka* vrnila kot prazne, zato smo podali dodatne pogoje v predprocesiranje.

Z funkcijo MAD, ki vzame dve matriki kot argumenta, izračunamo povprečno absolutno razliko ter jo zapišemo na i-to mesto v vektorju sprememb.

```
1 function s = mad(A,B)
```

Za alternativno računanje vektorja sprememb video posnetka imamo tudi skripto msd.m, ki izračuna MSD vrednost.

```
1 function ret = msd(A,B)
```

Na koncu se dopolni video posnetek *struct* z ustvarjenimi podatki in se ga vrne kot izhodni podatek funkcije za predprocesiranje.

3.5.1 Struktura Posnetek

V strukturi *Posnetek* hranimo vse potrebne spremenljivke za procesiranje. Zaradi števila argumentov, nujnih za delovanje, se je ustvarilo *struct*, ki nosi vse podatke znotraj ene spremenljivke. Takšna struktura je omogočila lažji pregled nad kodo. Struktura sprejme pot do video posnetka kot argument.

```
1 function pos = posnetek(video)
2  pos.datoteka=video;
3  pos.vektor_sprememb = [];
4  pos.podvzorcene_slike = [];
5  pos.pot_do_posnetka= "";
6  pos.hash = "";
7 endfunction
```

3.6 Proces analize

V procesu analize vzamemo strukturi video posnetkov in uporabimo njuna vektorja sprememb za iskanje križne korelacije ter kasneje tudi ujemanj z zamiki. Funkcija *iskanje_kandidatov* vrne možne kandidate za izračun zamikov, katere se potem poda tudi funkciji za iskanje ujemanj. Podajanje praga tej funkciji je neobvezno. V primeru, da prag ni podan, se ga določi na 0.5. Tako skozi več različnih testiranj kot v prejšnjem poglavju Metodologije se je izkazalo, da je 1 dovolj visok prag v primeru ujemanj. Prag se lahko določi nižje v primeru, da ni ujemanj, ampak temu lahko sledi tudi več možnih kandidatov za ujemanje ter daljši čas procesiranja. Kandidate se nato poda funkciji za iskanje ujemanj, s katero se preveri obseg ujemanja za vsakega kandidata.

3.6.1 Iskanje kandidatov za zamik

Funkcija za iskanje kandidatov sprejme obe Posnetek strukturi, podamo lahko tudi prag. Nazaj vrne obe strukturi in seznam kandidatov za zamike.

```
1  function [seznam_kandidatov prvi drugi] =
2  iskanje_kandidatov (prvi_posnetek,drugi_posnetek,prag)
```

Funkcija najprej primerja, če sta posnetka duplikata s prej izračunanim razpršilnim nizom. Sledi glavni korak, to je normirana križna korelacija. Kličemo našo prirejeno

funkcijo *normirana_korelacij*a, ker *normxcorr2* funkcija zahteva poseben vrstni red argumentov, kot je omenjeno v poglavju Metodologija. Tako preden podamo argumente *normxcorr2* funkciji, jih 'obrnemo' glede na njihovo dolžino vektorjev sprememb. Nato funkciji *find* podamo vektor korelacije in vrnemo seznam kandidatov za iskanje zamikov.

```
1 function [korelacija_rezultat prvi drugi] = normirana_korelacija(A,B)
2 #find funkcija
3 seznam_tau = find(vektor_korelacije>prag);
```

Izhodni podatek funkcije za iskanje kandidatov vrne strukturi video posnetkov in seznam kandidatov.

3.6.2 Iskanje pravih ujemanj

Za iskanje ujemanj uporabimo funkcijo *iskanje_ujemanj*. Ta sprejme seznam kandidatov iz križne korelacije, video strukturi in prag, ki je kot prej, spet neobvezen argument.

```
1 function[vsivektorji,zamik,obseg_ujemanja] =
2 iskanje_ujemanj (seznam_zamikov, prvivektor,drugivektor,treshold,
     stevilo_ujemanj)
```

Ker imamo seznam kandidatov, je celoten algoritem obdan s *for zanko*, katera iteratira skozi seznam zamikov. V telesu zanke se najprej izračuna indekse, ki so bili omenjeni v Metodologiji. Ko so izračunani, stopi program v drugo *for zanko*, ki primerja podvzorčene sličice od video posnetkov v obsegu danih indeksov.

Za določitev obsega ujemanj imamo na začetku tega koraka spremenljivko je_zacetek nastavljeno na *true*. *For zanka* gre skozi primerjalni vektor in si zapomni začetke in konce obsegov. Prvi *if* stavek preveri, če je i-ta vrednost manjša kot prag in je začetek. Naslednji *elseif* preverja, če je i-ta vrednost večja kot prag in ni začetek, v tem primeru je bila prva vrednost pod pragom označena in spremenljivka je_zacetek nastavljena na *false*. Ko se sproži prvi *elseif*, se nastavi je_zacetek na *true*, saj naslednjič, ko se bo ujemala vrednost pod pragom, pomeni, da je nov začetek obsega. Zadnji *elseif* je nastavljen v primeru, da so vrednosti primerjalnega vektorja pod pragom do konca vektorja.

3.7 Prikaz rezultatov

Funkcija p*rimerjanje slik* sprejme: obe strukturi video posnetkov, pravi zamik, vektor obsega ujemanj in prag. Podobna je funkciji za iskanje ujemanj, ampak kot izhodni

podatek nam poda animacijo ujemanja. Ta funkcija je uporabljena izključno za lepše prikazovanje rezultatov. Na koncu funkcija vrne video posnetek, ki prikazuje ujemanje glede na prag. Prikaže video posnetka, ko je vrednost točke v obsegu ujemanja pod pragom.

```
1 function = primerjanje_slik (prvi, drugi,vektor,zamik,treshold)
```

4 Testiranje

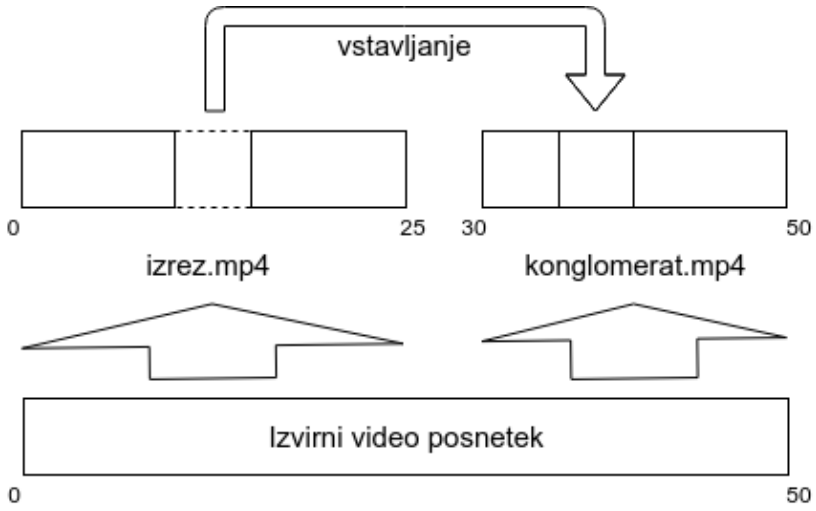
Kot omenjeno v uvodnem poglavju, želimo, da je proces zmožen najti ujemanja, tudi delna na krajših segmentih, med video posnetki, ki so bili podvzorčeni ali kompresirani. V tem poglavju bomo predstavili več primerov in poskusili najti ujemanja med njimi. Za primerjavo z drugimi testnimi primeri bomo naredili tudi analizo testnega primera brez transformacij.

4.1 Določitev in priprava testnih primerov

V testiranju se bo obravnavalo 3 različne testne primere. Uporabljena testna video posnetka smo izdelali iz izvirnega video posnetka, katerega uporabljamo skozi zaključno nalogo, *original.mp4*. Posnetek konglomerat.mp4 vsebuje del video posnetka izrezek.mp4. Posnetka sta ustvarjena tako, da se med sabo ne pokrivata, razen vsebovanega dela. Za stvaritev video posnetkov se je uporabilo funkcije *rezanje.m* in *lepljenje.m*. Treba je tudi upoštevati, da je lahko prišlo do morebitnih zanemarljivih razlik med video posnetki v procesu rezanja in sestavljanja video posnetkov z FFMPEG programom, saj se video posnetki ne delijo enostavno po sličicah ampak je odvisno od kodeka in formata v katerem so zapisani.

V testiranju se bo obravnavalo 3 različne testne primere. Uporabljena testna video posnetka smo izdelali iz izvirnega video posnetka, ki ga uporabljamo skozi zaključno nalogo, *original.mp4*. Posnetek konglomerat.mp4 vsebuje del video posnetka izrezek.mp4. Posnetka sta ustvarjena tako, da se med sabo ne pokrivata, razen v vsebovanem delu. Za stvaritev video posnetkov se je uporabilo funkcije *rezanje.m* in *lepljenje.m*. Treba je tudi upoštevati, da je prišlo do morebitnih zanemarljivih razlik med video posnetki v procesu rezanja in sestavljanja s FFMPEG programom, saj se video posnetki ne delijo enostavno po sličicah, ampak je to odvisno od kodeka in formata, v katerem so zapisani.

Za testne podatke smo priredili primer z vsebovanostjo. Iz izvirnega video posnetka smo izrezali dva različna video posnetka. Iz prvega, izrez.mp4, smo dodatno izrezali en del in ga vstavili v drugi video posnetek.



Slika 12: Prikaz načina kreacije testnega video posnetka. Iz izvirnega video posnetka smo izrezali 2 druga video posnetka. Iz izrez.mp4 smo vzeli en del in ga vstavili v konglomerat.mp4.

4.1.1 Ujemanje z vsebovanostjo in s podvzorčenostjo

Posnetek konglomerat.mp4 se je podvzorčilo na velikost 384x216 z pomočjo FFMPEG programa. Izhodni video posnetek je v velikosti 1.3MB. Uporabilo se je naslednji ukaz z parametri:

```
1 ffmpeg -i konglomerat.mp4 -vf scale=384:216 konglomerat_resized.mp4
```

Opis parametrov:

- *konglomerat.mp4* – z -i navedemo ime video posnetka;
- *-vf scale=384:216* – video filter za podvzorčenje video posnetka;
- *konglomerat_resized.mp4* – ime ciljnega video posnetka.

4.1.2 Ujemanje z vsebovanostjo in kompresijo

Podobno kot pri prejšnjem primeru se bo sestavljen video posnetek kompresiralo z različnimi kodeki in se ga poskusilo primerjati. Prvi video posnetek se je pretvorilo v flv format s pomočjo kodeka *flv*. Uporabili smo FFMPEG za enkodiranje in določitev jakosti kompresije.

FLV

Posnetek *konglomerat* je prešel z velikosti 3,2 MB na 2,1 MB, kar pomeni, da se je kakovost zmanjšala za tretjino. Konglomerat.flv ima dodatne 3 sličice več v zapisu

zaradi različnega enkodiranja, ki ga uporabljamo, na koncu pa sta zadnji dve sličici črni.

```
1 ffmpeg -i konglomerat.mp4 -qscale:v 31 -c:v flv konglomerat.flv
```

Opis parametrov:

- *konglomerat.mp4* z -i navedemo ime video posnetka;
- *-c:v* – je za navedbo kodeka, v našem primeru uporabljamo flv kodek;
- *-qscale:v 31* – se uporablja za variabilno bitno hitrost v video posnetku. Vrednost se giblje med 1 in 31, kjer je 1 najboljša kvaliteta in 31 najslabša.
- *konglomerat.flv* – ime ciljnega video posnetka.

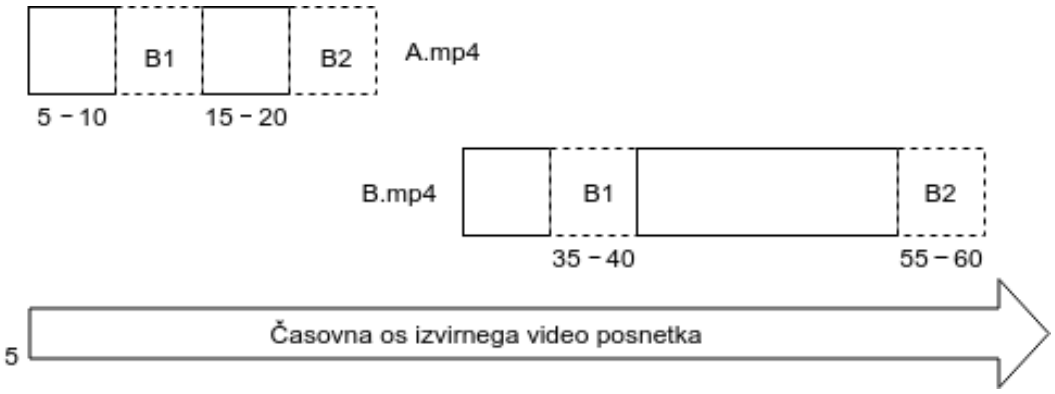
MPEG-4

Posnetek je po pretvorbi zmanjšan na velikost 2,4 MB. Uporabili smo MPEG-4 kodiranje, format je ostal enak. Za argumente velja enaka razlaga kot v prejšnjem razdelku. Parametri so enaki kot za FLV kodiranje, samo za kodiranje pa smo uporabili mpeg4.

```
1 ffmpeg -i konglomerat.mp4 -qscale:v 31 -c:v mpeg4 konglomeratmpeg4.mp4
```

4.1.3 Več razlčnih ujemanj

Večina testnih primerov, ki smo jih preizkusili, so imeli samo en pravi zamik. Vsebovanost se je ujemala izključno na enem zamiku. V tem primeru bi radi preizkusili, kaj se zgodi, če imamo dve vsebovanosti, ampak na drugačnih zamikih. Iz video posnetka B.mp4 smo vzeli dva izreza in jih vstavili v posnetek A.



Slika 13: Prikaz sestavljenih video posnetkov. Posnetka A.mp4 in B.mp4 se ne prekrivata. Iz B.mp4 smo izrezali fragmenta B1 in B2 in ju vstavili v posnetek A.

4.1.4 Alternativni izračun vektorja sprememb

MSD statistična metoda je bila predstavljena v Metodologiji, a je nismo uporabili, saj bi nam ovrednotila razlike, ki jih nismo želeli. Poleg tega ne upošteva šuma. Ker z MAD metodo nismo dosegli zadovoljivih rezultatov pri trenutnem testnem primeru, smo poskusili z MSD zaznati obe ujemanji.

Za testiranje se je v funkciji *predprocesiranje.m* zamenjalo klic MAD funkcije z MSD funkcijo, ki je vgrajena v Octave. Ta ovrednoti razlike z večjo vrednostjo. MSD funkcijo smo dodali samo v koraku predprocesiranja, saj nas zanimajo le korelacijske vrednosti, ki so pridobljene pri križni korelaciji med vektorji sprememb obeh video posnetkov.

4.1.5 Maksimalna absolutna razlika

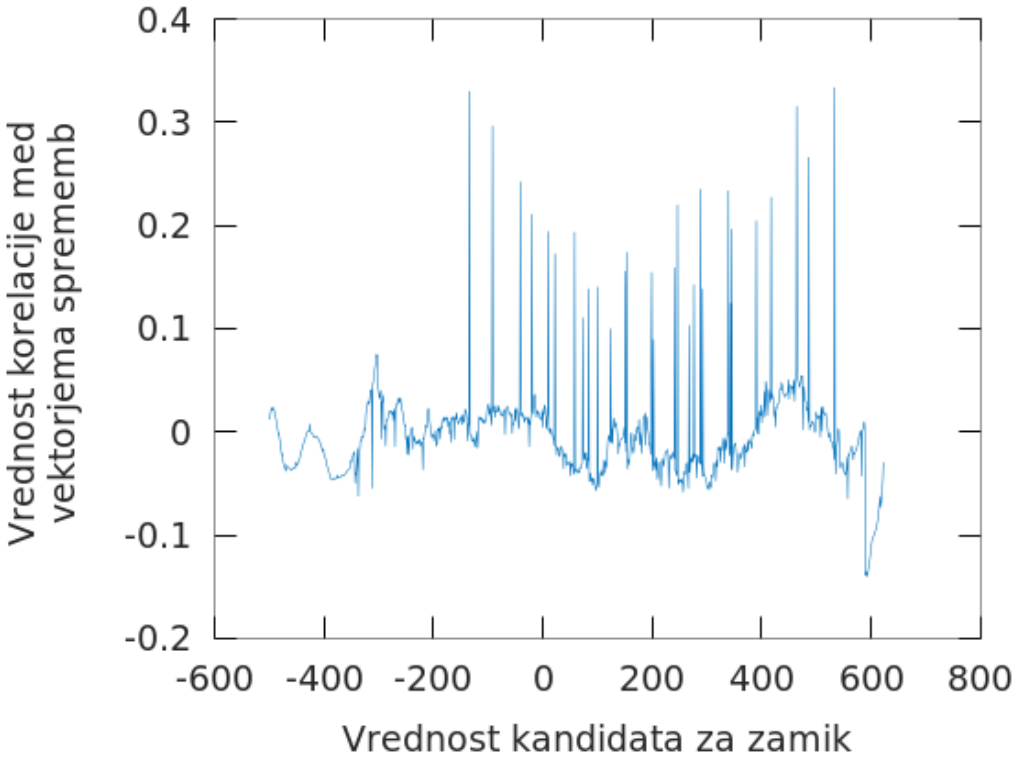
Primerjanje sličic za potrditev zamikov z MAD metodo nam je vrnilo tudi nekaj lažnih zamikov, kot opazimo na Sliki 24. Kot rešitev smo uporabili MAR, saj nam ta vrne največjo razliko med dvema sličicama. Posledično zato med ujemajočimi sličicami pričakujemo zelo majhne razlike v primerjavi z neujemajočimi. Razlike, ki se pojavijo na majhnih delih sličic, se z MAD povpreči po vseh vrednostih in so tako manj opazne v končnih vrednostih.

4.2 Testiranje in rezultati

V tem razdelku se bo testne video posnetke, ki smo jih pretvorili v druge oblike, analiziralo z našim programom. Sledi predstavitev rezultatov in razlaga.

4.2.1 Analiza testnega video posnetka brez transformacij

Testna video posnetka se ujemata v pragu za iskanje kandidatov 0.1 in pragu za iskanje obsega v velikosti 5. S pragom 0.5 za iskanje kandidatov nismo prišli do nikakršnih ujemanj. Na sliki korelacije vidimo, da so vse vrednosti pod pragom 0.5.



Slika 14: Prikaz korelacijskih vrednosti testnih video posnetkov brez transformacij. Graf ne prikaže nikakršnih izstopajočih vrednosti, zato smo prag za iskanje kandidatov znižali.

Da bi zajeli več različnih kandidatov za ujemanje, smo mejo znižali na 0.1. Funkcija nam tako vrne dva različna zamika.

- Tau = 101

  Obsegi ujemanj = [126 - 129] [177 - 200] [235 - 237] [244 - 245]

- Tau = 125

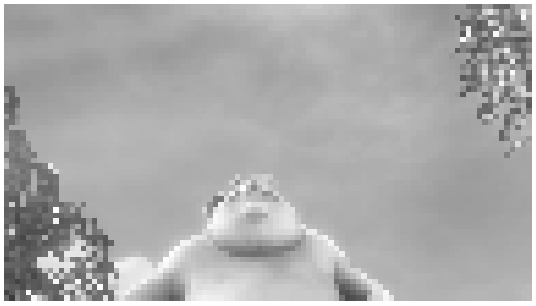
  Obseg ujemanj = [126 - 251]

Znano je, da je 125 prava vrednost zamika, ampak kot rezultat dobimo tudi 101. V naslednjem prikazu vidimo, da gre za majhno razliko med sličicama, kar naša funkcija za iskanje obsegov ujemanja zazna kot ujemanje, a v tem primeru ni tako. Vrednost MAD razlike med spodnjima neujemajočima sličicama je 3.972.

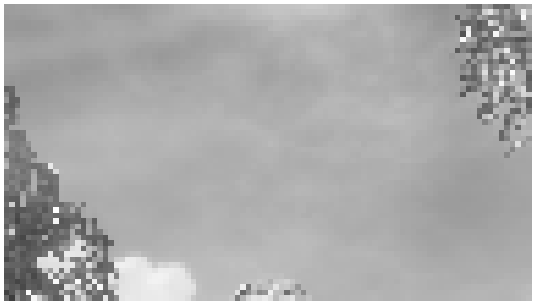
Na intervalu med 126 in 200 se opazi rahlo ujemanje, vendar je razlika med sličicami večja in se jih ujema manj, kot je dolžina vsebovanega video posnetka. Do takega ujemanja lahko pride, če so si sosednje sličice video posnetka na teh intervalih zelo podobne z minimalnimi razlikami.

Za zamik 101 dobimo nekaj obsegov, ki so dolgi le manjše število sličic. V resničnosti pa nam to ni pomembno, saj iščemo vsebovanosti, ki so dolge vsaj nekaj sekund.

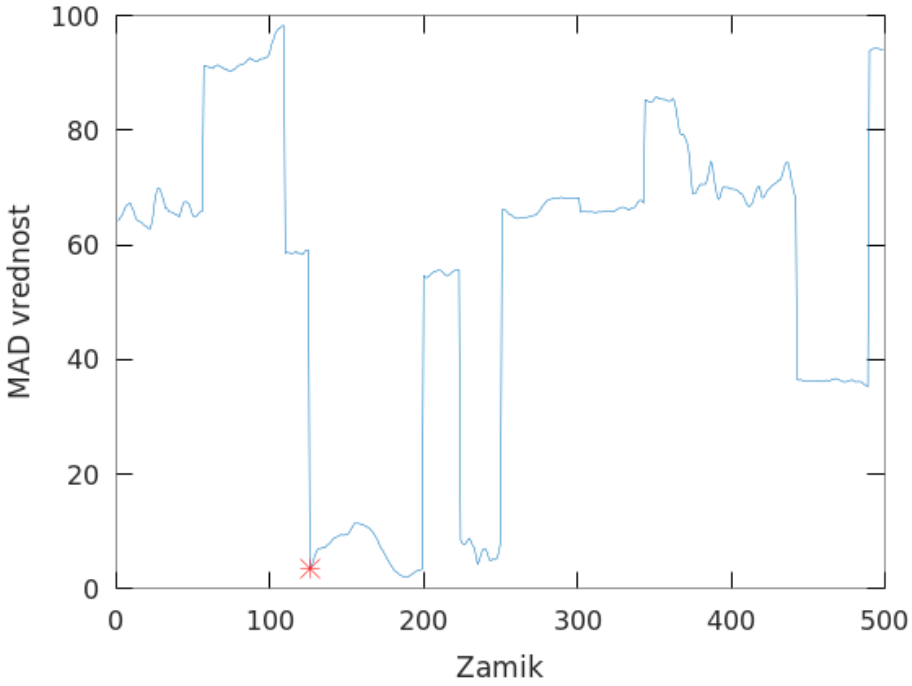
Takšne obsege lahko zavrnemo kot neustrezne. Rdeča pikica nakazuje, na katerem delu ujemanja sta prikazani sličici video posnetkov.



(a) Sličica prvega video posnetka



(b) Sličica drugega video posnetka



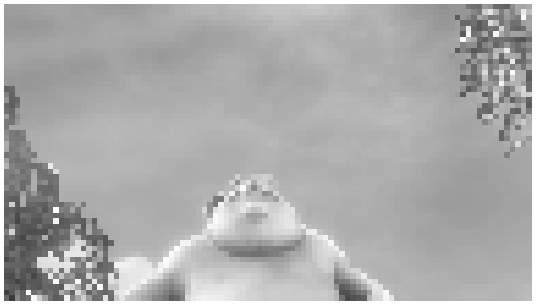
(c) Vektor razlik v obsegu ujemanja

Slika 15: Prikaz ujemanja z zamikom 101. Na sliki opazimo, da gre za zelo majhne razlike in naš algoritem je to zaznal kot ujemanje, a na sličicah vidimo, da to ni pravilno ujemanje.

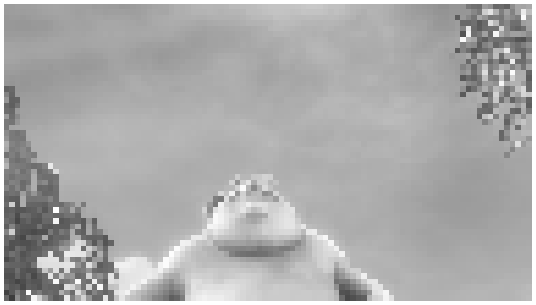
Z zamikom 125 dobimo obseg ujemanja v samo enem razponu in vrednost MAD razlike med sličicami obeh video posnetkov se za dani obseg močno zniža. Vrednost MAD razlike pade v celotnem obsegu pod 1 in če znižamo prag za iskanje obsega ujemanj na 1, dobimo za zamik samo vrednost 125.

4.2.2 Ujemanje z vsebovanostjo in s podvzorčenostjo

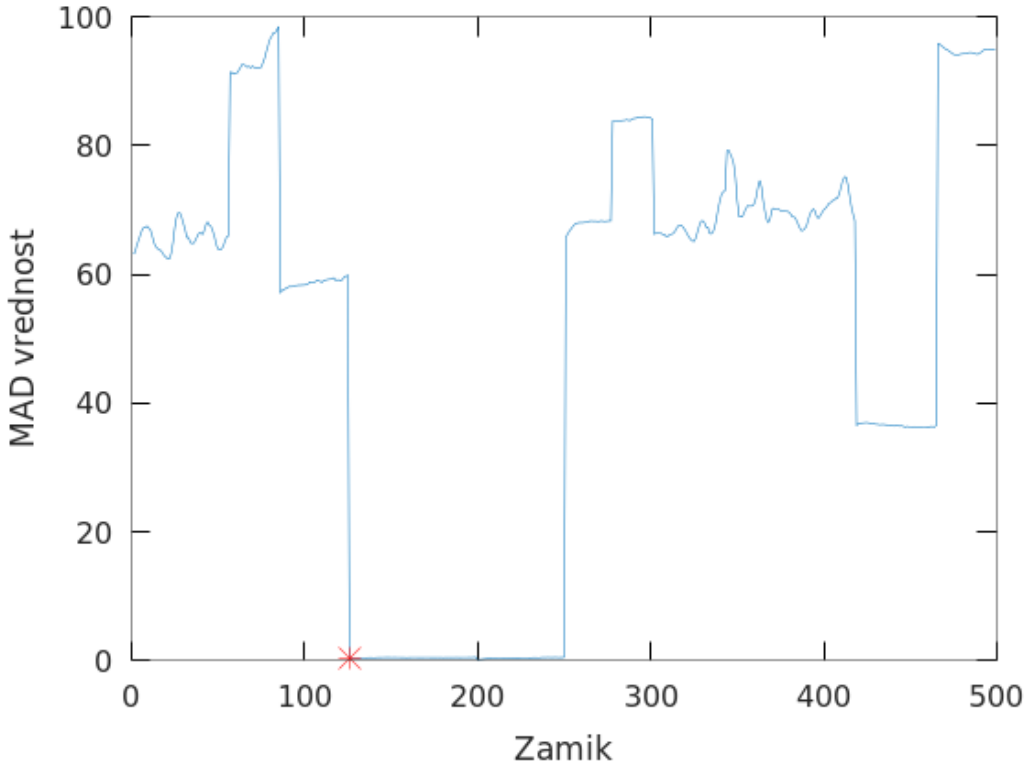
Podvzorčen video posnetek vrne vektor korelacije, ki je podoben ostalim. Pravega kandidata za zamik nam je vrnilo šele pri pragu za iskanje kandidatov 0.1, začeli pa



(a) Sličica prvega video posnetka



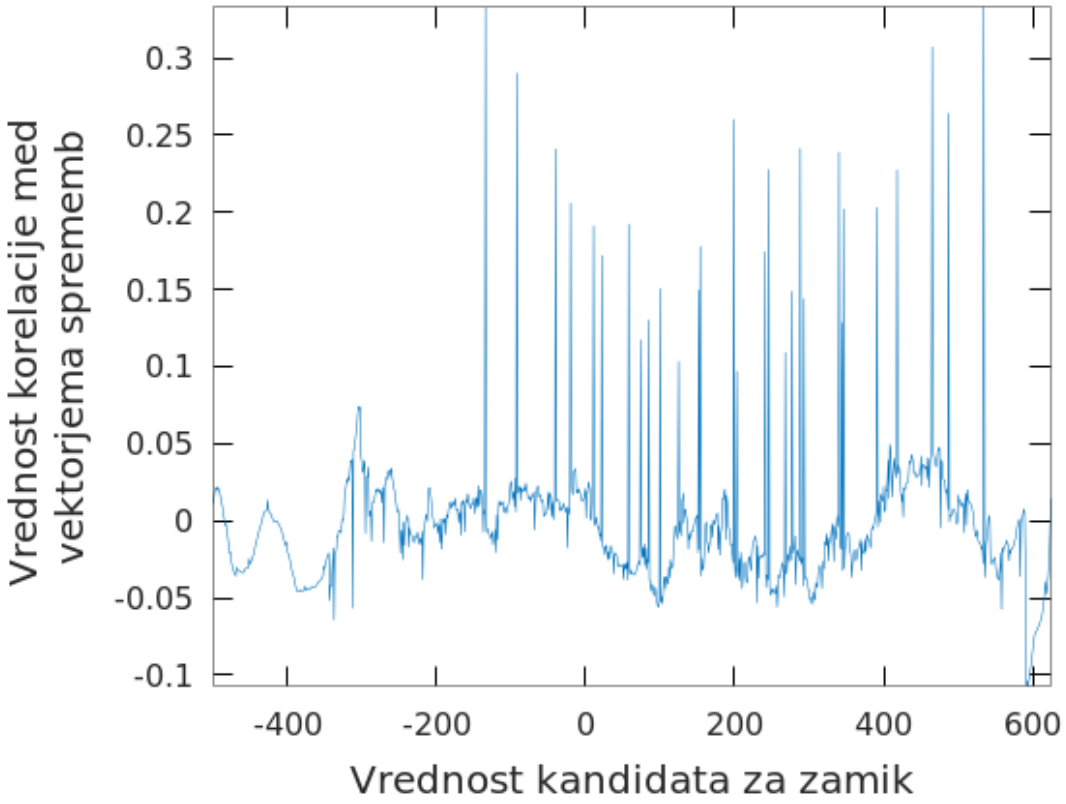
(b) Sličica drugega video posnetka



(c) Vektor razlik v obsegu ujemanja

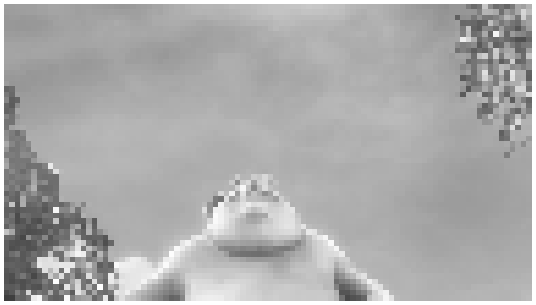
Slika 16: Prikaz ujemanja z zamikom 125. Obseg ujemanja je pod 1, kar pomeni popolno ujemanje, ki ga vidimo v obeh sličicah.

smo z 0.5. Za prag obsega ujemanja smo uporabili prag 5.

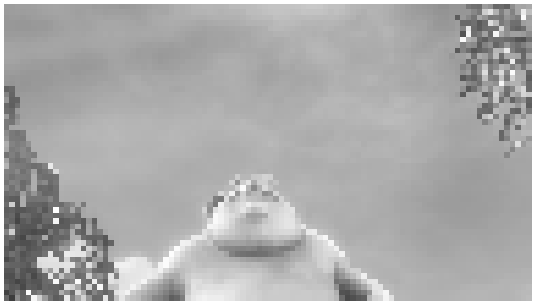


Slika 17: Prikaz križne korelacije testnega primera s podvzorčenostjo. Graf ne prikaže nobenih izstopajočih vrednosti, zato se zniža prag za zajem več različnih kandidatov.

- Tau = 124, Obsegi ujemanj = [127 - 252]



(a) Sličica prvega video posnetka



(b) Sličica drugega video posnetka

(c) Vektor razlik v obsegu ujemanja

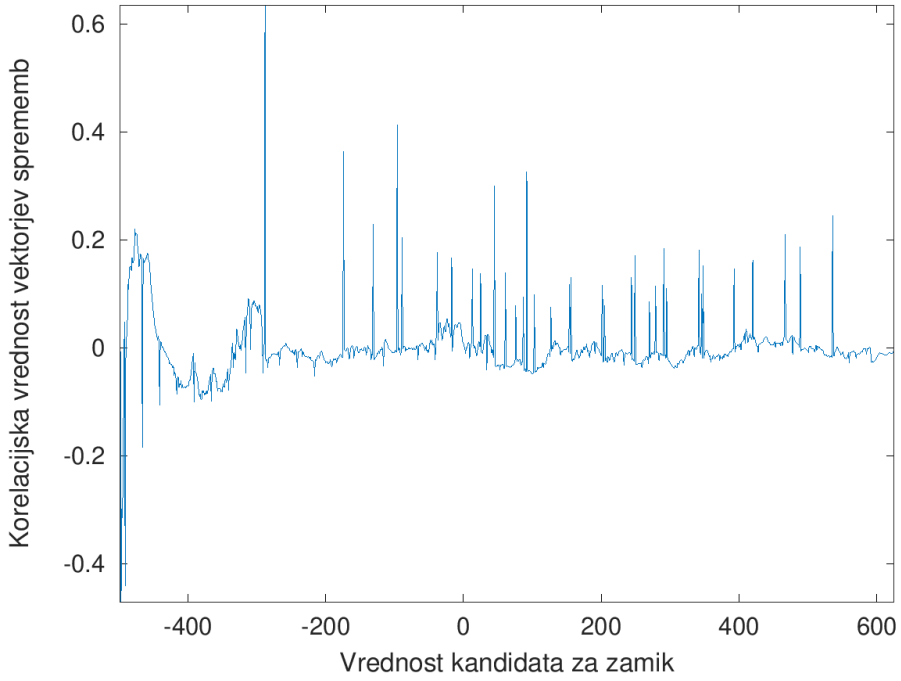
Slika 18: Prikaz primerjave sličic za zamik 124. Na točki 124 prikazujemo tudi obe sličici video posnetkov, kjer je bil uporabljen izračunan zamik. Na sličicah in na grafu vidimo, da pride do ujemanja na izbrani točki.

4.2.3 Ujemanje z vsebovanostjo in s kompresijo

Za kompresijo smo se odločili pri testiranju dveh izmed bolj uporabljenih in znanih standardov, in sicer FLV in MPEG4. Uporaba obeh nam vrne podobne rezultate.

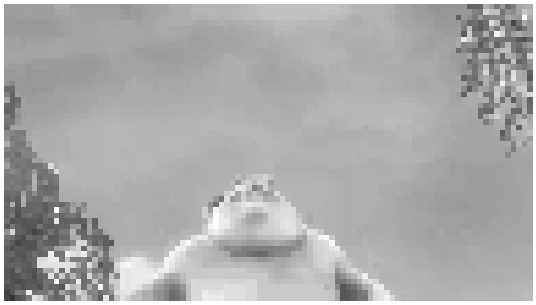
FLV

Z vhodnima podatkoma, *konglomerat.flv* in *izrez.mp4*, nam program vrne enega kandidata s pragom 0.5. Nato se kliče funkcijo za iskanje obsega ujemanj. Mejo nastavimo na 5 in iščemo ujemanja z najmanj 25 sličicami, kar pa nam ne vrne rezultata. Ker je znan pravi zamik, vemo, da ta ni pravi. Če pogledamo vektor korelacije, vidimo, da je vrednost korelacij v nižjih vrednostih.

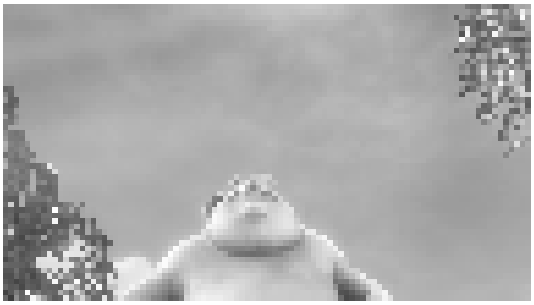


Slika 19: Prikaz korelacijskih vrednosti za testni primer s FLV enkodiranjem. V grafu opazimo izstopajočo vrednost nekje v okolici mesta -300, ki pa se je izkazalo za nepravilno.

Mejo iskanja kandidatov za zamik smo znižali postopoma, dokler nismo dobili obsega ujemanja. Rezultat smo dobili šele pri pragu za iskanje kandidatov pri vrednosti 0.05. Tako smo dobili 98 različnih kandidatov za ujemanje. Obseg ujemanja je bil 126 - 251 za *tau* velikosti 125 po izračunu.



(a) Sličica prvega video posnetka



(b) Sličica drugega video posnetka

(c) Vektor razlik v obsegu ujemanja

Slika 20: Prikaz obsega ujemanja in primerjanje sličic z izračunanim zamikom. Do ujemanja pride pri 126-ti sličici in se enakomerno nadaljuje do 251-te sličice.

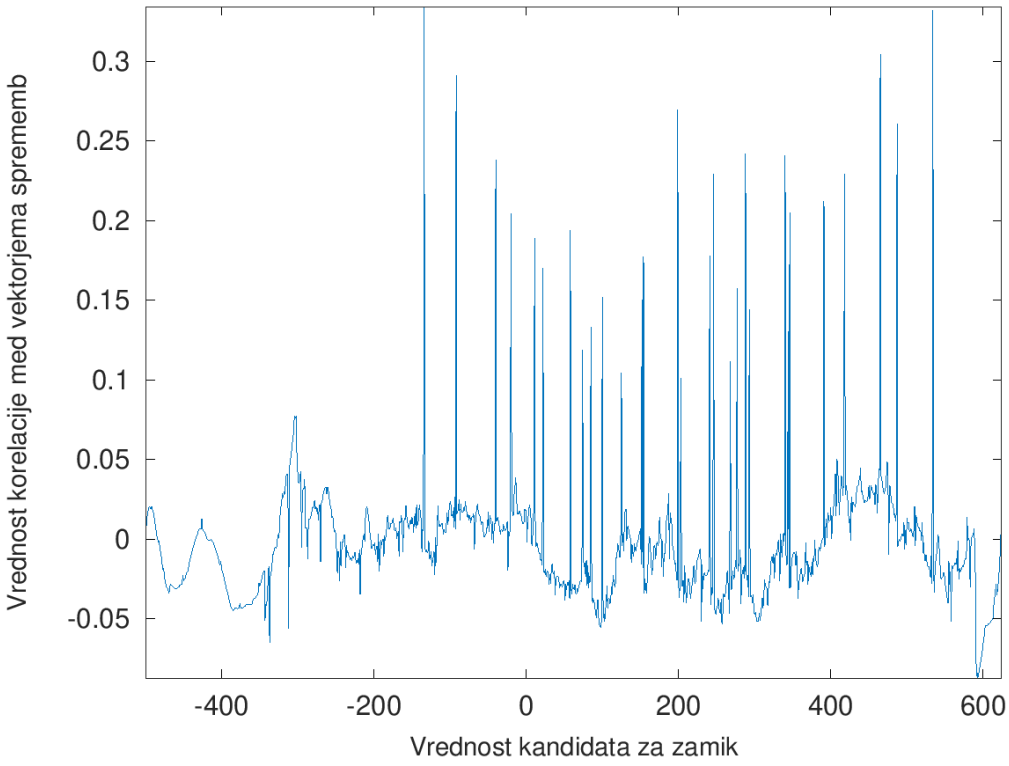
MPEG4

Z MPEG4 kodekom ni bilo veliko razlik. Meja za iskanje kandidatov za ujemanje je bila spuščena do 0.1, prag za iskanje obsega ujemanj je bil 5. Dobili smo samo en ujemajoč zamik.

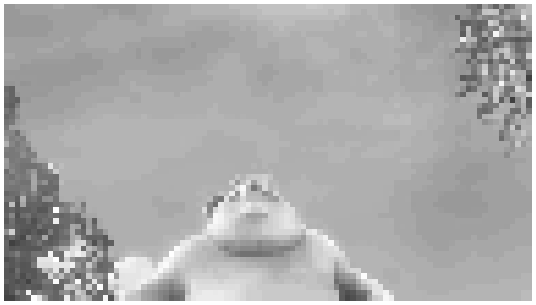
- Tau = 124

  Obseg ujemanj = [127 - 252]

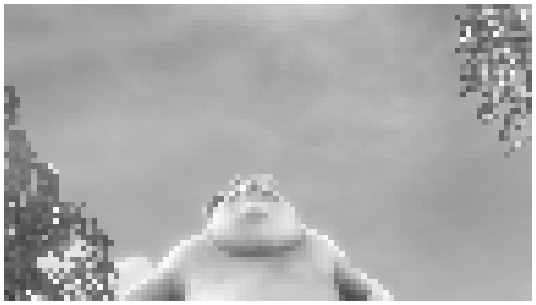
Večina korelacijskih vrednosti je bila nad 0.1. Naša iskana vrednost je bila okoli praga 0.1, kar je dosti nižje od drugih kandidatov.



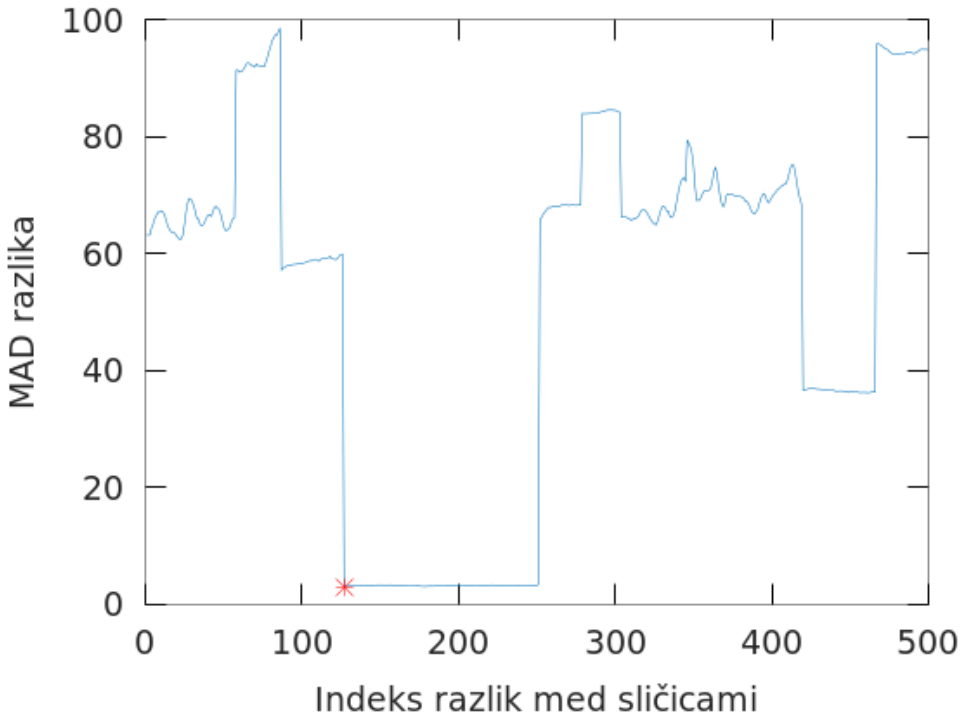
Slika 21: Prikaz korelacijskih vrednosti med posnetkoma A in B. Graf ne prikazuje izstopajočih vrednosti.



(a) Sličica prvega video posnetka



(b) Sličica drugega video posnetka



(c) Vektor razlik v obsegu ujemanja

Slika 22: Prikaz primerjave med ujemajočimi video posnetki za MPEG4. Na sličicah opazimo, da se video posnetka ujemata. Na grafu je označena tudi točka, ki prikazuje mesto ujemanja med sličicama v obsegu ujemanja.

4.2.4 Več različnih ujemanj

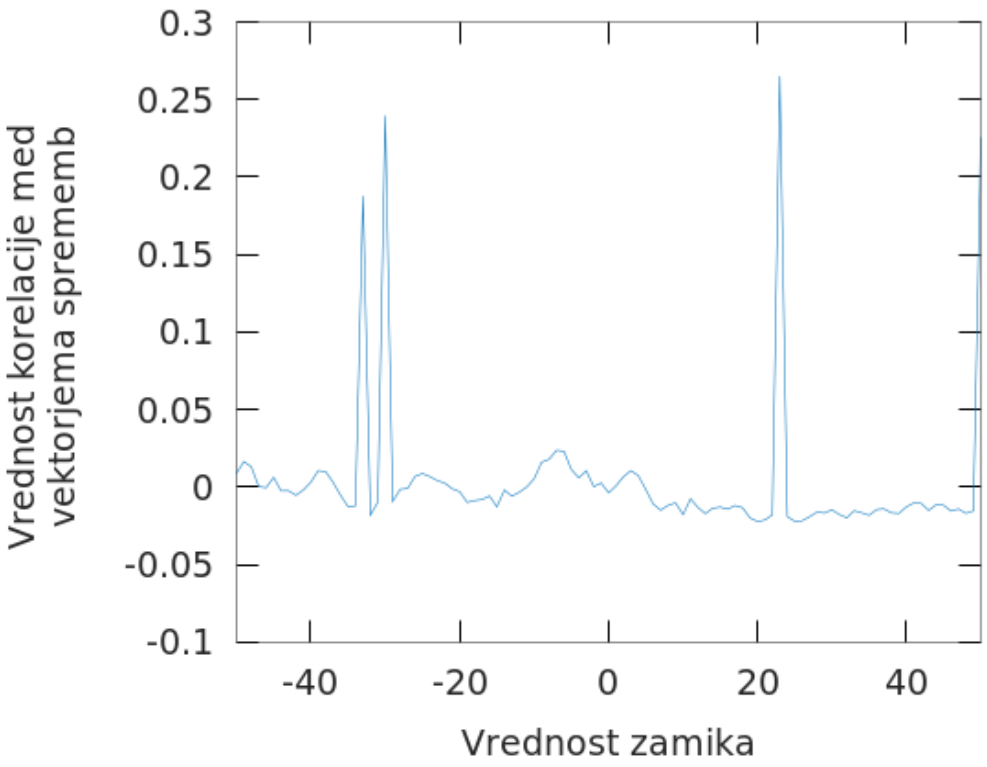
Posnetek *A.mp4* je dolg 500 sličic, posnetek *B.mp4* pa 750 sličic. Ker smo video posnetek sami pripravili, vemo, da obstajata dva zamika oz *tau-a*. Za ujemanje z B1, je zamik 0, za drugega pa 250, kandidata za ujemanje v tem primeru sta pa 499 in 749. Program nam z pragom za iskanje kandidatov 0.1 in pragom za obseg ujemanja 5 vrne nekaj rezultatov.

Posnetek *A.mp4* je dolg 500 sličic, posnetek *B.mp4* pa 750 sličic. Ker smo video posnetka pripravili sami, vemo, da obstajata dva zamika oz. *tau-a*. Za ujemanje z B1 je zamik 0, za drugega 250, kandidata za ujemanje v tem primeru pa sta 499 in 749.

Program nam pri pragu za iskanje kandidatov 0.1 in pragu za obseg ujemanja 5 vrne naslednje rezultate:

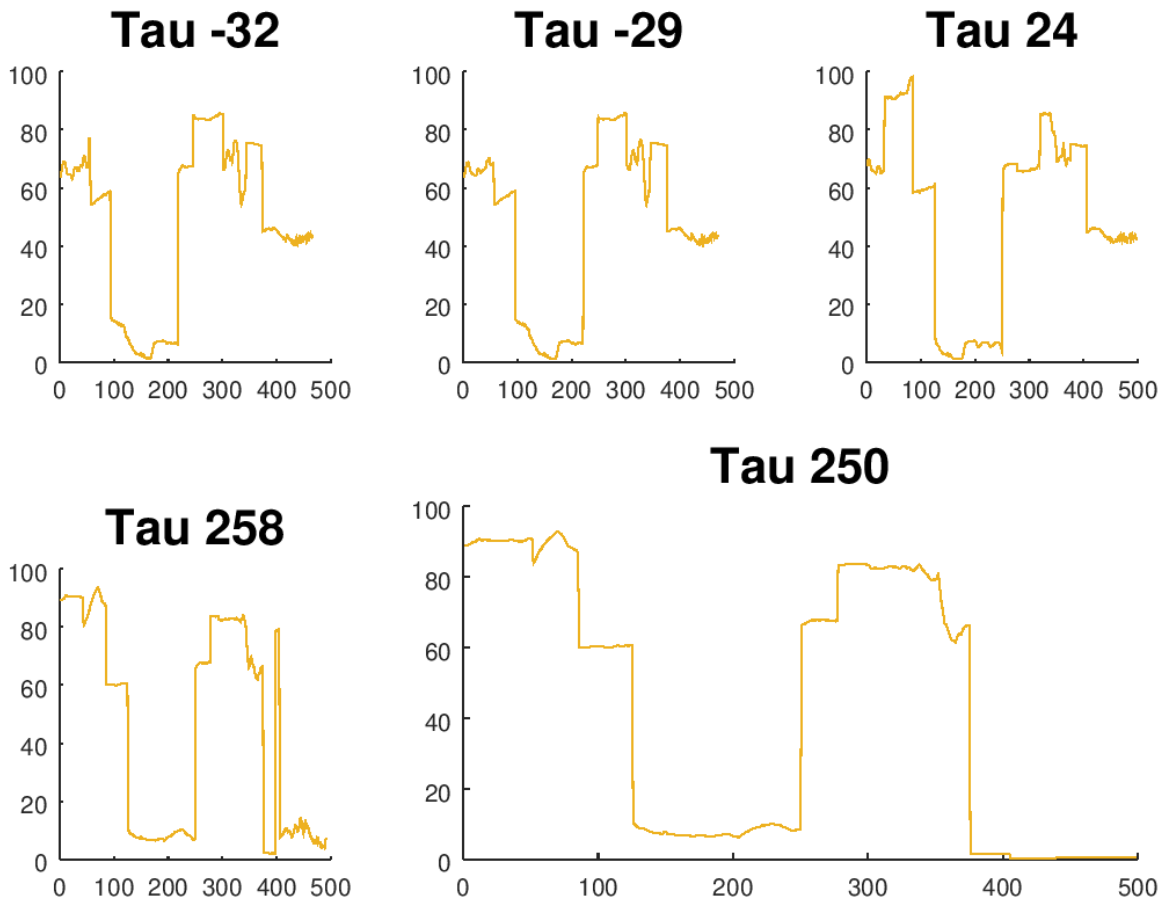
- tau: -32 obseg: [135 - 173],
- tau: -29 obseg: [134 - 176],
- tau: 24 obseg: [134 - 181] [249 - 251],
- tau: 250 obseg: [376 - 499],
- tau: 258 obseg: [376 - 398] [484 - 491].

V korelacijskem vektorju lahko preberemo, da obstaja kandidat za zamik, vrednost 750, za 500 pa ni visoke korelacije.



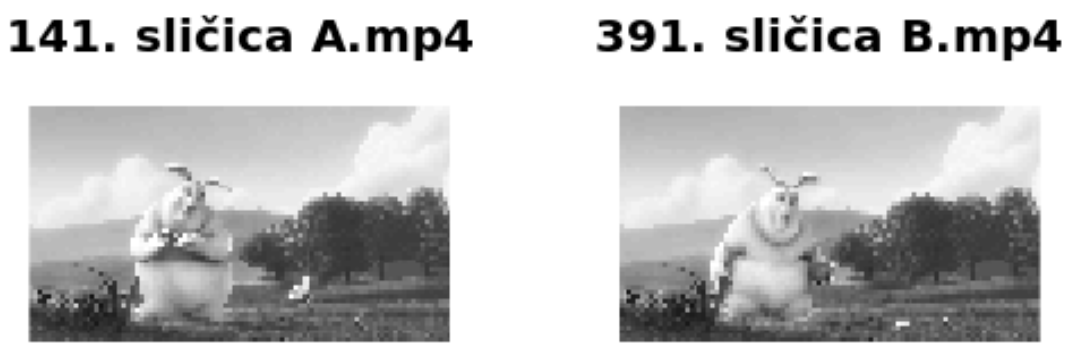
Slika 23: Korelacijska vrednost za prvi zamik. Graf prikazuje približan vpogled v vektor korelacijskih vrednosti. Za naš zamik 500 iščemo visoko vrednost na mestu 0 v grafu.

Pri prikazu obsega ujemanj vidimo, da se video posnetka pri *tau* 250 v zadnjem delu obsega skoraj popolnoma ujemata. Nekakšno ujemanje vidimo tudi v sredini, kar pa ni pravilno.



Slika 24: Prikaz rezultatov ujemanj za različne tau-e. Do ustreznega ujemanja pride pri tau 250, pri vseh drugih pa opazimo le delna ujemanja, ki niso pravilna.

Razlog za takšno ujemanje je v podobnosti sličic, čeprav sta časovno na drugačnih indeksih. Če pogledamo 141. sličico posnetka A.mp4 in 391. sličico posnetka B.mp4, vidimo, da sta si zelo podobni in MAD razlika med njima je 7.6.

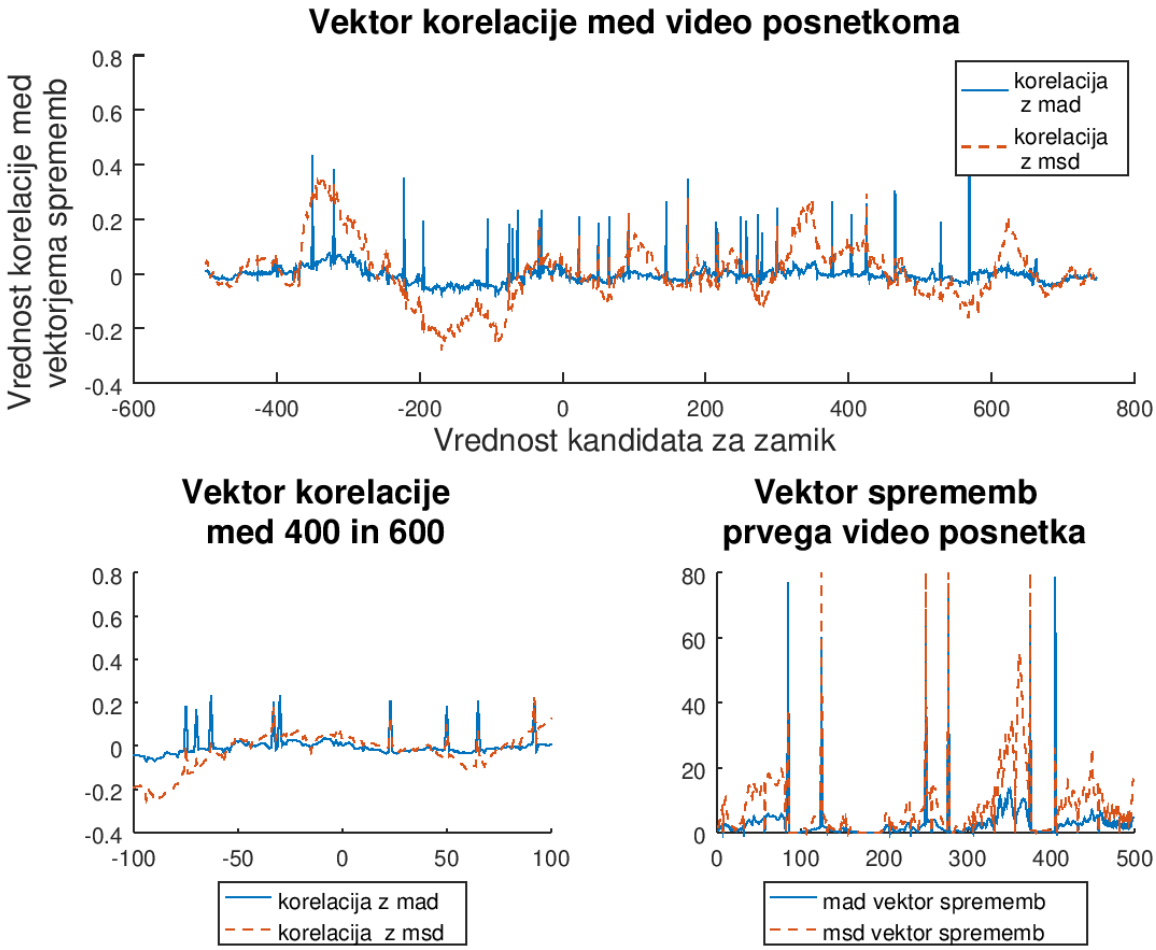


Slika 25: Primerjanje sličic za zamik 250. Opazimo, da nista enaki in se ne ujemata povsem, a sta si podobni in je razlika le v nekaj delih okoli subjekta.

Za drugi zamik, ki je 0, oziroma za kandidat ujemanja 500 ne dobimo točnega rezultata. Najbližje, kar vrne program, je za *tau* 24. Pri tem se vidi, da gre za delno ujemanje od 134 do 250, do katerega je prišlo zaradi podobnosti v sličicah v obsegu od 125 do 250. Gre za del izvirnega video posnetka, ki ima zelo malo sprememb med sličicami, saj so si zelo podobne.

MSD rezultati

Pri vektorju sprememb so vrednosti razlik mnogokrat večje. Spodaj je prikazan eden od vektorjev sprememb. Podan je tudi korelacijski vektor z izbranim pogledom med indeksoma 400 in 600. Na indeksu 500 ni nikakršne visoke korelacije.



Slika 26: Prikaz računanja z MSD vektorjem sprememb in prikaz vektorja korelacije za enak testni primer. Z računanjem MSD vektorja sprememb ni prišlo do zadovoljivih rezultatov, ki bi pokazali iskano ujemanje.

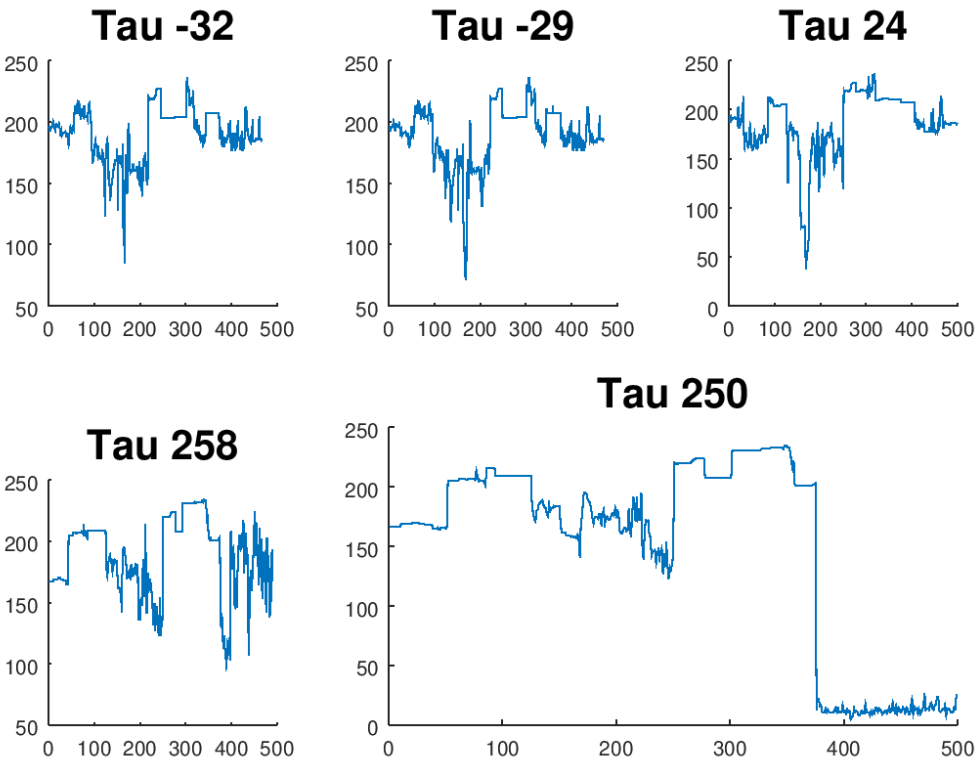
Ker korelacija bolje vrednoti višje vrednosti v vektorjih sprememb, pride do zanemarjenja manjših vrednosti. Za zelo visoke vrednosti lahko nastavimo prag in jih prepišemo s tem pragom. V našem primeru se manjše vrednosti gibljejo pod 80. Najvišje vrednosti v obeh vektorjih pa so tudi nad 100. Za testiranje se določi prag 80 in nato ponovno analizira. Rezanje vektorja sprememb ni vrnilo želenih rezultatov. Na mestu 500 ni vrednosti visokih korelacij.

Problem ostaja nerešen, saj smo se odločili, da ga pustimo odprtega.

Primerjanje sličic z MAR

Za *tau* 250 smo dobili lažna ujemanja na razmikih od 125 do 250, saj MAD funkcija ni odporna na zelo majhne razlike v video posnetkih. MAR pa se na drugi strani izkaže kot zanesljiva metoda v primeru več ujemanj, ki nam ne vrne lažnih ujemanj. Za obseg ujemanja smo uporabili prag 30, saj nam je vedno vrnilo maksimum razlike, zato so

vrednosti višje kot z MAD primerjavo.



Slika 27: Prikaz rezultatov obsega ujemanj z računanjem vrednosti MAR. Pri *tau* 250 opazimo na koncu oster padec v vrednostih, kjer obstaja ustrezno ujemanje.

Za primerjavo smo vzeli tudi lažne zamike, ki smo jih preverili z MAD in jih tokrat primerjali z MAR. Za pravilni tau 250 vidimo ujemanje na koncu, lažno ujemanje na indeksih od 125 do 250 pa je izginilo. Funkcija za iskanje obsega ujemanj nam vrne tudi:

- tau: 250 obseg: [376 - 499]

Sklepamo, da bi MAR lahko uporabili kot privzeto metodo za preverjanje obsega ujemanja. Težava bi nastala le, če bi MAR metoda vračala velike razlike, ko jih v sliki v bistvu ni. Recimo, če sta video posnetka skoraj enake kvalitete, ampak zaradi rekodiranja oz. procesiranja nastane razlika le v nekaj pikslih, vsebinsko pa je skoraj ni, to poda lažne rezultate. MAD pa takšne razlike povpreči, zato jih veliko ne upošteva.

5 Zaključek

V zaključni nalogi smo iskali vsebovanost video posnetkov s pomočjo statističnih metod. Težave smo imeli le v nekaj primerih, ko so bila ujemanja zelo kratka in ko je bila okolica video posnetka zelo statična, ter z video posnetki, ki imajo več drugačnih zamikov. Natančnost bi lahko izboljšali z različnimi funkcijami za izračun vektorja sprememb in ujemanja slik.

5.1 Zaključek

V zaključni nalogi sem bolje spoznal prej neznano domeno, to je procesiranje signalov. Odkril sem tudi okolje Octave, ki mi je omogočilo hitrejšo obdelavo numeričnih podatkov kot drugi konvencionalni programski jeziki. Največ težav pri izdelovanju metod je povzročilo soočenje z okoljem Octave in učenje jezika Octave. Menim, da smo s tem problemom zajeli le določen del procesiranja in primerjanja video posnetkov ter da bi se nalogo lahko še dodatno obogatilo z uporabo več različnih metod, kot so na primer razpršilne funkcije. V prihodnosti bi bilo priporočljivo in smiselno obdelovati video posnetke na strojni opremi z več pomnilnika in zmožnostjo procesiranja na več jedrih.

5.2 Nadaljevanje

Zaključno nalogo je možno še razširiti, v nadaljevanju bi lahko namreč pogledali tudi druge transformacije video posnetkov:

- različna razmerja sličic,
- iskanje vsebovanosti video posnetka v drugih posnetkih kot manjša projekcija,
- različne hitrosti sličic na sekundo.

Srečali smo se tudi s primerom, kjer je bilo na različnih zamikih več ujemanj. Primer nam je uspelo rešiti za en zamik, za drugega pa ni našlo ustrezne korelacijske vrednosti. Zelo možno je, da zaradi podobnosti posnetka program ni uspel najti zamika. Težavo smo poskusili rešiti tudi s programom video-comparer, ampak neuspešno.

Trenutna implementacija je časovno in prostorsko zahtevna. Največ časa porabi predprocesiranje. Zamudno je lahko tudi iskanje obsega ujemanj v primeru, da imamo

večje število kandidatov za zamike. Naštete operacije bi se lahko pospešilo z uporabo grafičnega pospeševanja, na primer z uporabo CUDA jeder v Octave okolju. Zadevo upočasnjuje tudi sprotni prevajalnik, kar bi lahko rešili s C++, kjer imamo že prevedeno programsko kodo, ki je bistveno hitrejša.

Videoposnetek vsebuje tudi zvočni posnetek, ki ima lahko edinstvene značilnosti in se ga obravnava kot signal, težava pa nastopi, če se zvočni posnetek ne ujema zaradi zakasnitve. Zalomi se pri primerjanju razlik, saj zvok ne nosi toliko informacij kot slika. Ta je na splošno le matrika z več različnimi vrednostmi, medtem ko ima zvok samo jakost in amplitudo.

6 Literatura in viri

[1] A. A. Bahl. Izvorna koda programa predstavljenega v zaključni nalogi. `https://github.com/aleksanderarun/videoprocess/tree/diploma`, zadnje dostopno 2019-07-02. *(Citirano na strani 1.)*

[2] E. Bohain. Video Comparer. `https://www.video-comparer.com/`, zadnje dostopno 2019-04-11. *(Citirano na strani 4.)*

[3] Bolide Software. Duplicate Video Search -. `http://duplicatevideosearch.com`, zadnje dostopno 2019-04-02, 2019. *(Citirano na strani 2.)*

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2019. *(Citirano na strani 23.)*

[5] Jon Crowcroft. *Jon Crowcroft*. CRC Press, 1999, 1999. *(Citirano na strani 7.)*

[6] FFMPEG Devleopers. FFmpeg Developers. (2016). ffmpeg tool (Version 3.4.4-1) [Software]. Available from. `http://ffmpeg.org/`, zadnje dostopno 2019-04-02. *(Citirano na strani 22.)*

[7] D. Starkweather E. Klinger. pHash Design. `https://www.phash.org/docs/design.html`, zadnje dostopno 2019-04-02. *(Citirano na strani 3.)*

[8] D. Starkweather E. Klinger. pHash.org. `https://www.phash.org/`, zadnje dostopno 2019-04-02. *(Citirano na strani 3.)*

[9] J. W. Eaton. GNU Octave: C-Style I/O Functions. `https://octave.org/doc/v4.2.0/C{\_}002dStyle-I{\_}002fO-Functions.html{\#}C{\_}002dStyle-I{\_}002fO-Functions`, zadnje dostopno 2018-12-26. *(Citirano na straneh 20 in 22.)*

[10] F. Zhao, Q. Huang, W. Gao. Image Matching by Normalized Cross-Correlation. In *2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, volume 2, pages II–729–II–732. IEEE, 2006. *(Citirano na strani 10.)*

[11] Octave forge. Octave Forge. `https://octave.sourceforge.io/`, zadnje dostopno 2019-04-08. *(Citirano na strani 22.)*

[12] Grieviant. Savvanah octave bug. http://savannah.gnu.org/bugs/?35191, zadnje dostopno 2019-05-02. *(Citirano na strani 22.)*

[13] J. W. Eaton. About. https://www.gnu.org/software/octave/about.html, zadnje dostopno 2018-12-26, 2018. *(Citirano na straneh 20 in 21.)*

[14] J. W. Eaton, D. Bateman, S. Hauberg, R. Wehbring. {GNU Octave} version 4.2.1 manual: a high-level interactive language for numerical computations. https://www.gnu.org/software/octave/doc/v4.2.1/, zadnje dostopno 2018-12-26, 2017. *(Citirano na straneh 20 in 22.)*

[15] S. Bombaywala J.N. Sarvaiya, S. Patnaik. Image registration by template matching using normalized cross-correlation. *ACT 2009 - International Conference on Advances in Computing, Control and Telecommunication Technologies*, (3):819–822, 2009. *(Citirano na strani 10.)*

[16] Kyamagu. Mexopencv. http://kyamagu.github.io/mexopencv/, zadnje dostopno 2019-06-02. *(Citirano na strani 23.)*

[17] O. Levenspiel. Wikipedia article. https://en.wikipedia.org/wiki/Octave_Levenspiel, zadnje dostopno 2019-04-02, 2019. *(Citirano na strani 22.)*

[18] Mathlab. Normalized 2-D cross-correlation . https://www.mathworks.com/help/images/ref/normxcorr2.html, zadnje dostopno 2019-04-02. *(Citirano na strani 11.)*

[19] Matlab. MathWorks - MATLAB & Simulink. https://www.mathworks.com/company/aboutus/contact{_}us.html?s{_}tid=gn{_}cntus, zadnje dostopno 2018-12-26. *(Citirano na strani 20.)*

[20] M. Miller. Octave PPA. https://launchpad.net/octave, zadnje dostopno 2019-04-08. *(Citirano na strani 23.)*

[21] Sample Videos developers. Sample Videos https://sample-videos.com. *(Citirano na straneh 1 in 11.)*

[22] K. Yamaguchi. Mexopencv. https://github.com/kyamagu/mexopencv, zadnje dostopno 2019-04-12. *(Citirano na strani 23.)*