Univerzitet u Nišu
Elektronski fakultet

Metodi i sistemi za obradu signala

Seminarski rad

Tema:

Video kompresija primenom H262 standarda

Profesor:
Prof. Miloš Radmanović

Studenti:
Petar Kovačević 18212
Sreten Matić 18246

Sadržaj

Video kompresija

Video kompresija se zasniva na dva principa. Prva je prostorna redundantnost koja postoji u svakom okviru. Druga je činjenica da je u većini vremena video okvir veoma sličan njegovim neposrednim susedima. Ovo se zove temporalna redundantnost. Tipična tehnika za kompresiju video zapisa bi stoga trebalo da počne kodiranjem prvog kadra koristeći metod kompresije nepokretne slike. Zatim bi trebalo da kodira svaki sledeći okvir identifikovanjem razlika između okvira i njegovog prethodnika i kodiranje tih razlika. Ako se okvir veoma razlikuje od svog prethodnika (kao što se dešava sa prvim okvirom snimka), trebalo bi da bude kodiran nezavisno od bilo kog drugog kadra. U literaturi video kompresije, okvir koji je kodiran korišćenjem svog prethodnika naziva se međuokvir (ili samo inter), dok se okvir koji je kodiran nezavisno naziva unutar okvira (ili samo intra).

Video kompresija je obično sa gubicima. Kodiranje okvira Fi u smislu njegovog prethodnika Fi−1 unosi neka izobličenja. Kao rezultat, okvir za kodiranje Fi+1 u smislu Fi povećava izobličenje. Čak i u video kompresiji bez gubitaka, okvir može izgubiti bitove. Ovo se može desiti tokom prenosa ili nakon dužeg nekorišćenja. Ako okvir Fi izgubi neke bitove, onda se dekodiraju svi okviri koji slede, do sledećeg unutrašnjeg okvira nepravilno, možda čak i dovodeći do nagomilanih grešaka. Zbog toga bi unutrašnji okvira trebalo da se s vremena na vreme koristi unutar sekvence, a ne samo na njenom početku. Unutrašnji okvir je označen sa I, a međuokvir je označen kao P (za predviđanje). Kada se shvati ova ideja, moguće je generalizovati koncept međuokvira.
Takav okvir može biti kodiran na osnovu jednog od svojih prethodnika, a takođe i na osnovu jednog od njegovih naslednika. Znamo da koder ne bi trebalo da koristi informacije koje nisu dostupne dekoderu, ali je video kompresija posebna zbog velike količine podataka koji su uključeni. Obično nam ne smeta ako je koder spor, ali dekoder mora biti brz. Tipičan slučaj je video snimljen na hard disku ili DVD-u, koji treba da se reprodukuje.

Koderu može biti potrebno nekoliko minuta ili sati da kodira podatke. Dekoder, međutim, mora da reprodukuje pravilnom brzinom kadrova (toliko kadrova u sekundi), tako da mora biti brz.
Zbog toga tipičan video dekoder radi paralelno. Ima nekoliko kola za dekodiranje radeći istovremeno na nekoliko ramova. Imajući ovo na umu, sada možemo zamisliti situaciju u kojoj enkoder kodira okvir 2 na osnovu oba okvira 1 i 3, i upisuje okvire u komprimovani tok u redosled 1, 3, 2. Dekoder ih čita ovim redosledom, dekodira frejmove 1 i 3 paralelno, emituje okvir 1, a zatim dekodira okvir 2 na osnovu okvira 1 i 3. Okviri bi trebalo da, budu jasno (vremenski) označeni. Okvir koji je kodiran na osnovu oba prošli i budući okviri su označeni sa B (za dvosmerne). Predviđanje okvira na osnovu njegovog naslednika ima smisla u slučajevima kada kretanje objekta na slici postepeno otkriva pozadinu. Takva oblast može biti samo delimično poznata u trenutnom kadru, ali može biti poznatija u sledećem kadru. Dakle, sledeći okvir je prirodan kandidat za predviđanje ove oblasti u trenutnom okviru. Ideja B okvira je toliko korisna da većina okvira u kompresovanoj video prezentaciji može biti ovog tipa. Stoga završavamo sa nizom komprimovanih okvira tri tipa I, P i B. I okvir se dekodira nezavisno od bilo kog drugog okvira. P okvir se dekodira korišćenjem prethodnog I ili P okvira. B okvir se dekodira pomoću prethodnog i sledećeg I ili P okvira.

Na slici 6.9a prikazan je niz takvih okvira redosledom kojim ih generiše koder (i unosi dekoder). Slika 6.9b prikazuje isti niz u redosledu kojim se frejmovi izlaze od strane dekoder i prikazan. Okvir sa oznakom 2 treba da bude prikazan posle okvira 5, dakle svaki okvir treba da ima dve vremenske oznake, vreme kodiranja i vreme prikaza.

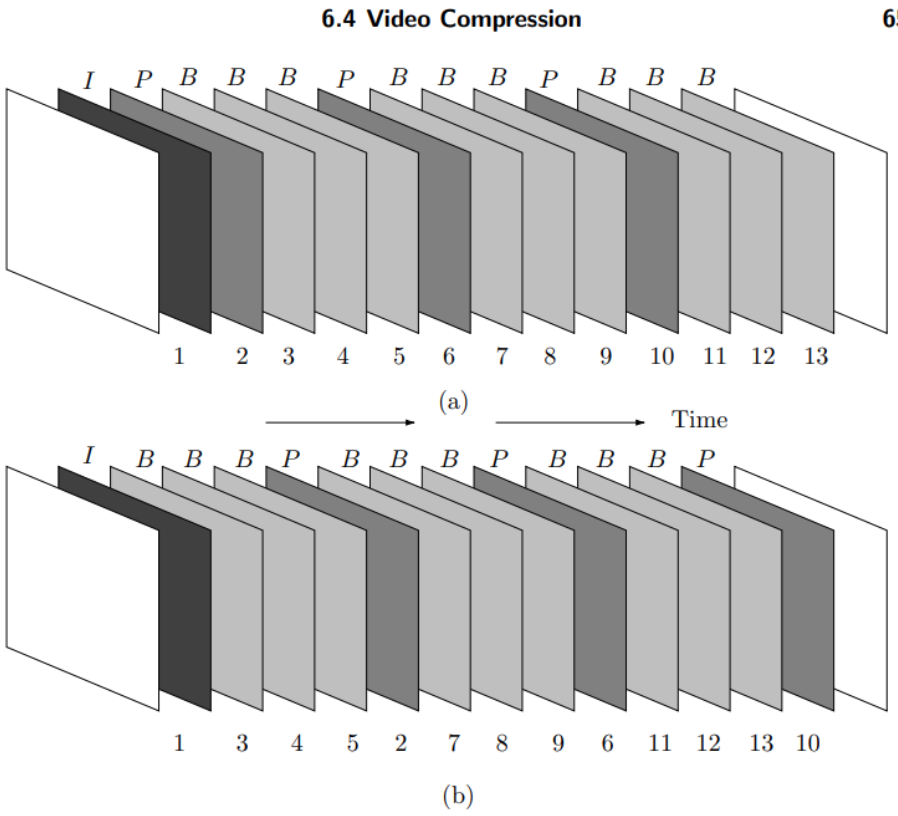


Figure 6.9: (a) Coding Order. (b) Display Order.

Počinjemo sa nekoliko intuitivnih metoda kompresije video zapisa.
Poduzorkovanje: Koder bira svaki drugi okvir i upisuje ga u komprimovani tok. Ovo daje faktor kompresije 2. Dekoder unosi okvir i duplira ga da stvori dva okvira.
Razlika: okvir se upoređuje sa prethodnikom. Ako je razlika između njih mala (samo nekoliko piksela), enkoder kodira piksele koji se razlikuju za tri broja u komprimovani tok za svaki piksel: njegove koordinate slike, i razlika između vrednosti piksela u dva okvira. Ako je razlika između okvira velika, trenutni okvir je ispisan na izlazu u sirovom formatu. Verzija razlikovanja sa gubicima gleda na količinu promene u pikselu. Ako je razlika između intenziteta piksela u prethodnom i u trenutnom okviru manja od određenog praga, piksel se ne smatra drugačijim.
Razlikovanje blokova: Ovo je dalje poboljšanje razlikovanja. Slika je podeljena na blokove piksela, a svaki blok B u trenutnom okviru se poredi sa odgovarajućimm blokom P u prethodnom okviru. Ako se blokovi razlikuju za više od izvesnog iznosa, onda se B komprimuje pisanjem njegovih koordinata slike, praćenih vrednostima svih njegovih piksela (izraženih kao razlike) na komprimovanom toku. Prednost je da su koordinate bloka mali brojevi (manji od koordinata

piksela), i ove koordinate se moraju napisati samo jednom za ceo blok. Sa druge strane, vrednosti svih piksela u bloku, čak i onih koji se nisu promenili, moraju biti napisane na izlazu. Međutim, pošto su ove vrednosti izražene kao razlike, one jesu mali brojevi. Shodno tome, ovaj metod je osetljiv na veličinu bloka.

Kompenzacija pokreta: Svako ko je gledao filmove zna da je razlika između uzastopnih kadrova mala jer je rezultat pomeranja scene, kamera, ili oboje između kadrova. Ova funkcija se stoga može iskoristiti da bi se poboljšala kompresija. Ako enkoder otkrije da je deo P prethodnog okvira bio kruto pomeren na drugu lokaciju u trenutnom okviru, onda P može biti komprimovan pisanjem sledeće tri stavke u komprimovani tok: njegova prethodna lokacija, njegovu trenutnu lokaciju i informacije koje identifikuju granice P. U principu, takav deo može imati bilo koji oblik. U praksi smo ograničeni na blokove jednake veličine (obično kvadratni, ali mogu biti i pravougaoni). Koder skenira trenutni okvir blok po blok. Za svaki blok B pretražuje prethodni okvir za identičan blok C (ako kompresija treba da bude bez gubitaka) ili za sličan (ako može da bude sa gubicima). Pronalaskom takvog bloka, koder upisuje razliku između njegovih prošlih i sadašnjih lokacija na izlazu. Ova razlika je oblika

$$(Cx - Bx, Cy - By) = (\Delta x, \Delta y),$$

pa se naziva vektor kretanja.
Slika 6.10a,b prikazuje jednostavan primer gde se sunce i drveće se pomera kruto udesno (zbog kretanja kamere) dok se dete pomera za drugačiju distancu ulevo (ovo je kretanje scene).

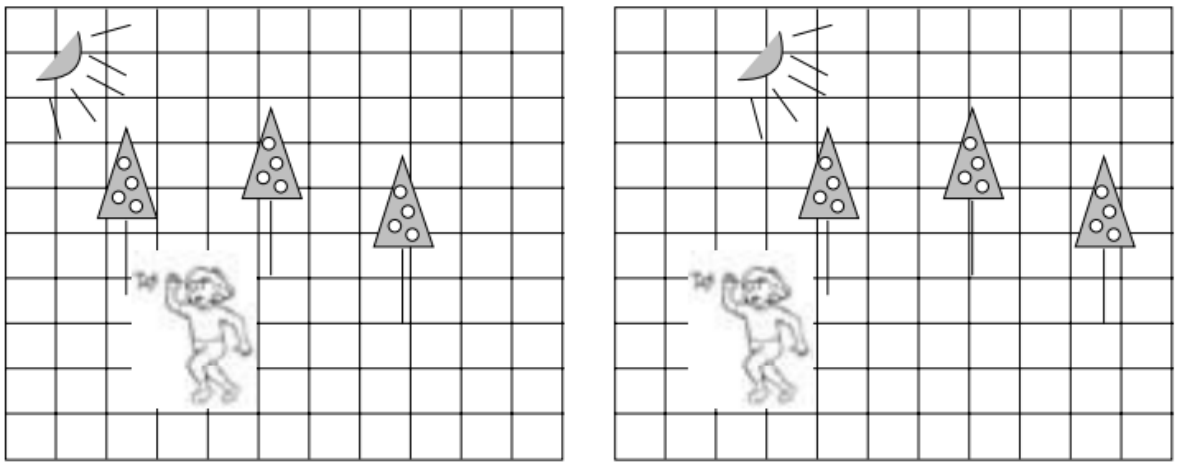


Figure 6.10: Motion Compensation.

Kompenzacija pokreta je efikasna ako su objekti samo prevedeni, a ne skalirani ili rotirani. Drastične promene u osvetljenju od okvira do okvira takođe smanjuju efikasnost ove metode. Generalno, kompenzacija pokreta je sa gubicima. U sledećim paragrafima se govori o glavnim aspektima kompenzacije pokreta u detalje.

Segmentacija okvira: Trenutni okvir je podeljen na ping blokove jednake veličine koji se ne preklapaju. Blokovi mogu biti kvadratni ili pravougaoni. Poslednji izbor pretpostavlja da je kretanje u videu uglavnom horizontalno, tako da horizontalni blokovi smanjuju broj pokreta vektora bez degradacije kompresije. Veličina bloka je važna, pošto je velika blokovi smanjuju šansu za pronalaženje podudaranja, a mali blokovi rezultiraju mnogim pokretima vektora. U praksi se koriste veličine blokova koji su celobrojni stepena 2, kao što su 8 ili 16, pošto ovo pojednostavljuje softver.
Prag pretrage: Svaki blok B u trenutnom okviru se prvo upoređuje sa svojim duplikatom C u prethodnom okviru. Ako su identični, ili ako je razlika između manja od unapred podešenog praga, enkoder pretpostavlja da blok nije pomeren.
Pretraga blokova: Ovo je dugotrajan proces i zato mora biti pažljivo potpisan. Ako je B trenutni blok u trenutnom okviru, onda prethodni okvir mora biti traži blok identičan ili veoma blizak B. Pretraga je obično ograničena na malo područje (nazvano područje pretrage) oko B, definisano maksimalnim pomeranjem parametri dx i dy. Ovi parametri određuju maksimalnu horizontalna i vertikalna rastojanja, u pikselima, između B i bilo kog odgovarajućeg bloka u prethodnom okviru. Ako je B kvadrat sa stranom b, oblast pretrage će sadržati (b + 2dx)(b + 2dy) piksela (slika 6.11) i sastojaće se od (2dx + 1)(2dy + 1) različitih, preklapajućih b×b kvadrata. Broj kandidatskih blokova u ovoj oblasti je stoga proporcionalan dx·dy.

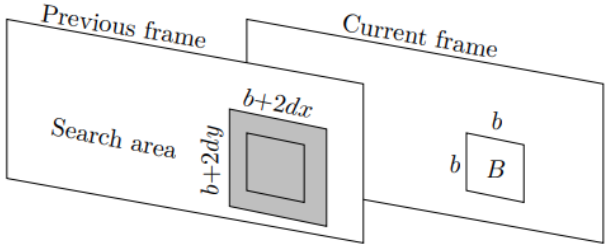


Figure 6.11: Search Area.

Srednja apsolutna razlika (ili srednja apsolutna greška) izračunava prosek vrednosti apsolutne razlike između piksela Bij u B i njegovog dvojnika Cij u kandidatu blok C:

$$\frac{1}{b^2}\sum_{i=1}^{b}\sum_{j=1}^{b}|B_{ij}-C_{ij}|.$$

Ovo uključuje b^2 oduzimanja i operacije apsolutne vrednosti, b^2 sabiranja i jedno deljenje. Ova mera se izračunava za svaki od (2dx + 1)(2dz + 1) različitih, preklapajućih b×b blokova kandidata, a ispituje se najmanja distorzija (recimo za blok Ck). Ako je manji od praga pretrage, tada je Ck izabran kao podudaranje za B. U suprotnom, nema podudaranja za B, a B mora biti kodiran bez kompenzacije kretanja. Prirodno pitanje u ovom trenutku je kako se tako nešto može dogoditi? Kako može bloku u trenutnom okviru da ne odgovara ništa u prethodnom okviru? Odgovor je da zamislite pomeranje kamere s leva na desno. Novi objekti će ući u vidno polje sa desne strane sve vreme. Blok na desnoj strani okvira može tako da sadrži objekte koji ne postoji u prethodnom okviru.

Suboptimalne metode pretrage: Ove metode pretražuju neke, umesto svih blokova kandidata u (b+2dk)(b+2di) oblasti. Oni ubrzavaju potragu za podudaranjem blokova, na račun efikasnosti kompresije.
Korekcija vektora kretanja: Jednom kada je blok C izabran kao najbolji za B, vektor kretanja se izračunava kao razlika između gornjeg levog ugla C i ono B. Bez obzira na to kako je poklapanje određeno, vektor kretanja može biti pogrešan zbog šuma, lokalnih minimuma u kadru ili zbog algoritma podudaranja koji nije idealan. Moguće je primeniti tehnike izglađivanja na vektore kretanja posle oni su izračunati, u pokušaju da se poboljša podudaranje. Prostorne korelacije na slici sugerišu da vektori kretanja takođe treba da budu u korelaciji. Ako se utvrdi da određeni vektori ovo krše, mogu se ispraviti. Ovaj korak je skup i može se čak i vratiti. Video prezentacija može uključivati spore, glatko kretanje većine objekata, ali i brzo, trzavo kretanje nekih malih objekata. Ispravljanje vektora kretanja može ometati vektore kretanja takvih objekata i izazivaju izobličenja u komprimovanim okvirima.
Kodiranje vektora kretanja: Veliki deo trenutnog kadra (možda blizu polovine može se konvertovati u vektore kretanja, tako da je način na koji su ovi vektori kodirani od ključnog značaja; takođe mora biti bez gubitaka. Dva svojstva vektora kretanja pomažu u njihovom kodiranju: (1) Oni su u korelaciji i (2) njihova distribucija je neujednačena. Dok skeniramo blok okvira po bloku, susedni blokovi obično imaju vektore kretanja koji se ne razlikuju mnogo; oni su u korelaciji. Vektori takođe ne pokazuju u svim pravcima. Obično postoje jedan ili dva poželjna pravca u kojima su svi ili većina vektora kretanja usmereni i vektori su tako neujednačeno raspoređeni. Nijedan metod se nije pokazao idealnim za kodiranje vektora kretanja. Aritmetika kodiranja, adaptivno Hafmanovo kodiranje i razni kodovi prefiksa su isprobani, i sve izgleda da se dobro ponaša.
Evo dve različite metode koje mogu biti bolje:
1. Predvidite vektor kretanja na osnovu njegovih prethodnika u istom redu i njegovih prethodnika u istoj koloni trenutnog okvira. Izračunajte razliku između predviđanja i stvarnog vektora, i Hafman ga kodira. Ovaj metod je važan. Koristi se u MPEG i drugim metodama kompresije.
2. Grupišite vektore kretanja u blokove. Ako su svi vektori u bloku identični, blok je kodiran kodiranjem ovog vektora. Ostali blokovi su kodirani kao u 1 iznad. Svaki kodirani blok počinje kodom koji identifikuje njegov tip.

MPEG

Započet 1988. godine, MPEG projekat je razvila grupa od stotina stručnjaka pod pokroviteljstvom ISO (Međunarodne organizacije za standardizaciju) i the IEC (Međunarodni elektrotehnički komitet). Naziv MPEG je akronim za grupu stručnjaka za pokretne slike. MPEG je metoda za video kompresiju, koja uključuje kompresiju digitalne slike i zvuka. Trenutno postoji nekoliko MPEG standarda. MPEG-1 je namenjen za srednje brzine prenosa podataka, reda veličine 1,5 Mbit/s. MPEG-2 je namenjen za visoke brzine prenosa podataka od najmanje 10 Mbit/s. MPEG-3 je bio namenjen za HDTV kompresiju, ali je utvrđeno da jeste suvišan i spojen je sa MPEG-2. MPEG-4 je namenjen za veoma niske brzine prenosa podataka manje od 64 Kbit/s. Treće međunarodno telo, ITU-T, je uključeno u dizajn i MPEG-2 i MPEG-4. Očigledna je važnost široko prihvaćenog standarda za video kompresiju iz činjenice da mnogi proizvođači (kompjuterskih igara, CD-ROM filmova, digital televizija, i digitalni snimači, između ostalih) implementirali i počeli da koriste MPEG-1 čak i pre nego što je konačno odobren od strane MPEG komiteta. Ovo je takođe bio jedan od razloga zašto je MPEG-1 morao da bude zamrznut u ranoj fazi, a MPEG-2 da se razvije prilagođavajući video aplikacije sa visokim brzinama prenosa podataka. MPEG-2 je, s druge strane, bio posebno dizajniran za digitalnu televiziju i ovaj proizvod je imao ogroman komercijalni uspeh.
H.262[2] ili MPEG-2 je format video kodiranja standardizovan i zajednički održavaju ITU-T Studi Group 16 Video Coding Ekperts Group (VCEG) i ISO/IEC Moving Picture Ekperts Group (MPEG), a razvijen je uz učešće mnogih kompanija.To je drugi deo standarda ISO/IEC MPEG-2.

MPEG-2 video je veoma sličan MPEG-1, ali takođe pruža podršku za prepleteni video (tehnika kodiranja koja se koristi u analognim NTSC, PAL i SECAM televizijskim sistemima). MPEG-2 video nije optimizovan za niske brzine prenosa (npr. manje od 1 Mbit/s), ali donekle nadmašuje MPEG-1 pri većim brzinama (npr. 3 Mbit/s i više), iako ne u velikoj meri osim ako video nije isprepleten. Svi MPEG-2 video dekoderi koji su u skladu sa standardima su takođe u potpunosti sposobni za reprodukciju MPEG-1 video tokova.

Semplovanje slike
HDTV kamera sa 8-bitnim uzorkovanjem generiše sirovi video tok od 25 × 1920 × 1080 × 3 = 155 520 000 bajtova u sekundi za video zapis od 25 frejmova u sekundi (koristeći format uzorkovanja 4:4:4). Ovaj tok podataka mora biti komprimovan ako digitalna TV stane u propusni opseg dostupnih TV kanala i ako filmovi trebaju stati na DVD. Video kompresija je praktična jer su podaci na slikama često suvišni u prostoru i vremenu. Na primer, nebo može biti plavo preko vrha slike i to plavo nebo može da se zadrži za kadar za kadrom. Takođe, zbog načina na koji oko funkcioniše, moguće je izbrisati ili aproksimirati neke podatke iz video slika sa malo ili nimalo primetne degradacije u kvalitetu slike.

Uobičajeni (i stari) trik za smanjenje količine podataka je da se svaki kompletan „okvir“ videa odvoji na dva „polja“ nakon emitovanja/kodiranja: „gornje polje“, koje su horizontalne linije neparnog broja, i „ donje polje", a to su parne linije. Nakon prijema/dekodiranja, dva polja se prikazuju naizmenično sa redovima jednog polja koji se prepliću između redova prethodnog polja; ovaj format se naziva “interlaced video“. Tipična brzina polja je 50 (Evropa/PAL) ili 59,94 (US/NTSC) polja u sekundi, što odgovara 25 (Evropa/PAL) ili 29,97 (Severna Amerika/NTSC) celih frejmova u sekundi. Ako video nije isprepleten, onda se naziva video s progresivnim

skeniranjem i svaka slika je kompletan okvir. MPEG-2 podržava obe opcije.

Digitalna televizija zahteva da ove slike budu digitalizovane kako bi se mogle obraditi kompjuterskim hardverom. Svaki element slike (piksel) je tada predstavljen jednim brojem luma i dva broja boje. Oni opisuju osvetljenost i boju piksela. Dakle, svaka digitalizovana slika je u početku predstavljena sa tri pravougaona niza brojeva.

Još jedna uobičajena praksa za smanjenje količine podataka koji se obrađuju je poduzorkovanje dve ravni hroma (posle niskopropusnog filtriranja da bi se izbegao aliasing). Ovo funkcioniše zato što ljudski vizuelni sistem bolje rešava detalje svetline nego detalje u nijansi i zasićenosti boja. Izraz 4:2:2 se koristi za video sa subsemplovanim hromom u odnosu 2:1 po horizontali, a 4:2:0 se koristi za video sa hroma subsemplovanim 2:1 i vertikalno i horizontalno. Video koji ima jačinu svetlosti i boje u istoj rezoluciji naziva se 4:4:4. MPEG-2 Video dokument razmatra sva tri tipa uzorkovanja, iako je 4:2:0 daleko najčešći za video za potrošače, a ne postoje definisani „profili“ MPEG-2 za video 4:4:4 (pogledajte ispod za dalju diskusiju o profilima).
Osim karakteristika za rukovanje poljima za isprepletano kodiranje, MPEG-2 video je veoma sličan MPEG-1 video (pa čak i prilično sličan ranijem standardu H.261).

Analiza video frejmova

Prvo se kreira grafički korisnički interfejs (GUI) za analizu video frejmova koristeći dugmad i panele. Zatim se kreira objekat za odabir fajlova. Nakon toga definišemo varijable za obradu video frejmova, korišćenjem FFmpegFrameGrabber i lista za čuvanje različitih vrsta frejmova (I, P, B frejmovi).

```
FFmpegFrameGrabber rawVideo;
ArrayList<BufferedImage> videoFrames = new ArrayList<BufferedImage>();
ArrayList<BufferedImage> videoIFrames = new ArrayList<BufferedImage>();
ArrayList<BufferedImage> videoPFrames = new ArrayList<BufferedImage>();
ArrayList<BufferedImage> videoBFrames = new ArrayList<BufferedImage>();
```

Slika 1.

YUVI i Chroma Frejmovi: Lista koja sadrži (Y, U, V) vrednosti za svaki frejm, kao i 4:2:0 Chroma uzorkovanje.

```
ArrayList<ArrayList<int[]>> YUVIFrames = new ArrayList<ArrayList<int[]>>();
ArrayList<ArrayList<int[]>> YUVPFrames = new ArrayList<ArrayList<int[]>>();
ArrayList<ArrayList<int[]>> YUVBFrames = new ArrayList<ArrayList<int[]>>();

ArrayList<ArrayList<int[]>> ChromaIFrames = new ArrayList<ArrayList<int[]>>();
ArrayList<ArrayList<int[]>> ChromaPFrames = new ArrayList<ArrayList<int[]>>();
ArrayList<ArrayList<int[]>> ChromaBFrames = new ArrayList<ArrayList<int[]>>();
```

Slika 2.

Blokirani, Predviđeni, Rezidualni i Transformisani Frejmovi: Liste koje sadrže frejmove podeljene u blokove, predviđene frejmove, rezidualne frejmove i transformisane frejmove.

```
ArrayList<ArrayList<ArrayList<int[][]>>> BlockerIFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> BlockerPFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> BlockerBFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();

ArrayList<ArrayList<ArrayList<int[][]>>> PredictedIFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> PredictedBFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();

ArrayList<ArrayList<ArrayList<int[][]>>> ResidualIFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> ResidualPFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> ResidualBFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();

ArrayList<ArrayList<ArrayList<int[][]>>> IntegerTransformIFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> IntegerTransformPFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> IntegerTransformBFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();

ArrayList<ArrayList<ArrayList<int[][]>>> IntegerInverseTransformIFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> IntegerInverseTransformPFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
ArrayList<ArrayList<ArrayList<int[][]>>> IntegerInverseTransformBFrames = new ArrayList<ArrayList<ArrayList<int[][]>>>();
```

Slika 3.

Nakon svega ovoga definišemo svojstva videa: širinu, visinu, vrednosti Y, U, V komponenti i ulazne vrednosti, zatim podesimo i rastojanje između I-frejmova, maksimalan broj frejmova i faktor kvaliteta.
Na kraju se kreira glavni panel (totalGUI) i pod-panel (buttonGUI), postavlja raspored i dodaje dugmad sa odgovarajućim akcijama.

U našem GUI-ju prvo imamo dugme za otvaranje fajla (Slika 4.), a zatim još tri dugmeta:

- Dugme za promenu QP vrednosti za I-frame
- Dugme za promenu QP vrednosti za P-frame
- Dugme za promenu QP vrednosti za B-frame

Na Slici 5. vidimo način implementiranja dugmića za promenu vrednosti frejmova.

```
if (evnt.getSource() == buttonOpen) {
    int result = m_fc.showOpenDialog(totalGUI);
    if (result == JFileChooser.APPROVE_OPTION) {
        File file = m_fc.getSelectedFile();
        String filename = file.toString();
        rawVideo = new FFmpegFrameGrabber(filename);
        reset();
        extractFrames();
        separateFrames();
        JOptionPane.showMessageDialog(totalGUI,
                "File Successfully Opened.",
                "Success",
                JOptionPane.PLAIN_MESSAGE);
    }
}
```

Slika 4.

```
else if (evnt.getSource() == buttonQP) {
    String inputStringQP =
    JOptionPane.showInputDialog(null, "ENTER NEW QP IFRAME VALUE");
    QP = Integer.parseInt(inputStringQP);
    buttonQP.setText("QP I-VALUE=" + QP);
    buttonQPP.setText("QP P-VALUE=" + QP);
    buttonQPB.setText("QP B-VALUE=" + QP);
    IntegerTransformIFrames.clear();
    IntegerInverseTransformIFrames.clear();
    IFrameTransform();
    test();
    totalGUI.revalidate();
    totalGUI.repaint();
}
```

Slika 5.

Zatim slede 3 dugmeta za izbor okvira (Slika 6.), kao i 3 dugmeta za transformaciju okvira (Slika 7):

```
else if (evnt.getSource() == buttonCurrentIFrame)
{
    String inputStringFrameNum = JOptionPane.showInputDialog
    (null, "ENTER I-FRAME NUMBER BETWEEN 0 - " + (videoIFrames.size() - 1));
    FRAME_NUM = Integer.parseInt(inputStringFrameNum);
    buttonCurrentIFrame.setText("I-FRAME SELECTED = " + FRAME_NUM);
    buttonCurrentPFrame.setText("P-FRAME SELECTED = " + FRAME_NUM);
    buttonCurrentBFrame.setText("B-FRAME SELECTED = " + FRAME_NUM);
    test();
    totalGUI.revalidate();
    totalGUI.repaint();
}
```

Slika 6.

```
else if (evnt.getSource() == buttonIFrame) {
    resetAll();
    convertYUV(0);
    subSampling(YUVIFrames, 0);
    for (int i = 0; i < ChromaIFrames.size(); i++) {
        BlockerIFrames.add(blocker(ChromaIFrames.get(i), 4));
    }
    // Dalji proces transformacije i predikcije za I-frameove
}
```

Slika 7.

Implementacija funkcija

Implementacija transformisanja frejmova: iteriraju kroz okvire, komponente i blokove, primenjuju transformacije, čuvaju rezultate i na kraju primenjuju inverzne transformacije. Razlika je samo u tipu okvira koji se obrađuju (I, P ili B).

```
public void IFrameTransform() {
        for(int i=0; i < ResidualIFrames.size(); i++) {
            ArrayList<ArrayList<int[][]>> currentFrame = new ArrayList<ArrayList<int[][]>>();

            for(int j=0; j < ResidualIFrames.get(i).size(); j++) {

                ArrayList<int[][]> Yres = new ArrayList<int[][]>();
                ArrayList<int[][]> Ures = new ArrayList<int[][]>();
                ArrayList<int[][]> Vres = new ArrayList<int[][]>();

                for(int k=0; k < ResidualIFrames.get(i).get(j).size(); k++) {
                    int[][] blockTransformed = integerTransform(ResidualIFrames.get(i).get(j).get(k), QP);

                    if(j == 0) {
                        Yres.add(blockTransformed);
                    }
                    else if(j == 1) {
                        Ures.add(blockTransformed);
                    }
                    else {
                        Vres.add(blockTransformed);
                    }
                }
                if(j == 0) {
                    currentFrame.add(Yres);
                }
                else if(j == 1) {
                    currentFrame.add(Ures);
                }
                else {
                    currentFrame.add(Vres);
                }
            }
            IntegerTransformIFrames.add(currentFrame);
        }
    for(int i=0; i < IntegerTransformIFrames.size(); i++) {
        ArrayList<ArrayList<int[][]>> currentFrame = new ArrayList<ArrayList<int[][]>>();

        for(int j=0; j < IntegerTransformIFrames.get(i).size(); j++) {

            ArrayList<int[][]> Yres = new ArrayList<int[][]>();
            ArrayList<int[][]> Ures = new ArrayList<int[][]>();
            ArrayList<int[][]> Vres = new ArrayList<int[][]>();

            for(int k=0; k < IntegerTransformIFrames.get(i).get(j).size(); k++) {

                int[][] blockTransformed = inverseIntegerTransform(IntegerTransformIFrames.get(i).get(j).get(k), QP);

                if(j == 0) {
                    Yres.add(blockTransformed);
                }
                else if(j == 1) {
                    Ures.add(blockTransformed);
                }
                else {
                    Vres.add(blockTransformed);
                }
            }

            if(j == 0) {
                currentFrame.add(Yres);
            }
            else if(j == 1) {
                currentFrame.add(Ures);
            }
            else {
                currentFrame.add(Vres);
            }
        }

        IntegerInverseTransformIFrames.add(currentFrame);
    }
}
```

Slika 8.

Ekstraktovanje: Ekstraktuje prvih 50 okvira iz video zapisa i smešta ih u ArrayList objekata BufferedImage. (Slika 10.)

```
public void extractFrames() {
        try {
            rawVideo.start();

            //Grabs the first 50 frames of the video
            for (int i = 0 ; i < MAX_FRAMES ; i++) {

                //Creates a buffered image from the video frame
                BufferedImage currentFrame =
                new Java2DFrameConverter().convert(rawVideo.grabImage());
                //Place the current frame into an array list
                videoFrames.add(currentFrame);

            }

            rawVideo.stop();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Slika 10.

Separacija: Razdvaja ekstraktovane okvire i sortira ih u I, P, i B okvire za dalju obradu(Slika 11.)

```
public void separateFrames() {

        //IBBPBBPBBI
        int Bframe = 0;

        for(int i=0; i < videoFrames.size(); i++) {
            if(i % IframeDistance == 0 || i == 0) {
                videoIFrames.add(videoFrames.get(i));
                Bframe = 1;
            }
            else if(Bframe == 1 || Bframe == 2) {
                videoBFrames.add(videoFrames.get(i));
                Bframe++;
            }
            else {
                videoPFrames.add(videoFrames.get(i));
                Bframe = 1;
            }

        }
    }
```

Slika 11.

Konvertovanje: Konvertuje okvire u YUV format i smešta ih u odgovarajuće liste(Slika 12.)

```
public void convertYUV(int currentMethod) {
        ArrayList<BufferedImage> videoFramesTemp = new ArrayList<BufferedImage>();

        if(currentMethod == 0) {
            videoFramesTemp = videoIFrames;
        }
        else if(currentMethod == 1) {
            videoFramesTemp = videoPFrames;
        }
        else {
            videoFramesTemp = videoBFrames;
        }

        for(int i = 0; i < videoFramesTemp.size(); i++) {
            width = videoFramesTemp.get(i).getWidth(null);
            height = videoFramesTemp.get(i).getHeight(null);

            ArrayList<int[]> currentYUVFrame = new ArrayList<int[]>();
            inputValues = new int[width*height];
            YValues = new int[width*height];
            UValues = new int[width*height];
            VValues = new int[width*height];

            PixelGrabber grabber =
            new PixelGrabber(videoFramesTemp.get(i).getSource(), 0, 0, width, height, inputValues, 0, width);
            try{
                if(grabber.grabPixels() != true){
                    try {
                        throw new AWTException("Grabber returned false: " + grabber.status());
                    } catch (AWTException e) {
                        e.printStackTrace();
                    };
                }
            } catch (InterruptedException e) {};

            for (int index = 0; index < height * width; ++index)
            {
                int red = ((inputValues[index] & 0x00ff0000) >> 16);
                int green =((inputValues[index] & 0x0000ff00) >> 8);
                int blue = ((inputValues[index] & 0x000000ff) );
                YValues[index] = (int)((0.299 * (float)red) + (0.587 * (float)green) + (0.114 * (float)blue));
                UValues[index] = (int)((-0.14713 * (float)red) + (-0.28886 * (float)green) + (0.436 * (float)blue));
                VValues[index] = (int)((0.615 * (float)red) + (-0.51499 * (float)green) + (-0.10001 * (float)blue));
            }

            currentYUVFrame.add(YValues);
            currentYUVFrame.add(UValues);
            currentYUVFrame.add(VValues);

            if(currentMethod == 0) {
                // Add the YUV array list to a global array list containing all frames
                YUVIFrames.add(currentYUVFrame);
            }
            else if(currentMethod == 1) {
                // Add the YUV array list to a global array list containing all frames
                YUVPFrames.add(currentYUVFrame);
            }
            else {
                // Add the YUV array list to a global array list containing all frames
                YUVBFrames.add(currentYUVFrame);
            }
        }
    }
```

Slika 12.

Subsemplovanje: Ova funkcija vrši poduzorkovanje (subsampling) U i V komponenti YUV okvira, smanjujući količinu podataka za kromu (boju), dok ostavlja Y (osvetljenost) komponentu nepromenjenom. Ovo je poznato kao Chroma Subsampling i često se koristi u video kompresiji da bi se smanjila veličina podataka, jer ljudsko oko manje primećuje detalje boje u poređenju sa detaljima osvetljenosti(Slika 13.)

- Priprema nove dimenzije (širina i visina) za poduzorkovane okvire.
- Prolazi kroz svaki okvir i inicijalizuje nizove za U i V komponente.
- Kopira svaku drugu U i V vrednost u 2x2 blokove za poduzorkovane okvire.
- Dodaje rezultate u odgovarajuće globalne liste na osnovu tipa okvira (I, P ili B).

```
public void subSampling(ArrayList<ArrayList<int[]>> inputYUVFrames, int currentMethod) {
        int newWidth = ((int) Math.floor(width/4)) * 4;
        int newHeight = ((int) Math.floor(height/4)) * 4;

        for(int i=0; i<inputYUVFrames.size(); i++) {
            ArrayList<int[]> outputRes = new ArrayList<int[]>();

            // set UV values for chroma use
            int[] UChroma = new int[newWidth*newHeight];
            int[] VChroma = new int[newWidth*newHeight];

            // adding every other U and V value to a block of 4
            for (int y = 1; y < newHeight; y+=2)
            {
                for (int x = 1; x < newWidth; x+=2)
                {
                    UChroma[((y - 1)*newWidth + (x - 1))] = (inputYUVFrames.get(i).get(1)[(y - 1)*newWidth + (x - 1)]);
                    UChroma[((y - 1)*newWidth + x)] = (inputYUVFrames.get(i).get(1)[(y - 1)*newWidth + (x - 1)]);
                    UChroma[(y*newWidth + (x - 1))] = (inputYUVFrames.get(i).get(1)[(y - 1)*newWidth + (x - 1)]);
                    UChroma[(y*newWidth + x)] = (inputYUVFrames.get(i).get(1)[(y - 1)*newWidth + (x - 1)]);

                    VChroma[((y - 1)*newWidth + (x - 1))] = (inputYUVFrames.get(i).get(2)[(y - 1)*newWidth + (x - 1)]);
                    VChroma[((y - 1)*newWidth + x)] = (inputYUVFrames.get(i).get(2)[(y - 1)*newWidth + (x - 1)]);
                    VChroma[(y*newWidth + (x - 1))] = (inputYUVFrames.get(i).get(2)[(y - 1)*newWidth + (x - 1)]);
                    VChroma[(y*newWidth + x)] = (inputYUVFrames.get(i).get(2)[(y - 1)*newWidth + (x - 1)]);
                }
            }

            outputRes.add(inputYUVFrames.get(i).get(0));
            outputRes.add(UChroma);
            outputRes.add(VChroma);

            if(currentMethod == 0) {
                // Add the ChromaSubsampled array list to a global array list containing all frames
                ChromaIFrames.add(outputRes);
            }
            else if(currentMethod == 1) {
                // Add the ChromaSubsampled array list to a global array list containing all frames
                ChromaPFrames.add(outputRes);
            }
            else {
                // Add the ChromaSubsampled array list to a global array list containing all frames
                ChromaBFrames.add(outputRes);
            }

        }
    }
```

Slika 13.

Rastavljanje na blokove: Funkcija blocker deli ulazne YUV okvire na manje blokove određene veličine, npr. 8x8(Slika 14.)

- Inicijalizacija novih dimenzija
- Kreiranje lista za blokove
- Petlja za svaki blok
- Dodavanje blokova u liste

```
public ArrayList<ArrayList<int[][]>> blocker(ArrayList<int[]> frame, int size)
    {
        int newWidth = ((int) Math.floor(width/size)) * size;
        int newHeight = ((int) Math.floor(height/size)) * size;
        int xcount = 0;
        int ycount = 0;
        int blockNum = (newWidth * newHeight) / (size * size);
        ArrayList<ArrayList<int[][]>> res = new ArrayList<ArrayList<int[][]>>();
        ArrayList<int[][]> resY = new ArrayList<int[][]>();
        ArrayList<int[][]> resU = new ArrayList<int[][]>();
        ArrayList<int[][]> resV = new ArrayList<int[][]>();
        int[] frameY = frame.get(0); // get all corresponding YUV values of current frame
        int[] frameU = frame.get(1);
        int[] frameV = frame.get(2);

        for (int x = 0; x < blockNum; x++)
        {
            int[][] blockY = new int[size][size];
            int[][] blockU = new int[size][size];
            int[][] blockV = new int[size][size];

            for (int e = 0; e < size; e++)
            {
                for (int f = 0; f < size; f++)
                {
                    if (xcount % newWidth == 0 && x != 0 && xcount != 0)
                    {
                        ycount += size;
                        xcount = 0;
                    }
                    blockY[e][f] = frameY[(ycount + e) * width + ((x % (newWidth/size)) * size) + f];
                    blockU[e][f] = frameU[(ycount + e) * width + ((x % (newWidth/size)) * size) + f];
                    blockV[e][f] = frameV[(ycount + e) * width + ((x % (newWidth/size)) * size) + f];
                }
                xcount++;
            }
            resY.add(blockY);
            resU.add(blockU);
            resV.add(blockV);
        }

        res.add(resY);
        res.add(resU);
        res.add(resV);

        return res;
    }
```

Slika 14.

Sastavljanje blokova: Funkcija unblocker ponovo sastavlja sliku iz blokova(Slika 15.)

- Inicijalizacija novih dimenzija
- Priprema nizova za povratak
- Petlja za svaki blok

```
public ArrayList<int[]> unblocker(ArrayList<ArrayList<int[][]>> frame) {
        ArrayList<int[]> res = new ArrayList<int[]>();
        ArrayList<int[][]> Yblocks = frame.get(0);
        ArrayList<int[][]> Ublocks = frame.get(1);
        ArrayList<int[][]> Vblocks = frame.get(2);

        int blockDimension = Yblocks.get(0).length;
        int blockSize = blockDimension * blockDimension;
        int newWidth = ((int) Math.floor(width/blockDimension)) * blockDimension;
        int newHeight = ((int) Math.floor(height/blockDimension)) * blockDimension;
        int xcount = 0;
        int count = 0;

        int[] unblockedY = new int[blockSize];
        int[] unblockedU = new int[blockSize];
        int[] unblockedV = new int[blockSize];

        int[] resY = new int[newWidth * newHeight];
        int[] resU = new int[newWidth * newHeight];
        int[] resV = new int[newWidth * newHeight];

        for (int x = 0; x < Yblocks.size(); x++) { // iterate through each block

            unblockedY = flatten2D(Yblocks.get(x));
            unblockedU = flatten2D(Ublocks.get(x));
            unblockedV = flatten2D(Vblocks.get(x));

            for(int y = 0; y < blockSize; y++)
            {
                int remainder = y % blockDimension;

                if (x < newWidth/blockDimension)
                {
                    xcount = (x % (newWidth/blockDimension)) * blockDimension;
                }
                else if (x == newWidth/blockDimension)
                {
                    xcount = x * blockSize;
                }
                else
                {
                    int rowCount = x/(newWidth/blockDimension);
                    xcount = rowCount * newWidth * blockDimension + (x % (newWidth/blockDimension)) * blockDimension;
                }

                if (remainder == 0 && y != 0)
                {
                    count += newWidth;
                }

                int index = count + remainder + xcount;
                if (index > newWidth * newHeight) {
                    break;
                }
                resY[index] = unblockedY[y];
                resU[index] = unblockedU[y];
                resV[index] = unblockedV[y];
            }
            count = 0;
        }
        res.add(resY);
        res.add(resU);
        res.add(resV);
        return res;
    }
```

Slika 15.

Logaritamska pretraga vektora: Ovaj deo koda implementira logaritamsku pretragu za pronalaženje vektora pokreta između dva okvira slike (trenutnog i referentnog okvira). Vektor pokreta predstavlja premik/blokiranje slike koja se koristi za predviđanje trenutnog okvira iz referentnog, čime se smanjuje količina podataka potrebnih za kodiranje(Slika 16. i Slika 17.)

- Inicijalizacija
- Pronalazak početnih pozicija blokova za pretragu
- Logaritamska pretraga
- Vraćanje rezultata

```
public static ArrayList<Integer> motionLogSearch(ArrayList<ArrayList<int[][]>> currentFrame, ArrayList<ArrayList<int[][]>> referenceFrame){
    ArrayList<int[][]> Yblocks = currentFrame.get(0);
    int numBlocks = Yblocks.size(); // get number of Y blocks
    int rowSize = (int) Math.sqrt(numBlocks);
    int p = (int) Math.ceil(Math.sqrt(numBlocks)/2);
    int offset = (int) Math.ceil(p/2);
    ArrayList<int[][]> nineBlocks = new ArrayList<int[][]>();
    ArrayList<int[][]> refBlocks = new ArrayList<int[][]>();
    ArrayList<int[][]> refY = referenceFrame.get(0);
    ArrayList<Integer> result = new ArrayList<Integer>();
    boolean last = false;

    int centerCoord = (int) numBlocks/2 - 1; //find middle point
    double minMAD = 666;
    int minIndex = 0;
    int lastPosition = centerCoord;

    int tl = (int)centerCoord/2 - offset;
    int tm = (int)centerCoord/2;
    int tr = (int)centerCoord/2 + offset;
    int l = centerCoord - offset;
    int m = centerCoord;
    int r = centerCoord + offset;
    int bl = (int)centerCoord/2 + centerCoord - offset;
    int bm = (int)centerCoord/2 + centerCoord;
    int br = (int)centerCoord/2 + centerCoord + offset;
while (last != true) {
    tl = centerCoord - offset * rowSize - offset;
    tm = centerCoord - offset * rowSize;
    tr = centerCoord - offset * rowSize + offset;
    l  = centerCoord - offset;
    m  = centerCoord;
    r  = centerCoord + offset;
    bl = centerCoord + offset * rowSize - offset;
    bm = centerCoord + offset * rowSize;
    br = centerCoord + offset * rowSize + offset;

    int[] positions = new int [] {tl, tm, tr, l, m, r, bl, bm, br};
    System.out.println("offset: tl, tm, tr, l, m, r, bl, bm, br: "
        + offset + " " + tl + " " + tm + " " + tr + " " + l + " " + m + " " + r + " " + bl + " " + bm + " " + br);
    nineBlocks.add(0, Yblocks.get(tl));
    nineBlocks.add(1, Yblocks.get(tm));                         // 1/4 spot
    nineBlocks.add(2, Yblocks.get(tr));
    nineBlocks.add(3, Yblocks.get(l));
    nineBlocks.add(4, Yblocks.get(m));                                          // 1/2 spot
    nineBlocks.add(5, Yblocks.get(r));
    nineBlocks.add(6, Yblocks.get(bl));
    nineBlocks.add(7, Yblocks.get(bm));         // 3/4 spot
    nineBlocks.add(8, Yblocks.get(br));

    refBlocks.add(0, refY.get(tl));
    refBlocks.add(1, refY.get(tm));
    refBlocks.add(2, refY.get(tr));
    refBlocks.add(3, refY.get(l));
    refBlocks.add(4, refY.get(m));
    refBlocks.add(5, refY.get(r));
    refBlocks.add(6, refY.get(bl));
    refBlocks.add(7, refY.get(bm));
    refBlocks.add(8, refY.get(br));
```

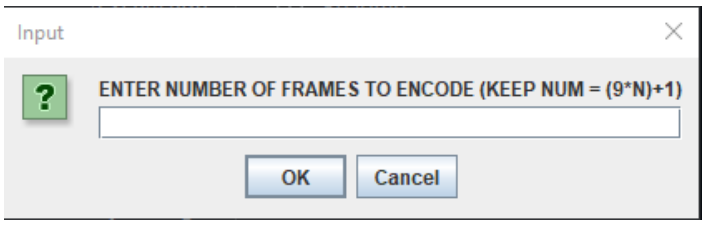
Slika 16.

```
        for (int x = 0; x < 9; x++) {
            double temp = meanAD(nineBlocks.get(x), refBlocks.get(x));
            if (temp < minMAD) {
                minMAD = temp;
                minIndex = x;
            }
        }

        if (offset == 1) {
            last = true;
            lastPosition = positions[minIndex];
        }

        System.out.println("centerCoord: " + centerCoord);
        centerCoord = positions[minIndex];
        offset = (int) Math.ceil(offset/2);

    }
    result = motionVector((int) numBlocks/2, lastPosition, rowSize);
    System.out.println("motionvector result: " + result.get(0) + ", " + result.get(1));

    return result;
}
```

Slika 17.

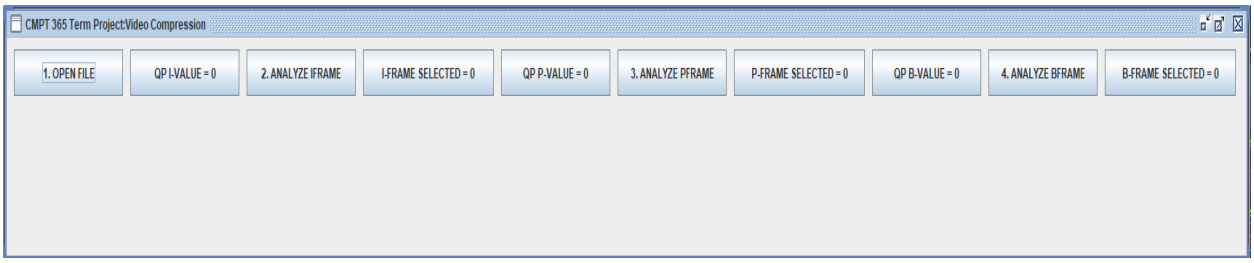
Nakon svih ovih implemmentacija, na kraju dolazimo do uređivanja GUI-ja,postavljanje dugmića, određivanje pozicija na kontrolnom panelu, kao i prikaz kompresije na slikama u svim fazama kompresovanja.
Prilikom pokretanja programma pojaviće se prozor u kom trebamo uneti broj frejmova za enkodiranje koji zadovoljava uslov NUM=(9*N)+1. Izgled prozora je prikazan na slici 18.

GUI



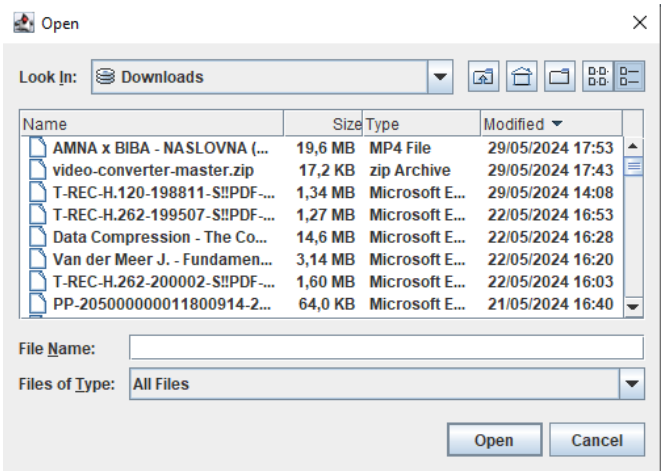
Slika 18.

Na slici 19 je prikazan izgled grafičkog korisničkog interfejsa koji se pojavljuje nakon unosa vrednosti za broj frejmova.



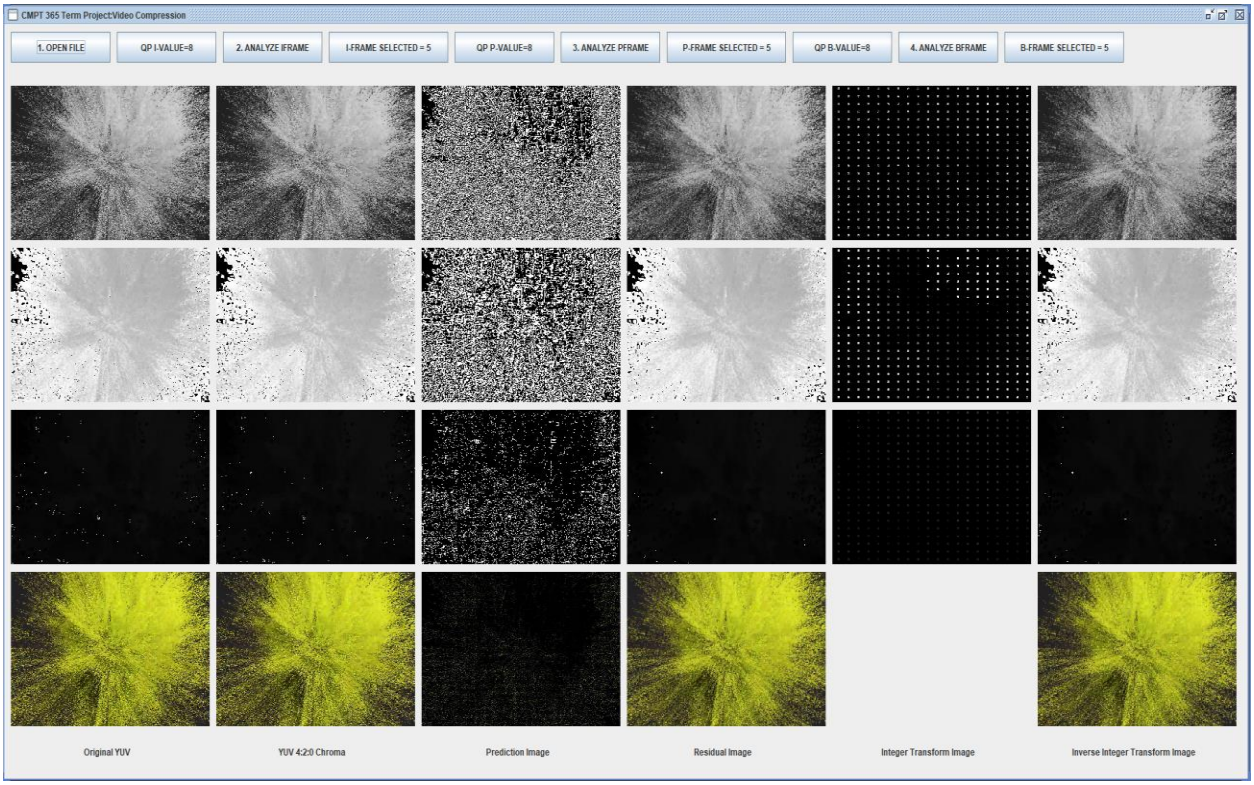
Slika 19.

Potrebno je učitati fajl(video zapis),pritiskom na dugme “OPEN FILE”, koji želimo da obradimo, tj. kompresujemo. Nakon pritiska dugmeta, otvara se prozor, koji prikazuje naš File System, gde želimo naći video zapis koji želimo da obradimo(Slika 20.)



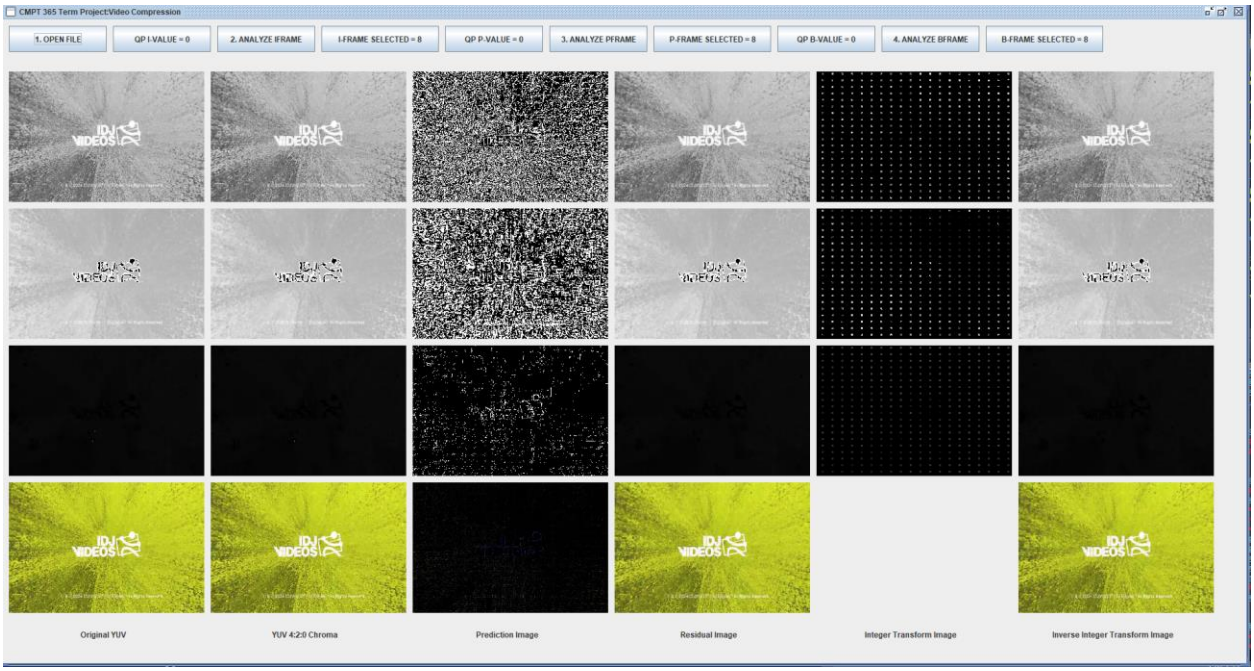
Slika 20.

Ukoliko smo uneli fajl odgovarajućeg formata, pojaviće se prozor sa porukom “File Successfully opened”. Nakon unošenja željenih parametara za vrednosti I, P i B frejmova, na našem ekranu će biti prikazani određeni frejmovi za unešene vrednosti(primer Slika 21.)

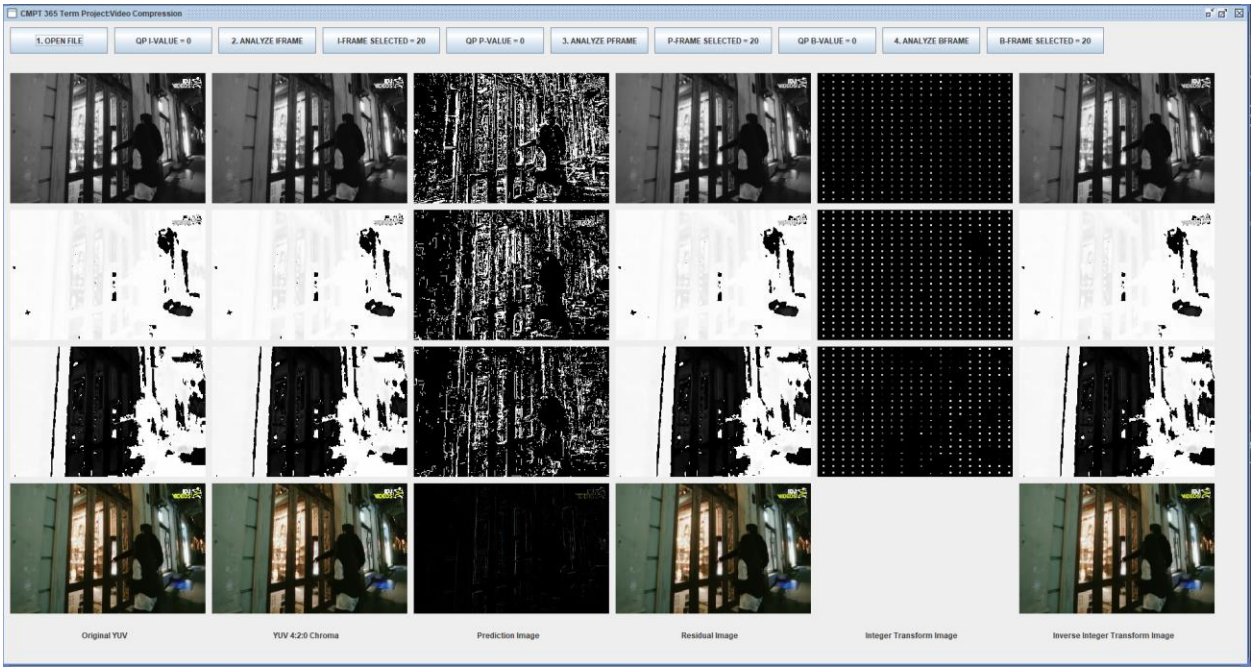


Slika 21.

Za unošenje drugih vrednosti parametara za broj frejmova koje želimo da enkodiramo, a zatim i vrednosti parametara I, P, i B, dobijamo drugačije rezultate obrade(primer Slika 22. i Slika 23.)



Slika 22.



Slika 23.

Zaključak

Kompresija podataka je postala neophodna u digitalnom dobu kako bi se omogućilo efikasno upravljanje sve većim količinama informacija. U tom kontekstu, H.262 standard, poznat i kao MPEG-2, ima značajnu ulogu u oblasti video kompresije. Iako je ovaj standard već dugo prisutan i široko korišćen, postoje određeni izazovi i perspektive koje treba razmotriti.

Jedan od glavnih izazova sa kojima se suočava H.262 standard je sve veća potreba za visokokvalitetnim video sadržajem, naročito u eri HD i Ultra HD rezolucija. S porastom popularnosti platformi za strimovanje video sadržaja kao što su Netflix, YouTube i druge, raste i zahtev za efikasnijim algoritmima kompresije koji mogu očuvati visok nivo kvaliteta slike pri minimalnom gubitku informacija. U ovom kontekstu, noviji standardi kao što su H.264 (AVC) i H.265 (HEVC) donose naprednije tehnike kompresije koje omogućavaju postizanje boljeg balansa između veličine fajla i kvaliteta slike.

Pored toga, H.262 standard se suočava i sa izazovima kompatibilnosti i interoperabilnosti sa novijim tehnologijama i uređajima. Dok su stariji uređaji i infrastruktura možda podržavali H.262, noviji uređaji i softveri često zahtevaju podršku za naprednije standarde. Ovo može predstavljati prepreku u usvajanju H.262 u novim aplikacijama i okruženjima gde je potrebna integracija sa modernim tehnologijama.

Međutim, uprkos ovim izazovima, H.262 i dalje zadržava svoju važnost u određenim sektorima i aplikacijama. Na primer, industrija emitovanja televizijskih programa i proizvodnje DVD-ova i dalje koristi H.262 zbog svoje pouzdanosti i kompatibilnosti sa postojećim sistemima. Takođe, H.262 se može koristiti kao deo hibridnih rešenja gde se kombinuje sa naprednijim standardima kako bi se postigla optimalna ravnoteža između kvaliteta, efikasnosti i kompatibilnosti.

U pogledu budućnosti, moguće je da će se H.262 standard i dalje koristiti u određenim segmentima industrije, ali će se istovremeno nastaviti i prelazak na nove standarde koji nude veću efikasnost i kvalitet. Očekuje se da će dalji razvoj tehnologije dovesti do novih inovacija u oblasti kompresije podataka, što bi moglo dovesti do novih rešenja koja će nadmašiti postojeće standarde.

U zaključku, H.262 standard ostaje relevantan u digitalnom dobu, pružajući efikasno rešenje za određene primene. Dok tehnologija nastavlja da napreduje, važno je pažljivo balansirati između iskorišćavanja naprednih tehnologija i održavanja kompatibilnosti sa postojećim sistemima kako bi se obezbedila optimalna performansa i funkcionalnost u digitalnom svetu.

Literatura

- https://en.wikipedia.org/wiki/H.262/MPEG-2_Part_2
- https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-H.262-200002-S!!PDF-E&type=items
- https://www.loc.gov/preservation/digital/formats/fdd/fdd000028.shtml
- https://github.com/hjiangsu/video-converter
- https://drive.google.com/file/d/18qt5ZIeRwHOOimXVtKVaxSE61M1h0Di6/view
- https://www.researchgate.net/publication/288552060_Fundamentals_and_Evolution_of_MPEG-2_Systems_Paving_the_MPEG_Road
- Shilpa P Metkar, Yogesh H Dandawate, Mehul S. Raval, Madhuri A. Joshi, Kalyani R. Yoshi, IMAGE AND VIDEO COMPRESSION – Fundamentals, Techniques, and Applications