Ubrzavanje kodiranja videa koristeći izvorne slike u Bayerovom formatu

Pavel, Dominik

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:541881

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-14**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 645

UBRZAVANJE KODIRANJA VIDEA KORISTEĆI IZVORNE SLIKE U BAYEROVOM FORMATU

Dominik Pavel

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 645

UBRZAVANJE KODIRANJA VIDEA KORISTEĆI IZVORNE SLIKE U BAYEROVOM FORMATU

Dominik Pavel

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 645

Pristupnik:	**Dominik Pavel (0035212648)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	izv. prof. dr. sc. Daniel Hofman

Zadatak: **Ubrzavanje kodiranja videa koristeći izvorne slike u Bayerovom formatu**

Opis zadatka:

Na ulazu u video koder obično se nalazi niz slika u formatu RGB koji je interpoliran pikselima koji nedostaju i zbog toga ima tri puta veću količinu podataka od izvorne slike sa senzora. Potrebno je proučiti načine i formate kojima kamere pružaju informacije o snimljenoj slici. Proučiti koji od navedenih formata je prikladan za kodiranje videa. Implementirati kôd pomoću kojega će se strujanje podataka u Bayerovom formatu s kamere proslijediti do video kodera. Proučiti koje se transformacije trebaju izvršiti nad podatcima iz kamere i implementirati iste kako bi se pripremili podatci za ulaz u video koder. Proučiti način rada video kodera i na strani dekodera implementirati potrebne transformacije kao što su proces interpolacije i konverzije boja. Testirati implementirano rješenje te usporediti rad s osnovnim video koderom. Napraviti dokumentaciju i objaviti programski kôd na repozitoriju Git.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

1. Uvod

Ovaj rad istražuje način kako ubrzati prijenos slike snimljene u Bayerovom, odnosno RGGB formatu, tako što maskiramo kanale boja u YUV komponente. Također sadrži implementaciju sustava za prikupljanje i obradu slikovnih podataka s ASI ZWO kamere korištenjem C++ programskog jezika i različitih biblioteka. Glavni cilj ovog rada je razviti učinkovit i pouzdan sustav za prikupljanje, obradu i prikaz slikovnih podataka u realnom vremenu.

Kamera koristi ASICamera2 SDK za povezivanje i inicijalizaciju, dok se OpenCV biblioteka koristi za prikaz slike tijekom snimanja i prikazivanje nakon dekodiranja. FFmpeg se koristi za kodiranje i dekodiranje videa koristeći H.265 (HEVC) kodek, omogućujući visoku kvalitetu slike uz učinkovitu kompresiju. Također se može koristiti FFmpeg-ov FFmplay za prikaz YUV444 i YUV422 videa.

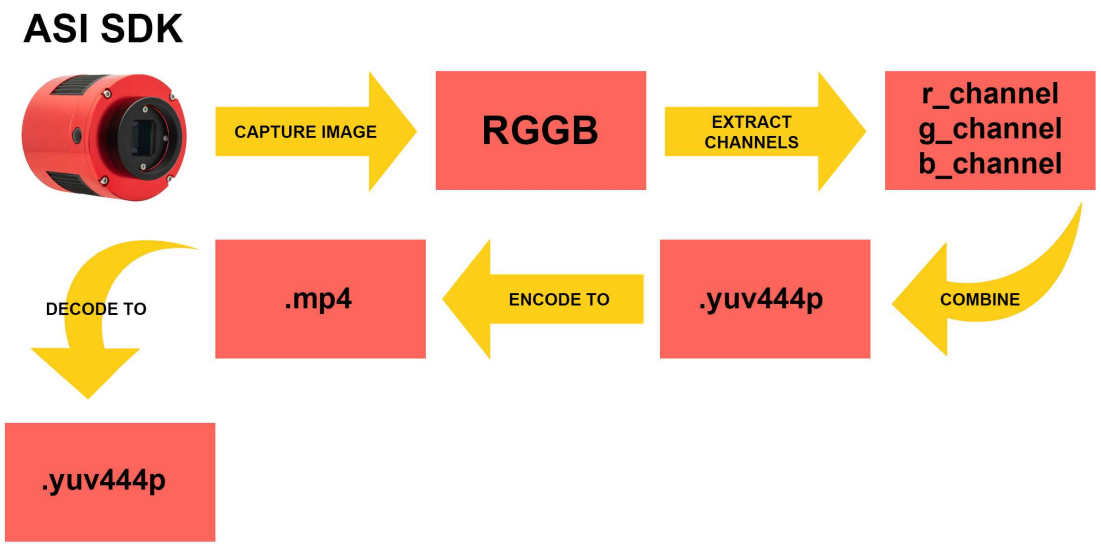
Rad je strukturiran u nekoliko ključnih poglavlja. Nakon pregleda toka podataka, podataka kamere i formata, formata i standardnih video kodiranja, u radu se opisuje dobivanje podataka s kamere, zatim obrada slike, nakon čega slijede enkodiranja i dekodiranja, te cijeli praktični dio završava spremanjem finalnih videa i uspoređivanjem rezultata.

Nadalje, detaljno je opisana arhitektura rješenja koja se temelji na modularnom pristupu. Ovaj pristup osigurava lakoću adaptacije programa kao i čitljivosti koda. Rad također sadrži pregled korištenih alata i tehnologija, objašnjavajući kako svaki alat doprinosi ukupnoj funkcionalnosti sustava. Na samom kraju uspoređujemo rezultate snimanih videa i radimo analizu.

2. Pregled toka podataka

2.1. Pregled toka podataka za normalno kodiranje

Slika 2.1 pokazuje proces normalne obrade podataka dobivenih iz kamere pomoću ASI SDK-a preuzetog s njihove web-stranice [3]. Proces se sastoji od pet glavnih koraka: prikupljanje RGGB podataka, ekstrakcija piksela, pretvaranje u YUV444p format, kodiranje podataka u MP4 koristeći FFmpeg i u konačnici dekodiranje nazad u YUV444p koristeći FFmpeg.



Slika 2.1. Pregled toka podataka normalnog videa

2.1.1. Slike s kamere u Bayerovom formatu

Proces počinje s kamerom, što je u ovom slučaju ASI kamera. Koristi se ASI SDK za dobivanje slikovnih podataka u RGGB formatu u kodu. U Bayerovom formatu, pikseli su raspoređeni u mrežu crvenih (R - red), zelenih (G - green) i plavih (B - blue) filtara. "RGGB" specificira određeni uzorak u kojem su raspoređene boje: crvena, zelena, zelena, plava, gdje je zelenih više nego ostalih radi osjetljivosti ljudskog oka na zelenu boju.

2.1.2. Ekstrakcija crvenih, zelenih i plavih piksela iz slike

Nakon što dobijemo RGGB sliku, slijedi ekstrakcija crvenih, zelenih i plavih kanala piksela iz same slike. Ti kanali će biti znatno manje veličine nego prava slika stoga je potrebno napravit upsampling, odnosno popuniti rupe koje nastanu kada odvajamo kanale. Pikseli su zapisani kao R G G B, što znači da će crveni kanal biti četvrtina potpune veličine, zeleni kanal će biti pola potpune veličine dok će plavi kao i crveni biti četvrtina potpune veličine.

2.1.3. Konverzija u YUV444 format

Jednom kada su izvučeni kanali piksela iz slike, ta polja piksela su razmjerno uvećana koristeći okolne piksele pomoću kojih se računa prosjek i taj prosjek se koristi za popunjavanje informacija koje nedostaju. Nakon toga će kanali biti potpune veličine i kao takvi se šalju direktno na FFmpeg enkoder.

2.1.4. Kodiranje pomoću FFmpeg

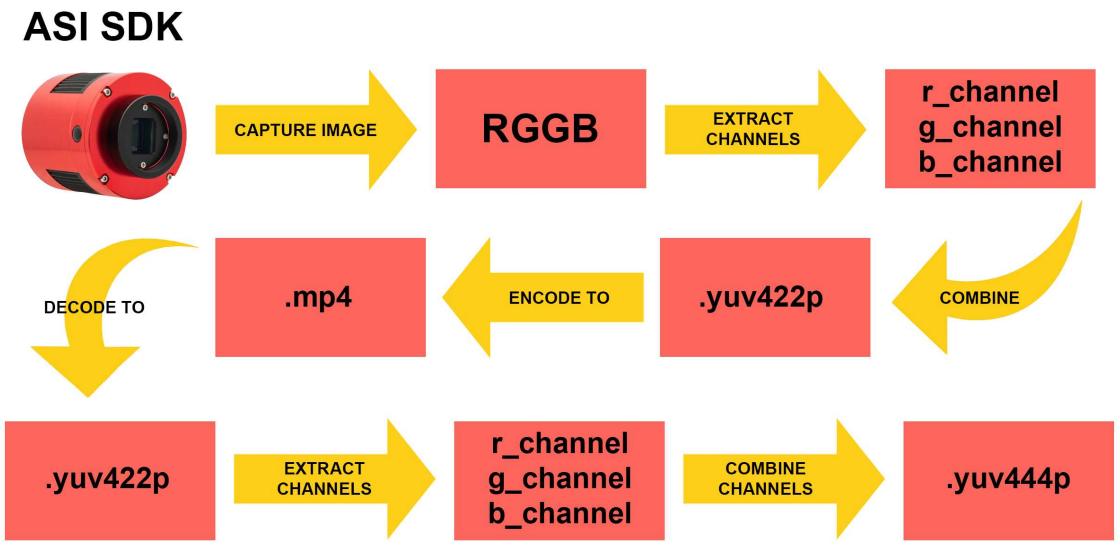
Sljedeći korak u procesu je kodiranje YUV slike. Ovaj korak se izvodi pomoću FFmpeg enkodera. Proces kodiranja komprimira slikovne podatke, čineći ih prikladnima za pohranu ili prijenos. Svrha kodiranja je pretvoriti podatke u format kojim je lako upravljati i koji je učinkovit za kasniju upotrebu, kao što je reprodukcija videa ili daljnja obrada slike, a za svrhu ovog rada koristiti će se za usporedbu s kodiranjem bez demozaika.

2.1.5. Dekodiranje pomoću FFmpeg

Sljedeći kao i zadnji korak u procesu je dekodiranje MP4 videa. Ovaj korak se izvodi pomoću FFmpeg dekodera. Proces dekodiranja pretvara komprimirane video podatke u nekomprimirani format, omogućujući njihovu reprodukciju ili daljnju obradu. Svrha dekodiranja je pretvoriti podatke u format kojim je lako upravljati i koji je prikladan za analizu ili daljnje obrade, kao što je konverzija u drugi format ili prilagodba specifičnim zahtjevima projekta. U ovom slučaju, dekodirani podaci će se koristiti za pretvorbu u YUV444 format, što će omogućiti usporedbu s drugim formatima ili tehnikama obrade slike.

2.2. Pregled toka podataka za maskirano kodiranje

Slika 2.2 pokazuje proces maskirane (ubrzane) obrade podataka dobivenih iz kamere pomoću ASI SDK-a preuzetog s njihove web-stranice. Proces se sastoji od sedam glavnih koraka: prikupljanje RGGB podataka, ekstrakcija piksela, pretvaranje u YUV422p format, kodiranje podataka u MP4 format koristeći FFmpeg, dekodiranje nazad u YUV422p koristeći FFmpeg, ekstrakcija crvenih, zelenih i plavih kanala te u konačnici spajanje tih kanala u finala YUV444p video.



Slika 2.2. Pregled toka podataka maskiranog videa

2.2.1. Slike s kamere u Bayerovom formatu

Proces počinje s kamerom, što je u ovom slučaju ASI kamera. Koristi se ASI SDK za dobivanje slikovnih podataka u RGGB formatu u kodu. U Bayerovom formatu, pikseli su raspoređeni u mrežu crvenih (R - red), zelenih (G - green) i plavih (B - blue) filtara. "RGGB" specificira određeni uzorak u kojem su raspoređene boje: crvena, zelena, zelena, plava, gdje je zelenih više nego ostalih radi osjetljivosti ljudskog oka na zelenu boju.

2.2.2. Ekstrakcija crvenih, zelenih i plavih piksela iz slike

Identično kao i u 2.1.2 poglavlju, iz slike dimenzija visina * širina izvlačimo redove piksela. Redoslijed piksela je R G G B, što znači da će crveni kanal biti četvrtina potpune veličine, zeleni kanal će biti pola potpune veličine dok će plavi kao i crveni biti četvrtina potpune veličine.

2.2.3. Konverzija u YUV422 format

Za razliku od normalnog kodiranja, gdje se ovdje prvo razmjerno uvećavaju kanali, pa zatim pretvara u YUV444 format, ovdje polja spajamo u jednu datoteku na način da prvo ide zeleno polje visina * širina / 2, zatim plavo polje veličine visina * širina / 4 te u konačnici crveno polje veličine visina * širina / 4. YUV422 je format kodiranja boja koji se obično koristi u video obradi i kompresiji. U ovom formatu, informacije o svjetlini (Y) pohranjuju se u višoj rezoluciji od informacija o boji (U i V), što pomaže uštedi propusnosti uz zadržavanje visoke vizualne kvalitete. Y mora imati najveću kvalitetu jer on nosi većinu podataka same slike. Pretvorba u YUV422 uključuje ponovno izračunavanje RGB boja u YUV spektar boja i primjenu poduzorkovanja na komponente boje.

2.2.4. Kodiranje pomoću FFmpeg

Sljedeći korak u procesu je kodiranje YUV slike. Ovaj korak se izvodi pomoću FFmpeg enkodera. Proces kodiranja komprimira slikovne podatke, čineći ih prikladnima za pohranu ili prijenos. Svrha kodiranja je pretvoriti podatke u format kojim je lako upravljati i koji je učinkovit za kasniju upotrebu, kao što je reprodukcija videa ili daljnja obrada slike, a za svrhu ovog rada koristiti će se za usporedbu s kodiranjem bez demozaika.

2.2.5. Dekodiranje u YUV422 pomoću FFmpeg

Sljedeći korak u procesu je dekodiranje MP4 videa. Ovaj korak se izvodi pomoću FFmpeg dekodera. Proces dekodiranja pretvara komprimirane video podatke u nekomprimirani format, omogućujući njihovu reprodukciju ili daljnju obradu. Svrha dekodiranja je pre-

tvoriti podatke u format kojim je lako upravljati i koji je prikladan za analizu ili daljnje obrade, kao što je konverzija u drugi format ili prilagodba specifičnim zahtjevima projekta. U ovom slučaju, dekodirani podaci će se koristiti za pretvorbu u YUV422p format, što će omogućiti ekstrakciju zelenog, plavog i crvenog kanala bez poteškoća.

2.2.6. Izvlačenje crvenih, zelenih i plavih piksela iz slike

Maskirali smo zeleni kanal veličine visina * širina / 2 kao Y komponentu, plavi kanal veličine visina * širina / 4 kao U komponentu, te crveni kanal jednake veličine visina * širina / 4, što znači da nakon dekodiranja videa ti kanali se "izvlače" iz frame-ova tako što samo "odrežemo" količinu podataka koja nam treba.

2.2.7. Konverzija u YUV444 format

Jednom kada imamo ponovno zeleni, plavi i crveni kanal, razmjerno ih uvećamo koristeći prosjek susjeda oko samih piksela i dobivamo sliku u punoj rezoluciji, nakon toga potpunu sliku pune širine i visine možemo jednostavno pretvoriti u YUV444 format korištenjem formule, bitno je naglasiti da pretvorba YUV mora ići u GBR format. Formula glasi:

$B = 1.164(Y - 16) + 2.018(U - 128)$

$G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)$

$R = 1.164(Y - 16) + 1.596(V - 128)$

nakon čega video postaje potpun viden, u punoj rezoluciji kao i u boji, spreman za usporedbu.

3. Podaci kamere i formati

3.1. ZWO ASI kamere

Posljednjih godina područje astrofotografije doživjelo je revoluciju, prvenstveno zahvaljujući brzom napretku digitalnih fotoaparata i softvera za obradu slika. ZWO je jedno od najznačajnijih imena u ovom tehnološkom sektoru, poznato po svojim visoko specijaliziranim ASI kamerama.

ZWO, također poznat kao Zhong Wei Optical and Electronic Co., Ltd., osnovan je 2011. i brzo je postao dobro poznato ime u astrofotografskoj zajednici. ASI serije kamera tvrtke, u početku prepoznate po svojim jednostavnim i pristupačnim rješenjima za astronome, razvile su se u sofisticirane uređaje koji nude impresivnu kvalitetu slike i preciznost.

Kamere ZWO ASI poznate su po svojoj svestranosti, jer mogu raditi s različitim teleskopima i dodacima, kao i za svakodnevno korištenje kamere, kako bi se prilagodile širokom rasponu korisnika.

Osim toga, tvrtka je postavila snažan sustav podrške za svoje klijente, koji uključuje aktivnu zajednicu entuzijasta astrofotografije, detaljne priručnike i vodiče, kao i redovita ažuriranja firmvera koja pružaju nove značajke i poboljšanja, zajedno s izravnim pristupom kameri putem ASI SDK dostupnog na njihovoj web stranici. Iako postoji njihov program koji snima i sprema video, u ovom radu to radimo "iz nule".

3.2. RGB format i interpolacija

RGB format temeljna je komponenta tehnologije digitalne slike i naširoko se koristi u ispisu fotoaparata, uključujući one koji se koriste u astrofotografiji i općim fotografskim primjenama. Ovaj se format temelji na tri primarne boje - crvenoj, zelenoj i plavoj - koje se kombiniraju kako bi stvorile cijeli spektar boja.

RGB format predstavlja svaku boju kao kombinaciju intenziteta crvenog, zelenog i plavog kanala. Svaki piksel u RGB slici sastoji se od tri vrijednosti, po jedna za svaku primarnu boju. Ove vrijednosti mogu biti u rasponu od 0 do 255 u formatu od 8 bita po kanalu, što znači da jedan piksel sadrži 24 bita podataka (8 bita za svaki kanal). Na primjer, slika razlučivosti 4000 x 3000 piksela može lako doseći veličinu datoteke od 36 megabajta, budući da svaki piksel zahtijeva 3 bajta (24 bita). [4]

Velike veličine datoteka RGB slika stvaraju potrebu za smanjenjem formata, posebno za pohranu i obradu. Neobrađene RGB slike, sadrže sve izvorne podatke koje je uhvatio senzor kamere. Ovi neobrađeni podaci potrebni su za visokokvalitetnu obradu slike, ali su preveliki i neučinkoviti za svakodnevnu upotrebu. Stoga se primjenjuje kompresija slike i smanjenje formata. To uključuje smanjenje količine podataka bez značajnog gubitka kvalitete slike.

Interpolacija je tehnika koja se koristi za popunjavanje informacija o boji koje nedostaju na slici. Digitalni fotoaparati koriste niz filtara boja, kao što je Bayerov uzorak, za hvatanje boja. Svaki piksel na senzoru ima samo jedan filter u boji - crveni, zeleni ili plavi. Interpolacija koristi boje okolnih piksela za izračunavanje vrijednosti boja koje nedostaju, čime se stvara potpuna RGB slika. Ovaj proces, također poznat kao debayering, neophodan je za postizanje glatke i živopisne slike, ali ako se ne izvede ispravno, ponekad može rezultirati artefaktima u boji ili gubitkom detalja. [5]

Postoje nekoliko razloga zbog kojih su često potrebne korekcije u obradi RGB slika. Prvo, kromatska aberacija, koja je uzrokovana lećama koje fokusiraju svjetlost različitih valnih duljina na neznatno različitim mjestima, može dovesti do pomaka boja na rubovima područja visokog kontrasta. Drugo, vinjetiranje ili potamnjenje kutova slike mora se

ispraviti kako bi se osigurala jednolika ekspozicija. Ostali korektivni koraci uključuju prilagodbu ravnoteže bijele za točnu obnovu boja i smanjenje šuma kako bi se smanjili učinci digitalnog šuma.

4. Formati i standardi video kodiranja

U digitalnoj obradi slike i video kodiranju, ključno je kako se informacije o boji pohranjuju i obrađuju. Postoje različite tehnike kodiranja i formati koji se koriste za učinkovito upravljanje tim informacijama, a svaki od njih ima svoje jedinstvene prednosti i primjene. Ovaj rad se usredotočuje na četiri važna sustava kodiranja: RGGB, RGB, YUV 422 i YUV 444.

4.1. RGGB format

RGGB format, poznatiji kao Bayerov filter, je jedan od najčešćih formata rasporeda boja u digitalnoj fotografiji. Nazvan je po Bryceu Bayeru, znanstveniku koji ga je patentirao 1976. godine, i koristi se u većini digitalnih kamera, uključujući DSLR i pametne telefone.

U ovom formatu, svaki piksel na senzoru kamere ima samo jednu od četiri osnovne boje: crvenu (R), zelenu (G), zelenu (G) ili plavu (B). Ovaj raspored boja omogućava snimanje punog spektra boja, ali zahtijeva daljnju obradu kako bi se dobila puna RGB slika. Bayerov filter koristi mrežu od 2x2 piksela, pri čemu je svaki piksel osjetljiv samo na jednu od boja.[3]

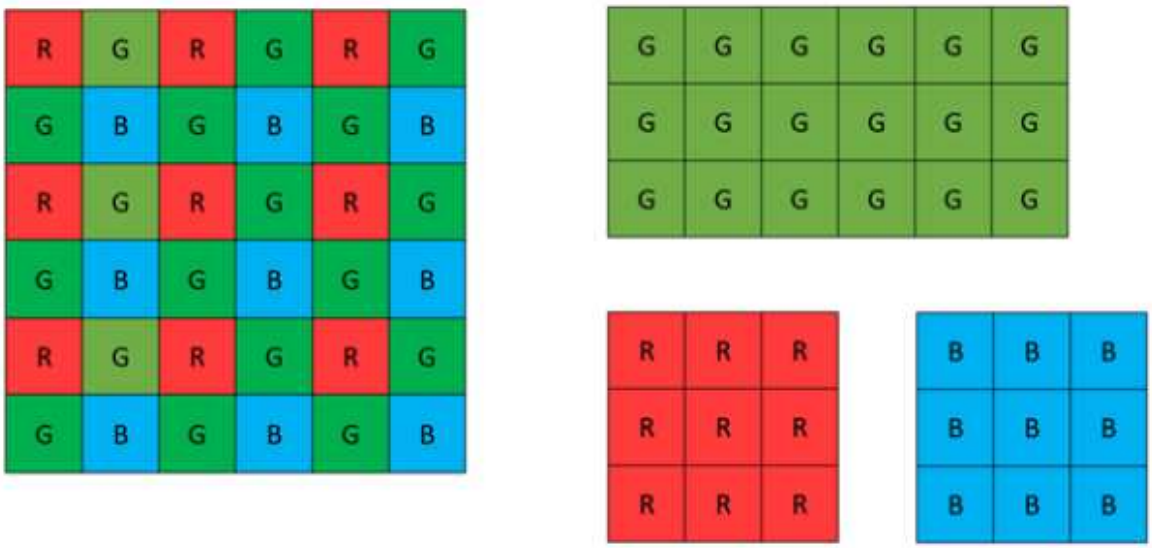
Na primjer, u RGGB rasporedu, prvi red piksela bi bio:

R G R G R G

Drugi red bi bio:

G B G B G B

Svaki piksel u ovom rasporedu zapravo predstavlja samo jednu od tri osnovne boje. Da bi se dobila puna RGB slika, potrebno je interpolirati nedostajuće boje za svaki piksel i podići veličinu slike. To se često radi korištenjem različitih tehnika interpolacije, kao što su bilinear ili bikubična interpolacija. [2] (Slika 4.1.)



Slika 4.1. Bayerov uzorak

Proces interpolacije se koristi za izračunavanje vrijednosti nepoznatih piksela na temelju poznatih piksela oko njih. Na primjer, u Bayerovom rasporedu, ako je piksel označen kao crveni (R), nedostajuće zelene i plave vrijednosti se interpoliraju iz susjednih piksela. Postoje različite metode interpolacije, od jednostavnih do složenijih, koje pokušavaju što vjernije rekonstruirati nedostajuće boje.

Korištenje RGGB formata omogućava efikasno snimanje slike sa manjim brojem senzora, što rezultira nižim troškovima i manjim zahtjevima za prostorom na senzoru. Međutim, to također zahtijeva sofisticiranu obradu slike kako bi se dobila konačna RGB slika visoke kvalitete.

4.2. RGB format

RGB format je jedan od najosnovnijih i najčešće korištenih formata za prikaz boje u digitalnoj fotografiji i videozapisima. Ovaj format se temelji na kombinaciji tri primarne boje: crvenoj (R), zelenoj (G) i plavoj (B). Svaka boja je predstavljena vrijednostima intenziteta u rasponu od 0 do 255, pri čemu 0 označava potpunu odsutnost boje, dok 255 označava maksimalnu zasićenost boje.

Demosaikiranje je proces koji se koristi za pretvaranje podataka dobivenih iz Bayerovog filtera u RGB format. Bayerov filter je tip filtera za boje koji se koristi u digitalnim kamerama i CCD senzorima. U Bayerovom uzorku, pikseli senzora su prekriveni crvenim (R), zelenim (G) i plavim (B) filterima raspoređenim u matrici 2x2. Svaka četveropikselska grupa ima samo jedan crveni piksel, dva zelena piksela i jedan plavi piksel.

Demosaikiranje je proces rekonstrukcije punih RGB slika iz ovog Bayerovog uzorka. Postupak se sastoji od interpolacije boje iz susjednih piksela kako bi se dobila potpuna RGB vrijednost za svaki piksel. Postoje različite tehnike interpolacije koje se mogu koristiti, kao što su bilinearna, bikubična interpolacija i druge, a odabir ovisi o potrebama i zahtjevima primjene.

4.3. YUV format

YUV format je popularan format za prikaz boje kod videozapisa. Umjesto RGB (Crvena-Zelena-Plava) modela, koji koristi tri kanala za prikaz svake boje, YUV koristi tri komponente: Y (luminancija) i U, V (krominancije). Ovaj format je posebno koristan u kompresiji videozapisa, jer ljudsko oko ima veću osjetljivost na promjene u svjetlosti (luminanciji) nego na promjene u boji (krominanciji), pa je iz tog razloga ovaj format poprilično korišten danas.

Y komponenta (luminancija): Ova komponenta predstavlja svjetlinu piksela i nosi većinu informacija o sivoj skali slike. Visoke vrijednosti Y komponente odgovaraju svijetlim pikselima, dok niske vrijednosti odgovaraju tamnim pikselima.

U i V komponente (krominancije): Ove komponente predstavljaju razliku između boje piksela i sive razine (luminancije). U komponenta nosi informacije o plavoj i žutoj boji, dok V komponenta nosi informacije o crvenoj i zelenoj boji.

Različiti tipovi YUV formata variraju u načinu organizacije ovih komponenti i mogućnosti prijenosa boje i svjetline. Neki od najčešćih tipova YUV formata uključuju:

- YUV444: Svaka komponenta (Y, U i V) ima jednaku rezoluciju kao i sam videozapis. Ovo je bez gubitaka u kvaliteti, ali zahtijeva više prostora za pohranu i daje video najbolje kvalitete.

- YUV422: U ovom formatu, svaka linija slike ima polovicu uzorka U i V komponenti u odnosu na Y komponentu. To smanjuje potrebnu propusnost za prijenos podataka u odnosu na YUV444 i ne gubi previše na kvaliteti jer malo isključuje kvalitetu boje.

- YUV420: Ovaj format dodatno smanjuje potrebnu propusnost tako što uzorkuje U i V komponente na svakoj drugoj liniji i svakom drugom pikselu. Ova tehnika omogućuje veću kompresiju, no može dovesti do gubitka kvalitete slike zbog čega nije previše korištena.

5. Dobivanje podataka s kamere

Podaci s kamere se dobivaju putem postojećeg ASI SDK-a koji je specifično napravljen za kontroliranje kamere pomoću C++ programskog jezika. U ovom dijelu rada vidi se detaljan pregled načina prikupljanja i obrade podataka s ASI ZWO kamere korištenjem C++ koda. Kod koristi različite biblioteke poput ASICamera2 i OpenCV za čitanje, prikaz i spremanje slikovnih podataka s kamere.

5.1. Spajanje i inicijalizacija kamere

Prvi korak u procesu je povezivanje i inicijalizacija kamere. Broj dostupnih kamera određuje se pomoću funkcije ASIGetNumOfConnectedCameras. Ukoliko je kamera dostupna, svojstva kamere se dobivaju pomoću funkcije ASIGetCameraProperty. Nakon toga, kamera se otvara i inicijalizira pomoću funkcija ASIOpenCamera i ASIInitCamera. Ukoliko su ove operacije uspješne, prikazuju se informacije o kameri, kao što su najveća razlučivost i je li kamera u boji ili crno-bijela.

```
int numDevices = ASIGetNumOfConnectedCameras();
if (numDevices <= 0)
{
    printf("no camera connected, press any key to exit\n");
    getchar();
    return -1;
}
else
    printf("attached cameras:\n");
for (i = 0; i < numDevices; i++)
{
    ASIGetCameraProperty(&CamInfo, i);
```

```
    printf("%d %s\n", i, CamInfo.Name);
}
```

5.2. Postavljanje formata slike i parametara

Korisnici mogu odabrati predefinirani format slike ili prilagoditi format. Kod koristi ASISetROIFormat za postavljanje razlučivosti, grupiranja i vrste slike. Ti parametri određuju veličinu i vrstu slika koje će fotoaparat proizvesti. Odabir odgovarajućeg formata i grupiranja važan je za postizanje ravnoteže između potrebne kvalitete slike i brzine obrade.

```
if (inputformat){
    printf("0:Size 1024 X 768, BIN 2, ImgType raw8\n");
    // dodatni formati...
    scanf("%d", &definedformat);
    // postavljanje formata...
} else {
    printf("Please input the <width height bin image_type> with one space\n");
    scanf("%d %d %d %d", &width, &height, &bin, &Image_type);
    if (width == 0 || height == 0) {
        width = iMaxWidth;
        height = iMaxHeight;
    }
    ASISetROIFormat(CamInfo.CameraID, width, height, bin,
     ↪  (ASI_IMG_TYPE)Image_type);
}
```

5.3. Učitavanje i prikaz slikovnih podataka

Za čitanje podataka u glavnoj petlji se koristi ASIGetVideoData za dobivanje slikovnih podataka s kamere. Nakon toga, ti se podaci prikazuju pomoću funkcija OpenCV-a poput cvShowImage. Važno je napomenuti razliku između RAW8 formata i RAW16 formata, gdje je za RAW16 format potrebno nadodati velčinu uint16, dok će kod RAW8 formata

ostati samo širina * visina * uint_8. Dohvaćena slika sprema se u pRawBuf i kasnije će se koristiti putem iste varijable.

```
uint8_t* pRawBuf = nullptr;
if (Image_type == ASI_IMG_RAW16) {
    pRawBuf = (uint8_t*)malloc(width * height * sizeof(uint16_t));
} else {
    pRawBuf = (uint8_t*)malloc(width * height);
}
int ret;
if (Image_type != ASI_IMG_RAW8) {
    ret = ASIGetVideoData(CamInfo.CameraID,
    (unsigned char*)pRgb->imageData, pRgb->imageSize, CALLBACK_PERIOD);
}
else {
    ret = ASIGetVideoData(CamInfo.CameraID, pRawBuf, width * height,
     ↪  CALLBACK_PERIOD);
}
```

6. Obrada slike

Ovaj dio programa se bavi obradom Bayer formata slike. Cilj je pretvoriti ga u RGB format te izdvojiti pojedinačne kanale boja (crveni, zeleni i plavi) za daljnju obradu i spremanje u YUV formatu. Prvo, koristimo OpenCV funkciju cvtColor za pretvorbu Bayer slike u RGB sliku, ovo koristimo isključivo radi prikazivanja slike tokom snimanja videa. Nakon toga, kopiramo RGB podatke u odgovarajući buffer. Zatim, koristimo funkciju extract_channels za izdvajanje crvenog, zelenog i plavog kanala iz Bayer slike, te koristimo funkciju save_yuv_frame za spremanje izdvojenih kanala u YUV formatu. Konačno, koristimo funkciju write_yuv_frame za pisanje izdvojenih kanala u YUV datoteku.

```
while (bMain)
{
    frames += 1;
    cv::Mat bayer_img(height, width, CV_8UC1, pRawBuf);
    cv::Mat rgb_img;

    cv::cvtColor(bayer_img, rgb_img, cv::COLOR_BayerRG2RGB);

    memcpy(pRgb -> imageData, rgb_img.data, rgb_img.total() *
    ↪  rgb_img.elemSize());
    cv::Mat r_channel, g_channel, b_channel;

    extract_channels(bayer_img, r_channel, g_channel, b_channel);

    save_yuv_frame(g_channel, b_channel, r_channel, oFile);

    std::string outputFile = "output_video.yuv";
    write_yuv_frame(g_channel, b_channel, r_channel, outputFile);
}
```

6.1. Ekstrakcija kanala

Ovaj dio koda opisuje funkciju extract_channels koja izdvaja crveni, zeleni i plavi kanal iz Bayer slike te ih sprema u odgovarajuće OpenCV matrice. Prvo se inicijaliziraju matrice za crveni, zeleni i plavi kanal s odgovarajućim veličinama, a zatim se iterira kroz Bayer sliku korakom od dva piksela. U svakom koraku, pikseli se raspoređuju u odgovarajuće kanale: crveni kanal dobiva piksele iz parnih redaka i stupaca, zeleni kanal se dijeli na dva privremena kanala (g1 i g2), a plavi kanal dobiva piksele iz neparnih redaka i stupaca. Nakon toga, dva privremena zelena kanala (g1 i g2) se kombiniraju u konačni zeleni kanal (g), čime se osigurava pravilno raspoređivanje zelenih piksela u izvornoj slici.

```
void extract_channels(const cv::Mat& bayer, cv::Mat& r, cv::Mat& g, cv::Mat& b) {
    int rows = bayer.rows;
    int cols = bayer.cols;

    r = cv::Mat(rows / 2, cols / 2, CV_8UC1);
    cv::Mat g1 = cv::Mat(rows / 2, cols / 2, CV_8UC1);
    cv::Mat g2 = cv::Mat(rows / 2, cols / 2, CV_8UC1);
    b = cv::Mat(rows / 2, cols / 2, CV_8UC1);
    g = cv::Mat(rows / 2, cols, CV_8UC1);

    for (int i = 0; i < rows; i += 2) {
        for (int j = 0; j < cols; j += 2) {
            r.at<uchar>(i / 2, j / 2) = bayer.at<uchar>(i, j);
            g1.at<uchar>(i / 2, j / 2) = bayer.at<uchar>(i, j + 1);
            g2.at<uchar>(i / 2, j / 2) = bayer.at<uchar>(i + 1, j);
            b.at<uchar>(i / 2, j / 2) = bayer.at<uchar>(i + 1, j + 1);
        }
    }

    for (int i = 0; i < g1.rows; i++) {
        for (int j = 0; j < g1.cols; j++) {
            g.at<uchar>(i, 2 * j + 1) = g1.at<uchar>(i, j);
            g.at<uchar>(i, 2 * j) = g2.at<uchar>(i, j);
        }
    }
}
```

6.2. Spremanje kanala u YUV datoteku

Ovaj dio koda opisuje funkciju save_yuv_frame, koja sprema izdvojene kanale boja (zeleni, plavi i crveni) iz matrica u YUV formatu u otvoreni izlazni tok (std::ofstream). Funkcija prvo izračunava punu širinu i visinu za YUV kanale, pri čemu se visina zelenog kanala udvostručuje kako bi odgovarala veličini izvornog okvira. Nakon toga se koriste OpenCV funkcije cv::resize za skaliranje zelenog, plavog i crvenog kanala na punu širinu i visinu korištenjem linearne interpolacije (cv::INTER_LINEAR). Nakon skaliranja, podaci iz skaliranih YUV kanala (y_channel, u_channel i v_channel) se zapisuju u izlaznu datoteku koristeći metodu write, pri čemu se svaka matrica pretvara u niz bajtova (reinterpret_cast<const char*>). Na taj način, YUV okviri se ispravno skaliraju i pohranjuju u datoteku. Ovaj YUV video biti će korišten za normalno kodiranje i uspoređivanje na samom kraju.

```
1  void save_yuv_frame(const cv::Mat& g_channel, const cv::Mat& b_channel,
2  const cv::Mat& r_channel, std::ofstream& outFile) {
3      int full_width = g_channel.cols;
4      int full_height = g_channel.rows * 2;
5
6      cv::Mat y_channel, u_channel, v_channel;
7      cv::resize(g_channel, y_channel, cv::Size(full_width, full_height), 0, 0,
       ↪  cv::INTER_LINEAR);
8      cv::resize(b_channel, u_channel, cv::Size(full_width, full_height), 0, 0,
       ↪  cv::INTER_LINEAR);
9      cv::resize(r_channel, v_channel, cv::Size(full_width, full_height), 0, 0,
       ↪  cv::INTER_LINEAR);
10
11     outFile.write(reinterpret_cast<const char*>(y_channel.data),
       ↪  y_channel.total() * y_channel.elemSize());
12     outFile.write(reinterpret_cast<const char*>(u_channel.data),
       ↪  u_channel.total() * u_channel.elemSize());
13     outFile.write(reinterpret_cast<const char*>(v_channel.data),
       ↪  v_channel.total() * v_channel.elemSize());
14 }
```

6.3. Maskiranje G, B i R kanala kao Y, U i V kanale

Ovaj dio koda opisuje funkciju write_yuv_frame, koja sprema izdvojene boje (zelenu, plavu i crvenu) iz matrica u YUV formatu u datoteku. Funkcija otvara izlaznu datoteku outputFile u binarnom načinu rada s opcijom dodavanja (std::ios::binary | std::ios::app). U slučaju nemogućnosti otvaranja datoteke, ispisuje se poruka o grešci i funkcija se prekida. Nakon uspješnog otvaranja datoteke, podaci iz zelenog, plavog i crvenog kanala se zapisuju u datoteku korištenjem metode write, gdje se svaka matrica pretvara u niz bajtova (reinterpret_cast<const char*>). Na kraju, datoteka se zatvara kako bi se osiguralo da su svi podaci pravilno pohranjeni. Kanali se zapisuju tako da se zeleni kanal zapisuje kao Y komponenta, plavi kanal kao U komponenta i crveni kanal kao V komponenta.

```
1 void write_yuv_frame(const cv::Mat& g_channel, const cv::Mat& b_channel
2 , const cv::Mat& r_channel, const std::string& outputFile) {
3     outFile.write(reinterpret_cast<const char*>(g_channel.data),
      ↪  g_channel.total() * g_channel.elemSize());
4     outFile.write(reinterpret_cast<const char*>(b_channel.data),
      ↪  b_channel.total() * b_channel.elemSize());
5     outFile.write(reinterpret_cast<const char*>(r_channel.data),
      ↪  r_channel.total() * r_channel.elemSize());
6
7     outFile.close();
8 }
```

7. Procesiranje video zapisa

U ovom dijelu koda program prvo definira niz varijabli koje predstavljaju ulazne i izlazne datoteke u različitim formatima, kao i odgovarajuće log datoteke za PSNR i kodiranje. Nakon toga, program enkodira maskirane YUV podatke u MP4 format koristeći funkciju encode_yuv_with_ffmpeg, a zatim enkodira normalne YUV podatke pomoću funkcije encode_yuv_with_ffmpeg_for_normal. Nakon svakog koraka enkodiranja, izračunava se PSNR između originalnog YUV i enkodiranog MP4 pomoću funkcije calculate_psnr, te se rezultati spremaju u log datoteke. Program također izvlači i prikazuje metrike iz tih log datoteka. Slijedi dekodiranje maskiranih i normalnih MP4 datoteka natrag u YUV format koristeći convert_mp4_to_yuv. Konačno, izdvojeni kanali iz dekodiranih YUV okvira spremaju se u novi YUV format, te se izračunava PSNR između konačnih YUV datoteka radi procjene kvalitete obrade.

```
1  //niz varijabli koje predstavljaju ulazne i izlazne datoteke
2  encode_yuv_with_ffmpeg(input_file, encoded_file, width, height / 2,
   ↪  yuv_format_422p, crf);
3  encode_yuv_with_ffmpeg_for_normal(input_file_normal, encoded_file_normal, width,
   ↪  height, yuv_format_444p, crf);
4
5  calculate_psnr(input_file, encoded_file, psnr_log_file, width, height,
   ↪  yuv_format_422p);
6  calculate_psnr(input_file_normal, encoded_file_normal, psnr_log_file_normal,
   ↪  width, height, yuv_format_444p);
7
8  extract_metrics(psnr_log_file, encode_log_file);
9  extract_metrics(psnr_log_file_normal, encode_log_file_normal);
10
```

```
convert_mp4_to_yuv(encoded_file, decoded_yuv_file, yuv_format_422p);
convert_mp4_to_yuv(encoded_file_normal, decoded_yuv_file_normal,
    yuv_format_444p);

cv::Mat g_channel, b_channel, r_channel;
std::ofstream outFile(final_yuv_output, std::ios::binary | std::ios::trunc);

for (int i = 0; i < frames; i++) {
    read_yuv_frame(decoded_yuv_file, i, g_channel, b_channel, r_channel, width,
        height);
    save_yuv_frame(g_channel, b_channel, r_channel, outFile);
}
outFile.close();

display_yuv_video(decoded_yuv_file, frames, width, height);

display_yuv444_video(decoded_yuv_file_normal, frames, width, height);

calculate_psnr2(final_yuv_output, decoded_yuv_file_normal, psnr_log_file_final,
    width, height, yuv_format_444p);
```

7.1. Enkodiranje

Ovaj kod definira funkciju encode_yuv_with_ffmpeg koja koristi FFmpeg za enkodiranje YUV datoteka u MP4 format koristeći H.265 kodek. Funkcija prihvaća ulaznu datoteku, izlaznu datoteku, širinu i visinu videa, YUV format i CRF (Constant Rate Factor) vrijednost. CRF vrijednost kontrolira kvalitetu videa, gdje niže vrijednosti znače bolju kvalitetu, a više vrijednosti manju veličinu datoteke. U ovom procesu, za "normalno" enkodiranje koristi se YUV444p format koji zadržava punu rezoluciju boja, dok se za "maskirano" enkodiranje koristi YUV422p format.

```
void encode_yuv_with_ffmpeg(const string& input_file, const string& output_file,
    int width, int height, const string& yuv_format, int crf) {
    ostringstream command;
```

```
    command << "ffmpeg -s " << width << "x" << height
        << " -pixel_format " << yuv_format
        << " -i " << input_file
        << " -c:v libx265 -crf " << crf
        << " -y " << output_file
        << " > ffmpeg_output.log 2>&1";

    string command_str = command.str();
    system(command_str.c_str());
}
```

7.2. Dekodiranje

Ovaj kod opisuje funkciju convert_mp4_to_yuv koja koristi FFmpeg za dekodiranje MP4 datoteka u YUV format. Funkcija prima enkodiranu MP4 datoteku, naziv izlazne YUV datoteke i željeni YUV format. Prilikom dekodiranja, FFmpeg koristi rawvideo kodek za ekstrakciju sirovih video podataka i specificira odgovarajući YUV format. Dekodiranje se provodi radi daljnje obrade video zapisa u izvornom formatu. Za "normalno" dekodiranje koristi se YUV444p format koji zadržava punu rezoluciju boja, dok se za "maskirano" dekodiranje koristi YUV422p format koji smanjuje rezoluciju boja.

```
void convert_mp4_to_yuv(const string& encoded_mp4, const string& output_yuv,
    const string& yuv_format) {
    ostringstream command;
    command << "ffmpeg"
        << " -i " << encoded_mp4
        << " -c:v rawvideo"
        << " -pix_fmt " << yuv_format
        << " -y " << output_yuv
        << " > ffmpeg_decode_output.log 2>&1";
    string command_str = command.str();
    system(command_str.c_str());
}
```

7.3. Računanje PSNR

Ovaj dio koda opisuje funkciju calculate_psnr koja koristi FFmpeg za izračunavanje PSNR (Peak Signal-to-Noise Ratio) vrijednosti između dvije video datoteke. Funkcija prima naziv dvije datoteke, naziv log datoteke za spremanje rezultata, širinu i visinu videa te YUV format. FFmpeg komanda se generira pomoću ostringstream objekta i koristi filter kompleks za izračunavanje PSNR-a. Rezultati se zatim spremaju u log datoteku koja je navedena kao parametar. Ova funkcija se koristi u glavnom programu kako bi se izračunale PSNR vrijednosti između originalne YUV datoteke i enkodirane MP4 datoteke, kako za "maskirane" tako i za "normalne" video zapise. U ovom dijelu koda prikazano je kako se funkcija calculate_psnr koristi za izračunavanje PSNR vrijednosti između originalnih YUV datoteka i enkodiranih MP4 datoteka, bez obzira na format (YUV422p ili YUV444p). Nakon enkodiranja i dekodiranja, PSNR se računa i rezultati se spremaju u odgovarajuće log datoteke, što omogućava analizu enkodiranih video zapisa.

```
void convert_mp4_to_yuv(const string& encoded_mp4, const string& output_yuv,
    const string& yuv_format) {
    ostringstream command;
    command << "ffmpeg"
        << " -i " << encoded_mp4
        << " -c:v rawvideo"
        << " -pix_fmt " << yuv_format
        << " -y " << output_yuv
        << " > ffmpeg_decode_output.log 2>&1";

    string command_str = command.str();
    system(command_str.c_str());
}
```

8. Ekstrakcija kanala i spremanje upscaliranog YUV videa

Unutar petlje koja prolazi kroz sve okvire videa, ovaj dio koda izvodi ekstrakciju kanala boja iz dekodiranih YUV okvira i njihovo spremanje u upscalirani YUV format. Funkcija read_yuv_frame čita YUV podatke za trenutni okvir i razdvaja ih u zelene (G), plave (B) i crvene (R) kanale. Nakon toga, funkcija save_yuv_frame sprema ove izdvojene kanale u otvorenu izlaznu datoteku outFile u YUV formatu. Kada su svi okviri obrađeni, izlazna datoteka se zatvara, a korisniku se prikazuje poruka koja potvrđuje da je upscalirani video uspješno spremljen kao "final_output_video.yuv".

```
for (int i = 0; i < frames; i++) {
    read_yuv_frame(decoded_yuv_file, i, g_channel, b_channel, r_channel, width,
    ↪  height);
    save_yuv_frame(g_channel, b_channel, r_channel, outFile);
}
std::cout << "Upscaled video saved to \"final_output_video.yuv\" " << std::endl;

outFile.close();
```

8.1. Ekstrakcija kanala iz dekodiranog videa

Ovaj dio koda opisuje funkciju read_yuv_frame koja se koristi za čitanje pojedinačnog okvira iz YUV datoteke i izdvajanje kanala boja (zelene, plave i crvene) u zasebne OpenCV matrice. Funkcija prima naziv ulazne datoteke, indeks okvira koji se treba pročitati, reference na matrice za zelene, plave i crvene kanale, kao i dimenzije videa (širina i visina).

Prvo se izračunavaju dimenzije i veličine za svaki kanal na temelju ulazne rezolucije. Zatim se izračunava veličina svakog okvira i pomak unutar datoteke za traženi okvir. Datoteka se otvara u binarnom načinu rada, a pokazivač datoteke se postavlja na početak željenog okvira. Za svaki kanal se stvaraju OpenCV matrice, a podaci se čitaju iz datoteke i pohranjuju u odgovarajuće matrice. Nakon čitanja podataka, datoteka se zatvara.

```
void read_yuv_frame(const std::string& inputFile, int frameIndex, cv::Mat&
↪  g_channel, cv::Mat& b_channel, cv::Mat& r_channel, int width, int height) {
    int g_rows = height / 2;
    int g_cols = width;
    int b_rows = height / 2;
    int b_cols = width / 2;
    int r_rows = height / 2;
    int r_cols = width / 2;

    size_t g_size = height / 2 * width;
    size_t b_size = height / 2 * width / 2;
    size_t r_size = height / 2 * width / 2;

    size_t frame_size = g_size + b_size + r_size;
    size_t frame_offset = frameIndex * frame_size;

    std::ifstream inFile(inputFile, std::ios::binary);
    if (!inFile.is_open()) {
        std::cerr << "Could not open input file: " << inputFile << std::endl;
        return;
    }

    inFile.seekg(frame_offset, std::ios::beg);

    g_channel.create(g_rows, g_cols, CV_8UC1);
    b_channel.create(b_rows, b_cols, CV_8UC1);
    r_channel.create(r_rows, r_cols, CV_8UC1);

    inFile.read(reinterpret_cast<char*>(g_channel.data), g_size);
```

```
    inFile.read(reinterpret_cast<char*>(b_channel.data), b_size);
    inFile.read(reinterpret_cast<char*>(r_channel.data), r_size);

    inFile.close();
}
```

8.2. Upsampling i spremanje konačnog videa

Ovaj dio koda definira funkciju save_yuv_frame, koja omogućava spremanje izdvojenih kanala boja (zeleni, plavi i crveni) u upscalirani YUV format u otvorenu izlaznu datoteku. Funkcija najprije određuje punu širinu i visinu YUV kanala na osnovu dimenzija zelenog kanala, gdje se visina udvostručuje. Zatim se koristi OpenCV funkcija cv::resize za skaliranje zelenog, plavog i crvenog kanala na punu širinu i visinu pomoću linearne interpolacije (cv::INTER_LINEAR). Nakon što su kanali skalirani, podaci iz skaliranih YUV kanala (y_channel, u_channel i v_channel) se pišu u otvorenu izlaznu datoteku koristeći metodu write, pri čemu se svaka matrica pretvara u niz bajtova (reinterpret_cast<const char*>).

```
void save_yuv_frame(const cv::Mat& g_channel, const cv::Mat& b_channel, const
↪  cv::Mat& r_channel, std::ofstream& outFile) {
    int full_width = g_channel.cols;
    int full_height = g_channel.rows * 2;

    cv::Mat y_channel, u_channel, v_channel;
    cv::resize(g_channel, y_channel, cv::Size(full_width, full_height), 0, 0,
    ↪  cv::INTER_LINEAR);
    cv::resize(b_channel, u_channel, cv::Size(full_width, full_height), 0, 0,
    ↪  cv::INTER_LINEAR);
    cv::resize(r_channel, v_channel, cv::Size(full_width, full_height), 0, 0,
    ↪  cv::INTER_LINEAR);

    outFile.write(reinterpret_cast<const char*>(y_channel.data),
    ↪  y_channel.total() * y_channel.elemSize());
```

```
11      outFile.write(reinterpret_cast<const char*>(u_channel.data),
        ↪  u_channel.total() * u_channel.elemSize());
12      outFile.write(reinterpret_cast<const char*>(v_channel.data),
        ↪  v_channel.total() * v_channel.elemSize());
13  }
```

9. Prikazivanje videa pomoću OpenCV

Ovaj dio koda opisuje funkciju display_yuv_video koja koristi OpenCV za prikazivanje YUV video zapisa. Funkcija prima naziv ulazne datoteke, broj okvira, širinu i visinu videa. U petlji koja prolazi kroz sve okvire, funkcija read_yuv_frame čita trenutni YUV okvir i izdvaja kanale boja (zeleni, plavi i crveni). Svaki kanal se zatim skalira natrag na punu veličinu slike koristeći funkciju cv::resize i kubičnu interpolaciju (cv::INTER_CUBIC). Nakon skaliranja, kanali se spajaju u jedan kolorni okvir pomoću cv::merge, a zatim se prikazuju na ekranu pomoću cv::imshow. Funkcija cv::waitKey osigurava prikaz svakog okvira otprilike 33 milisekunde, što rezultira približno 30 okvira u sekundi. Na kraju, svi prozori se zatvaraju pozivom cv::destroyAllWindows.

```
1  void display_yuv_video(const std::string& inputFile, int frames, int width, int
   ↪  height) {
2      cv::Mat g_channel, b_channel, r_channel;
3
4      for (int i = 0; i < frames; i++) {
5          read_yuv_frame(inputFile, i, g_channel, b_channel, r_channel, width,
           ↪  height);
6
7          cv::Mat g_resized, b_resized, r_resized;
8          cv::resize(g_channel, g_resized, cv::Size(width, height), 0, 0,
           ↪  cv::INTER_CUBIC);
9          cv::resize(b_channel, b_resized, cv::Size(width, height), 0, 0,
           ↪  cv::INTER_CUBIC);
10         cv::resize(r_channel, r_resized, cv::Size(width, height), 0, 0,
           ↪  cv::INTER_CUBIC);
11
```

```
        std::vector<cv::Mat> channels = { b_resized, g_resized, r_resized };
        cv::Mat color_image;
        cv::merge(channels, color_image);

        cv::imshow("YUV Video", color_image);

        cv::waitKey(33);
    }

    cv::destroyAllWindows();
}
```

Ovaj kod se koristi za prikazivanje videa nakon što su dekodirani, dok za vrijeme snimanja se koristi ovaj kod za prikazivanje, također koristeći OpenCV.

```
static void Display(IplImage* pImg)
{
    cvNamedWindow("video", 1);
    while (bDisplay)
    {
        cvShowImage("video", pImg);
        char c = cvWaitKey(1);
        if (c == 27) {
            bDisplay = false;
            bMain = false;
            break;
        }
    }
    cvDestroyWindow("video");
    ASIStopVideoCapture(CamInfo.CameraID);

```

Ovaj program poziva se u dretvi zbog paralelnog snimanja u liniji koda.

```
std::thread displayThread(Display, pRgb);
```

10. Interpolacije

10.1. Pretvorba prostora boja

Pretvorba prostora boja je proces transformacije slikovnih podataka iz jednog prostora boja u drugi. Svrha je prilagoditi prikaz boja slike kako bi bio prikladniji za određenu aplikaciju ili uređaj. Na primjer, u digitalnoj fotografiji slike se često snimaju u RGB prostoru boja, dok za video kompresiju i emitiranje je ponekad potrebno pretvoriti slike u YUV prostor boja.

RGB je aditivni prostor boja koji se koristi za prikaz na zaslonima, dok je YUV prostor boja koji se koristi za video kompresiju i emitiranje. YUV odvaja informaciju o svjetlini od informacije o boji, što ga čini prikladnijim za određene svrhe. Ljudsko oko je osjetljivije na osvjetljenje nego na boju, što omogućuje kodiranje informacija o boji u nižoj razlučivosti bez gubitka kvalitete slike.

U kodu koji analiziramo, neobrađena slika u RGGB uzorku se konvertira u YUV422p format. Proces započinje izdvajanjem crvenog, zelenog i plavog kanala iz neobrađene slike. Nakon toga se primjenjuje balans bijele boje kako bi se osiguralo da se boje pravilno reproduciraju. Balans bijele boje uključuje podešavanje intenziteta crvenog, zelenog i plavog kanala kako bi bijeli objekti na slici izgledali zaista bijelo, bez obzira na boju izvora svjetla. Zatim se slike normaliziraju i primjenjuje se gama korekcija prije skaliranja na raspon od 0 do 255. Na kraju, slike se sprema u formatu YUV422p, koji je pogodniji za video kompresiju i emitiranje.

10.2. Normalizacija podataka

Normalizacija podataka je važan korak u obradi slike kako bi se osiguralo da vrijednosti piksela budu unutar određenog raspona. To je ključno za postizanje dosljednih rezultata tijekom daljnje obrade slike, poput primjene filtera i izdvajanja značajki. Normalizacija također poboljšava robusnost algoritama za analizu slike i osigurava da podaci budu unutar dinamičkog raspona alata za obradu. [7]

U danom kodu, crveni, zeleni i plavi kanali slike su normalizirani na raspon od 0 do 1 dijeljenjem vrijednosti piksela s 4095. Ovaj postupak omogućuje usporedbu vrijednosti piksela na različitim slikama i uređajima. Odabir broja 4095 sugerira da su originalne slike vjerojatno 12-bitni podaci, što je često slučaj kod visokokvalitetnih fotoaparata. Normalizacijom ovih vrijednosti smanjujemo utjecaj varijacija u osvjetljenju i postavkama kamere.

Normalizirane vrijednosti se nakon toga transformiraju u raspon od 0 do 255 kako bi se dalje obrađivale i pohranile u YUV formatu. Ova transformacija omogućuje pohranu podataka u standardnom 8-bitnom formatu, koji se često koristi u video kodiranju i emitiranju.

10.3. Metode interpolacije piksela

Interpolacija piksela je tehnika koja se primjenjuje kako bi se procijenile nove vrijednosti piksela na temelju postojećih podataka o pikselima. Ova tehnika je posebno važna u procesu demosaikiranja, gdje senzor mora interpolirati podatke o pikselima kako bi dobio točnu boju za svaki piksel. Demosaikiranje je ključno jer digitalni fotoaparati obično koriste niz senzora prekrivenih Bayerovim filtrom, koji uzrokuje da svaki piksel u nizu bilježi određenu boju, crvenu, zelenu ili plavu.

Kod prikazuje jednostavnu tehniku interpolacije piksela za konverziju RGGB neobrađene slike u potpunu RGB sliku. Ovaj pristup uključuje kopiranje vrijednosti piksela

kako bi se nadopunile informacije o boji koje nedostaju za svaki piksel. Na primjer, vrijednosti crvenog kanala se koriste za popunjavanje crvenih informacija koje nedostaju u zelenom i plavom kanalu. Slični koraci se primjenjuju za zelene i plave kanale. Ova jednostavna metoda interpolacije može biti brza i jednostavna za implementaciju, ali može rezultirati artefaktima poput obojenih rubova i moiré uzoraka. [6]

Napredniji algoritmi interpolacije, poput bilinearnog, bikubičnog i adaptivnog interpolacije, mogu pružiti bolje rezultate. Na primjer, bilinearna interpolacija koristi ponderirani prosjek četiri najbliže vrijednosti piksela, dok bikubična interpolacija koristi ponderirani prosjek šesnaest najbližih vrijednosti piksela, što u konačnici rezultira u boljom kvalitetom same slike.

11. Arhitektura rješenja

11.1. Alati i tehnologije

U procesu obrade videa koristimo nekoliko ključnih alata i tehnologija kako bismo osigurali visokokvalitetnu i efikasnu obradu podataka. Jedan od glavnih alata u ovom procesu je OpenCV, koji se koristi za prikazivanje videa nakon dekodiranja, kao i za vrijeme samog snimanja. OpenCV je biblioteka za računalni vid koja omogućava jednostavno upravljanje videozapisima, manipulaciju slikama i integraciju s različitim hardverskim uređajima.

Za kodiranje i dekodiranje videa koristimo FFmpeg multimedijalnu platformu koja omogućava konverziju između različitih formata videa i audija, streaming i manipulaciju video i slikovnim datotekama. FFmpeg je izuzetno učinkovit u obradi velikih količina podataka, podržava širok spektar kodeka i formata te omogućava prilagodbu parametara kodiranja kako bi se postigli optimalni rezultati u pogledu kvalitete i veličine datoteke. Integracijom FFmpeg-a u naš sustav, možemo osigurati da video materijal bude pravilno komprimiran i dekodiran.

11.1.1. FFmpeg

U procesu obrade videa, FFmpeg igra ključnu ulogu zahvaljujući svojoj sposobnosti da učinkovito kodira i dekodira video sadržaj koristeći H.265 (HEVC) kodek. H.265 je moderni standard za kompresiju videa koji nudi značajno bolju efikasnost u usporedbi s prethodnim kodecima kao što je H.264. Ovaj kodek omogućava smanjenje veličine video datoteka uz održavanje visoke kvalitete slike, što je posebno važno za aplikacije koje

zahtijevaju prijenos ili pohranu velike količine video podataka.

FFmpeg-ov H.265 enkoder i dekoder omogućavaju prilagodbu različitih parametara kodiranja, kao što su brzina bita, nivo kompresije i kvaliteta slike. Ova fleksibilnost omogućava nam da optimiziramo proces kodiranja u skladu s specifičnim potrebama naših projekata, bilo da je prioritet smanjenje veličine datoteke za brži prijenos ili održavanje visoke kvalitete slike za analizu i prikaz.

Korištenje H.265 enkodera u FFmpeg-u omogućava nam da stvorimo komprimirane video datoteke koje zauzimaju manje prostora bez gubitka detalja, što je ključno za učinkovito upravljanje prostorom za pohranu i smanjenje troškova prijenosa podataka. Nakon enkodiranja H.265 dekoder u FFmpeg-u osigurava da se komprimirani video može brzo i pouzdano dekodirati, omogućavajući fluentan prikaz i analizu videa u stvarnom vremenu.

11.1.2. OpenCV

OpenCV je važan alat za prikazivanje videomaterijala tijekom snimanja i nakon dekodiranja. Tijekom snimanja, omogućava real-time prikaz onoga što kamera snima, pružajući trenutan vizualni feedback. Ova sposobnost je ključna za interaktivno praćenje i analizu, omogućavajući odmah uočavanje i reakciju na događaje u video streamu.

Nakon dekodiranja videa pomoću FFmpeg-a, OpenCV se koristi za prikazivanje dekodiranog videa. Funkcije za manipulaciju slikama i videozapisima omogućavaju glatko prikazivanje videa bez prekida. Također, možemo izvršiti dodatne operacije na videu, poput filtriranja, detekcije objekata i prilagođavanja vizualnih parametara, poboljšavajući kvalitetu prikaza i omogućavajući detaljniju analizu sadržaja.

11.2. Arhitektura rješenja

Arhitektura predloženog rješenja za strujanje podataka s kamere do video kodera temelji se na modularnom pristupu koji osigurava fleksibilnost, lakoću održavanja i skalabilnost sustava. Sustav je podijeljen u nekoliko ključnih komponenti, svaka sa specifičnim zadacima i funkcionalnostima.

11.2.1. Ulazni modul (Camera Input Module)

Modul za unos je odgovoran za inicijalizaciju i konfiguraciju kamere, kao i za prikupljanje sirovih podataka u Bayerovom formatu. Ovaj modul uključuje:

- Spajanje i inicijalizacija kamere: Ovaj korak uključuje uspostavljanje veze s kamerom i konfiguraciju parametara kao što su rezolucija, brzina okidanja i ostale postavke.
- Prikupljanje slika: Modul implementira funkcionalnosti za kontinuirano prikupljanje slika u stvarnom vremenu, koje se zatim šalju na daljnju obradu.

11.2.2. Modul za obradu slike (Image Processing Module)

Ovaj modul provodi potrebne transformacije nad prikupljenim slikama kako bi ih pripremio za kodiranje. Glavne funkcionalnosti uključuju:

- Demozaik: Konverzija Bayerovog formata u RGB format kroz proces interpolacije.
- Pretvorba prostora boja: Konverzija RGB formata u YUV format koji je prikladniji za video kodiranje.
- Normalizacija podataka: Osigurava da podaci imaju odgovarajuće vrijednosti i format za ulaz u video koder.

11.2.3. Modul za procesiranje videa (Video Processing Module)

Modul za enkodiranje videa koristi FFmpeg za pretvaranje obrađenih slika u MP4 video. Ovaj modul obuhvaća sljedeće:

- Konfiguracija FFmpeg-a: Postavljanje parametara za enkodiranje, kao što su bitrate, framerate i format izlaznog videa.
- Enkodiranje videa: Implementacija funkcionalnosti za enkodiranje ulaznog videa tipa YUV u MP4 video.
- Dekodiranje videa: Implementacija funckionalnosti za dekodiranje enkodiranog videa tipa MP4 nazad u YUV oblik.

11.2.4. Modul za izlaz (Output Module)

Modul za izlaz je odgovoran za distribuciju enkodiranog video toka prema odabranom odredištu. To može uključivati:

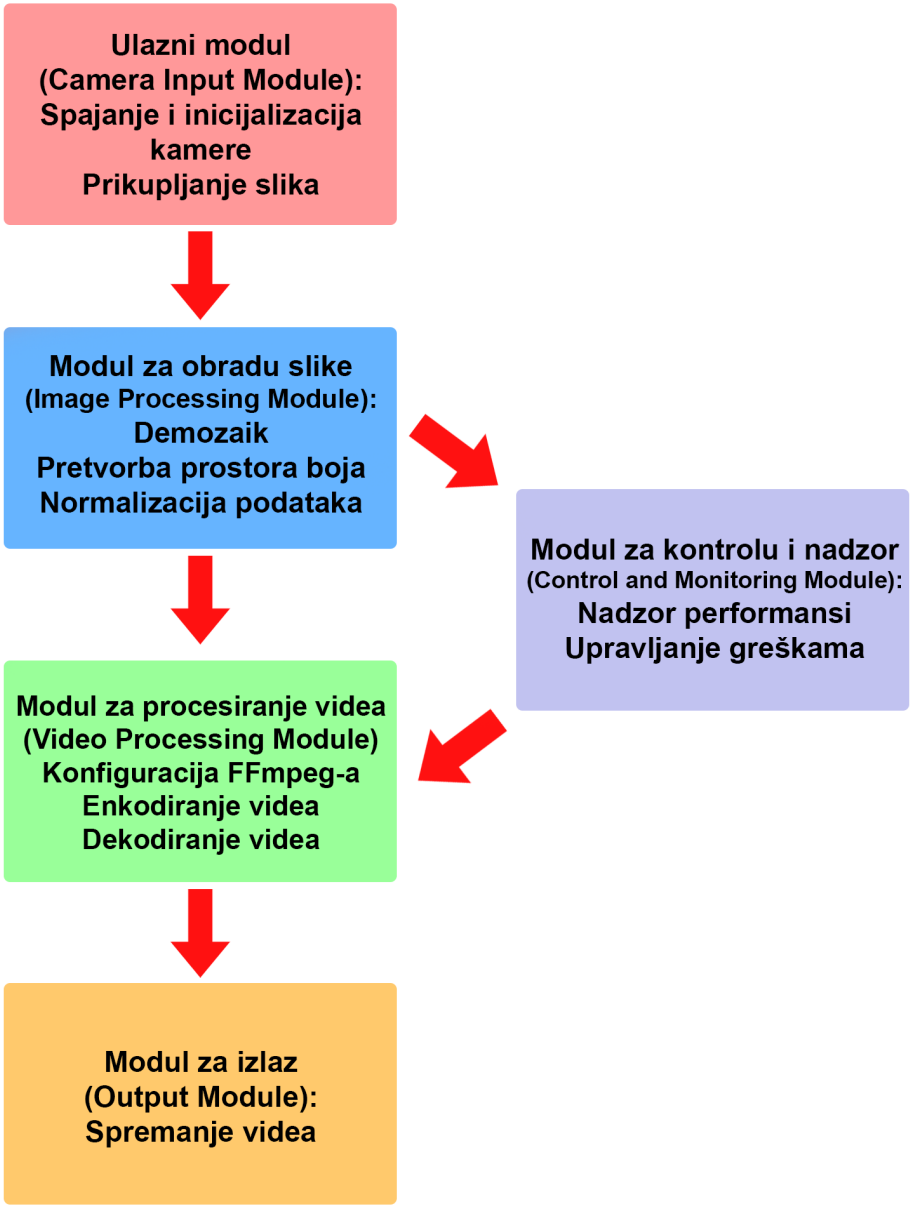
- Spremanje videa: Pohranjivanje enkodiranog videa na lokalni disk ili mrežno spremište.

11.2.5. Modul za kontrolu i nadzor (Control and Monitoring Module)

Ovaj modul omogućava nadzor nad cijelim procesom i upravljanje različitim komponentama sustava. Funkcionalnosti ovog modula izvedene su pomoću OpenCV-a i zapisivnje, odnosno logiranje u .log datoteke. Funkcionalnosti uključuju:

- Nadzor performansi: Praćenje performansi sustava, uključujući brzinu obrade, iskorištenost resursa i latenciju.
- Upravljanje greškama: Detekcija i upravljanje greškama te osiguravanje pouzdanosti sustava.

11.2.6. Dijagram arhitekture rješenja



Slika 11.1. Dijagram arhitekture rješenja

12. Rezultati i rasprava

12.1. Rezultati

Prosječni srednji PSNR				
Resolution	BIN 1	BIN 2	BIN 3	BIN 4
720x480	42.5394	41.5652	41.0751	40.5540
1280x720	43.4352	40.7704	41.2114	42.4757
1920x1020	43.1524	42.6526	x	x
2560x1440	44.6557	44.2788	x	x
3840x2140	44.5509	x	x	x

Prosječni srednji okviri po sekundi				
Resolution	BIN 1	BIN 2	BIN 3	BIN 4
720x480	26.16	26.12	25.01	25.05
1280x720	25.37	25.58	24.54	24.92
1920x1020	25.18	25.44	x	x
2560x1440	24.33	25.11	x	x
3840x2140	22.44	x	x	x

Ubrzanje prosječnog vremena enkodiranja				
Resolution	BIN 1	BIN 2	BIN 3	BIN 4
720x480	26.88%	19.16%	14.17%	8.27%
1280x720	30.00%	21.62%	18.83%	19.42%
1920x1020	36.42%	25.54%	x	x
2560x1440	43.19%	34.68%	x	x
3840x2140	48.05%	x	x	x

Smanjenje prosječne veličine datoteke				
Resolution	BIN 1	BIN 2	BIN 3	BIN 4
720x480	66.45%	66.55%	66.53%	66.59%
1280x720	66.66%	66.66%	66.66%	66.66%
1920x1020	66.66%	66.66%	x	x
2560x1440	66.66%	66.66%	x	x
3840x2140	66.66%	x	x	x

12.2. Analiza performansi

U ovoj sekciji će biti analizirane performanse različitih rezolucija i metoda kodiranja (BIN 1, BIN 2, BIN 3, BIN 4) koristeći četiri metrike: prosječni srednji PSNR, broj slika po sekundi (fps), ubrzanje vremena kodiranja i razlika u veličini datoteke.

12.2.1. Prosječni srednji PSNR

Prosječni srednji PSNR (Peak Signal-to-Noise Ratio) je važna mjera za procjenu kvalitete slike nakon kompresije. Više vrijednosti PSNR-a ukazuju na bolju kvalitetu slike. Sve iznad 40 uglavnom ukazuje na vrlo visoku kvalitetu slike.

- 720x480: Sve četiri metode (BIN 1, BIN 2, BIN 3, BIN 4) pokazuju relativno visoke vrijednosti PSNR-a, pri čemu BIN 1 ima najvišu vrijednost (42.5394), dok BIN 4 ima najnižu (40.5540).
- 1280x720: BIN 1 također ima najviši PSNR (43.4352), dok BIN 2 ima nešto niži PSNR (40.7704).
- 1920x1020 i više: Rezultati su dostupni samo za BIN 1 i BIN 2, gdje BIN 1 ponovno ima najbolji PSNR od 43.1524 za 1920x1020 i 44.6557 za 2560x1440.

12.2.2. Broj slika po sekundi

Broj slika u sekundi (fps) pokazuje koliko brzo video može biti procesuiran. Više vrijednosti fps-a ukazuju na efikasnije kodiranje.

- 720x480: Sve metode (BIN 1, BIN 2, BIN 3, BIN 4) imaju slične vrijednosti fps-a, krećući se od 25.01 fps do 26.16 fps, što ukazuje na konzistentne performanse kodiranja za ovu rezoluciju.
- 1280x720: Metode također pokazuju bliske vrijednosti fps-a, s BIN 1 i BIN 2 malo ispred (25.37 fps i 25.58 fps), dok su BIN 3 i BIN 4 nešto niže (24.54 fps i 24.92 fps).
- 1920x1020 i više: BIN 1 i BIN 2 pokazuju postepeni pad fps-a s povećanjem rezo-

lucije, što je očekivano zbog veće kompleksnosti obrade, dok su podaci za BIN 3 i BIN 4 ne mogući za proizvesti.

12.2.3. Ubrzanje vremena kodiranja

Ubrzanje vremena kodiranja mjeri koliko je brže novo kodiranje u odnosu na referentno kodiranje. Očekivani rast bi trebao biti linearan no kao što vidimo to u praksi jednostavno nije moguće.

- Za sve rezolucije, BIN 1 pokazuje najviše ubrzanje, sa postotcima koji se kreću od 26.88% za 720x480 do 48.05% za 4K rezoluciju.
- BIN 2 također pokazuje visoko ubrzanje za dostupne rezolucije, dok su BIN 3 i BIN 4 nešto niži ali i dalje blizu vrha performansi.
- Podaci za BIN 3 i BIN 4 nisu mogući za rezolucije veće od 1280x720, kao niti BIN 2 za 4k rezoluciju.

12.2.4. Razlika u veličini datoteka

Razlika u Veličini Datoteke Ova metrika pokazuje postotnu razliku u veličini datoteke nakon kompresije.

- 720x480: Sve metode pokazuju vrlo slične rezultate, oko 66.55%.
- 1280x720 i više: Veličina datoteke ostaje konzistentna između 66.66% za sve metode i rezolucije, što pokazuje na to da metode kompresije održavaju sličan nivo efikasnosti bez obzira na povećanje rezolucije.

12.3. Izazovi i ograničenja

Prilikom upotrebe ASI ZWO kamere za snimanje slika putem USB 3.0 kabla, postoje neki izazovi i ograničenja koji mogu utjecati na performanse i kvalitetu snimljenih slika.

- Portovi i kablovi: USB 3.0 kablovi omogućuju brz prijenos podataka, mogu se pojaviti problemi s kompatibilnošću, posebno ako su kablovi ili priključci loše kvalitete ili predugački, što može rezultirati gubitkom signala ili prekidima u prijenosu podataka.
- Software i upravljački program: Zastarjeli ili nekompatibilni upravljački programi mogu uzrokovati nestabilnost sustava, probleme s prepoznavanjem kamere ili smanjenje performansi. Također, postavljanje optimalnih parametara za snimanje, poput ekspozicije, gaina i balansa bijele boje, može biti složeno i zahtijeva eksperimentiranje kako bi se postigli najbolji rezultati.
- Hardware: Iako USB 3.0 omogućuje brzi prijenos podataka, performanse mogu biti ograničene sposobnostima računala da obradi te podatke u stvarnom vremenu. To može dovesti do problema poput kašnjenja, zastajkivanja ili gubitka okvira, posebno pri snimanju slika visoke rezolucije ili pri visokim brzinama snimanja.

13. Zaključak

U ovom radu detaljno je detaljno opisan proces prikupljanja, obrade i prikazivanja slikovnih podataka s ASI ZWO kamere koristeći C++ programski jezik i ključne biblioteke poput ASICamera2 i OpenCV. Arhitektura temelji se na modularnom pristupu koji je veoma koristan za adaptaciju i lako snalaženje u kodu ukoliko je potrebno. Kod je napisan čitko i temeljno sa mnogo izlaznih linija u terminalu pomoću kojih se lagano orijentirati ovisno o kontekstu.

Kod je isproban i testiran na ASI ZWO 183MC Pro kameri, YUV videi su testirani uz pomoć FFmpeg-ovog FFmplay-a. Svi videi nalaze se u root direktoriju i potrebno ih je brisati prilikom svakog ponovnog pokretanja. Finalno testiranje isprobano je na SD, HD, FHD, 2k i 4k rezolucijama na različitim kromabitnim poduzorkovanjima (1, 2, 3 i 4). Neki od kromabitnih poduzorkovanja nisu radili zbog ne mogućnosti kamere da snimi tu sliku.

U konačnici možemo vidjeti i zaključiti da korištenje maskiranog videa znatno smanjuje vrijeme enkodiranja videa kao i veličinu datoteke, bez gubitka na kvalitetni konačne slike i okvirima po sekundi (fps).

14. Dokumentacija i izvorni kôd

Sav kod može se pronaći na ovome url: `https://github.com/Dodica/Masters-Thesis`

Literatura

[1] M. Holik, "Sustav za prenošenje 4K video snimke s niskom latencijom na male udaljenosti", Diplomski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2023. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:168:595177

[2] D. Hofman and J. Benjak, "Offloading Video Encoding Energy Consumption to the Decoder," 2022 7th International Conference on Smart and Sustainable Technologies (SpliTech), Split / Bol, Croatia, 2022, pp. 1-5, doi: 10.23919/SpliTech55088.2022.9854368.

[3] Software center -> other -> for developers: https://www.zwoastro.com/software/

[4] ISO 17321, WD 4, Graphic Technology and Photography – Colour characterisation of digital still cameras (DSCs) using colour targets and spectral illumination, November 1999.

[5] Fadnavis, Shreyas. (2014). Image Interpolation Techniques in Digital Image Processing: An Overview. International Journal Of Engineering Research and Application. 4. 2248-962270.

[6] Thévenaz, Philippe & Blu, Thierry & Unser, Michael & Bankman, I.. (2000). Image Interpolation and Resampling. Handbook of Medical Image Processing and Analysis. 10.1016/B978-012077790-7/50030-8.

[7] Finlayson, G.D., Schiele, B., Crowley, J.L. (1998). Comprehensive colour image normalization. In: Burkhardt, H., Neumann, B. (eds) Computer Vision — ECCV'98. ECCV 1998. Lecture Notes in Computer Science, vol 1406. Springer, Berlin, Heidelberg. ht-

tps://doi.org/10.1007/BFb0055685

Sažetak

Ubrzavanje kodiranja videa koristeći izvorne slike u Bayerovom formatu

Dominik Pavel

Ovaj rad temelji se na obradi slike dobivene s ASI ZWO kamere koristeći ASI_SDK biblioteku, fokusirajući se na dva različita načina enkodiranja i dekodiranja: normalno enkodiranje i maskirano enkodiranje. Normalno enkodiranje uključuje proces dohvaćanja slike s kamere u RGGB obliku, zatim izvlačenje crvenog, zelenog i plavog kanala iz slike kao i proces spajanja tih kanala u YUV444p format, enkodiranje u MP4 format pa zatim dekodiranje nazad u YUV444p format. Za razliku od normalnog, maskirano enkodiranje koristi dodatne korake koji uključuju pretvaranje u YUV422p format prije enkodiranja, maskiranje zelenog kanala kao Y, plavog kanala kao U te crvenog kanala kao V u YUV formatu. Nakon dekodiranja maskirani video ponovno vrši ekstrakciju zelenog, plavog i crvenog kanala i spaja ih u konačnu sliku u YUV444p formatu. Nakon što oba videa prođu proces enkodiranja i dekodiranja, konačni videi uspoređuju se uz pomoć FFmpeg-ovog PSNR-a za dobivanje razlike u kvaliteti videa.

Ključne riječi: ASI ZWO kamera; C++ programski jezik; ASI_SDK biblioteka; OpenCV; FFmpeg; Video enkoder; Video dekoder; Bayerov format; YUV format; Demosaik; Pretvorba prostora boja;

Abstract

Acceleration of video encoding using source images in Bayer format

Dominik Pavel

This thesis is based on the processing of an image obtained from an ASI ZWO camera using the ASI_SDK library, focusing on two different encoding and decoding methods: normal encoding and masked encoding. Normal encoding involves the process of retrieving the image from the camera in RGGB format, then extracting the red, green and blue channels from the image as well as the process of combining those channels into YUV444p format, encoding into MP4 format and then decoding back into YUV444p format. Unlike normal, masked encoding uses additional steps that include converting to YUV422p format before encoding, masking the green channel as Y, the blue channel as U, and the red channel as V in YUV format. After decoding, the masked video re-extracts the green, blue and red channels and merges them into the final image in YUV444p format. After both videos go through the process of encoding and decoding, the final videos are compared using FFmpeg's PSNR to get the difference in video quality.

Keywords: ASI ZWO camera; C++ programming language; ASI_SDK library; OpenCV; FFmpeg; Video encoder; Video decoder; Bayer's format; YUV format; Demosaic; Color space transformation;