Filtriranje slike po boji na realnoj razvojnoj platformi za napredne sustave za pomoć vozaču

Budak, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Filtriranje slike po boji na realnoj razvojnoj platformi za napredne sustave za pomoć vozaču

Diplomski rad

Luka Budak

Osijek, 2021.

SADRŽAJ

1. UVOD *1*

2. PROBLEM FILTRIRANJA SLIKE PO BOJI *3*

2.1. Osnovna terminologija za izradu programskog rješenja za filtriranje slike po boji **3**

2.1.1. Prostor i model boja 3

2.1.2. RGB model boja 4

2.1.3. HSV model boja 4

2.1.4. YUV model boja 5

2.2. Pregled postojećih rješenja za filtriranje slike po boji **7**

3. PREDLOŽENO RJEŠENJE ZA FILTRIRANJE SLIKE PO BOJI *12*

3.1. Koncept rješenja za filtriranje slike po boji **12**

3.2. Opis implementacije programskog rješenja na osobnom računalu **16**

3.2.1. Implementacija u programskom jeziku *Python* 16

3.2.2. Implementacija u programskom jeziku C 18

3.3. ADAS razvojna platforma **19**

3.4. Razvojno okruženje *Vision SDK* **21**

3.5. Opis programskog rješenja na ADAS razvojnoj platformi **23**

3.5.1. Opis implementacije na ADAS razvojnu platformu 23

3.5.2. Optimizacija i paralelizacija rješenja 25

3.5.3. Pokretanje algoritma na ADAS razvojnoj platformi 31

4. TESTIRANJE PREDLOŽENOG RJEŠENJA ZA FILTRIRANJE SLIKE PO BOJI *33*

4.1. Opis skupa testnih slika i načina provođenja testiranja **33**

4.2. Testiranje ispravnosti rješenje za filtriranje slike po boji **35**

4.3. Rezultati testiranja brzine izvođenja rješenja implementiranih na osobnom računalu i na ADAS razvojnoj ploči **38**

4.3.1. Osnovna i optimizirana implementacija rješenja na osobnom računalu 38

4.3.2. Optimizirana implementacija rješenja na jednom DSP odnosno A15 procesoru Alpha ploče 40

4.3.3. Optimizirana implementacija rješenja na dva DSP procesora Alpha ploče 41

4.3.4. Optimizirana implementacija rješenja na dva DSP procesora i A15 procesor Alpha ploče 42

4.4. Osvrt na dobivene rezultate **43**

5. ZAKLJUČAK *44*

LITERATURA *45*

SAŽETAK *48*

ABSTRACT *49*

ŽIVOTOPIS *50*

PRILOZI *51*

1. UVOD

Automobilska industrija pripada skupini industrija koje se danas brzo razvijaju. Takav brz razvoj uz automatizaciju velikog broja funkcionalnosti, koje je dosad obavljao vozač, za cilj ima učiniti vožnju udobnijom i sigurnijom. Stoga ne čudi što čitava automobilska industrija teži ka potpuno autonomnoj vožnji.

Društvo automotiv inženjera (engl. *Society of Automotive Engineers* - SAE) definira šest razina automatizacije vožnje. Na nultoj razini su svi automobili koji su potpuno upravljani od strane vozača. Razina jedan predstavlja najnižu razinu automatizacije i sadrži jedan automatizirani sustav za pomoć vozaču, npr. nadzor brzine pomoću kontrole vožnje. Odlika druge razina je djelomična automatizacija vožnje gdje napredni sustavi pomoći vozaču mogu kontrolirati upravljanje, ubrzavanje i usporavanje vozila. Druga razina ne spada u autonomnu vožnju jer vozač mora imati kontrolu nad vozilom i pratiti stanje u prometu. Treća razina se zove i uvjetovana automatizacija vožnje. Ova razina sadrži sposobnost detekcije okoline i sposobnost donošenja informiranih odluka, no i dalje zahtjeva uključivanje vozača u proces vožnje. Visoka automatizacija vožnje je četvrta razina gdje vozilo izvodi sve zadaće u vožnji u svojstvenim situacijama te ne treba upliv vozača, no ono je dostupno vozaču. Posljednja peta razina predstavlja potpunu automatizaciju vožnje koja ne zahtjeva uključivanje vozača u proces vožnje niti njegovu pozornost [1].

Da bi se postigla automatizacija vožnje, potrebno je implementirati različite sustave za pomoć vozaču u vožnji (engl. *Advanced Driver Assistance System* - ADAS) [2]. ADAS koristi razne senzore kao što su kamere, LiDAR (engl. *Light Detection and Ranging*), radar, ultrazvučne i infracrvene senzore kako bi prikupio informacije iz okoline automobila. Pomoću tih višestrukih ulaznih podataka ADAS može detektirati obližnje prepreke i/ili potencijalne opasnosti te pravovremenom reakcijom izbjeći nesreću. Neke od aplikacija ADAS-a jesu automatsko parkiranje, sustav izbjegavanja sudara, sustav za nadzor vozača, nadgledanje mrtvog kuta te noćni vid. Navedeni sustavi su uvelike ovisni o obradi slike. Obradom je slike često puta potrebno prepoznati i pratiti određene objekte, a prilikom rješavanja navedenih zadataka potrebno je koristiti filtriranje objekata po boji.

U ovom radu predložen je takav algoritam koji za različite prostore boja filtrira sliku po određenoj boji. Algoritam se temelji na metodama obrade elemenata slike gdje je za svaku boju eksperimentalno određen njezin podprostor unutar zadanog prostora boja. Za razvoj programskog rješenja na osobnom računalu su se koristili programski jezici C i *Python* [3] uz pomoć biblioteke otvorenog pristupa za računalni vid u stvarnome vremenu *OpenCV* [4] i *NumPy* [5] biblioteke.

Algoritam za filtriranje slike po boji implementiran je i na razvojnu ADAS Alpha ploču te se koristio programski jezik C uz pomoć *Vision SDK* programskog okruženja.

Rad je strukturiran na sljedeći način. U drugom poglavlju je dana potrebna teorijska podloga, kao i pregled postojećih rješenja uz istaknute prednosti i nedostatke tih rješenja. U trećem poglavlju je opisano vlastito rješenje za filtriranje slike po boji, korištene tehnologije, razvojni proces i dijagram toka rješenja. Četvrto poglavlje je posvećeno testiranju rada predloženog rješenja. Opisana je baza slika na kojoj je provedeno testiranje te sami način testiranja. Na kraju se rada nalazi zaključak.

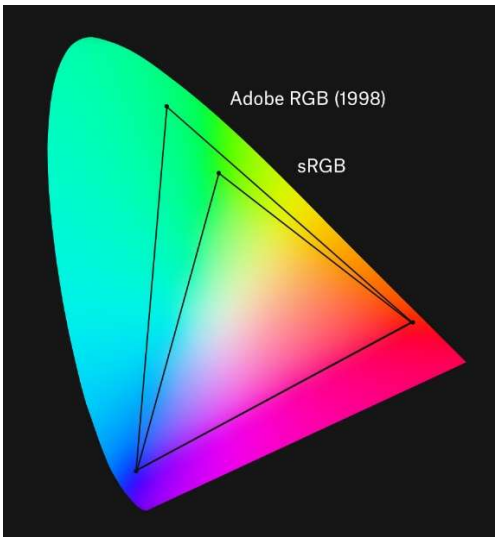
2. PROBLEM FILTRIRANJA SLIKE PO BOJI

2.1. Osnovna terminologija za izradu programskog rješenja za filtriranje slike po boji

Filtriranje slike po boji je proces obrade digitalne slike koji podrazumijeva izdvajanje svih elemenata slike koji pripadaju određenoj boji u zadanom modelu boja. U ovom poglavlju predočena je osnovna terminologija problema filtriranja slike po boji, kao što su prostor i modeli boja.

2.1.1. Prostor i model boja

Prostor boja [6] i model boja [7] su pojmovi koji služe pri opisivanju položaja boje unutar spektra boja. Prostor boja predstavlja definirane kvantitativne veze između distribucija valnih duljina u elektromagnetskom vidljivom spektru i fiziološki percipiranih boja u ljudskom vidu boja. Takva konkretna organizacija boja bitan je alat za upravljanje bojama, važna kada se radi s tintama u boji, osvijetljenim zaslonima i uređajima za snimanje kao što su digitalne kamere jer se njome omogućuje ponovljivi prikazi boja. Dva najpoznatija prostora boja jesu Adobe RGB [8] i sRGB [9] čiji su potpuni podskupovi boja (engl. *color gamut*) vidljivi na slici 2.1.

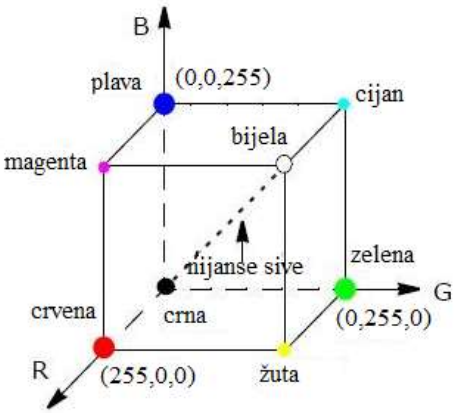


Slika 2.1 Dijagram kromatičnosti s ucrtanim potpunim podskupovima boja za Adobe RGB i sRGB [6]

Boje se opisuju n-torkama brojeva te tako predstavljaju apstraktni matematički model kojeg nazivamo modelom boja. Model boja se također može gledati kao vizualizacija spektra boja u višedimenzionalnom prostoru. Dodavanjem se specifične funkcije mapiranja između modela boja i referentnog prostora boja uspostavlja unutar tog prostora boja određeni raspon boja. Primjerice, Adobe RGB [8] i sRGB [9] su prostori boja koji se baziraju na RGB modelu boja.

2.1.2. RGB model boja

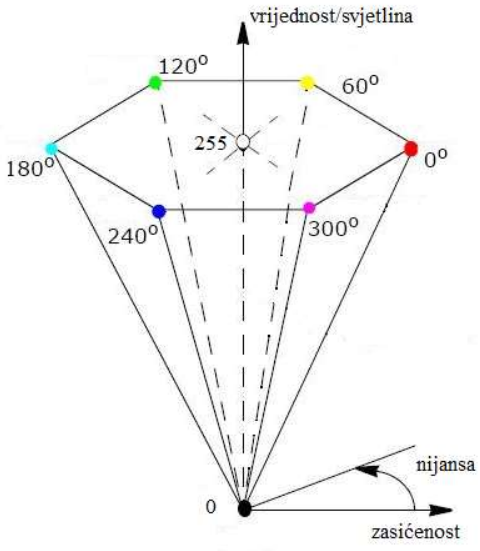
U RGB modelu [10] svaka se boja pojavljuje kao kombinacija primarnih boja (crvene, zelene i plave). Zato se RGB model zove i aditivni model boja. Vrijednosti intenziteta koje primarne boje mogu poprimiti su u intervalu od 0 do 255. Dvije primarne boje jednakog intenziteta se mogu dodati za proizvodnju sekundarnih boja, npr. magenta (crvena i plava), cijan (zelena i plava) i žuta (crvena i zelena). U slučaju da jedna primarna boja ima najjači intenzitet, nastala boja je nijanse te primarne boje (crvenkasta, plavkasta ili zelenkasta), a kada dvije komponente imaju isti najjači intenzitet, tada je boja nijanse sekundarne boje. Kombinacija primarnih boja pri punom intenzitetu (vrijednost jednaka 255) čini bijelu boju, dok je kombinacija primarnih boja pri najmanjem intenzitetu (vrijednost jednaka 0) čini crnu boju. RGB model boja kao vizualnu reprezentaciju u trodimenzionalnom prostoru ima kocku (slika 2.2).



Slika 2.2 RGB model boja [10]

2.1.3. HSV model boja

HSV model boja [10], nastao transformacijom RGB modela, opisuje boje s tri komponente: nijansa (engl. *hue*), zasićenost (engl. *saturation*) i svjetlina (engl. *value*). Razvijen je kako bi se intuitivnije manipuliralo bojama te je dizajniran da približi način na koji ljudi percipiraju i interpretiraju boju. Nijansa definira samu boju. Vrijednosti za os nijansi mogu varirati od 0 do 360 ili od 0 do 180; počevši i završavajući crvenom i prolazeći kroz zelenu, plavu i sve međuboje. Zasićenost označava stupanj do kojeg se nijansa razlikuje od neutralne sive. Vrijednosti se kreću od 0, što znači da nema zasićenja boje, do 255, što je najpotpunija zasićenost dane nijanse pri danom osvjetljenju. Komponenta vrijednosti označava razinu osvjetljenja (svjetlinu), varira od 0 (crno) do 255 (bijelo). HSV model je cilindričnog oblika, ali obično je predstavljen kao stožac ili heksagonalni stožac kao što je prikazano na slici 2.3, jer stožac definira podskup HSV prostora s važećim RGB vrijednostima unutar RGB kocke.

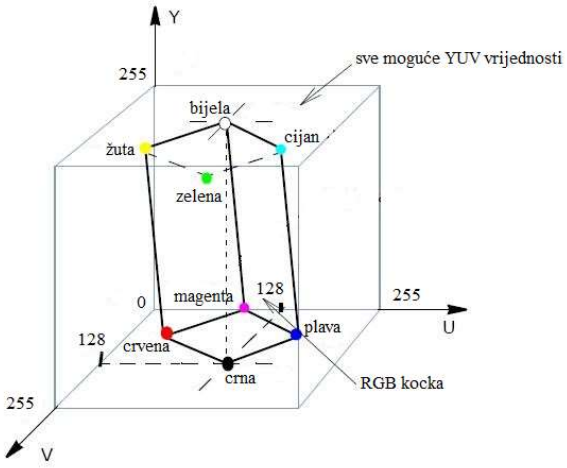


Slika 2.3 HSV model boja [10]

Važno je napomenuti da su tri komponente HSV modela boja međusobno ovisne. Primjerice, ako je komponenta vrijednosti boje postavljena na 0, količina nijanse i zasićenosti nije bitna jer će boja biti crna.

2.1.4. YUV model boja

YUV model boja [10] je izveden iz RGB modela te za svoju vizualnu reprezentaciju ima kocku unutar koje se može upisati RGB kocka (slika 2.4). YUV model boja se sastoji od komponente osvjetljenja (Y) i dvije kromatske komponente koje određuju boju (U i V).



Slika 2.4 YUV model boja [10]

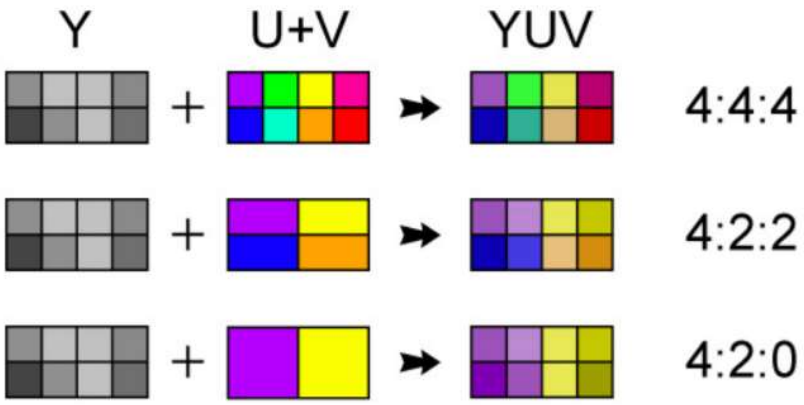
Svjetlina se može izračunati kao ponderirani zbroj crvene (*R*), zelene (*G*) i plave (*B*) komponente RGB modela prema (2-1); kromatske komponente se formiraju oduzimanjem svjetline od plave i od crvene prema (2-2) i (2-3).

$$Y = 0{,}299 \times R + 0{,}587 \times G + 0{,}144 \times B \tag{2-1}$$

$$U = -0{,}147 \times R - 0{,}289 \times G + 0{,}436 \times B + 128 \tag{2-2}$$

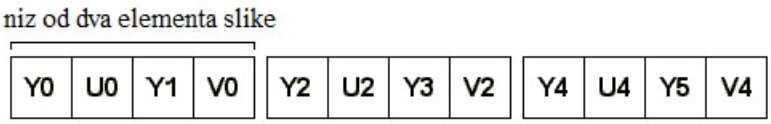
$$V = 0{,}615 \times R - 0{,}515 \times G - 0{,}1 \times B + 128 \tag{2-3}$$

Glavna prednost YUV modela u obradi slike je razdvajanje informacija o svjetlini i boji. Važnost ovog razdvajanja je da se komponenta svjetline slike može obraditi bez utjecaja na njenu komponentu boje. Također, zbog veće osjetljivosti ljudskog oka na promjene u svjetlini nego na promjene u sjajnosti boje (engl. *chrominance*), moguće je odraditi redukciju bitova U i V komponenti. Postupak kojim se svakome elementu slike postavlja vrijednost komponente Y, a odbacuje komponente U i V pojedinih elemenata slike naziva se poduzorkovanje. Postoji više različitih vrsta poduzorkovanja, no neki od njih su poduzorkovanje 4:2:2 i poduzorkovanje 4:2:0 [11].



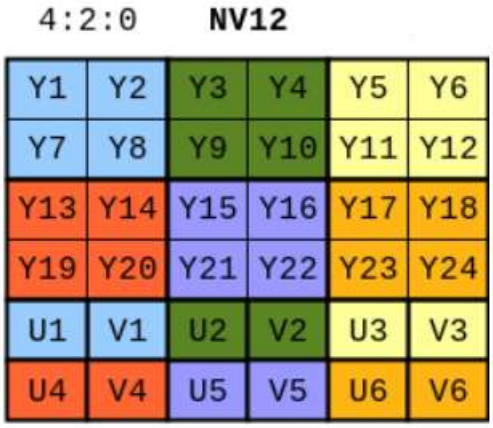
Slika 2.5 Načini poduzorkovanja slike u YUV modelu boja

Poduzorkovanje 4:2:2 će za svaki element slike uzorkovati komponentu Y, a komponente U i V će uzorkovati za svaki drugi element slike u horizontalnom smjeru (slika 2.5). Ovim poduzorkovanjem se prepolovi horizontalna rezolucija sjajnosti boje, a jedan od formata u kojem bude slika pakirana je YUYV [12]. YUYV format pakira podatke na način da nizove od dva elementa slike započne sa Y komponentom prvog elementa, zatim redoslijedom idu U komponenta prvog elementa i Y komponenta drugog elementa slike te na posljetku V komponenta prvog elementa (slika 2.6).



Slika 2.6 Zapis unutar YUYV formata [12]

Poduzorkovanje 4:2:0 će za svaki element slike uzorkovati komponentu Y, a komponente U i V će uzorkovati za svaki drugi element slike u horizontalnom i vertikalnom smjeru, odnosno svaki blok slike veličine 2x2 elementa slike (slika 2.5). Ovim poduzorkovanjem se prepolove horizontalna i vertikalna rezolucija krominacije, a jedan od formata u kojem bude slika pakirana je NV12 [12]. NV12 format pakira podatke na način da izdvoji sve Y komponente, zatim redoslijedom idu parovi U i V komponenata (slika 2.7).



Slika 2.7 Zapis unutar NV12 formata [13]

2.2. Pregled postojećih rješenja za filtriranje slike po boji

Budući da je filtriranje slike po boji srž svake predobrade slike kada se žele izvući pojedine značajke, postoji mnogo radova koji u sklopu svoga rješenja imaju implementirano filtriranje slike po boji.

U radu [14] je predstavljena robusna tehnika detekcije semafora tijekom dana i tijekom noći te procjena udaljenosti. Detekcija semafora se izvorno temelji na značajkama boje. Boje su grupirane pomoću algoritma nejasnog grupiranja (engl. *fuzzy clustering*). Prilagodljivim stjecanjem slike s kontrolom ekspozicije poboljšala se kvaliteta boje s obzirom na ukupnu osvijetljenost slike. Upravo tom prilagodljivošću se omogućila detekcija tijekom dana i tijekom noći. Nakon dohvaćanja slike se izvodi filtriranje slike po boji koristeći nejasnu logiku grupiranja (engl. *fuzzy logic clustering*). Postavljeno je pet grupa koje odgovaraju crvenoj, žutoj, zelenoj, crnoj i bijeloj

boji. Dakle, svaki element slike bi bio sadržan u jednoj od tih grupa. Crvena, žuta i zelena grupa sadrže komponente RGB modela boja koje odgovaraju crvenoj, žutoj i zelenoj boji semafora. Grupa za crnu boju sadrži elemente slike s vrlom tamnim bojama. Problem postajanja boja koje su slične crvenoj, žutoj i zelenoj boji semafora je riješen uvođenjem grupe lažno pozitivnih (engl. *false positive cluster*). Elemente slike koji pripadaju toj grupi bojaju se bijelom bojom. Korišteni RGB model boje se normalizirao kako bi se dobio prostor boja koji je invarijantan na promjenu osvjetljenja i geometriju objekta. Kombinirajući osnovnim aritmetičkim operacijama komponente RGB modela (R, G, B) i komponente normaliziranog RGB modela (R_N, G_N, B_N) svakog elementa slike dobiveno je devetnaest različitih *fuzzy* setova koji se dalje koristili za proces grupiranja. Ovim pristupom se za slike dimenzije 752x480 elementa slike zapisane u RGB formatu postiglo srednje izvođenje cijelog algoritma od 55.09 milisekundi, od čega je vrijeme izvođenja grupiranja 12.10 milisekundi.

U radu [15] predstavljeno je rješenje detekcije boje semafora koristeći strukturalne informacije. Kako bi se detektirala boja uključenog ili isključenog semafora, elementi slike su se filtrirali po bojama koristeći metodu grupiranja. Koristila su dva prostora boje u tu svrhu: RGB i normalizirani RGB. Vrijednosti komponenti normaliziranog RGB modela se dobivaju prema izrazu (2-4).

$$\begin{cases} R_N = G_N = B_N = 0, & za\ R+B+G=0 \\ R_N = \dfrac{R}{R+B+G}, \quad G_N = \dfrac{G}{R+B+G}, \quad B_N = \dfrac{B}{R+B+G}, & za\ R+B+G \neq 0 \end{cases} \tag{2-4}$$

Na temelju pravila (2-5) grupiraju se elementi slike u jednu od pet grupa. Prve tri grupe redom predstavljaju crvenu, žutu i zelenu boju uključenog semafora. Četvrta grupa predstavlja isključeni semafor te joj je dodijeljena crna boja, a peta grupa je za sve ostale boje te se elementima slike ove grupe postavlja vrijednost bijele boje.

$$\begin{cases} R > 100, & R_N > 0.4, & G_N < 0.3, & B_N < 0.3 \\ R > 100, & R_N > 0.4, & G_N > 0.3, & B_N < 0.3 \\ G + B > 100, & R_N < 0.3, & G_N > 0.45, & B_N > 0.45 \\ & & R + G + B < 300 & \end{cases} \tag{2-5}$$

Ovim pristupom se za slike dimenzije 640x480 elementa slike zapisane u RGB formatu postiglo srednje izvođenje cijelog algoritma od 0.15 milisekundi uz 89% točnost detekcije semafora.

Rad [16] u svrhu detekcije prometnih znakova predstavlja metodu prilagodljivih pragova boje za filtriranja slike po boji. Metoda je suzbila smetnje osvijetljenih područja i poboljšala kontrast boja prometnih znakova u slikama sa scenama iz prometa. U izvođenju metode najprije se vrši

transformacija slike prometa u crveno-plavu sliku u sivoj skali (engl. *red-blue gray scale image*) vršeći crveno-plavu normalizaciju prema izrazu (2-6), gdje je *RB* crveno-plava slika u sivoj skali, a *R,G* i *B* su komponente RGB zapisa slike.

$$RB = max\left(\frac{R}{R+B+G}, \frac{B}{R+B+G}\right) \tag{2-6}$$

Na tako dobivenu sliku još uvijek utječe visoki kontrast i svjetlina kompleksne scene iz prometa. Kao rezultat toga, prometni znak u prednjem planu možda neće biti istaknut na vrlo kontrastnoj i vrlo svijetloj slici. Kako bi se izbjegao taj problem, određuje vrijednost praga boje kumulativnom funkcijom distribucije sivog histograma slike na temelju kojeg se vrši približna maksimum-minimum normalizacija (engl. *approximate maximum-minimum normalization*). Prema izrazu (2-7) postavlja se nova vrijednost za svaki elementa slike, gdje je *RB(x)* pojedini element crveno-plavo slike u sivoj skali, a m_1 je prag.

$$RB'(x) = \begin{cases} 0, & RB(x) < m_1 \\ 255, & RB(x) = m_1 \\ \dfrac{RB(x) - m_1}{max\ (RB) - m_1} \times 255, & RB(x) > m_1 \end{cases} \tag{2-7}$$

Tako se većina pozadine eliminirala te su se istaknuli blokovi boja prednjeg plana. Previše izložena područja također su potisnuta kako bi se smanjili utjecaji niske razine kontrasta, visoka svjetlina i drugi čimbenici kompleksne scene iz prometa. Ovim pristupom se za slike dimenzije 1360x800 elementa slike zapisane u RGB formatu postiglo srednje vrijeme izvođenja cijelog algoritma od 0.93 sekundi, od čega je vrijeme izvođenja izdvajanje boja prilagodljivim pragom 0.38 sekundi.

U radu [17] uvodi se nova metoda povećanja kvalitete boje kako bi se pojačala uočljivost boje kritičnih dijelova slike i unaprijedila dosljednost rezultata filtriranja slike po boji. Metoda povećanja kvalitete boje je učinkovita i lako se primjenjuje jer je dizajnirana za rad u RGB prostoru boja te zahtijeva samo jedno procesiranje ulazne slike pri samoj obradi. Metoda predložena ovim radom se sastoji od dva dijela: utvrđivanje praga i povećavanje kvalitete boja. Najprije se određuje prag povećavanja kvalitete boje prema šarenilu zadane slike koristeći srednju razinu boje. Kritičnim područjima slike se zatim povećava kvaliteta boje maksimiziranjem intenziteta boje uz očuvanje nijanse, što se postiže filtriranjem svih triju komponenti RGB modela određenim pragom. Srednja razina boje se računa prema izrazu (2-8), gdje R_i, G_i, B_i predstavljaju vrijednosti komponenti pojedinog elementa slike, M_i najveću i m_i najmanju vrijednost komponente pojedinog

elementa slike, n je ukupan broj elemenata slike te je C_i razina boje pojedinog elementa slike, a $\overline{\mathrm{C}}$ srednja razina boje.

$$\begin{aligned} M_i &= max(R_i, G_i, B_i) \\ m_i &= min(R_i, G_i, B_i) \\ C_i &= M_i - m_i \\ \bar{C} &= \frac{\sum_i^n C_i}{n} \end{aligned} \tag{2-8}$$

Nadalje, prag za povećavanje kvalitete boje se utvrđuje prema srednjoj vrijednosti kao:

$$C_T = max(\frac{1}{8}, \frac{1}{4} - \bar{C}) \tag{2-9}$$

gdje je C_T prag za povećavanje kvalitete boje.

Slika se zatim filtrira na način da se za tri komponente RGB modela svakog elementa slike provjerava razina boje u odnosu na prag. Ako je njezina razina boje veća ili jednaka pragu, element slike će biti dodijeljen kritičnom području boja i povećavanje kvalitete boje bit će primijenjeno maksimiziranjem intenziteta boje uz očuvanje nijanse. U suprotnom, element slike će se okarakterizirati kao šum i bit će uklonjen pretvorbom vrijednosti elementa slike u nijanse sive. Nove vrijednosti komponenti se računaju prema (2-10), (2-11) i (2-12).

$$B_i{}' = \begin{cases} \frac{R_i + B_i + G_i}{3}, & C_i < C_T \\ 1, & C_i \geq C_T, B_i = M_i \\ 0, & C_i \geq C_T, B_i = m_i \\ \frac{B_i - m_i}{|G_i - R_i|}, & inače \end{cases} \tag{2-10}$$

$$G_i{}' = \begin{cases} \frac{R_i + B_i + G_i}{3}, & C_i < C_T \\ 1, & C_i \geq C_T, G_i = M_i \\ 0, & C_i \geq C_T, G_i = m_i \\ \frac{G_i - m_i}{|B_i - R_i|}, & inače \end{cases} \tag{2-11}$$

$$R_i{}' = \begin{cases} \frac{R_i + B_i + G_i}{3}, & C_i < C_T \\ 1, & C_i \geq C_T, R_i = M_i \\ 0, & C_i \geq C_T, R_i = m_i \\ \frac{R_i - m_i}{|B_i - G_i|}, & inače \end{cases} \tag{2-12}$$

Testovi u raznim modelima boja koji koriste metodu izdvajanja pomoću euklidske udaljenosti su pokazali da su slike prethodno obrađene s predloženom metodom povećavanja kvalitete boje polučile točnije i pouzdanije rezultate filtriranja boja od slika bez povećavanja kvalitete i slika poboljšane korištenjem drugih metoda povećavanja kvalitete boje.

3. PREDLOŽENO RJEŠENJE ZA FILTRIRANJE SLIKE PO BOJI

U ovome poglavlju detaljno su objašnjeni koncepti za filtriranje slike po boji i predloženo rješenje za realnu ugradbenu ADAS platformu. Poznato je da razvoj programskog rješenja za ugradbene sustave traje dugo te se za ubrzanje procesa razvoja koristio iterativni pristup. Prvo je izvršena implementacija rješenja za filtriranje slike po boji na osobnom računalu koristeći programski jezik *Python* uz pomoć biblioteke otvorenog pristupa za računalni vid u stvarnome vremenu *OpenCV* i njezinih ugrađenih algoritama za obradu slike. Zatim se pristupilo implementiranju rješenja na osobnom računalu u programskom jeziku C standardnim bibliotekama. Potom slijedi opis ADAS razvojne platforme, njezinog sklopovlja (engl. *hardware*) s popratnim sučeljima i opis razvojnog okruženja *Vision SDK* (engl. *software development kit* - SDK). Na kraju je predloženo rješenje implementirano na ADAS razvojnu platformu uz dodatnu optimizaciju rješenja i paralelizaciju posla korištenjem više procesora (engl. *central processing unit* - CPU).

3.1. Koncept rješenja za filtriranje slike po boji

Kao što je već navedeno na početku ovog poglavlja, za razvoj rješenja filtriranja slike po boji korišten je iterativni pristup. Rješenje se prvo implementiralo na osobnom računalu unutar *Spyder* integriranog razvojnog okruženja (engl. *Integrated Development Environment* - IDE) u programskom jeziku *Python* uz pomoć biblioteke *OpenCV*. Sljedeći korak je bila implementacija rješenja u C programskog jeziku unutar razvojnog okruženja *Visual Studio*. Obje implementacije dijele isti tok izvođenja operacija koji je prikazan slikom 3.1.

Slika 3.1 Dijagram toka koncepta rješenja za filtriranje slike po boji

Prva zadaća je učitavanje slike koja ima elemente slike zapisane u jednom od modela boja objašnjenih u drugom poglavlju ovog rada, a to su: RGB, HSV i YUV. Zatim slijedi postavljanje raspona vrijednosti za pojedine boje u obliku granica. Pronalazak raspona vrijednosti pojedine boje u RGB i YUV modelu boja izvršen je pomoću alata za ugađanje granica napisanog u *Python*-u i prikazan je na slici 3.2. Alat za svaku komponentu modela boje stvara klizače čijim se pomicanjem mijenja donja i gornja granica raspona vrijednosti te komponente. Alat je pomoću klizača za učitane slike obavljao ručno izdvajanje obojenih predmeta. Ovim su se alatom brzo i lagano mogle podesiti vrijednosti granica pojedinih boja. Za podešavanje vrijednosti granica pojedinih boja korištene su dvije slike koje su dane slikom 3.3 i slikom 3.4. Slika 3.3 predstavlja obojene pravokutnike različitih razina prozirnosti, a slika 3.4 predstavlja figurice u raznim boja.

Linija Kod

```
1:      while (True):
2:          l_1 = cv2.getTrackbarPos("L - 1", "Trackbars")
3:          l_2 = cv2.getTrackbarPos("L - 2", "Trackbars")
4:          l_3 = cv2.getTrackbarPos("L - 3", "Trackbars")
5:          u_1 = cv2.getTrackbarPos("U - 1", "Trackbars")
6:          u_2 = cv2.getTrackbarPos("U - 2", "Trackbars")
7:          u_3 = cv2.getTrackbarPos("U - 3", "Trackbars")
8:          lower = np.array([l_1, l_2, l_3])
9:          upper = np.array([u_1, u_2, u_3])
10:         mask = cv2.inRange(oblik, lower, upper)
11:         result = cv2.bitwise_and(oblik, oblik, mask=mask)
12:         result = cv2.cvtColor(result, oblik_cv1)
13:         cv2.imshow("result", result)
14:         k = cv2.waitKey(1) & 0xFF
15:         if k == 27:
16:             break
```

Slika 3.2 Dio programskog kod alata za ugađanje granica

Slika 3.3 Slika za ugađanje donje i gornje granice raspona s pravokutnicima različitih razina prozirnosti

Slika 3.4 Ulazna slika korištena za ugađanje donje i gornje granice raspona

Kod HSV modela boja nije bilo potrebno koristiti alat jer je informacija o boji sadržana u komponenti nijanse te se rasponi pojedinih boja mogu iščitati iz vizualne reprezentacije modela (slika 2.3). Eksperimentalno dobiveni pragovi za raspone svih boja svakog modela boja dani su u tablici 3.1 na temelju slike 3.3 i slike 3.4.

Tablica 3.1 Rasponi boje svakog modela boja

	RGB model [R, G, B]		HSV model [H, S, V]		YUV model [Y, U, V]	
	Donja granica	**Gornja granica**	**Donja granica**	**Gornja granica**	**Donja granica**	**Gornja granica**
Crvena	[188,0,0]	[255,85,80]	[0,90,25] i [170,90,25]	[10,255,255] i [180,25 5,255]	[0,91,186]	[255,113,255]
Plava	[0,0,220]	[153,229,245]	[94,90,25]	[126,255,255]	[0,124,0]	[255,255,125]
Zelena	[0,0,0]	[113,255,168]	[36,90,25]	[76,255,255]	[0,0,0]	[255,130,120]
Žuta	[26,127,0]	[242,230,87]	[26,90,25]	[35,255,255]	[0,0,0]	[255,69,172]
Narančasta	[132,0,0]	[241,165,100]	[11,90,25]	[25,255,255]	[0,55,164]	[255,95,240]
Ljubičasta	[51,0,178]	[238,144,255]	[132,90,25]	[150,255,255]	[0,151,143]	[255,211,190]
Ružičasta	[155,33,111]	[255,236,255]	[155,90,25]	[168,255,255]	[37,120,169]	[237,185,255]

Nakon koraka postavljanja granica za pojedine boje izvršava se generiranje maski te se za svaku boju (tablica 3.1) izvršava logička I operacija ulazne slike i maske odgovarajuće boje. Tako će svi elementi slike koji se nalaze unutar definiranih granica pojedine boje biti očuvani, a ostali elementi će biti postavljeni na vrijednost 0. Rezultat te operacije se sprema u posebnu binarnu datoteku, time je omogućeno da se u daljnjoj upotrebi dobiveni rezultati lakše učitavaju.

Nakon razvoja osnovnog koncepta, pristupilo se optimizaciji rješenja. Optimizacija predstavljenog rješenja za filtriranje slike po boji temelji se na aproksimaciji raspona boja, a opisani su u obliku jednostavne jednadžbe. Budući da svaka boja zauzima podprostor u prostoru definiranog modelom boje, opisom pojedinog raspona jednadžbom omogućilo se znatno ubrzanje provjere pripadnosti elementa slike pojedinoj boji. Korištenjem Lagrange-ove interpolacije za funkcije s više varijabli ostvarena je ideja optimizacije predloženog rješenja. Lagrange-ova interpolacija [20] osnovna je metoda numeričke analize čiji je zadatak pronaći funkciju koja najbolje opisuje konačno uzorkovani skup podataka. Uzorkovani je skup podataka predstavljen rasponima boje (tablica 3.1) kao tri različite točke u trodimenzionalnom prostoru. Te točke (x_i, y_i, z_i) jednoznačno definiraju linearnu funkciju z od dvije varijable x i y za koeficijente α_1, α_2 i α_3 (3-1). Matrica M u (3-2) proizlazi iz sustava linearnih jednadžbi definirane funkcijama (3-1). Matrice M_i u (3-2) nastaju zamjenom i-tog retka matrice M uređenom trojkom varijabli x i y.

$$z_i = \alpha_1 x_i + \alpha_2 y_i + \alpha_3, \qquad 1 \leq i \leq 3 \tag{3-1}$$

$$M = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}, M_1 = \begin{pmatrix} x & y & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}, M_2 = \begin{pmatrix} x_1 & y_1 & 1 \\ x & y & 1 \\ x_3 & y_3 & 1 \end{pmatrix}, M_3 = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{pmatrix} \tag{3-2}$$

$$\Delta = det(M), \Delta_1 = det(M_1), \Delta_2 = det(M_2), \Delta_3 = det(M_3) \quad (3\text{-}3)$$

$$z = z_1 \frac{\Delta_1}{\Delta} + z_2 \frac{\Delta_2}{\Delta} + z_3 \frac{\Delta_3}{\Delta} \quad (3\text{-}4)$$

Metoda izračunavanjem determinanti Δ_i matrica M_i (3-3) i prema izrazu (3-4) daje jednadžbu prvog stupnja od dvije varijable. Dobivene jednadžbe se koriste u implicitnom zapisu i određuju pripadnost elementa slike pojedine boje na sljedeći način. Vrijednosti komponenti pojedinog elementa slike se uvrštavaju u jednadžbu. Za rezultate jednadžbe veće ili jednake nuli, elementi slike pripada pojedinoj boji. U slučaju kada su rezultati jednadžbe manji od nule elementi slike ne pripadaju zadanoj boji.

3.2. Opis implementacije programskog rješenja na osobnom računalu

U ovom potpoglavlju su predstavljene implementacije predloženog rješenja za filtriranje slike po boji na osobnom računalu. Razvoj programskog rješenja na osobnom računalu se sastoji od dvije implementacije. Rješenje se prvo implementiralo u programskom jeziku *Python* uz pomoć biblioteke *OpenCV*. Druga implementacija rješenja na osobnom računala je u C programskog jeziku unutar razvojnog okruženja *Visual Studio*.

3.2.1. Implementacija u programskom jeziku *Python*

Slika 3.5 prikazuje kôd u kojem se učitava ulazna slika pomoću funkcije `imread()` koja za argumente prima naziv slike i način učitavanja zadan kao učitavanje slike u boji. Budući da su ulazne slike učitane u BGR model boje, potrebno je odraditi pretvorbu u odgovarajući model boje korištenjem funkcije `cvtColor()` i unaprijed definiranih oznaka za pretvorbu boje. Pri obradi slika u HSV modelu boja potrebna je oznaka `COLOR_BGR2HSV`, za YUV model boja potrebna je oznaka `COLOR_BGR2YUV`, dok je za RGB model boja potrebna oznaka `COLOR_BGR2RGB`. Nadalje, postavljanje granica pojedinih boja vrijednostima iz tablice 3.1 prati generiranje maski. Generiranje pojedine maske je ostvareno funkcijom `inRange()` kojoj se predaju slika pretvorena u odgovarajući model boja i granice pojedine boje. Funkcijom `inRange()` za svaku komponentu provjerava se svaki element slike i njegova pripadnost u raspon definiran predanim granicama. Za sve one vrijednosti izvan raspona postavlja se vrijednost 0, a za pripadajuće vrijednosti postavlja se vrijednost 255. U listu `results` se pohranjuju rezultati logičke operacije I na razini elementa slike između maske i slike pomoću funkcije `bitwise_and()`. Treba napomenuti da je kod YUV modela boja, zbog činjenice da je crna boja definirana kao [0, 128, 128], bilo potrebno odraditi dodatno postavljanje vrijednosti elemenata slike koji ne pripadaju zadanoj boji. Tim se elementima

za vrijednosti kromatskih komponenti postavila vrijednost 128 (slika 3.6). Konačno se svaki element liste `results` pohranjuje u posebnu binarnu datoteku.

Linija ***Kod***

```
1:      img = cv.imread(image_string, 1)
2:      hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
3:
4:      lower_red = np.array([0,90,25])
5:      upper_red = np.array([10,255,255])
6:      lower_red1 = np.array([170,90,25])
7:      upper_red1 = np.array([180,255,255])
8:      lower_blue = np.array([94,90,25])
9:      upper_blue = np.array([126,255,255])
10:     lower_green = np.array([36,90,25])
11:     upper_green = np.array([76,255,255])
12:     lower_yellow = np.array([26,90,25])
13:     upper_yellow = np.array([35,255,255])
14:     lower_orange = np.array([11,90,25])
15:     upper_orange = np.array([25,255,255])
16:     lower_purple = np.array([132,90,25])
17:     upper_purple = np.array([150,255,255])
18:     lower_pink = np.array([155,90,25])
19:     upper_pink = np.array([168,255,255])
20:
21:     mask_green = cv.inRange(hsv, lower_green, upper_green)
22:     mask_blue = cv.inRange(hsv, lower_blue, upper_blue)
        mask_red = cv.inRange(hsv, lower_red, upper_red) | cv.inRange(hsv,
23:     lower_red1, upper_red1)
24:     mask_y = cv.inRange(hsv, lower_yellow, upper_yellow)
25:     mask_o = cv.inRange(hsv, lower_orange, upper_orange)
26:     mask_p = cv.inRange(hsv, lower_purple, upper_purple)
27:     mask_pi = cv.inRange(hsv, lower_pink, upper_pink)
28:     results = []
29:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_red))
30:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_green))
31:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_blue))
32:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_y))
33:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_o))
34:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_p))
35:     results.append(cv.bitwise_and(hsv, hsv, mask = mask_pi))
36:     for i, result in enumerate(results):
37:         result.tofile(path_string + str(i+1) + ".bin")
```

Slika 3.5 Programski kôd rješenja implementiran u Python programskom jeziku

Linija ***Kod***

```
1:      for result in results:
2:          for k in range(0, result.shape[0]):
3:              for j in range(0, result.shape[1]):
                    if(result[k,j,0] == 0 and result[k,j,1] == 0 and
4:      result[k,j,2] == 0):
5:                      result[k,j,1] = 128
6:                      result[k,j,2] = 128
```

Slika 3.6 Programski kôd dodatnog postavljanja vrijednosti za YUV model boja

Na temelju ovog programskog rješenja je slijedio razvoj predloženog rješenja u programskom jeziku C.

3.2.2. Implementacija u programskom jeziku C

Implementacija programskog rješenja u programskom jeziku C prati sličan tok kao implementacija u programskom jeziku *Python.* Razlike nastupaju kod učitavanja slike te potrebe za vlastitom implementacijom korištenih ugrađenih funkcija biblioteke *OpenCV.* Za učitavanje slike se dinamičkom alokacijom stvara ulazni spremnik podataka koji je tipa `uint8_t` i veličine trostrukog umnoška širine i visine slike. Također se na isti način kreirao izlazni spremnik za filtriranu sliku po boji. Nadalje su se iz ulaznog spremnika izdvajale vrijednosti komponenti modela boje učitane slike. Za svaku od tri komponente je kreiran zasebni spremnik tipa `uint8_t` i veličine umnoška širine i visine slike. Ovim pristupom se omogućilo jednostavnije filtriranje slike po boji i popunjavanje izlaznog spremnika. Funkcija `filterColor()` izdvaja elemente slike koji pripadaju pojedinoj boji (slika 3.7). Funkcija `filterColor()` za argumente prima tri spremnika komponente modela boje `data1, data2, data3`; odgovarajuće donje granice raspona pojedine boje `lower1, lower2, lower3`; odgovarajuće gornje granice raspona boje `upper1, upper2, upper3` te veličinu spremnika `DATASIZE`. Iteracijom po polju elemenata spremnika komponenti provjera se njihova pripadnosti pojedinoj boji pozivom funkcije `isWithinBounds()` (slika 3.8). U slučaju neispunjavanja uvjeta pripadnosti elementima spremnika komponenti se postavlja vrijednost crne boje.

Linija ***Kod***

```
        void filterColor(uint8_t* data1, uint8_t lower1, uint8_t upper1,
                         uint8_t* data2, uint8_t lower2, uint8_t upper2,
                         uint8_t* data3, uint8_t lower3, uint8_t upper3,
1:      uint32_t DATASIZE){
2:          int dataSize = (int)DATASIZE;
3:          for (int i = 0; i < dataSize; i++){
```

```
                if (!IsWithinBounds(data1[i], lower1, upper1) ||
        !IsWithinBounds(data2[i], lower2, upper2) ||
4:      !IsWithinBounds(data3[i], lower3, upper3)){
5:                  data1[i] = 0;
6:                  data2[i] = 0;
7:                  data3[i] = 0;
8:              }
9:          }
10:     }
```

Slika 3.7 Programski kôd funkcije `filterColor()`

Linija Kod

```
        bool IsWithinBounds(uint8_t value, uint8_t lowerLimit, uint8_t
1:      upperLimit){
2:          if(value >= lowerLimit && value <= upperLimit){
3:              return true;
4:          }else{
5:              return false;
6:          }
7:      }
```

Slika 3.8 Programski kôd funkcije `isWithinBounds()`

Na kraju se rezultat filtriranja slike po boji sprema u izlazni spremnik pozivom funkcije `fillOutput()` (slika 3.9). Funkcija `fillOutput()` za primljeni pojedini spremnik komponente modela boje (`data`) i njegove veličine (`DATASIZE`) popunjava izlazni spremnik (`buffer`) s obzirom na odgovarajuću poziciju (`START`) memorijskog zapisa.

Linija Kod

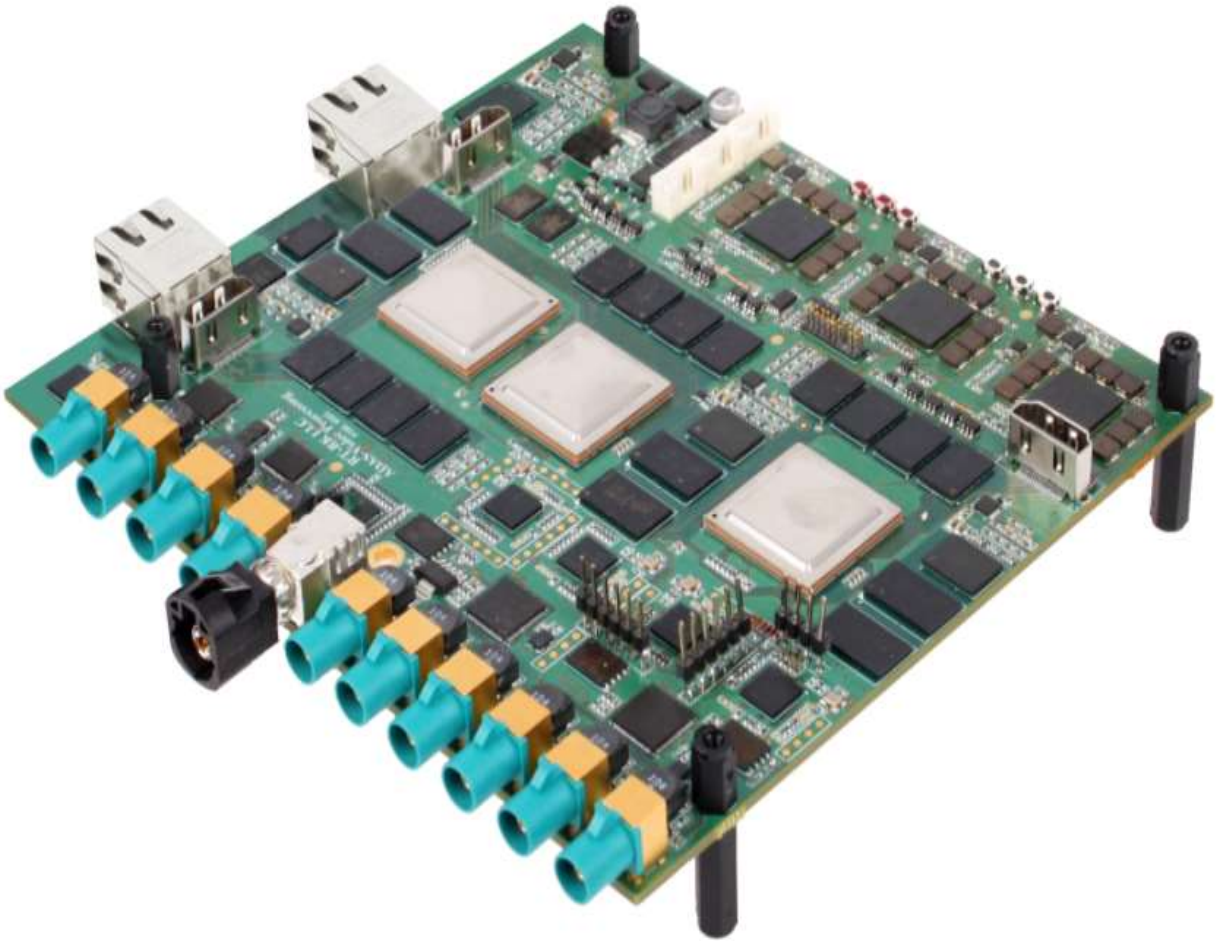
```
        void fillOutput(uint8_t* data, uint32_t DATASIZE, uint8_t* buffer,
1:      uint32_t START){
2:          int start = (int)START;
3:          int dataSize = (int)DATASIZE;
4:          int j = 0;
5:          for (int i = start; i < dataSize; i += 3, j++){
6:              buffer[i] = data[j];
7:          }
8:      }
```

Slika 3.9 Programski kôd funkcije *fillOutput()*

3.3. ADAS razvojna platforma

U okviru diplomskog rada se za realizaciju rješenja koristila razvojna ADAS platforma imena Alpha ploča [18]. Ploču, prikazanu slikom 3.10, je razvio RT-RK te podržava aktivne sustave za upravljanje vozilom, osnovne i napredne sustave upozorenja vozaču i polu-autonomne operacije. Skup ciljanih automotiv aplikacija za cilj ima povećanje udobnosti i cjelokupno iskustvo vožnje.

U tu svrhu su istaknuti algoritmi pomoći vozaču za pogled od naprijed i pogled u okružju, sustav nadzora vozača, noćni vid te sustav zrcaljenja kamere.

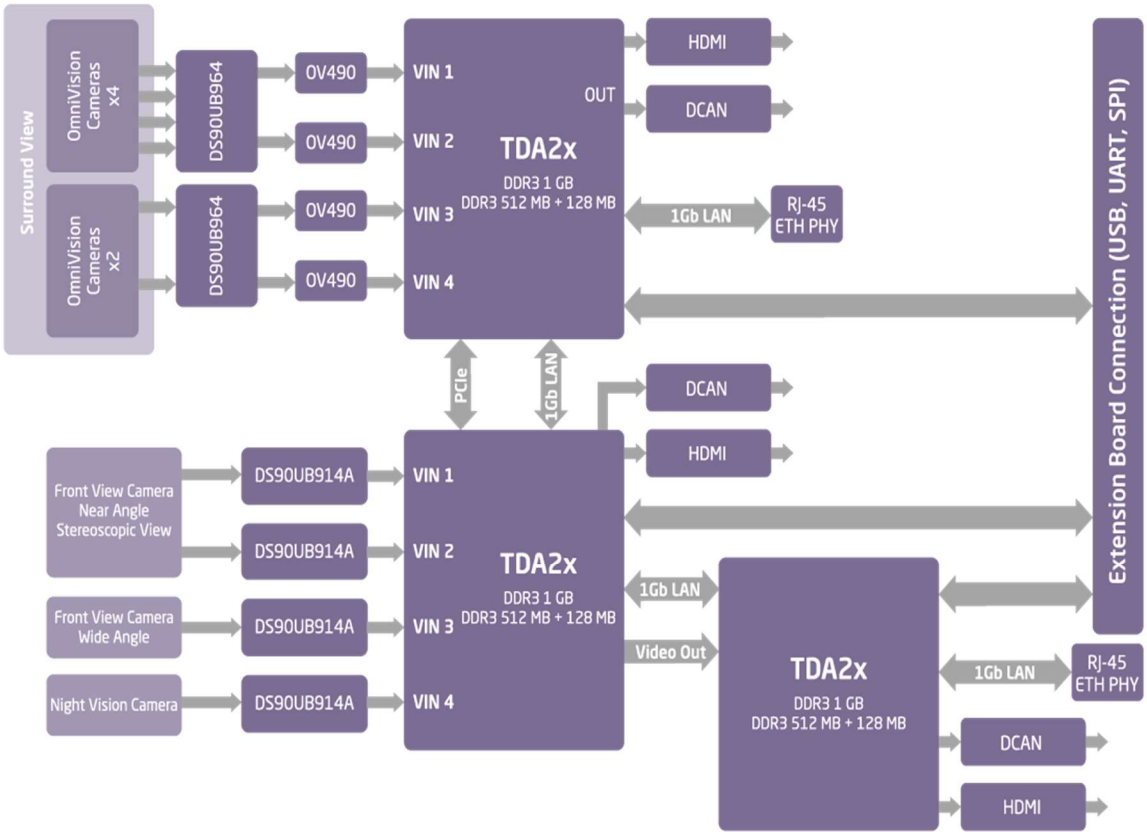


Slika 3.10 Alpha ploča [18]

Na Alpha ploči su sve njene komponente upakirane u tri jedinstvene cjeline koje se zovu sustavi na čipu (engl. *System on Chip* - SoC). SoC-ove proizvodi tvrtka Texas Instruments, tipa su TDA2x te svaki SoC ima određenu upotrebu. SC (engl. *Surround Camera*) SoC se koristi za pogled u okružju vozila i podržava do šest kamera. FFN (engl. *Front view camera near angle stereoscopic view, Front view camera wide angle, Night vision camera*) je SoC opće namjene korišten za pogled od naprijed i podržava do četiri kamere. FUS (engl. *Fusion*) SoC služi za spajanje podataka iz drugih SoC-ova i ne koristi kamere.

Svaki SoC Alpha ploče sadrži dva ARM Cortex A15 procesora, dva ARM Cortex M4 procesora, dva C66x DSP procesora i četiri EVE procesora. Također, SoC-evi imaju pristup po 1.5GB DDR3 RAM memorije, jedan HDMI izlaz za prikaz video signala, UART sučelje pomoću kojeg se uspostavlja komunikacija između ploče i računala, JTAG sučelje za otklanjanje pogrešaka (engl. *debugging*) i MicroSD sučelje za učitavanje programskog rješenja na ploču. Pristup Ethernet

priključku imaju SC i FUS SoC-ovi i njime je moguće slati i primati podatke s računala. Blok dijagram arhitekture Alpha ploče prikazan je na slici 3.11.



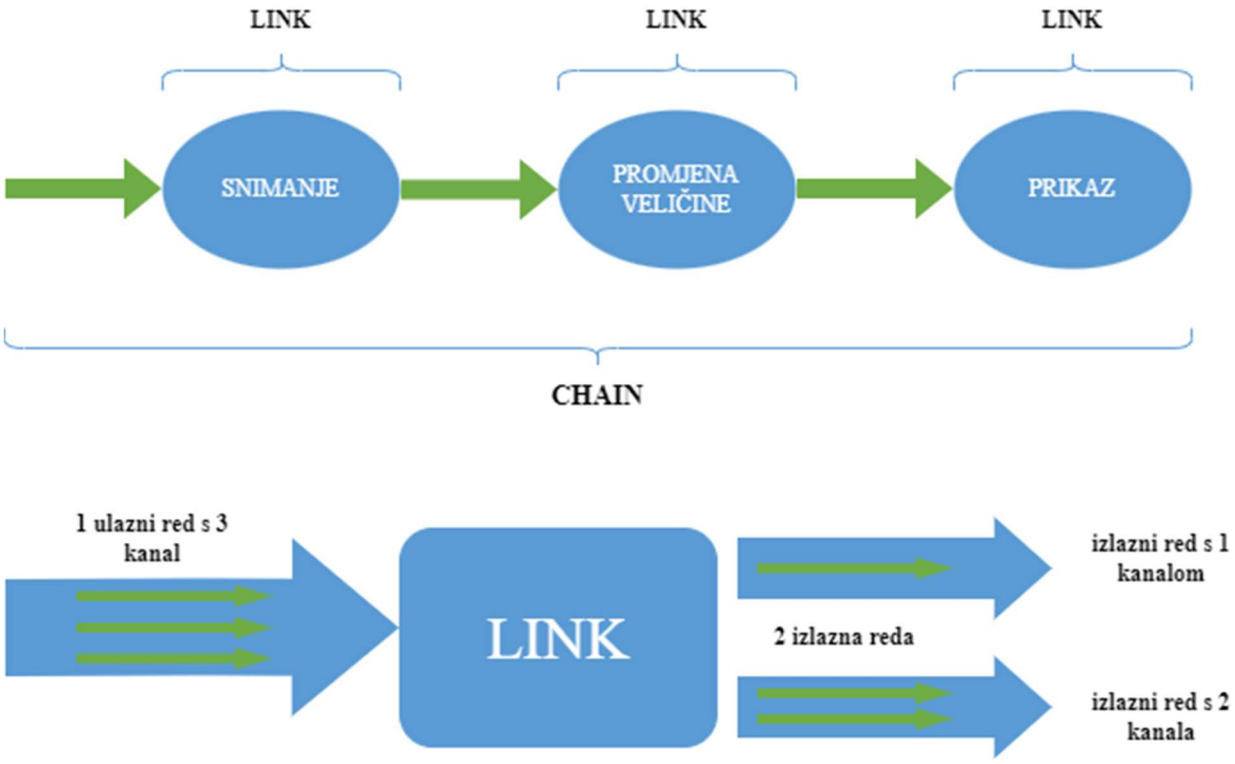
Slika 3.11 Blok dijagram arhitekture Alpha ploče [18]

3.4. Razvojno okruženje *Vision SDK*

Kako bi se na sklopovlje Alpha ploče implementiralo programsko rješenje, potrebno je koristiti razvojno okruženje *Vision SDK* firme Texas Instruments. Razvojno je okruženje napisano u programskom jeziku C te generira operacijski sustav zasnovan na Linux-u unutar binarne datoteke *MLO*. Također, prema uputama sadržanim u *.mk* datotekama generira sva korisnička programska rješenja u binarnu datoteku *AppImage* korištenjem *gmake* alata.

Vision SDK podržava višeprocesorski razvoj softvera za ADAS SoC-ove te pruža okvir (engl. *framework*) koji omogućuje stvaranje različitih tokova podataka u ADAS aplikacijama. Okvir razvojnog okruženja se temelji na konceptu „Veza i lanaca“ (engl. *Links and Chains*). Veza je osnovna jedinica obrade u protoku video podataka. Veza se sastoje od niti operacijskog sustava spojene s okvirom za poruke. Implementiranjem semafora na razini operacijskog sustava omogućeno je višenitno izvođenje pa se veze mogu izvoditi paralelno, svaka u svojoj niti. Okvir s

porukom povezan s vezom omogućuje korisničkoj aplikaciji kao i drugim vezama komunikaciju s tom vezom. Posebno je sučelje implementirano za vezu kako bi se drugim vezama omogućila razmjena video okvira i/ili tokova podataka s vezom. Pristup je vezama, kao i njihovo stvaranje, kontrola i međusobno povezivanje, pruženo Link API-ijem (engl. *Application Programming Interface* - API). Na taj način korisnik ne treba voditi brigu o detaljima međuprocesorske komunikacije niže razine. Veze složene serijski i/ili u paraleli čine lanac (engl. *Chain*) te predstavljaju gotovo programsko rješenje. Prikaz koncepta „Veza i lanaca“ dan je slikom 3.12.



Slika 3.12 Prikaz koncepta „Veza i lanaca“ [19]

Veze podatke međusobno izmjenjuju preko strukture koju *Vision SDK* naziva redom (engl. *Queue*). Svaka veza može imati proizvoljan broj ulaznih i izlaznih redova (slika 3.3). *Vision SDK* implementira redove koristeći podatkovnu strukturu red temeljen na nizu fiksne veličine. Element koji se umeće ili čita iz reda je sistemski spremnik (engl. *system buffer*). Sistemski spremnik je podatkovna struktura koja može imati više različitih tipova spremnika za različite vrste podataka. Neki od tipova spremnika su: spremnik video okvira (engl. *videoframe buffer*), spremnik toka podatka u binarnom obliku (engl. *bitstream buffer*) i spremnik metapodataka (engl. *metadata buffer*).

Osnovne veze dostupne u razvojnom okruženju *Vision SDK*:

- *Capture* - veza koja predstavlja senzor kamere platforme, omogućava dohvaćanje slike,

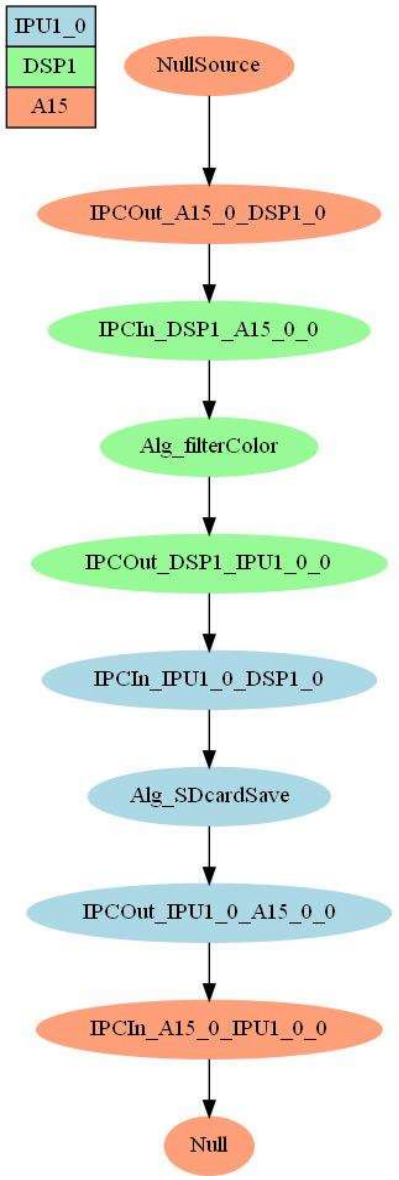
- *Display* - veza za slanje video spremnika na video izlaz HDMI i njegov prikaz,
- *NullSource* - veza koja dohvaća podatke s Ethernet sučelja u spremnik,
- *Null* - veza za preuzimanje spremnika podataka i slanje na Ethernet sučelje,
- *Dup* - veza čiji ulaz ima jedan red, a izlaz ima dva reda u kojima su kopije reference sistemskog spremnika,
- *Merge* - veza koji se koristi za spajanje 2 ulazna reda u jedan izlazni,
- *Alg_NameOfAlgo* - veza koja omogućava korištenje vlastitih i/ili unaprijed definiranih algoritama.

3.5. Opis programskog rješenja na ADAS razvojnoj platformi

U ovome potpoglavlju predstavljena je implementacija predloženog rješenja na ADAS razvojnu platformu. Koristeći iterativni postupak razvoja i implementacije koncepta rješenja na osobnom računalu uvelike se olakšalo implementiranje na ADAS razvojnu platformu. Pod konceptom rješenja se podrazumijeva osnovni tok izvođenja programa za filtriranje slike po boji predstavljen potpoglavljem 3.1.

3.5.1. Opis implementacije na ADAS razvojnu platformu

Implementacija na ADAS razvojnu platformu je izrađena u *Vision SDK* okruženju koristeći koncept „veze i lanaca“. Sama implementacija rješenja predstavljena je jednim lancem veza slučaja upotrebe (engl. *use case*). Dijagram toka za rješenje implementirano na ADAS razvojnu platformu dan je slikom 3.13. Na dijagramu su narančastom bojom prikazane veze koje se izvršavaju na procesoru A15, zelenom bojom se prikazuju veze čije se izvršavanje odvija na DSP1 procesoru, a plavom bojom veze za izvršavanje na IPU1 procesoru.



Slika 3.13 Dijagram toka rješenja implementiranog na ADAS razvojnoj platformi

Vezom *NullSource* se preko Ethernet sučelja dohvaća slika u sistemski spremnik toka podataka u binarnom obliku (engl. *bitstream buffer*). Time se omogućilo učitavanje svih vrsta zapisa slike, tj. omogućeno je čitanje RGB, YUV i HSV zapisa slike. Također je za *NullSource* vezu definirana rezolucija slike 1242x375 elemenata slike te je širina proširena za tri puta kako bi se mogle učitati sve tri komponente odabranog zapisa slike. U slučaju kada se koristi poduzorkovani YUV zapis YUYV potrebno je za širinu zadati proširenje od dva puta, dok je za poduzorkovani YUV zapis NV12 potrebno za širinu zadati proširenje od jedan i pola puta. Sljedeća veza je *Alg_filterColor* čiji je zadatak filtriranje slike po boji. Veza se sastoji od metode filtriranja slične metodi implementirane na osobnom računalu u programskom jeziku C koja je opisana u potpoglavlju 3.2.2. Jedina razlika je postavljanje granica pojedinih boja u trodimenzionalno polje `colors` tipa `Uint8` prikazano slikom 3.14. Budući da veza *Null* šalje podatke filtriranih slika po boji na

računalo i pakira ih u jednu datoteku, bilo je potrebno dodati vezu *SDcardSave*. Korištenjem veze *SDcardSave* omogućena je pohrana filtriranih slika po boji na SD karticu kao zasebne binarne datoteke.

Linija ***Kod***

```
1:      UInt8 colors[7][3][2] = {
2:           {{0, 255}, {91, 113}, {190, 255}},
3:           {{0, 255}, {0, 130}, {0, 120}},
4:           {{0, 216}, {124, 255}, {0, 125}},
5:           {{0, 255}, {0, 69}, {0, 173}},
6:           {{0, 255}, {55, 95}, {164, 240}},
7:           {{0, 255}, {151, 211}, {143, 190}},
8:           {{37, 237}, {120, 185}, {169, 255}}
9:      }
```

Slika 3.14 Prikaz kôda trodimenzionalnog polja `colors` s podatcima granica boja za RGB model boja

3.5.2. Optimizacija i paralelizacija rješenja

Po završetku implementacije rješenja na Alpha ploču i postizanja željene funkcionalnosti, slijedilo je optimiziranje implementacije u svrhu poboljšavanja brzine izvođenja i efikasnosti. Prva metoda optimizacije bila je razgradnja petlje (engl. *loop unrolling*) [21]. Metoda razgradnje petlje povećava brzinu izvođenja programa tako što se smanjuje broj potrebnih iteracije petlje. U jednoj iteraciji petlje dohvaća i obrađuje se što veći broj susjednih elemenata polja podataka. U slučaju RGB, HSV i YUV zapisa slike moguće je petlju iterirati za po tri elementa polja (slika 3.15).

Linija ***Kod***

```
1:      for(rowIdx = 0; rowIdx < height; rowIdx++){
2:        for(colIdx = 0; colIdx < wordWidth; colIdx += 3){
            if( !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 0],
3:      colors[iterator][0][0], colors[iterator][0][1])
            || !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 1],
4:      colors[iterator][1][0], colors[iterator][1][1])
            || !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 2],
5:      colors[iterator][2][0], colors[iterator][2][1])){
6:             inputPtr[rowIdx * wordWidth + colIdx + 0] = 0;
7:             inputPtr[rowIdx * wordWidth + colIdx + 1] = 0;
8:             inputPtr[rowIdx * wordWidth + colIdx + 2] = 0;
9:           }
10:       }
11:     }
```

Slika 3.15 Prikaz kôd nakon razgradnje petlje

Nakon što se izvršila razgradnja petlje kod poduzorkovanog YUV zapisa slike u YUYV formatu petlju je bilo moguće iterirati po četiri elementa polja podataka. Kod poduzorkovanog YUV zapisa u NV12 formatu se izvršila razgradnje unutarnje i vanjske petlje te se svaka petlja iterirala po dva elementa polja. Također je za pristupanje vrijednostima komponenti U i V kod NV12 zapisa bilo potrebno uvođenje pomaka (engl. *offset*) koji se u unutarnjoj petlji iterirao za dva. Ovim pristupom je omogućeno dohvaćanje šest elemenata polja podataka.

Sljedeći korak je bilo optimiziranje uvjeta `if` izraza o donošenju odluka pripadnosti elementa slike pojedinoj boji. Prijašnji uvjet se sastojao od tri poziva funkcije `isWithinBounds()` nad čijim su se povratnim vrijednostima vršile logičke operacije ILI (slika 3.16).

Linija ***Kod***

```
        if( !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 0],
1:      colors[iterator][0][0], colors[iterator][0][1])
            || !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 1],
2:      colors[iterator][1][0], colors[iterator][1][1])
            || !IsWithinBounds(inputPtr[rowIdx * wordWidth + colIdx + 2],
3:      colors[iterator][2][0], colors[iterator][2][1]))
```

Slika 3.16 Prikaz kôda neoptimiziranog `if` izraza o donošenju odluke

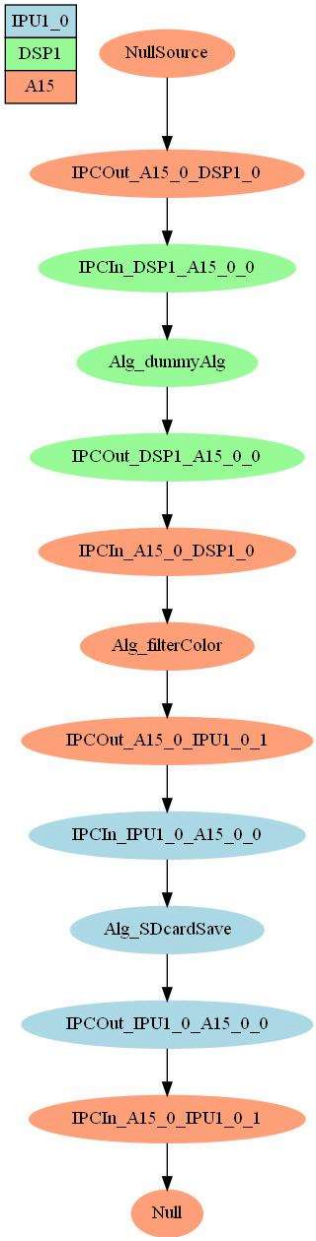
Koristeći koncept opisan u potpoglavlju 3.3. uvjet se sveo na jednu jednostavnu implicitnu jednadžbu s tri varijable koja opisuje podprostor boje. Slika 3.17 prikazuje kôd optimiziranog `if` izraza o donošenju odluke.

Linija ***Kod***

```
        if((Int32) (1350*inputPtr[rowIdx * wordWidth + colIdx + 0] -
1:      400*inputPtr[rowIdx * wordWidth + colIdx + 1] -
2:      190*inputPtr[rowIdx * wordWidth + colIdx + 2] - 239900)  < 0)
```

Slika 3.17 Prikaz kôda optimiziranog `if` izraza o donošenju odluke

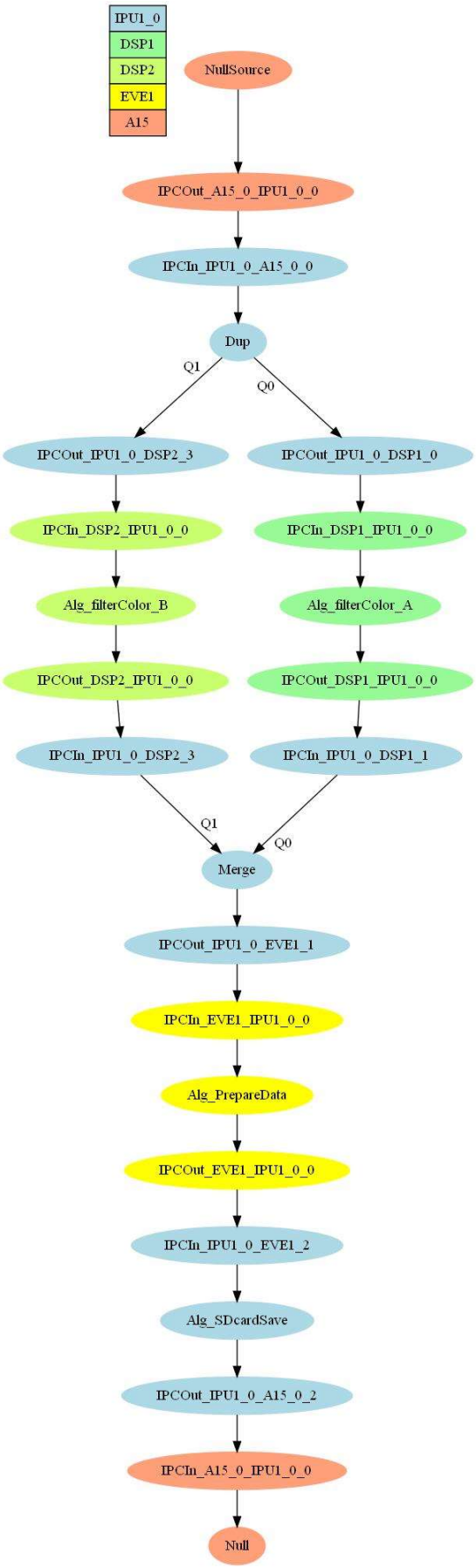
Po završetku algoritamske optimizacije pristupilo se raspoređivanju rješenja na više procesora, tj. paralelizaciji posla. Izvorno rješenje je vezu *Alg_filterColor* izvodilo na DSP1 procesoru. Po uzoru na to rješenje implementirano je rješenje koje vezu *Alg_filterColor* izvodi na A15 procesoru. Tok (slika 3.18) ove implementacije razlikuje se po dodavanju veze *Alg_dummyAlg*. Zadaća te veze bila je dohvatiti ulazni spremnika toka podataka sa veze *NullSource* i pripremiti ga za vezu *Alg_filterColor*.



Slika 3.18 Dijagram toka rješenja implementiranog na A15 procesor

Potreba za ovakvim rješenjem proizlazi iz činjenice da se veze *NullSource* i *Alg_filterColor* izvode na procesoru A15 pa te veze ne mogu međusobno izmijeniti sistemske spremnike zbog ograničenja u *cache* memoriji procesora.

Nadalje, raspodjela posla veze *Alg_filterColor* se izradila za DSP1 i DSP2 procesore. Dijagram toka (slika 3.20) je izmijenjen na način da se nakon veze *Nullsource* udvostručuje referenca sistemskog spremnika toka podataka pomoću veze *Dup*. Svaka referenca se predala zasebnoj vezi *Alg_filterColor* te se prema slici 3.21 obrada podatka sistemskog spremnika podijelila na jednake dijelove. Sa stajališta obrade je to značilo da je DSP1 procesor izvršavao filtriranje slike po boji za prvu, a DSP2 procesor filtriranje za drugu polovicu slike.



Slika 3.19 Dijagram toka rješenja implementiranog na DSP1 i DSP2 procesore

Linija Kod

```
1:      if(System_getSelfProcId() == SYSTEM_PROC_DSP1){
2:        filterRGB((UInt8*)(inPtr), width, 187);
3:      }
4:      else if(System_getSelfProcId() == SYSTEM_PROC_DSP2){
5:        filterRGB((UInt8*)(((UInt8*)inPtr) + 187*3*width), width, 188);
6:      }
```

Slika 3.20 Prikaz kôd za raspodjelu obrade na DSP1 i DSP2 procesor

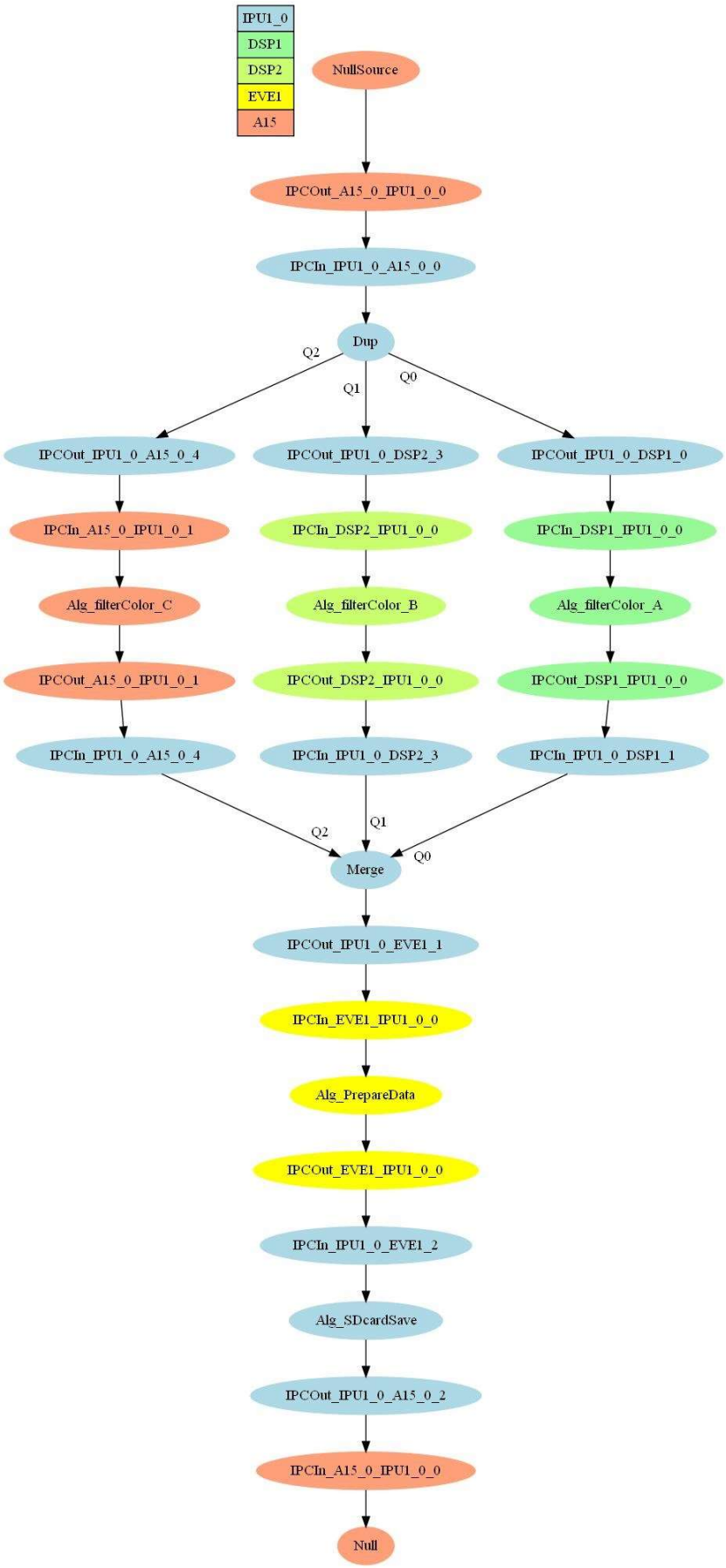
Nakon izvršavanja obrade slike potrebno je bilo pomoću veze *Merge* pohraniti rezultate filtriranja iz dva reda u jedan red. Ti rezultati su spremljeni u sistemski kompozitni spremnik i potrebno je korištenjem veze *Alg_PrepareData* rezultat spremiti u sistemski spremnik toka podataka za daljnje korištenje. Ovakvom paralelizacijom posla se očekuje dvostruko brže izvođenje od izvođenja jednim DSP procesorom.

Daljnja paralelizacija je ujedno bila i konačna. Raspodjela posla veze *Alg_filterColor* se izradila za DSP1 i DSP2 procesore i A15 procesor. Po uzoru na implementaciju od dva DSP procesore, korištene su veze *Dup* i *Merge* za pripremu podataka za veze *Alg_filterColor*, a veza *Alg_PrepareData* sprema rezultate za daljnju upotrebnu. Dijagram toka ove implementacije prikazan je slikom 3.22 te je slikom 3.21 prikazan kôd raspodjele obrade na sva tri procesora. Svaki je procesor obrađivao dio slike, A15 procesor je obrađivao elemente slike do visine slike od 209, DSP1 procesor je obrađivao elemente slike od visine slike 209 do 292, a DSP2 procesor je obrađivao elemente slike preostalog dijela slike. To znači da je A15 procesor obradio 56% slike, dok je svaki DSP procesor obradio 22% slike. Ovom podjelom je omogućeno jednako vrijeme izvođenja svakog procesora.

Linija Kod

```
1:      if(System_getSelfProcId() == SYSTEM_PROC_DSP1){
2:        filterRGB((UInt8*)(((UInt8*)inPtr) + 292*3*width), width, 83);
3:      }
4:      else if(System_getSelfProcId() == SYSTEM_PROC_DSP2){
5:        filterRGB((UInt8*)(((UInt8*)inPtr) + 209*3*width), width, 83);
6:      }
7:      else if(System_getSelfProcId() == SYSTEM_PROC_A15_0)
8:        filterRGB((UInt8*)(inPtr), width, 209);
        }
```

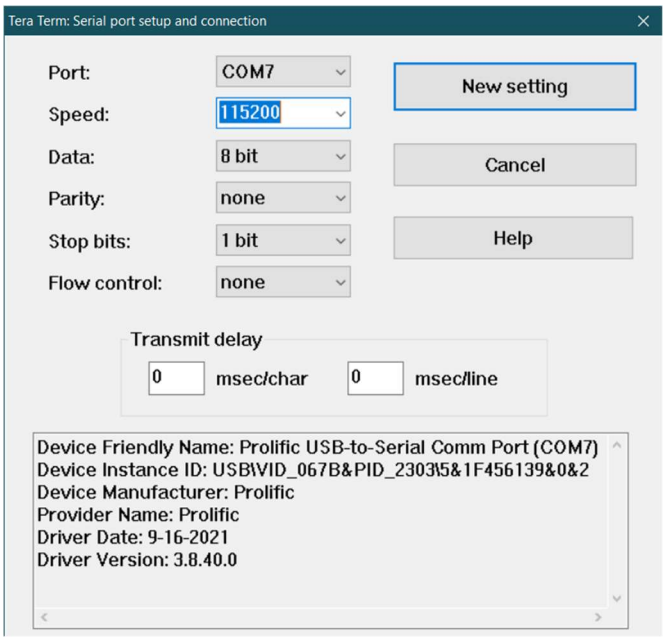
Slika 3.21 Prikaz kôd za raspodjelu obrade na A15, DSP1 i DSP2 procesor



Slika 3.22 Dijagram toka rješenja implementiranog na A15, DSP1 i DSP2 procesore

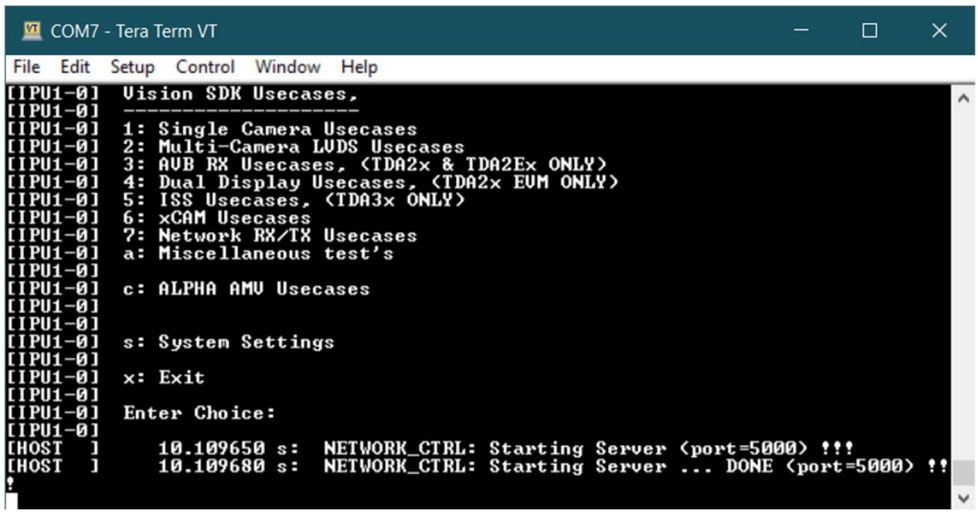
3.5.3. Pokretanje algoritma na ADAS razvojnoj platformi

Prilikom pokretanja algoritma na ADAS razvojnoj platformi potrebno je koristiti alat *network_tx* kako bi se s osobnog računala poslale slike na ploču. *Tera Term* [22] je program koji se koristi za komunikaciju između USB sučelja osobnog računala i UART sučelja ploče. Nakon što se USB na UART adapter priključi iz ploče na računalo potrebno je postaviti postavke komunikacije unutar *Tera Term*-a (slika 3.23).



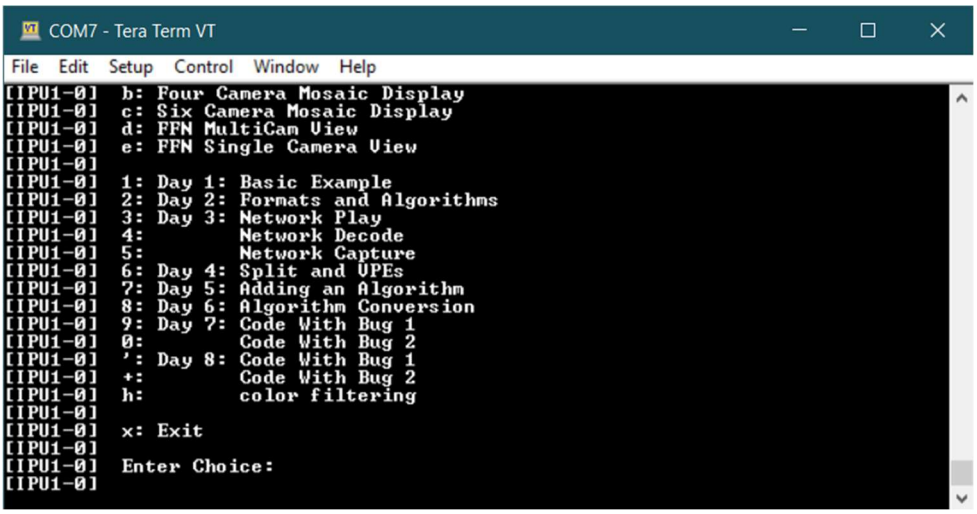
Slika 3.23 Prikaz postavki USB sučelja za komunikaciju računala i ploče

Uključivanjem napajanja ploče, uz uvjet da su na microSD kartici postavljene binarne datoteke *AppImage* i *MLO*, unutar *Tera Term* alata otvara se izbornik sa svim korisničkim slučajevima upotrebe (slika 3.24). Potrebno je odabrati opciju *APLHA AMV Usecases* pritiskom tipke „*c*“.



Slika 3.24 Prikaz izbornika svih korisničkih slučajeva upotrebe

Odabirom opcije *APLHA AMV Usecases* se otvara novi izbornik koji prikazuje korisničke slučajeve upotrebe razvijane za Alpha ploču (slika 3.25). Odabirom opcije „*h*“ pokreće se korisnički slučaj upotrebe za filtriranje slike po boji.



Slika 3.25 Prikaz izbornika korisnički rješenja na Alpha ploči

Kako bi se podatci mogli s računala poslati na ploču, potrebno je pozicionirati se u mapu „*C:\VISION_SDK_02_12_01_00\vision_sdk\tools\network_tools\bin*“ unutar terminala i pokrenuti naredbu za slanje slika. Naredba za slanje podataka na ploču je „*network_tx --host_ip 192.168.10.1 --target_ip 192.168.10.2 --no_loop --files naziv.bin*“. Po završetku se izvođenja rezultati filtriranja slike po boji nalaze na microSD kartici u binarnim datotekama.

4. TESTIRANJE PREDLOŽENOG RJEŠENJA ZA FILTRIRANJE SLIKE PO BOJI

U ovome poglavlju su predstavljeni rezultati testiranja predloženog rješenja za filtriranje slike po boji. Najprije je opisan skup slika korišten za testiranje nakon čega je predstavljen način provođenja testiranja uz vrednovanje brzine izvođenja pojedinih implementacija rješenja. Opisana je evaluacija podataka s obzirom na ispravnost filtriranja slike po boji za pojedine implementacije. Zatim su prikazani rezultati za osnovnu i optimiziranu implementaciju rješenja na osobnom računalu. Također su predstavljeni rezultati optimiziranog rješenja u slučaju implementacije koja koristi više dostupnih procesora unutar SoC-a na ADAS razvojnoj ploči. Na kraju je dana rasprava postignutih rezultate brzine izvođenja implementiranih rješenja.

4.1. Opis skupa testnih slika i načina provođenja testiranja

Skup slika korišten u testiranju rada predloženog rješenja za filtriranje slike po boji sastoji se od 50 slika prikupljenih s „*KITTI*“ javne baze podataka [23]. Slike „*KITTI*“ baze podataka su snimljene tijekom vožnje i sadržavaju scene iz prometa u gradu, u naseljima i na otvorenoj cesti. Slike su u boji, rezolucija slike je 1242x375 i spremljene su kao osam bitne PNG datoteke korištenjem kompresije bez gubitaka. Slike „*KITTI*“ baze podataka su već obrađene na način da se izrezala hauba motora te područje slike koje je predstavljalo nebo. Kako bi se dobio uvid u brzinu izvođenja pojedine implementacije predloženog rješenja, skup testnih slika je proširen istovjetnim slikama veće (1862x562) i manje rezolucije (622x188). Nadalje, sve slike su iz PNG datoteka zapisane u zasebne binarne datoteke za formate zapisa RGB, HSV, YUV, YUYV i NV12. Ukupni testni skup od 150 slika nalazi se u elektroničkom prilogu P. 4.1. Tri primjera testnih slika dan je slikom 4.1.

a)

b)

c)

Slika 4.1 Primjeri testnih slika; a) scena prometa u gradu, b) scena prometa u naselju, c) scena prometa na otvorenoj cesti

Testiranje je provedeno na osobnom računalu i Alpha ploči. Skup testnih slika se koristio kako bi se za sve implementacije rješenja izmjerila brzina izvođenja. Za provođenje testiranja je odabrano filtriranje slike po crvenoj boji jer se za crvenu boju odrađuje najveći broj operacija. Potreba za većim brojem operacija naspram ostalih boja proizlazi iz filtriranja slike po crvenoj boji u HSV zapisu. Filtriranja slike u HSV zapisu zahtjeva provjeru dva uvjeta o pripadnosti elementa slike crvenoj boji. Osobno računalo, specifikacija: procesor Intel® Core™ i5-9300H CPU @ 2.40GHz, RAM memorija 16GB DDR4, PCIe NVMe SSD kapaciteta 512GB i grafička kartica NVIDIA Geforce GTX 1650, korišteno je za testiranje osnovne i optimizirane implementacije u programskom jeziku C za formate zapisa RGB, HSV, YUV, YUYV i NV12. Vrijeme izvođenja se dohvaćalo pomoću *time.h* biblioteke metodom *clock()*.

Mjerenje brzine izvođenja na Alpha ploči učinjeno je za implementaciju koja koristi jedan DSP procesor, za implementaciju koja koristi A15 procesor, za implementaciju koja koristi dva DSP procesora te za implementaciju koja koristi dva DSP procesora i A15 procesor. Vrijeme izvođenja veze algoritma (*Alg_filterColor*) u tim implementacijama dohvaćalo se pomoću metode *Utils_getCurTimeInMsec()* i putem serijskog sučelja slalo na osobno računalo. Za svaku su se sliku

pojedinog formata zapisa rezultati vremena izvođenja zapisivali u *log* datoteku koja se korištenjem *Python* skripte na osobnom računalu izdvojila u tablice.

4.2. Testiranje ispravnosti rješenje za filtriranje slike po boji

Za testiranje ispravnosti rješenja za filtriranje slike po boji korištene su granice boja dobivene alatom opisane u potpoglavlju 3.1. Ispravnost pojedinih implementacija rješenja se provjeravala usporedbom rezultata filtriranja slike po boji svake od implementacija. Najprije je odrađeno testiranje ispravnosti između implementacije u programskom jeziku *Python* i implementacije u programskom jeziku C na osobnom računalu. Za svaku se sliku rezultata filtriranja slike po određenoj boji obavljalo oduzimanje. U slučaju da je rezultat oduzimanja crna slika, implementacije se smatraju točnima, odnosno rezultati rješenja na osobnom računalu se podudaraju. Nadalje, tako usklađene implementacije na osobnom računalu su se uspoređivale s implementacijom na Alpha ploči. Oduzimanjem slike rezultata filtriranja slike po određenoj boji implementacije u programskom jeziku C i implementacije na Alpha ploči je utvrđeno da je rješenje implementacije na Alpha ploči ispravno. Prikaz nekoliko rezultata filtriranja slike po boji dan je slikama 4.1, 4.2 i 4.3.

a)

b)

c)

d)

Slika 4.2 Primjer rezultata filtriranja slike po žutoj boji; a) ulazna slika, b) rezultat za RGB zapis, c) rezultat za HSV zapis, d) rezultat YUV zapisa

a)

b)

c)

d)

Slika 4.3 Primjer rezultata filtriranja slike po crvenoj boji; a) ulazna slika, b) rezultat za RGB zapis, c) rezultat za HSV zapis, d) rezultat YUV zapisa

a)

b)

c)

d)

Slika 4.4 Primjer rezultata filtriranja slike po plavoj boji; a) ulazna slika, b) rezultat za RGB zapis, c) rezultat za HSV zapis, d) rezultat YUV zapisa

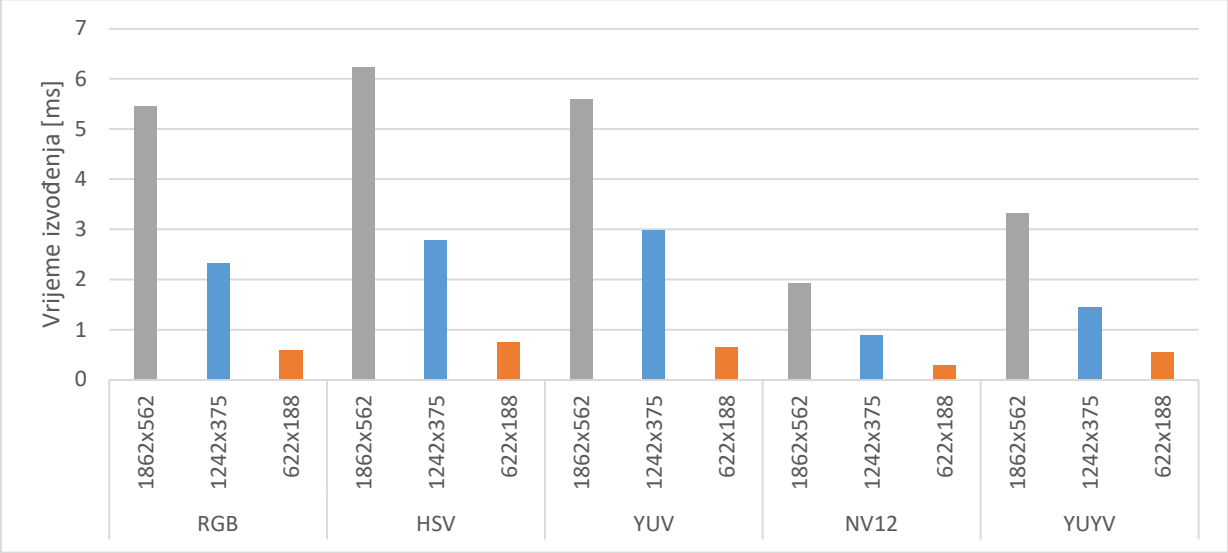
4.3. Rezultati testiranja brzine izvođenja rješenja implementiranih na osobnom računalu i na ADAS razvojnoj ploči

U ovom su potpoglavlju predstavljeni rezultati testiranja brzine izvođenja implementiranih na osobnom računalu i na ADAS razvojnoj platformi. Najprije su prikazani rezultati za osnovnu i optimiziranu implementaciju rješenja na osobnom računalu u programskom jeziku C. Zatim su predstavljeni rezultati optimizirane implementacije rješenja na jednom DSP odnosno A15 procesoru Alpha ploče. Rezultati optimizirane implementacije na dva DSP procesora su sljedeći prikazani, a na kraju su dani rezultati implementacije rješenja na dva DSP procesora i A15 procesor Alpha ploče.

4.3.1. Osnovna i optimizirana implementacija rješenja na osobnom računalu

Rezultati osnovne implementacije rješenja na osobnom računalo u C programskom jeziku za filtriranje slike po boji svih zapisa dani su u elektroničkom prilogu P. 4.2.

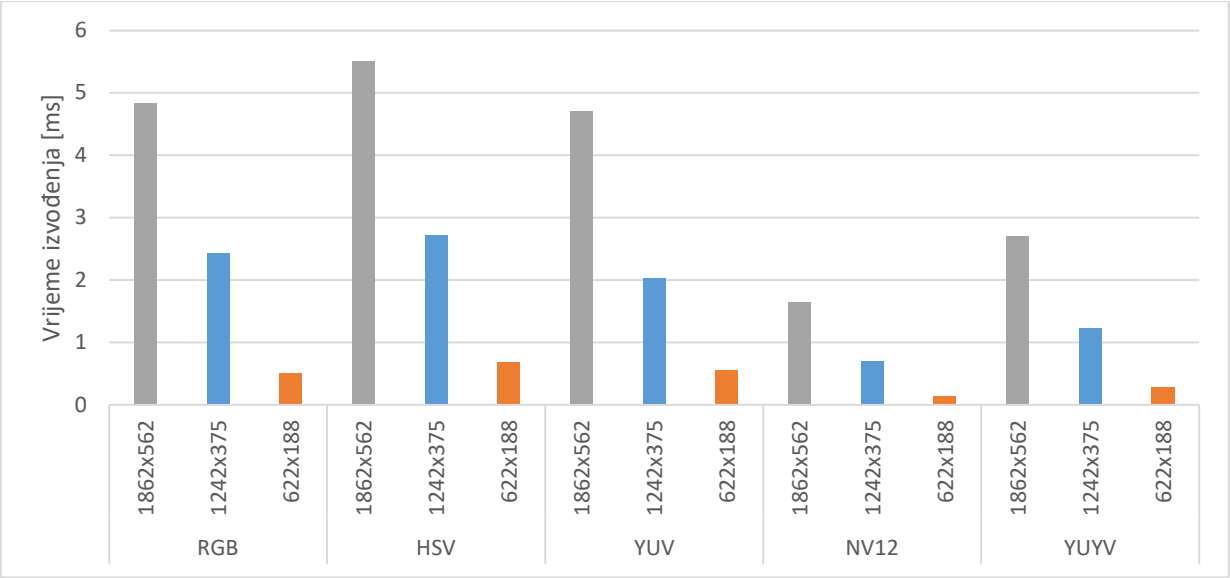
Ovim testiranjem se potvrdilo da je vrijeme izvođenja najveće za slike većih dimenzija zbog veće količine elemenata slike koje mora obraditi. Ovo rješenje je pokazalo najbolje rezultate filtriranja za NV12 format zapisa, a razlog tome je manji broj elemenata slike nastao poduzorkovanjem. Najsporije se izvodi filtriranje HSV zapisa jer kod toga zapisa je bilo potrebno izvoditi dvije provjere granica za svaki element slike. Filtriranje slike po boji u RGB zapisu je brže od YUV zapisa zbog toga što RGB prostor boja čini manji dio YUV prostora boja te za opis pojedinih boja u RGB zapisu treba manji raspon vrijednosti. Slikom 4.5 prikazano je srednje vrijeme izvođenja svih zapisa boja.



Slika 4.5 Prikaz srednjeg vremena izvođenja svih zapisa slike kod osnovne implementacije rješenja na PC-u

Rezultati optimizirane implementacije rješenja na osobnom računalo u C programskom jeziku za filtriranje slike po boji svih zapisa dani su u elektroničkom prilogu P. 4.3.

Uvođenjem optimizacijskih rješenja opisanih u potpoglavlju 3.5.2. smanjeno je vrijeme izvođenja u odnosu na izvođenje osnovne implementacije rješenja. HSV zapis se izvodi najsporije u svim rezolucijama jer je potrebno obavljati dvije provjere uvjeta pripadnosti elementa slike boji. Uspoređujući vrijeme izvođenja RGB i YUV zapisa vidljiva je promjena u odnosu na osnovnu implementaciju. Vrijeme izvođenja za RGB je veće od vremena izvođenja YUV zapisa osim u najmanjoj rezoluciji. YUV zapis je imao za najmanju rezoluciju veći broj izvođenja od jedne milisekunde. YUYV format zapisa je imao brže vrijeme izvođenja od YUV zapisa što proizlazi iz jedan i pola puta manje elemenata slike u YUYV formatu. Najbrže vrijeme izvođenja se postiglo za NV12 format zapisa koji ujedno ima i najmanji broj elemenata slike. Slikom 4.6 prikazano je srednje vrijeme izvođenja svih zapisa boja kod optimizirane implementacije rješenja na osobnom računalu.

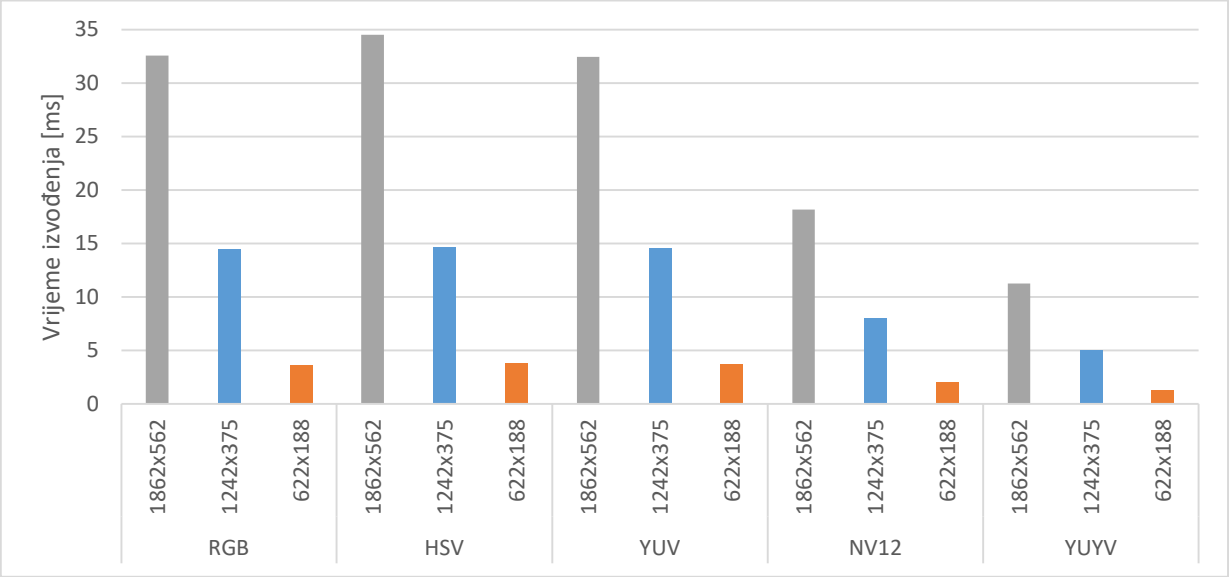


Slika 4.6 Prikaz srednjeg vremena izvođenja svih zapisa slike kod optimizirane implementacije rješenja na PC-u

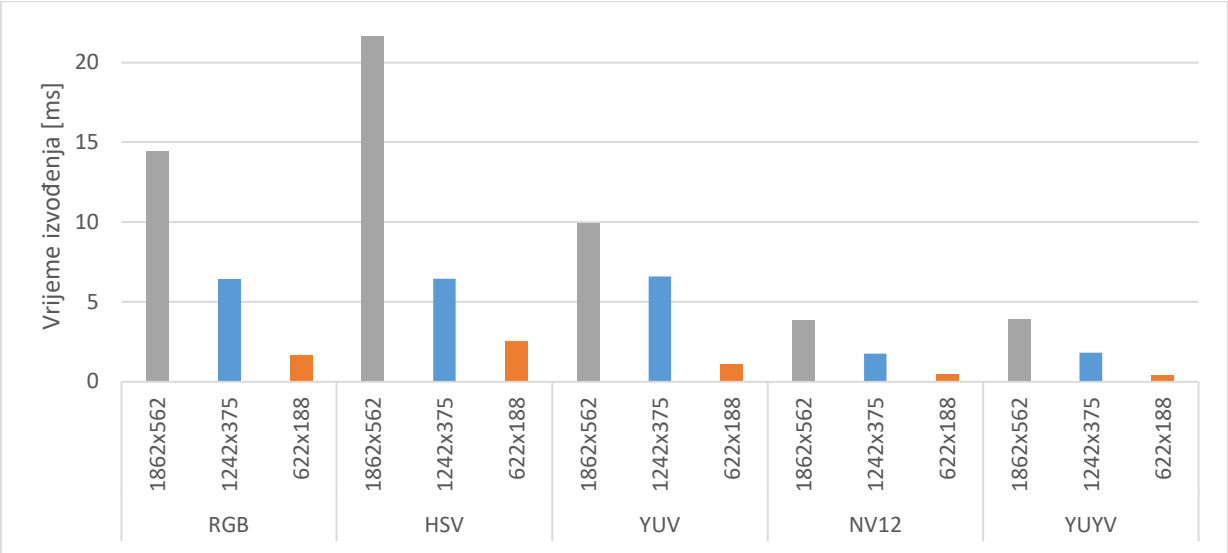
4.3.2. Optimizirana implementacija rješenja na jednom DSP odnosno A15 procesoru Alpha ploče

Rezultati optimizirane implementacije rješenja na DSP procesoru Alpha ploče za filtriranje slike po boji u svih zapisa dani su u elektroničkom prilogu P. 4.4, a rezultati optimizirane implementacije rješenja na A15 procesoru Alpha ploče dani su u elektroničkom prilogu P. 4.5.

Rezultati optimizirane implementacije rješenja na DSP procesoru Alpha ploče za filtriranje slike po boji za sve zapise dani su slikom 4.7, a rezultati optimizirane implementacije rješenja na A15 procesoru slikom 4.8.



Slika 4.7 Prikaz srednjeg vremena izvođenja svih zapisa slike kod optimizirane implementacije rješenja na DSP procesoru Alpha ploče



Slika 4.8 Prikaz srednjeg vremena izvođenja svih zapisa slike kod optimizirane implementacije rješenja na A15 procesoru Alpha ploče

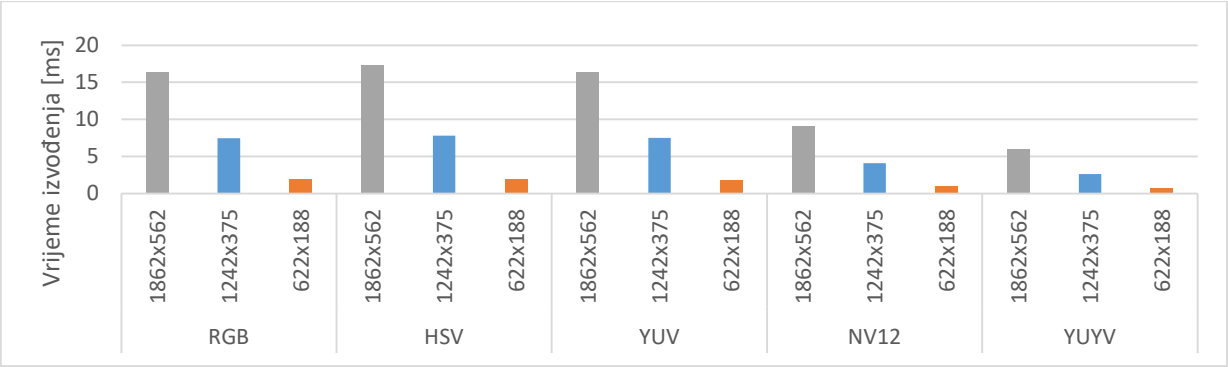
Rješenje na jednom DPS procesoru je imalo približno ista vremena izvršavanja za RGB, HSV i YUV zapise boja te iznosilo oko 14.5 milisekundi za rezoluciju 1242x375. U usporedbi s optimiziranom implementacijom rješenja na osobnom računalu za RGB zapis vrijeme izvođenja je šest puta sporije, za HSV zapis je pet, a za YUV zapis sedam puta sporije. Za razliku od optimiziranog rješenja na osobnom računalu, YUYV zapis boja je imao kraće vrijeme filtriranja slike po boji od NV12 zapisa što je direktno rezultat manje količine *cache* memorije SoC-ova Alpha ploče. Vrijeme izvođenja YUYV zapisa je 5 milisekunde dok je kod NV12 zapisa brzina izvođenja 8.04 milisekundi za rezoluciju 1242x375.

Uspoređujući rezultate rješenja za A15 procesor s rezultatima rješenja za DSP procesor može se utvrditi brže izvođenje za sve formate zapisa i svih dimenzija. Vremena izvođenja za slike rezolucije 1242x375 su 6.5 milisekundi za RGB, HSV i YUV zapis, 1.76 milisekundi za NV12 i 1.82 milisekunde za YUYV zapis. To je proizašlo iz činjenice da je brzina A15 procesora 750 MHz, a brzina DSP procesora 600 MHz [18]. Još jedan razlog je i različita arhitektura procesora zbog koje je DSP procesor efikasniji u radu sa podatcima sa pomičnim zarezom (engl. *floating-point*) [24].

4.3.3. Optimizirana implementacija rješenja na dva DSP procesora Alpha ploče

Rezultati optimizirane implementacije rješenja na dva DSP procesora Alpha ploče za filtriranje slike po boji svih zapisa dani su u elektroničkom prilogu P. 4.6.

Rezultati optimizirane implementacije rješenja na dva DSP procesora Alpha ploče za filtriranje slike po boji za sve zapise dani su slikom 4.9.



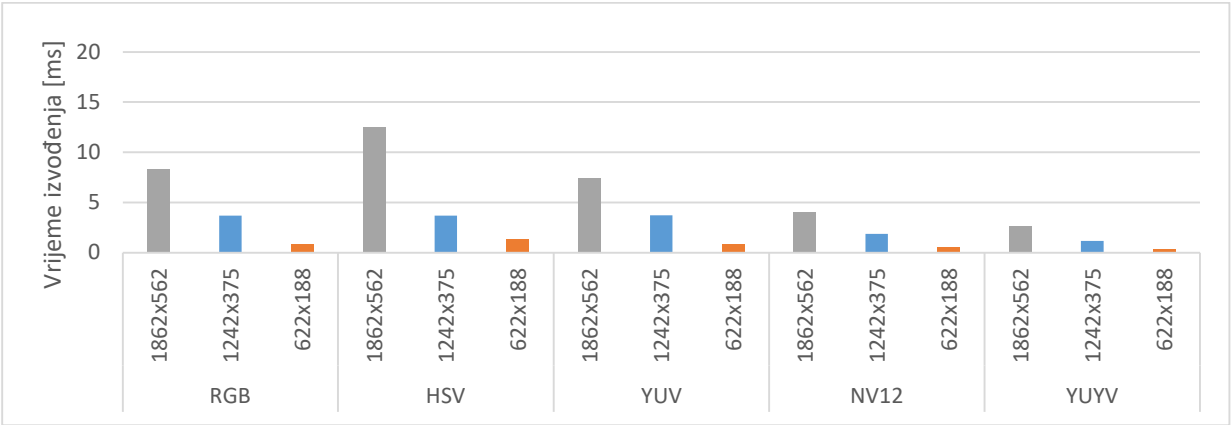
Slika 4.9 Prikaz srednjeg vremena izvođenja svih zapisa slike kod optimizirane implementacije rješenja na dva DSP procesora Alpha ploče

Ovo je rješenje je dvostruko brže od rješenja na jednom DSP procesoru te se svojom brzinom izvođenja približio izvođenju rješenja na A15 procesoru. Koristeći ovo saznanje pristupilo se daljnjem razvijanju rješenja na dva DSP i A15 procesoru čiji se rezultati nalaze u sljedećem potpoglavlju.

4.3.4. Optimizirana implementacija rješenja na dva DSP procesora i A15 procesor Alpha ploče

Rezultati optimizirane implementacije rješenja na dva DSP procesora i A15 procesoru Alpha ploče za filtriranje slike po boji svih zapisa dani su u elektroničkom prilogu P. 4.7.

Rezultati optimizirane implementacije rješenja na dva DSP procesora i A15 procesoru Alpha ploče za filtriranje slike po boji za sve zapise dani su slikom 4.10.



Slika 4.10 Prikaz srednjeg vremena izvođenja svih zapisa slike kod optimizirane implementacije rješenja na dva DSP procesora Alpha ploče

Implementacija filtriranja slike po boji na tri procesora postiglo je vrijeme izvođenja od 3.7 milisekundi za RGB, HSV i YUV zapis, vrijeme od 1.86 milisekundi za NV12 zapis i 1.14 milisekundi za YUYV zapis. Usporedbom rezultata vremena izvođenja na jednom DSP procesoru

dobilo se ubrzanje od četiri puta, odnosno ubrzanje od dva puta naspram rezultata izvođenja na A15 procesoru.

4.4. Osvrt na dobivene rezultate

Paralelizacijom posla optimizirane implementacije rješenja na tri procesora Alpha ploče ostvarena je brzina izvođenja filtriranja slike po pojedinoj boji od 270 okvira u sekundi za RGB, HSV i YUV zapis, 537 okvira u sekundi za NV12 zapis i 877 okvira u sekundi za YUYV zapis. Daljnjom je raspodjelom filtriranja slike po pojedinoj boji na više SoC-ova Alpha ploče moguće postizanje boljih rezultata izvođenja. Naime, u implementaciji se rješenja ovog rada nije izvodila navedena raspodjela zbog manjkave dokumentacije vezane uz komunikaciju SoC-ova ploče preko PCIe sučelja.

S obzirom da je u optimizaciji rješenja korištena metoda Lagrange-ove aproksimacije uvedene su određene pogreške u ispravnost filtriranja slike po boji (slika 4.11). Te pogreške su svjesno unesene radi ubrzavanja izvođenja algoritma. Moguće je odraditi bolju aproksimaciju uvođenjem jednadžbi višeg stupnja Lagrange-ovom interpolacijom za funkcije s više varijabli. Uvođenjem jednadžbe većeg stupnja postići će se manji broj pogrešno filtriranih elemenata slike, ali za posljedicu će imati smanjenje vremena izvođenja.

a)

b)

Slika 4.11 Primjer uvođenja pogreške u rezultat Lagrange-ovom aproksimacijom; a) rezultat filtriranja slike po boji bez aproksimacije, b) rezultat filtriranja slike po boji s aproksimacijom

5. ZAKLJUČAK

Jedan od ciljeva automobilske industrije je razvijanje sustava koji će omogućiti autonomnu vožnju. Takvi sustavi su uvelike ovisni o obradi slike. Jedan od osnovnih algoritama obrade slike je prepoznavanje i praćenje određenih objekata, a u rješavanju navedenih zadataka često se koristi filtriranje slike po boji. U radu je predloženo rješenje koje za različite prostore boja filtrira sliku po određenoj boji. Rješenje se temelji na metodama obrade elemenata slike gdje se za svaki element slike provjerava njegova pripadnost podprostoru koji predstavlja određenu boju unutar zadanog prostora boja. Izvršen je pronalazak tih podprostora za sedam jedinstvenih boja unutar tri različita prostora boja. Najprije je predloženo rješenje implementirano na osobnom računalu u programskom jeziku *Python* i programskom jeziku C. Korištenjem eksperimentalno određenih opisa podprostora i Lagrange-ove interpolacije za funkcije s više varijabli izvršena je optimizacija implementacije rješenja na osobnom računalu, a koja je onda implementirana na ADAS razvojnu platformu. U daljnjoj optimizaciji kreiranog rješenje iskoristili su se dostupni procesori SoC-a Alpha ploče. Najprije je provedena evaluacija pojedine implementacije s obzirom na ispravnost filtriranja slike po boji. Zatim je izvršeno testiranje uz vrednovanje brzine izvođenja rješenja na 50 slika u tri različite rezolucije za pet formata zapisa slike. Rezultati brzine izvođenja rješenja optimiziranog razgradnjom petlji i pojednostavljenjem uvjeta izraza o donošenju odluka ukazuju da je ovo rješenje pogodno za izvođenje u sustavima stvarnog vremena. YUYV format zapisa se pokazao kao najprikladniji sa stajališta brzine za filtriranje slike po boji te je imao srednje vrijeme izvođenja od 1.14 milisekunde.

LITERATURA

[1] „The 6 Levels of Vehicle Autonomy Explained | Synopsys Automotive“, dostupno na: https://www.synopsys.com/automotive/autonomous-driving-levels.html (pristupljeno 14. lipnja 2021.)

[2] „What is ADAS?“, dostupno na: https://www.synopsys.com/automotive/what-is-adas.html (pristupljeno 14. lipnja 2021.)

[3] „What is Python? Executive Summary“, dostupno na: https://www.python.org/doc/essays/blurb (pristupljeno 14. lipnja 2021.))

[4] „About“., dostupno na: https://opencv.org/about (pristupljeno 14. lipnja 2021.))

[5] „What is NumPy? — NumPy v1.19 Manual“, dostupno na: https://numpy.org/doc/stable/user/whatisnumpy.html (pristupljeno 14. lipnja 2021.))

[6] „Color: Color models and color spaces“, dostupno na: https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html (pristupljeno 18. studenog 2021.)

[7] N. Ibraheem, M. Hasan, R. Z. Khan, „Understanding Color Models: A Review“, Research Gate, 2012, dostupno na: https://www.researchgate.net/publication/266462481_Understanding_Color_Models_A_Review (pristupljeno 18. studenog 2021.)

[8] „Adobe digital imaging solutions: Adobe RGB (1998) color image encoding“, Adobe, dostupno na: https://www.adobe.com/digitalimag/adobergb.html (pristupljeno 18. studenog 2021.)

[9] „IEC 61966-2-1:1999: Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space – sRGB“, International Electrotechnical Commission, dostupno na: https://webstore.iec.ch/publication/6169 (pristupljeno 18. studenog 2021.)

[10] „Intel® Integrated Performance Primitives Developer Reference: Color Models“, Intel, 2021, dostupno na: https://www.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference/top/volume-2-image-processing/image-color-conversion/color-models.html (pristupljeno 18. studenog 2021.)

[11] „Intel® Integrated Performance Primitives Developer Reference: Image Downsampling“, Intel, 2021, dostupno na: https://www.intel.com/content/www/us/en/develop/documentation/ipp-dev-reference/top/volume-2-image-processing/image-color-conversion/image-downsampling.html#image-downsampling (pristupljeno 18. studenog 2021.)

[12] „YUV pixel formats“, dostupno na: https://www.fourcc.org/yuv.php (pristupljeno 18. studenog 2021.)

[13] „About YUV formats“, GitHub, dostupno na: https://gist.github.com/Jim-Bar/3cbba684a71d1a9d468a6711a6eddbeb (pristupljeno 18. studenog 2021.)

[14] M. Diaz-Cabrera, P. Cerri, P. Medici, „Robust real-time traffic light detection and distance estimation using a single camera“, Expert Systems with Applications, br. 8, sv. 48, str. 3911-3923, svibanj 2015.

[15] M. Omachi and S. Omachi, "Detection of traffic light using structural information," IEEE 10th INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING PROCEEDINGS, str. 809-812, 2010, dostupno na: https://ieeexplore.ieee.org/document/5655932

[16] Xu, X., Jin, J., Zhang, S., Zhang, L., Pu, S., & Chen, Z., „Smart data driven traffic sign detection method based on adaptive color threshold and shape symmetry“, Future Generation Computer Systems, sv. 94, str. 381-391., 2019.

[17] Su, Feng, and Gu Fang, „Chroma based colour enhancement for improved colour segmentation.“, 9th International Conference on Sensing Technology (ICST), str. 162-167, 2015.

[18] „Automotive Machine Vision ALPHA reference board on Texas Instruments SoCs“, dostupno na: https://www.rt-rk.com/download/rt-rk_ALPHA_ADAS_board.pdf (pristupljeno 20. studenog 2021.)

[19] M. Vranješ, D. Vajak, »Predlošci s laboratorijskih vježbi iz kolegija „Digitalna obrada slike i videa za autonomna vozila“, vježbe „Upoznavanje s ADAS razvojnom pločom“ i „Izrada vlastitog use-case-a za ADAS razvojnu ploču“«

[20] A. Lewis, „Multivariate Lagrange Interpolation“, dostupno na: https://lewisacplu11.files.wordpress.com/2010/09/alex-lewis4.pdf (pristupljeno 20. studenog 2021.)

[21] A.Fog, „Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms“, dostupno na: https://www.agner.org/optimize/optimizing_cpp.pdf (pristupljeno 20. studenog 2021.)

[22] „Tera Term - Terminal Emulator for Windows“ dostupno na: https://ttssh2.osdn.jp/ (pristupljeno 20. studenog 2021.)

[23] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, „The KITTI Vision Benchmark Suite“, dostupno na: http://www.cvlibs.net/datasets/kitti/ (pristupljeno 20. studenog 2021.)

[24] P. Moakes, „Floating-point multiprocessing with C66x DSPs from Texas Instruments“, dostupno na: http://signal-processing.mil-embedded.com/articles/floating-point-multiprocessing-c66x-dsps-texas-instruments/ (pristupljeno 5. prosinca 2021.)

SAŽETAK

Ovaj rad se bavi metodom filtriranja slike po boji u različitim prostorima boje za slike dohvaćene kamerom s pogledom od naprijed. Cilj ovog rada je bila implementacija metode filtriranja slike po boji u različitim prostorima boje na ugradbenu ADAS razvojnu platformu. Najprije je predstavljena teorijska podloga za metodu koja se koristila u ovom radu uz pregled postojećih rješenja. Metoda se temelji na obradi elemenata slike gdje se za svaki element slike provjerava njegova pripadnost podprostoru koji predstavlja određenu boju unutar zadanog prostora boja. Sljedeći korak u radu je bila izrada koncepta rješenja u *Python* programskom jeziku korištenjem biblioteke *OpenCV* na osobnom računalu. Zatim se rješenje implementiralo i optimiziralo u programskom jeziku C na osobnom računalu. Nadalje, to optimizirano rješenje se implementiralo na ADAS razvojnu platformu unutar *Vision SDK* razvojnog okruženja programskog jezika C. Zatim je izvršena evaluacija implementacija s obzirom na ispravnost filtriranja slike po boji na skupu slika u tri različite rezolucije za pet formata zapisa slike. Na razvojnoj platformi je također odrađena pravilna raspodjela poslova na procesore koji su dostupni na razvojnoj platformi. Rezultati brzine izvođenja pokazali su da je implementirano rješenje pogodno za uporabu u stvarnom vremenu.

COLOR IMAGE FILTERING ON REAL DEVELOPMENT PLATFORM FOR ADVANCED DRIVER ASSISTANCE SYSTEMS

ABSTRACT

This paper deals with the method of filtering an image by color in different color spaces for images captured by a front view camera. The aim of this paper was to implement the method of filtering image by color on the embedded ADAS development platform. First, the theoretical basis for the method used in this paper was presented along with an overview of existing solutions. The method is based on the processing of image pixels, where for each pixel its belonging to the subspace that represents a certain color within a given color space is checked. The next step in the paper was to develop a concept solution in the *Python* programming language using the *OpenCV* library on a personal computer. The solution was then implemented and optimized in the C programming language on a personal computer. Furthermore, this optimized solution was implemented on the ADAS development platform within the *Vision SDK* development environment of the C programming language. Additionally, the implementations were evaluated with regard to the correctness of color image filtering on a dataset with three different resolutions for five image formats. The proper distribution of jobs on the processors available on the development platform was performed. Execution time results showed that the implemented solution is suitable for real-time use.

ŽIVOTOPIS

Luka Budak rođen je 20.kolovoza 1994. godine u Vinkovcima. Završio je osnovnu školu Ivana Gorana Kovačića u Vinkovcima. Nakon završene osnovne škole upisao je Gimnaziju Matije Antuna Reljkovića smjer prirodoslovno-matematički u Vinkovcima. Maturirao 2013. godine i upisao preddiplomski sveučilišni studij Elektrotehnike i informacijskih tehnologija na Fakultetu elektrotehnike i računarstva u Zagrebu. Godine 2017. upisao je preddiplomski sveučilišni studij Elektrotehnike i informacijskih tehnologija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2019. završava preddiplomski sveučilišni studij i na istom fakultetu upisuje diplomski sveučilišni studij Računarstvo, izborni blok Informacijske i podatkovne znanosti.

_____

Potpis autora

PRILOZI

P. 4.1 Skup testnih slika

P. 4.2 Tablice vremena izvođenja filtriranja slike po boji svih zapisa na računalu u programskom jeziku C bez optimizacije rješenja

P. 4.3 Tablice vremena izvođenja filtriranja slike po boji svih zapisa na računalu u programskom jeziku C uz optimizaciju rješenja

P. 4.4 Tablice vremena izvođenja filtriranja slike po boji svih zapisa implementirano na DSP procesor Alpha ploče

P. 4.5 Tablice vremena izvođenja filtriranja slike po boji svih zapisa implementirano na A15 procesor Alpha ploče

P. 4.6 Tablice vremena izvođenja filtriranja slike po boji svih zapisa implementirano na dva DSP procesora Alpha ploče

P. 4.7 Tablice vremena izvođenja filtriranja slike po boji svih zapisa implementirano na dva DSP procesora i A15 procesor Alpha ploče