Prostorno filtriranje slike na realnoj razvojnoj platformi za napredne sustave za pomoć vozaču

Opačak, Kristina

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:164688

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni diplomski studij automobilsko računarstvo i komunikacije

PROSTORNO FILTRIRANJE SLIKE NA REALNOJ RAZVOJNOJ PLATFORMI ZA NAPREDNE SUSTAVE ZA POMOĆ VOZAČU

Diplomski rad

Kristina Opačak

Osijek, 2021.

SADRŽAJ

1. UVOD *1*

2. PROBLEM PROSTORNOG FILTRIRANJA SLIKE *3*

2.1. Uloga filtriranja slike u prostornoj domeni **3**

2.2. Postojeća rješenja prostornog filtriranja slike **6**

3. OPIS VLASTITIH IMPLEMENTIRANIH ALGORITAMA *9*

3.1. Postupak prostornog filtriranja slike **9**

3.1.1. Konvolucija 9

3.1.2. *Padding* 10

3.2. Niskopropusni filtri **13**

3.2.1. Medijan filtar 13

3.2.2. Gaussov filtar 15

3.2.3. *Moving average* filtar 18

3.3. Visokopropusni filtri **19**

4. IMPLEMENTACIJA ALGORITAMA NA REALNU ADAS PLATFORMU *22*

4.1. *ADAS* razvojna ploča i razvojna platforma *VisionSDK* **22**

4.2. Postupak implementacije **23**

4.2.1. Tipovi podataka i prilagodba *C* koda za *ADAS* ploču 23

4.2.2. Postupak dodavanja novog algoritma 26

4.2.3. Kreiranje slučajeva upotrebe 26

4.3. Provjera uspješnosti implementacije **29**

4.4. Optimizacija rješenja **31**

5. TESTIRANJE RADA IMPLEMENTIRANIH RJEŠENJA *37*

5.1.* Rezultati testiranja brzine izvođenja algoritma na *DSP* procesoru - *single threaded **37**

5.2.* Rezultati testiranja brzine izvođenja algoritma na *A15* procesoru - *single threaded **39**

5.3. Rezultati testiranja brzine izvođenja algoritma na dva paralelna *DSP* procesora **42**

5.1. Rezultati testiranja brzine izvođenja algoritma na osobnom računalu **44**

5.2. Usporedba dobivenih rezultata **46**

6. ZAKLJUČAK *48*

LITERATURA **49**
SAŽETAK **50**
ABSTRACT **51**
ŽIVOTOPIS **52**
PRILOZI **53**

1. UVOD

Sve veći broj prometnih nesreća kritična je stvar s kojom se suočava moderno društvo. Nepažnja vozača, umor i somnolencija najčešći su uzroci nesreća s velikim brojem smrtnih slučajeva. Zbog toga se u posljednjih nekoliko godina sve više razvijaju napredni sustavi za pomoć vozaču kako bi se broj nesreća smanjio ili čak sveo na nulu. Takva vozila nazivaju se inteligentnim transportnim sustavima, a posjeduju razne napredne tehnologije za primanje podataka u stvarnom vremenu koji se dalje analiziraju kako bi se osigurala veća udobnost i sigurnost za vozače i ostale sudionike prometa. Tehnike zasnovane na kamerama i radaru su one koje se najčešće koriste u inteligentnim vozilima. Tehnike zasnovane na viziji su robusne i mogu se koristiti za otkrivanje traka, prometnih znakova, pješaka, drugih vozila itd. Sustav pomoći koji to omogućuje je *ADAS* (engl. *Advanced Driver-Assistance System*). Razvijeni sustavi za pomoć mogu, osim navedenih, obavljati i zadatke kao što su upozorenje o napuštanju trake, upozorenje o promjeni trake, pomoć pri održavanju unutar trake, prilagodljivi tempomat i sl. Da bi sve to bilo moguće, nužno je provesti predobradu slike tijekom koje će se izvršiti filtriranje. To je nužno jer slike koje se koriste kao ulaz u sistem, tj. slike dobivene direktno iz video signala s kamera postavljenih na različitim lokacijama u vozilu, sadrže šum i druge neželjene čimbenike poput varijacije kod osvjetljenja, sjene od obližnjih predmeta i sl. Slike s takvim slučajnim šumovima zahtijevaju robusne metode predobrade za ispravno izvršavanje zadataka pomoći vozaču. Predobrada slike je presudna za sve naredne korake i izvedbe u stvarnom vremenu jer je njena glavna funkcija suzbijanje neželjenih izobličenja i poboljšavanje i naglašavanje značajki od interesa. Predobrada predstavlja pretvaranje slike u digitalni oblik i izvršavanje operacija na njemu. Digitalna slika prikaz je dvodimenzionalne slike kao konačni skup digitalnih vrijednosti, odnosno elemenata slike ili piksela. Prostornim filtriranjem izvršavaju se određene manipulacije nad elementima slike i na taj način rješavaju problemi šumova uzrokovanih maglom, kišom, snijegom, sjenama od drveća i sl. Filtriranje omogućuje daljnju obradu različitim *ADAS* algoritmima.

U ovom radu istražuju se metode i razni tipovi prostornih filtara (niskopropusnih i visokopropusnih). Najprije su u drugom poglavlju opisane uloge prostornih filtara te postojeća rješenja (engl. *State of the art*). Nakon toga je u trećem poglavlju opisan rad pojedinih prostornih filtara, a u četvrtom poglavlju objašnjena je implementacija filtara na realnu ugradbenu računalnu platformu za izvršavanje *ADAS* algoritma. Nakon izvršavanja optimizacije rada implementiranih filtara i raspoređivanja određenih podzadataka na različite procesore *ADAS*

razvojne platforme, testirana je brzina izvršavanja različitih implementiranih funkcija filtriranja na različitim procesorima, različite rezolucije i parametara filtara. Postupak testiranja opisan je u petom poglavlju te su navedeni i uspoređeni svi rezultati provedenih testova.

2. PROBLEM PROSTORNOG FILTRIRANJA SLIKE

2.1. Uloga filtriranja slike u prostornoj domeni

Signali iz stvarnog svijeta obično sadrže odstupanja od idealnog signala koji bi proizveo naš model procesa proizvodnje signala. Takva odstupanja nazivaju se šumom. Šum u digitalnoj slici predstavlja slučajnu varijancu svjetline ili podataka slike. Može ga proizvesti senzor slike i sklop skenera ili digitalnog fotoaparata, ali najčešće je posljedica nepovoljnih vremenskih uvjeta, kao npr. kiša i magla. Pri tome na slici nastaju izobličenja koja se u realnosti ne nalaze na objektu od interesa. Šum slike može se kretati od gotovo neprimjetnih mrlja na digitalnoj fotografiji snimljenoj u dobrim uvjetima do optičkih slika koje su gotovo u cijelosti šum i iz kojih se može dobiti samo mala količina informacija. Takva bi razina šuma bila u potpunosti neprihvatljiva jer bi bilo nemoguće odrediti subjekt promatranja. Svaki šum je nepoželjan nusprodukt snimanja i uglavnom ga je potrebno ukloniti filtriranjem. Filtriranje se može provoditi u prostornoj ili frekvencijskoj domeni. Postupci u prostornoj domeni provode se u ravnini slike, a temelje se na direktnoj manipulaciji elementima slike.

Prvi korak prostornog filtriranja je definiranje susjedstva u prostoru elementa slike koji se filtrira. Susjedstvo je najčešće kvadratno područje, a definira se ovisno o veličini kernela na način da je središnji element kernela pozicioniran na element slike koji se filtrira. Uzima se onoliko susjednih elemenata slike koliko je u kernelu preostalo vrijednosti. Središte kernela se pomiče od početnog pa sve do krajnjeg elementa slike u slici, počevši npr. od gornjeg lijevog ugla. Izraz 2.1. predstavlja pojednostavljen način primjene filtra:

$$g(x, y) = T[f(x, y)], \tag{2.1.}$$

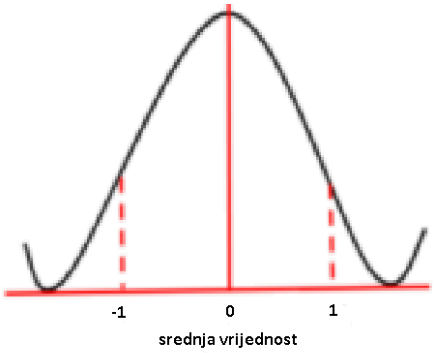
gdje je $f(x,y)$ ulazna slika, $g(x,y)$ filtriranu, a T operator koji se primjenjuje na svaki element slike.

U nastavku su navedeni različiti tipovi šuma te predloženi prostorni filtri za uklanjanje istih. Jedan od najčešćih tipova šuma je Gaussov šum. Ima funkciju gustoće vjerojatnosti jednaku onoj normalne raspodjele koja je također poznata i kao Gaussova raspodjela. Drugim riječima, vrijednosti koje šum može poprimiti su vrijednosti Gaussove distribucije. Funkcija gustoće vjerojatnosti p Gaussove slučajne varijable z dana je izrazom 2.2.:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}, \tag{2.2.}$$

gdje je z razina sive, μ srednja vrijednost sive boje, a σ njegovo standardno odstupanje [1].

Funkcija je zvonastog oblika sa srednjom vrijednosti 0 i standardnim odstupanjem 1, a prikazana je na slici 2.1.



Sl. 2.1. Gaussova raspodjela vjerojatnosti [1]

Količina Gaussovog šuma ovisi o standardnom odstupanju na način da su izravno proporcionalne. Glavni izvori Gaussova šuma u digitalnim slikama nastaju tijekom snimanja, npr. buka senzora uzrokovana lošim osvjetljenjem i/ili visokom temperaturom i/ili šumom električnog kruga. U digitalnoj obradi slike, ovakav se šum može smanjiti korištenjem prostornog filtra. Međutim, pri zaglađivanju slike filtriranjem neželjeni ishod može biti zamućenje rubova i detalja jer oni, kao i šum, odgovaraju visokim frekvencijama. Zbog toga je iznimno bitno pažljivo odabrati tip filtra i njegove parametre. Najučinkovitiji način za uklanjanje ovog šuma u predobradi digitalne slike je primjena Gaussovog filtra objašnjenog u 3.2.2 poglavlju. Slika 2.1. prikazuje usporedbu originalne slike (lijevo) i iste slike pod utjecajem Gaussovog šuma (desno).

Sl. 2.1. Originalna slika (lijevo) i slika pod utjecajem Gaussovog šuma (desno) [2]

Još jedan tip šuma za čije eliminiranje je potrebno prostorno filtriranje je impulsni šum. Razlikujemo 3 vrste impulsnog šuma:

- šum „soli“
- šum „papra“ i
- šum „soli i papra“.

Šum soli predstavlja slučajne svijetle vrijednosti, tj. pojavu slučajnih vrijednosti elemenata slike s vrijednostima 255 ili vrijednostima blizu tome iznosu. Nasuprot tome, šum „papra“ prepoznaje se kao slučajne tamne vrijednosti elemenata slike, odnosno elementi slike s vrijednostima 0 ili nešto iznad nule. Analogno tome, šum „soli i papra“ je pojava slučajnih svijetlih i slučajnih tamnih elemenata slike po cijeloj slici. Tamni pikseli pojavljuju se na svijetlim područjima, a svijetli na tamnim. Ovaj model poznat je i kao šum ispuštanja podataka jer statistički opada izvorna vrijednost podataka. Najčešći izvor ovog šuma je neispravan rad senzorske ćelije fotoaparata. Kako ovaj šum predstavlja elemente slike koji su ekstremnih vrijednosti u odnosu na svoje susjede, vrlo se efikasno uklanja medijan filtrom jer se upotrebom tog filtra element slike ekstremne vrijednosti zamijeniti median vrijednošću njegovih susjeda. Detaljniji rad medijan filtra objašnjen je u 3.2.1. poglavlju. Na slici 2.2. nalazi se originalna slika (lijevo) i ista slika pod utjecajem šuma soli i papra (desno).

Sl. 2.2. Originalna slika (lijevo) i slika pod utjecajem šuma „soli i papra“ (desno) [2]

Kvantni (fotonski ili Poissonov) šum nastaje zbog statističke prirode elektromagnetskih valova kao što su x-zrake, vidljiva svjetlost i gama zrake. Najčešće se opaža kod slabih intenziteta svjetlosti. Ovaj šum prati Poissonovu raspodjelu koja se, osim na vrlo visokim intenzitetima, približava Gaussovoj raspodjeli. Najefikasniji način uklanjanja fotonskog šuma pri digitalnoj

predobradi slike je primjena *moving average* filtra objašnjenog u 3.2.3. poglavlju. Na slici 2.3. prikazana je originalna slika (lijevo) i ista slika pod utjecajem kvantnog šuma (desno).

Sl. 2.3. Originalna slika (lijevo) i slika pod utjecajem kvantnog šuma (desno) [2]

Dakle, učinkoviti način za suzbijanje, smanjivanje ili potpuno eliminiranje opisanih šumova na digitalnoj slici je upotreba niskopropusnih prostornih filtara:

- medijan filtar
- Gaussov filtar
- *moving average* filtar

Osim za suzbijanje šuma, filtri se mogu koristiti i za isticanje detalja. Za to su potrebni visokopropusni filtri koji su u *ADAS* sustavima također potrebni za mnoge svrhe, kao npr. za očitavanje registracijske oznake na vozilu ili detektiranje objekata u okolini vozila. Isticanje detalja pomoću visokopropusnih filtara temelji se na propuštanju visokih frekvencija i njihovo naglašavanje. Upotreba niskopropusnih i visokopropusnih filtara odvija se u prostornoj domeni pri čemu se koriste različite maske, ovisno o tipu filtra. Svi navedeni filtri objašnjeni su u 3. poglavlju.

2.2. Postojeća rješenja prostornog filtriranja slike

Prostorno filtriranje slike srž je svake predobrade slike i predmet je mnogih istraživanja. U nastavku su navedeni neki od radova s istom ili sličnom tematikom te opis postignuća vezanih za tematiku prostornog filtriranja slike.

U radu [4] napravljena je sklopovska realizacija filtriranja slike korištenjem maski u prostornim domenama. Testiranje implementiranih filtara temelji se na usporedbi brzine izvođenja na

MATLAB-u i *Xilinx FPGA* ploči. Prednost *FPGA* pločice je velika brzina i mogućnost ponovnog podešavanja postavki. Cilj u ovome radu je implementirati filtre koji će biti prikladni za upotrebu u stvarnom vremenu. Arhitektura rješenja počinje od ulazne slike zapisane kao 8-bitna slika za koju vrijednosti elemenata slike variraju od 0 do 255. Slika je pretvorena u ljestvicu sive te je dobivena matrica istog. Za implementaciju na *FPGA*, slika je smanjena na 128x128 matricu i dodan joj je *zero padding* sa svih strana. Zatim je pretvorena u matricu stupca veličine 1x16900. Vektor je pohranjen u blok *RAM* za daljnju obradu. Obrađeni elementi slike se kasnije zapisuju u tekstualnu datoteku za upotrebu u *MATLAB*-u. Naposljetku su se usporedili rezultati s *MATLAB*-a i *FPGA* ploče. Također su uspoređeni rezultati mjerenja brzine obrade i filtriranja različitim filtrima, npr. Gausov filtar izvršen je u 304 ms, dok je za filtriranje medijan filtrom bilo potrebno 575 ms. Zaključeno je da medijan filtar troši više resursa i najsporije je među svima, ali daje najbolje rezultate za smanjenje šuma. Također je uočeno da su povećanjem maske Gausovog filtra rezultati bolji, tj. efikasnost je najlošija kod filtriranja s maskom veličine 3x3. Osim toga, uočeno je da su rezultati testiranja na hardveru analogni onima testiranja na softveru.

U radu [5] najprije su predstavljeni različiti tipovi šuma, a zatim načini filtriranja istih pomoću nekoliko filtara – medijan, bilateralni i *weighted average*. Ustanovljeno je koji je filtar najbolji za slike sa šumom uzrokovanim nepovoljnim uvjetima okoliša. Riječ je o medijan filtru. Također je pronađen najbolji način za očuvanje oštrih rubova, a to je filtriranje bilateralnim filtrom. Uspoređeni su filtri što je dovelo do zaključka o efikasnijim načinima obrade slike – ako je prioritet brzina obrade, najbolje je koristiti Gaussov filter. Bilateralni je najefikasnije koristiti za očuvanje rubova u pogodnim vremenskim uvjetima, a medijan za filtriranje u nepovoljnim uvjetima. Nažalost, nije postignuta obrada slike koja će na što brži način sačuvati rub neovisno o vremenskim uvjetima. Algoritmi su testirani na stvarnim slikama s maskama veličine 3x3, 5x5 i 7x7 te su rezultati međusobno uspoređeni.

U radu [6] predstavljeno je istraživanje o tehnologijama koje se koriste u autonomnim vozilima s naglaskom na značaj filtriranja slike s implementacijom na ugradbeni sustav u stvarnom vremenu. Rješenje je implementirano i testirano na *Nvidia Drive* računalnoj platformi s dva Parker procesora i dva grafička procesora. Slike se tretiraju kao dvodimenzionalni signal na koji se primjenjuju razne tehnike obrađivanja, ali prije same obrade, potrebna je i predobrada. Predobrada podrazumijeva prostorno filtriranje kako bi se uklonio šum, ali istovremeno i sačuvali rubni detalji. Predstavljen je osnovni princip takvog filtriranja koji se bazira na medijan filtru. U usporedbi s eksplicitnom razgradnjom problema, sustav u ovom radu istodobno se optimizira što na kraju dovodi do boljih performansi. Ustanovljena je najefikasnija metoda za

filtriranje slike uz očuvanje oštrih rubova. Riječ je o nelinearnoj niskopropusnoj tehnici filtriranja slike koja ju zaglađuje i na taj način smanjuje šum. Unatoč predobradama slike koje nisu vezane za prostorno filtriranje, nekoliko algoritama testiranih u stvarnom vremenu u ugradbenim sustavima detektiralo je zadane objekte s točnošću manjom od 75%, što je u autonomnoj industriji neprihvatljivo. Zbog toga je prostorno filtriranje nužno kako bi se učinkovito odvijalo detektiranje rubova. Prvi korak kompletnog rješenja u ovome radu je određivanje područja od interesa (engl. *Region of Interest – ROI*) kako bi se smanjilo vrijeme potrebno za predobradu jer se time skraćuje vrijeme izvršavanja algoritama koje je potrebno za filtriranje. Drugi korak je samo filtriranje, a tek nakon toga slijedi detektiranje rubova uz strojno učenje. Nakon uspješne obrade, rješenje se implementira na *Nvidia Drive* računalnoj platformi.

3. OPIS VLASTITIH IMPLEMENTIRANIH ALGORITAMA

3.1. Postupak prostornog filtriranja slike

Filtriranje u prostornoj domeni odnosi se na primjenu filtra na samu ravninu (plohu) slike. Nad elementima slike u ovom tipu filtriranja izravno se provode različite operacije. Može se odvijati na pojedinačnom elementu slike ili na grupi elemenata slike. Za razliku od filtra u frekvencijskoj domeni, koji se mogu koristiti samo za linearno filtrirane, prostorni filtri pogodni su kako za linearno, tako i za nelinearno filtriranje. Za svaki je filtar prije samog filtriranja na originalnu sliku potrebno dodati *padding*. Nakon dodavanja *padding*-a, središnji element kernela postavlja se na prvi element slike koji se u C kodu ovoga rada nalazi na lokaciji [*offset, offset*]. *Offset* je broj ovisan o veličini kernela, tj. broj redova i stupaca koje je potrebno dodati sa svake strane slike kao *padding*. Pojam *padding*-a detaljno je objašnjen u 3.1.2. poglavlju. Nakon toga kernel se pomiče preko svih ostalih elemenata, sve do elementa slike na lokaciji [*row-offset-1, col-offset-1*], gdje je *row* broj redova matrice s dodanim *padding-om,* a *col* broj stupaca iste matrice. U svakom koraku odrađuju se matematičke operacije ovisno o tipu filtra. Konačan rezultat matematičkih operacija je vrijednost kojom se zamjenjuje trenutna vrijednost na koju je postavljeno središte kernela te se dobiva novo filtrirana matrica koju se vraća.

Funkciji svakog prostornog filtra u C kodu predaje se matrica s dodanim *padding*-om, broj redova i stupaca originalne matrice, veličina kernela te sam kernel ako se filtar temelji na konvoluciji. Povratni tip funkcije je dvostruki pokazivač – vraća se 2D filtrirana matrica. U nastavku su detaljnije objašnjeni spomenuti pojmovi (*padding* i konvolucija) te opisani implementirani filtri s prototipima funkcija. Ostatak programskog koda nalazi se u prilogu P.3.1.

3.1.1. Konvolucija

Gaussov filtar i svi visokopropusni filtri temelje se na konvoluciji. Konvolucija je jednostavna matematička operacija koja je temelj za mnoge uobičajene operatore obrade slike pri prostornom filtriranju. Omogućava množenje dvaju polja brojeva, uglavnom različitih veličina, ali najčešće iste dimenzionalnosti. Predstavlja jednostavnu linearnu kombinaciju određenih vrijednosti ulaznih piksela. U kontekstu obrade slike, jedan od ulaznih nizova obično je dio digitalne slike zapisane kao razine sivih intenziteta. Drugi ulazni niz je maska (kernel) kojom se slika filtrira, a obično je puno manjih dimenzija od slike na koju se primjenjuje. Filtriranje s obzirom na dimenzije može biti:

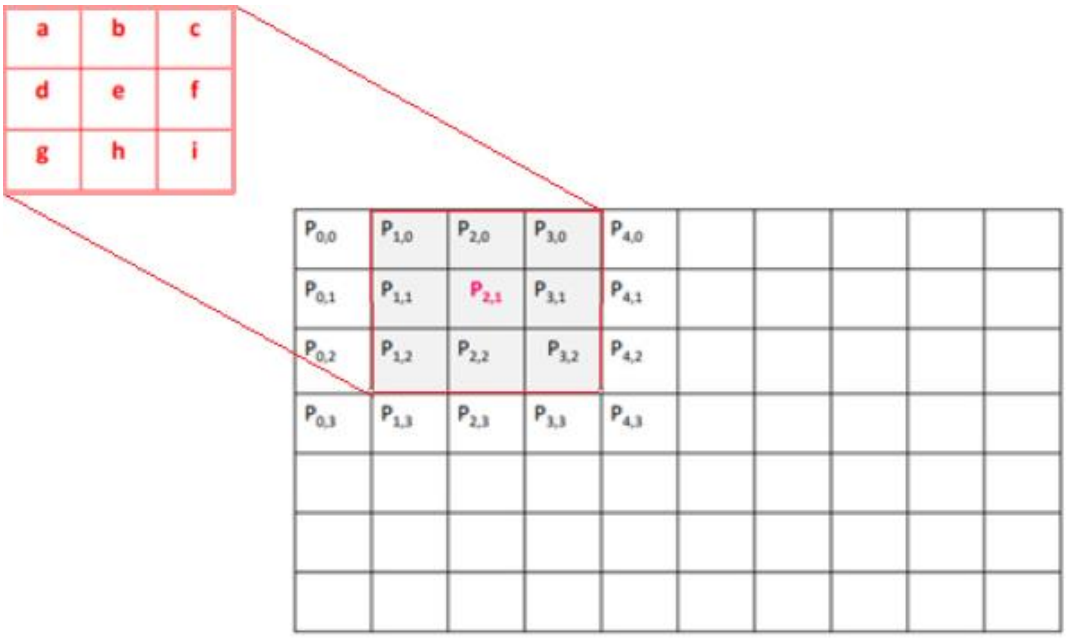
- 1D – konvolucijska maska je vektor

- 2D – konvolucijska maska je matrica

Maska koja se upotrebljava pri konvoluciji je matrica koja sadrži težinske faktore. Težinskim faktorima se množe vrijednosti elemenata slike i zbrajanjem umnožaka dobiva se nova vrijednost pojedinog elementa slika. Definiranje vrijednost izlaznog elementa slike konvolucijom izraženo je izrazom 3.1.:

$$\text{slikaIzlaz}[x, y] = \sum_{i=-m}^{m} \sum_{j=-m}^{m} maska[i,j] * slikaUlaz[x-i, y-j], \quad (3.1.)$$

gdje su *x* i *y* trenutni položaji elementa slike (*x*-red, *y*-stupac), a *m* i *n* broj redova/stupaca u krenelu podijeljen s dva te zaokružen na nižu cjelobrojnu vrijednost. Konvolucija se izvršava množenjem svakog elementa slike u susjedstvu s odgovarajućim koeficijentom kernela i sumiranjem umnožaka kako bi se dobio odziv za svaki element slike, odnosno točku (x,y). Izraz 3.2. primjer je matematičke operacije konvolucije koji se odnosi na prototip digitalne slike i konvolucijskog kernela na slici 3.1. Pri tome se manipulira elementom slike s položajem (2,1).



Sl. 3.1. Konvolucijska maska (lijevo) i slika osjenčanim područjem na koje se maska primjenjuje (desno) [7]

$$P_{2,1} = a * P_{1,0} + b * P_{2,0} + c * P_{3,0} + d * P_{1,1} + e * P_{2,1} + f * P_{3,1} + g * P_{1,2} + h * P_{2,2} + i * P_{3,2} \quad (3.2.)$$

3.1.2. *Padding*

Padding je pojam koji se odnosi na količinu elemenata slike dodanih na sliku prije primjene konvolucijsle maske. Dodaje se kako bi se omogućilo postavljanje središnjeg elementa kernela

na prvi red elemenata slike i sve ostale retke i stupce koji se ne bi mogli filtrirati kada ne bi bio dodan *padding*. Broj redova i stupaca koji se dodaju oko slike direktno ovisi o veličini kernela na način da kada je veličina kernela tri, dodaje se po jedan red/stupac sa svake strane slike, kada je veličina kernela pet, dodaju se dva reda/stupca itd. U C kodu ovog rada taj je broj deklariran kao *offset*. Ovisno o tipu filtra koji se primjenjuje na sliku, koriste se tri vrste *padding*-a:

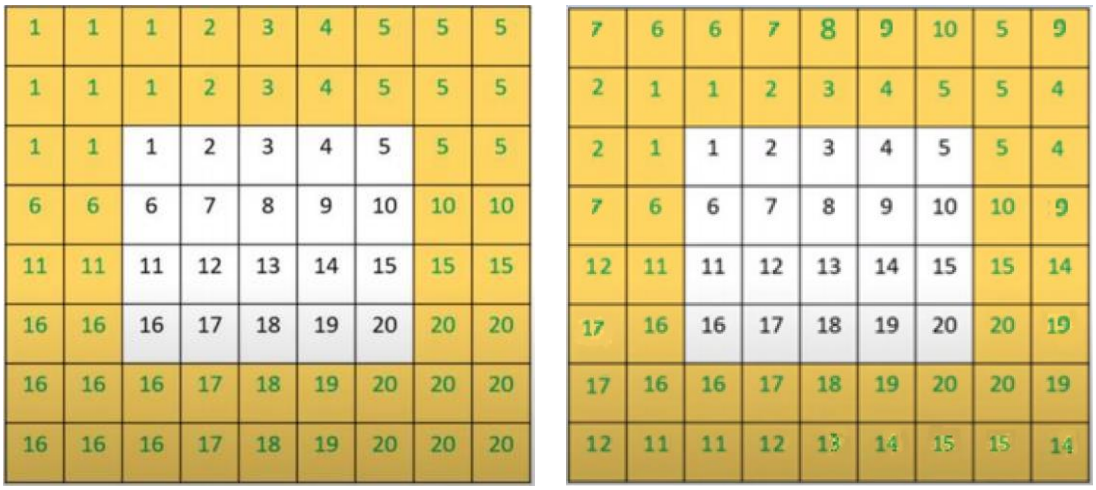
- *zero padding*
- *replicate padding*
- *mirror padding*

Zero padding podrazumijeva dodavanje elemenata slike oko originalne na način da su vrijednosti elemenata slike izvan granica originalne jednake nuli. Na slici 3.2. prikazan je primjer dodavanja *zero padding*-a oko dvodimenzionalnog polja vrijednosti.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

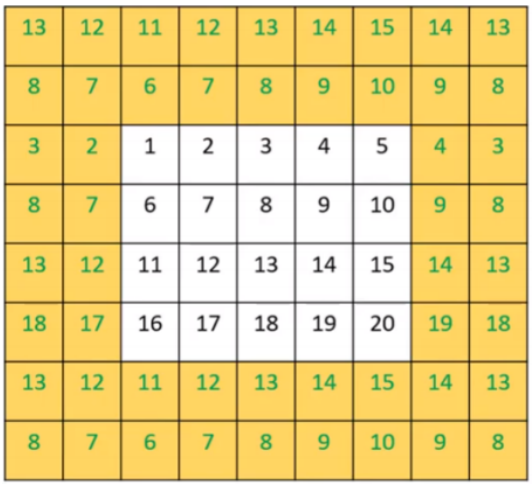
Sl. 3.2. Primjer *zero padding*-a [8]

Pri dodavanju *replicate padding*-a, vrijednosti elemenata slike izvan granice originalne slike jednake su vrijednosti najbližeg susjeda. Ovaj se tip *padding*-a koristi kada su područja u blizini rubova unutar originalne slike približno konstantna, tj. imaju približno jednake vrijednosti elemenata slike – malo detalja. Slika 3.3. prikazuje dvodimenzionalno polje vrijednosti kojemu je dodan *replicate padding*. Os simetrije u odnosu na koju se repliciraju vrijednosti originalne slike je postavljena na sam rub slike. Pri tome razlikujemo dva tipa *replicate padding*-a. Kod prvoga (u daljnjem tekstu tip 1) se repliciraju samo krajnji rubni redovi/stupci i prepisuju se u svim *padding* redovima/stupcima. Kod drugog tipa (u daljnjem tekstu tip 2) replicira se onoliko rubnih redova/stupaca koliko će biti dodanih *padding* redova/stupaca.



Sl. 3.3. Primjer dvaju tipova *replicate padding*-a [8]

Mirror padding je tip *padding*-a kod kojega su vrijednosti elemenata slike izvan granice originalne slike jednake zrcaljenim vrijednostima pri čemu su granice zrcaljenja rubni redovi/stupci originalne slike. Koriste se kada područja u blizini rubova unutar originalne slike imaju visoku frekvenciju, odnosno mnogo detalja. Primjer dvodimenzionalnog polja s dodanim *mirror padding*-om nalazi se na slici 3.4.



Sl. 3.4. Primjer *mirror padding*-a [8]

Prototip fukncije za dodavanje *padding*-a na originalnu sliku je sljedeći:

Linija Kod

```
1:        int** Padding(int** mat, int row, int col, int kernelSize);
```

Sl. 3.5. Prototip funkcije *padding*

Povratni tip je dvostuki pokazivač, riječ je o dvodimenzionajnoj matrici koja predstavlja izlaznu, filtriranu sliku. Parametri koji se predaju funkciji su originalna matrica, također dvodimenzionalna, broj redova, broj stupaca i željena veličina kernela. Unutar funkcije alocira se memorija za novu matricu na način da se broj redova i stupaca *padding* matrice povećava za broj ovisan o veličini kernela na sljedeći način:

Linija Kod

```
1:        row += (kernelSize - 1);
2:        col += (kernelSize - 1);
```

Sl. 3.6. Promjena broja redova i stupaca ovisno o veličini kernela

Osim toga, bitno je definirati *offset* koji je također ovisan o veličini kernela, a koji označava broj dodanih redova s pojedine strane slike i od kojega započinje promjena vrijednosti elemenata slike, tj. filtriranje. Inicijalizacija *offset*-a u C kodu nalazi se na slici 3.7.

Linija Kod

```
1:         offset = (kernelSize - 1) / 2;
```

Sl. 3.7. Inicijalizacija *offset*-a

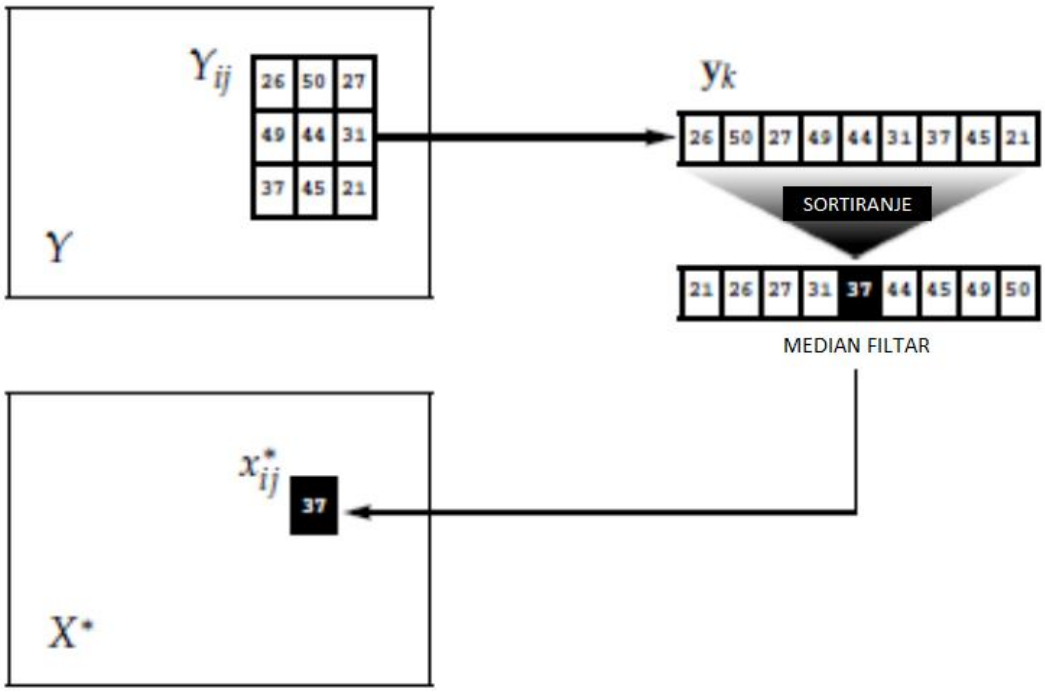
3.2. Niskopropusni filtri

Ovisno o namjeni obrade slike, neke frekvencije su na slici od veće važnosti u odnosu na druge. Niskopropusni filtri zaglađuju slike tako što propuštaju niske frekvencije, a prigušuju visoke. Drugim riječima, reduciraju male i nebitne detalje kako bi se bolje vidjeli bitni dijelovi i objekti od interesa na slici. Zaglađuju i zamućuju nepotrebne dijelove slike. Međutim, rubovi objekata od interesa nose karakteristiku oštrih prijelaza te se upotrebom niskopropusnih filtara mogu također zamutiti što je negativna nuspojava niskopropusnog filtriranja [9].

3.2.1. Medijan filtar

Medijan filtar pripada nelinearnih statističkim filtrima. Medijan filtrom se zauzima područje slike (3x3, 5x5, 7x7 itd.) i sve vrijednosti unutar tog područja poslože uzlazno te se središnji element slike zamjenjuje medijan vrijednošću promatranog polja. Ovaj filtar ne zahtijeva konvoluciju. Međutim, odziv ovog filtra temelji se na rangiranju elemenata, tj. potrebno je razvrstavanje vrijednosti po veličini kako bi se omogućio pronalazak medijan vrijednosti. Glavna prednost medijan filtra je ta što efikasno uklanja šum, ali istovremeno dijelove slike bez šuma

ostavlja naizgled nepromijenjenima. Osim toga, prednost medijan filtra je izlazna slika s malom količinom zamućenja, a znatno većim odnosom signal/šum u odnosu na ulaznu sliku. Konačna filtrirana vrijednost je takva da je manja (ili jednaka) od jedne polovice elemenata iz promatranog polja kernela, a veća (ili jednaka) od druge polovice. Na slici 3.8. nalazi se opisani proces medijan filtriranja pri čemu je Y ulazna, a X* izlazna, tj. filtrirana slika.



Sl. 3.8. Postupak medijan filtriranja [10]

Dio slike veličine kernela u svakom se koraku u C kodu sortira na sljedeći način:

Linija Kod

```
1:        if (arr[x] > arr[y]) {
2:         tmp = arr[x];
3:         arr[x] = arr[y];
4:         arr[y] = tmp;
5:        }
```

Sl. 3.9. Sortiranje središnjeg i susjednih elemenata

Nakon toga se određuje medijan vrijednost pomoću položaja središnjeg elementa sortiranog niza i ta se vrijednost pridružuje trenutnom elementu slike, prikazano na slici 3.10.

Linija Kod

```
1:        matPadFiltered[i][j] = arr[kernel * kernel / 2];
```

Sl. 3.10. Pridruživanje medijan vrijednosti trenutnom elementu slike

Prototip funkcije medijan filtra je sljedeći:

Linija Kod

```
1:      int** MedianFilter(int** paddingMat, int row, int col, int
kernelSize);
```

Sl. 3.11. Prototip funkcije medijan filtra

Dakle, osnovni princip uklanjanja šuma (najčešće „sol i papar“) je eliminiranje izoliranih elemenata slike, tj. elemenata slike koji su znatno svjetliji ili tamniji od svojih susjeda. Pri tome eliminirati takve elemente slike znači zamijeniti ih s medijan vrijednošću. Na slici 3.12. nalazi se originalna slika i slike filtrirane medijan filtrom s 3x3, 5x5 i 7x7 kernelom (redom). Originalna slika pod utjecajem je „sol i papar“ šuma koji je u ovom slučaju prisutan zbog kiše.

Sl. 3.12. Originalna slika i slike filtrirane medijan filtrom različitog kernela (redom 3x3, 5x5, 7x7) [12]

3.2.2. Gaussov filtar

Gaussov se filtar koristi za smanjivanje detalja i zaglađivanje slike na način da smanjuje razliku između susjednih elemenata slike. Primjena Gaussovog filtra utječe na smanjenje

visokofrekventnih komponenata slike – Gaussov je filtar prema tome niskopropusni. Također je pogodan za upotrebu *antialiasing*-a na rubovima, tj. smanjivanje artefakata izobličenja pri prikazivanju slike u manjoj rezoluciji nego što je u originalu. Vrijednosti unutar Gaussovog kernela računaju se prema izrazu 3.3.:

$$h_g(k,l) = e^{[\frac{(k^2-l^2}{2*\sigma^2}]}, \tag{3.3.}$$

gdje k i l predstavljaju trenutni položaj u kernelu, a σ standardno odstupanje. Prije samog filtriranja potrebno je napraviti Gaussov kernel pomoću kojega će se izvoditi konvolucija. Na slici 3.13. nalazi se prototip funkcije za kreiranje Gaussovog kernela. Parametri koji se predaju funkciji su veličina kernela i *sigma* (σ). Pomoću tih dvaju parametara kontrolira se razina zaglađivanja rubova i uklanjanja detalja šuma. *Sigma* je parametar koji određuje širinu Gaussove funkcije i predstavlja standardnu devijaciju, odnosno stupanj varijacije unutar grupe elemenata slike [9]. Na slici 3.14. prikazano je punjenje Gaussovog kernela vrijednostima prema formuli 2.2.

Linija ***Kod***

```
1:       double** GaussKernel(int kernel_size, double sigma);
```

Sl. 3.13. Prototip funkcije za kreiranje Gaussovog kernela

Linija ***Kod***

```
1:       double x = i - (kernel_size - 1) / 2.0;
2:       double y = j - (kernel_size - 1) / 2.0;
3:       gauss[i][j] = 1 * exp(((pow(x, 2) + pow(y, 2)) / ((2 * pow(sigma,
2)))) * (-1));
4:       sum += gauss[i][j];
5:       gauss[i][j] /= sum;
```

Sl. 3.14. Računanje vrijednosti Gaussovog kernela

Vrijednosti kernela su definirane na način da poprimaju oblik Gaussove krivulje. Na slici 3.15. prikazan je primjer Gaussovog kernela veličine 5x5 i standardne devijacije 1. Središnji element (na poziciji [2,2]) ima najveću vrijednost koja se smanjuje kako se povećava udaljenost od središta prema Gaussovoj funkciji. Vrijednosti elememata u maski su simetrične.

0.002969	0.013306	0.021938	0.013306	0.002969
0.013306	0.059634	0.09832	0.059634	0.013306
0.021938	0.09832	0.162103	0.09832	0.021938
0.013306	0.059634	0.09832	0.059634	0.013306
0.002969	0.013306	0.021938	0.013306	0.002969

Sl. 3.15. Primjer Gaussovog kernela

Nakon definiranja Gaussovog kernela, slijedi konvolucija. Kako bi konvolucija bila moguća, prije samog filtriranja na originalnu sliku potrebno je dodati *padding*. Tip *padding*-a koji se koristi kod Gaussovog filtriranja je *mirror padding*. Prototip funkcije Gausovog filtra je sljedeći:

Linija Kod

```
1:      int** GaussianFilter(int** paddingMat, int row, int col, int
kernelSize, double **kernelmat);
```

Sl. 3.16. Prototip funkcije medijan filtra

Funkciji je, osim originalne matrice s dodanim *padding*-om, broja redova i stupaca i veličine kernela, potrenmo predati i Gaussov kernel kreiran u prethodnoj funkciji. Gaussovo zaglađivanje obično se koristi prije detekcije ruba čiji algoritmi koriste operatore vrlo osjetljive na šum. Na slici 3.17. nalaze se originalna slika i slike filtrirane Gaussovim filtrom s kernelom veličine 3x3, 5x5 i 7x7 (redom). Originalna slika je pod utjecajem Gaussovog šuma. Povećanjem veličine kernela stupanj zamagljenosti također je veći.

Sl. 3.17. Originalna slika i slike filtrirane Gausovim filtrom različitog kernela (redom 3x3, 5x5, 7x7) [11]

3.2.3. *Moving average* filtar

Moving average filtar je niskopropusni filtar kod kojega svi elementi unutar kernela, koji se primjenjuje na sliku koja se filtrira, moraju biti jednake vrijednosti. Kao i ostali niskopropusni filtri, *moving average* filtar uklanja šum i zaglađuje sliku. Zasnovan je na principu računanja srednje vrijednosti svih elemenata slike unutar polja veličine kernela. Zbroj svih vrijednosti unutar maske mora biti jedan. Koeficijenti unutar maske *moving average* filtra računaju se prema izrazu 3.4., pri čemu je $h(a,b)$ izlazni koeficijent a M veličina kernela.

$$h(a,b) = \frac{1}{M^2} \tag{3.4.}$$

Na slici 3.18. nalazi se primjer *moving average* maske veličine 3x3 elementa slike. Ova se maska može zapisati i tako da svi elementi imaju vrijednost 1/9, ali zbog veće efikasnosti računanja svi su postavljeni na 1 i pomnoženi odgovarajućom vrijednošću. Prostorni filtar u kojemu su svi koeficijenti jednaki naziva se i *box* filtar.

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sl. 3.18. Primjer maske *moving average* filtra

Prije samog filtriranja, potrebno je dodati *padding* kako bi konvolucija bila moguća. Tip *padding*-a koji se koristi za ovaj filtar je *mirror padding*. Na slici 3.20. nalazi se linija C koda u kojemu se računa filtrirana vrijednost elementa slike nakon što je izračunata suma elemenata unutar polja veličine kernela. Prototip funkcije *moving average* filtra nalazi se na slici 3.19.

Linija ***Kod***

```
1:      int** MovingAverageFilter(int** paddingMat, int row, int col,
        int kernel);
```

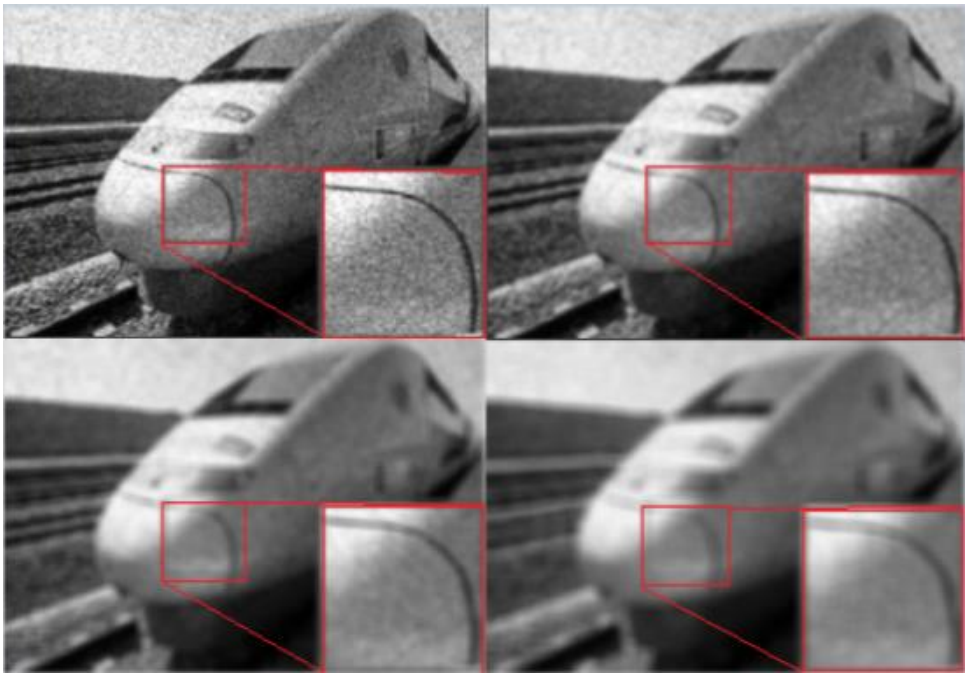
Sl. 3.19. Prototip funkcije *moving average* filtra

Linija Kod

```
1:        matFiltered[i][j] = round((double)sum / (kernel * kernel));
```

Sl. 3.20. Računanje vrijednosti elementa slike filtriranog *moving average* filtrom

Slika 3.21. prikazuje originalnu sliku i slike filtrirane *moving average* filtrom s kernelom veličine 3x3, 5x5 i 7x7 (redom). Originalna je slika pod utjecajem Poissonovog šuma. Stupanj zamagljenosti slike povećava se povećanjem veličine kernela.



Sl. 3.21. Originalna slika i slike filtrirane *moving average* filtrom različitog kernela (redom 3x3, 5x5, 7x7) [12]

3.3. Visokopropusni filtri

Za razliku od niskopropusnih, visokopropusni filtri koriste se za isticanje detalja, tj. propuštaju visoke frekvencije, a prigušuju niske. Na taj se način izoštravaju bitni dijelovi na slici. Detalji su u originalnoj slici zamagljeni ili greškom ili kao posljedica nekih metoda predobrade slike (npr. filtriranje niskopropusnim filtrom). Nakon filtriranja visokopropusnim filtrom pojačava se oštar prijelaz (rub) na slici. Filtriranje se odvija pomoću konvolucije s gotovim definiranim kernelima. Na slici 3.22. nalazi se primjer maski visokopropusnih filtara. Svaka od njih ostavlja homogeno područje slike nepromijenjenim. Također, svaka od maski ima zajedničku činjenicu da se u središtu nalazi vršna vrijednost, a negativne vrijednosti iznad, ispod, lijevo i desno od središnje

vrijednosti. Vrijednosti u rubovima kernela blizu su nuli. Međutim, prikazane tri maske proizvode različita pojačanja ovisno o primjeni filtra [13].

0	-1	0	-1	-1	-1	1	-2	1
-1	5	-1	-1	9	-1	-2	5	-2
0	-1	0	-1	-1	-1	1	-2	1

Sl. 3.22. Primjeri maski visokopropusnog filtra [13]

Ako među grupom elemenata slike nema promjene intenziteta, ništa se ne događa, tj. dio slike ostaje homogen. Međutim, ako je jedan element slike svjetliji od svojih neposrednih susjeda, pojačat će se. Nažalost, ako postoji šum u slici, visokopropusni filtri će ga pojačati. Zbog toga su pogodni kada je šum na slici beznačajan. Prototip funkcije visokopropusnog filtra nalazi se na slici 3.23. Funkciji je potrebno, osim originalne matrice s dodanim odgovarajućim *padding*-om, broja redova i stupaca i veličine kernela, predati i unaprijed definiran kernel. Kernel je inicijaliziran kao dvodimenzionalno polje prikazano primjerom na slici 3.24.

Linija Kod

```
1:      int** HighPassFilter(int** paddingMat, int row, int col, int
kernelSize, int* kernelmat);
```

Sl. 3.23. Prototip funkcije visokopropusnog filtra

Linija Kod

```
1:      int HighPassKernel[3][3] = { {0,-1,0}, {-1,5,-1}, {0,-1,0} };
```

Sl. 3.24. Inicijalizacija maske visokopropusnog filtra

Na slici 3.25. prikazuje se originalna slika i slike filtrirane visokopropusnim filtrom s drugim kernelom sa slike 3.22. Originalna slika nije pod utjecajem šuma – cilj visokopropusnog filtra je naglasiti granice i rubove objekata na slici.

Sl. 3.25. Originalna slika i slike filtriran visokopropusnim filtrom [14]

4. IMPLEMENTACIJA ALGORITAMA NA REALNU *ADAS* PLATFORMU

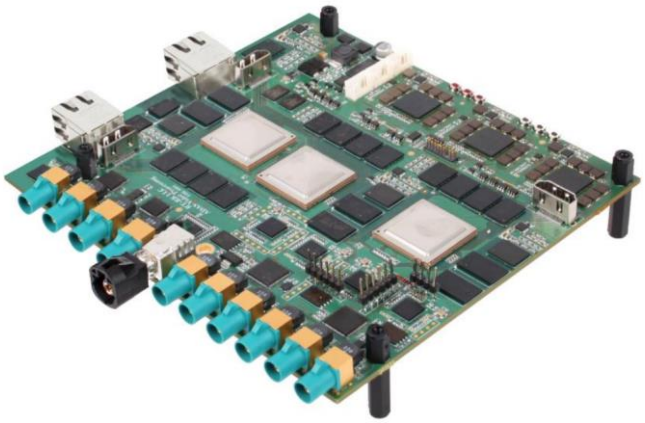
U ovom poglavlju opisane su specifikacije *ADAS* razvojne ploče, razvojna platforma *VisionSDK* i način na koji je odrađena implementacija algoritama opisanih u prethodnom poglavlju.

4.1. *ADAS* razvojna ploča i razvojna platforma *VisionSDK*

U ovom je diplomskom radu korištena *ADAS* razvojna ploča *Alpha*, razvojna ploča dizajnirana od strane tvrtke RT-RK u obrazovne svrhe. Ploča se sastoji od tri sustava na čipu (engl. *System on a Chip - SoC*) tvrtke Texas Instruments. 3 *SoC*-a koja sadrži ploča su

- *SC* (*SCV*)
- *FFN*
- *FUS*

Odabir *SoC*-a ovisi o namjeni. *SC* je primarno korišten za pogled na okruženje vozila ili bilo koju sličnu upotrebu koja zahtijeva veći broj kamera. *FFN* opće je namjene i koristi se za generičke namjene poput pogleda sprijeda. *FUS* je prvenstveno namijenjen fuzijskim aplikacijama: spajanje podataka iz drugih *SoC*-ova. Ovaj *SoC* nema priključke za kameru. Za izradu ovog diplomskog rada korišten je *SC*. Uz ove razlike, svim *SoC*-ovima su zajedničke neke stvari, kao npr. činjenica da sva tri imaju 10 *CPU*-ova. U ovom radu korišten je *DSP*, *A15* i kombinacija dvaju *DSP* procesora na način da se pola slike obrađuje na jednom, a druga polovica istovremeno, tj. paralelno, na drugom procesoru. Ploča sadrži 1,5 *GB RAM*-a, utor za *Micro SD* karticu – za pokretanje kompilirane slike iz *Vision SDK*, za čitanje podataka iz kartice ili za pisanje na nju. Također je korišten *HDMI* priključak za prikazivanje sadržaja na zaslonu, tj. za prikaz filtriranih slika i „grubu“ provjeru ispravnosti algoritma [3]. *UART* priključak korišten je za komunikaciju preko terminala na računalu (*Terra Term*). Izgled same ploče prikazan je na slici 4.1.



Sl. 4.1. *ADAS* razvojna ploča *ALPHA* [15]

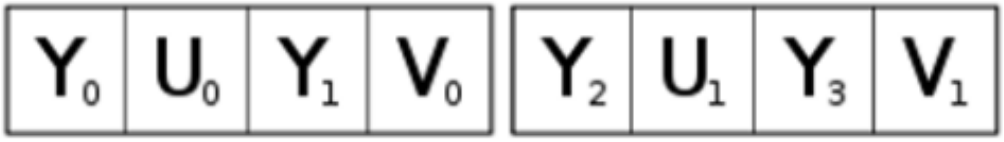
Vision SDK je odabrani softver koji se koristi uz ploču. Predstavlja programski paket orijentiran prema računalnom vidu koji je razvila tvrtka Texas Instruments, a tvrtka RT-RK je sudjelovala u kreiranju kako bi programski paket bio kompatibilan s *Alpha* pločom. Omogućuje programiranje *ADAS SoC*-ova i kreiranje različitih tokova podataka, algoritama i slučaja upotrebe (engl. *usecase*) te izvršavanje kreiranih algoritama na različitim procesorima. Također dolazi s unaprijed definiranim slučajevima upotrebe pomoću kojih se demonstriraju neke od mogućnosti upotrebe *ADAS Alpha* ploče te programskog okruženja. Neki od algoritama koje *Vision SDK* također omogućuje su algoritam za kompresiju, prikaz i analizu videa koji se mogu snimati, ali i preuzimati. Osnovni princip na kojemu se temelji *Vision SDK* je okvir *Links and Chains* i programsko sučelje *Link API*. [15]

4.2. Postupak implementacije

4.2.1. Tipovi podataka i prilagodba *C* koda za *ADAS* ploču

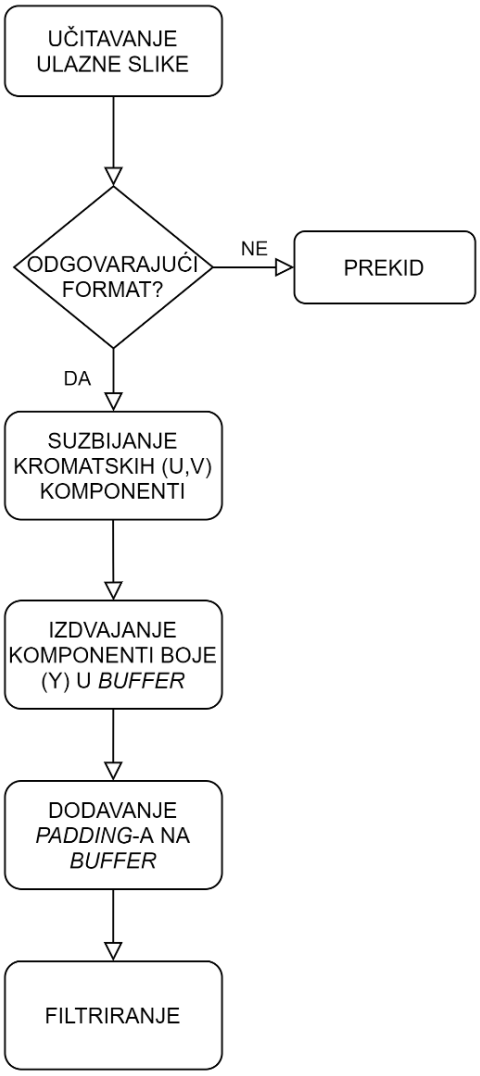
Slika koja je korištena kao ulaz u *C* kodu na osobnom računalu zapisana je kao dvodimenzionalno polje elemenata slike s vrijednostima od 0 do 255. Međutim, slika na ploči zapisana je u drugačijem formatu. Algoritmi implementirani na *ADAS* ploču mogu sadržavati ulaznu sliku zapisanu kao polje tipa *UInt8* ili *UInt32*. To znači da podaci o boji mogu biti raščlanjeni ovisno o primjeni. Algoritmi temeljeni na *UInt32* tipu sadrže četiri vrijednosti za svaki član polja: *Y, U, Y* i *V*. Ovo može biti korisno za algoritme kao što je npr. pretvaranje slike u boji u sliku zapisanu kao razine intenziteta sive, pri čemu je dovoljno samo maskirati dolazne podatke. Međutim, taj se format ne može koristiti u slučajevima u kojima je potrebno kanal u

boji obrađivati pojedinačno ili npr. pretvoriti *YUV* u *RGB* format. U algoritmu varijabla **inPtr* diktira kojim je tipom predstavljena ulazna slika. U ovom je radu korišten *YUYV* format, točnije *YUV_422I_YUYV*. Dakle, u **inPrt* zapisana je vrijednost svjetline (*Y* komponenta*)* svakog elementa slike, ali vrijednosti boje (*U* i *V* komponente) su uzorkovane na način da se izražavaju za svaki drugi element slike, tj. *U* za prvi, *V* za drugi, nakon toga opet *U* za treći itd. Prikaz navedenog formata nalazi se na slici 4.2. Ovo se uzorkovanje temelji na ljudskoj većoj osjetljivosti na luminanciju, nego na boju.



Sl. 4.2. Zapis *YUV_422I_YUYV* formata

Algoritam ovog rada temelji se na primjeni operacija filtriranja na kreirani *bufferY*. *bufferY* je polje svih vrijednosti svjetline (*Y*) elemenata ulazne slike. Komponente *U* i *V*, koje sadrže informacije o boji, nisu potrebne te ih se zbog toga prigušuje na način da se postavljaju na vrijednost 128. Na taj se način dobiva slika prikazana kao razine intenziteta sive. *bufferY* je popunjen na način da je u njega spremljen nulti, a zatim i svaki drugi element iz polja *inputPtr*. *inputPtr* je pokazivač koji je postavljen na vrijednost nultog elementa ulazne **inPtr* slike. Nakon toga se *bufferY* predaje funkciji za pretvorbu 1D u 2D polje, a zatim funkciji za *padding* i filtriranje koje su objašnjenje u trećem poglavlju. Dijagram toka implementiranog algoritma prikazan je na slici 4.3. i vrijedi za svaki filtar.



Sl. 4.3. Dijagram toka implementiranog algoritma

U odnosu na *C* kod napisan na osobnom računalu, osim u definiranju tipova podataka i ulazne slike, u *C* kodu prilagođenom za ploču postoji razlika i u alokaciji memorije. Alokacija memorije na ploči prikazana je na slici 4.5., gdje je *row* broj redova, *col* broj stupaca koji se alocira, a *2Dmat* alocirana matrica. Funkcija *Utils_memAlloc* omogućuje zauzimanje memorije iz jedne od hrpa (engl. *heap*). Osim alociranja memorije, postoje i sitnije razlike kao npr. nemogućnost deklariranja varijabli unutar zagrade *for* petlje.

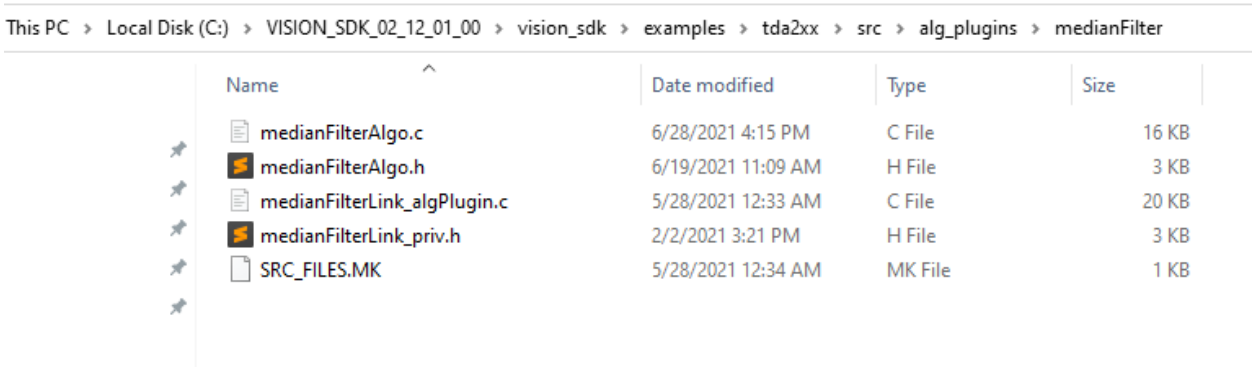
Linija Kod

```
1:      UInt8** 2Dmat = (UInt8**)Utils_memAlloc(HEAP_NAME,
        row*col, sizeof(UInt8*));
2:      for (i = 0; i < row; i++){
3:      2Dmat[i] = (UInt8*)Utils_memAlloc(HEAP_NAME,
        col, sizeof(UInt8));
```

Sl. 4.5. Alokacija memorije na ploči

4.2.2. Postupak dodavanja novog algoritma

Algorithm link je modul koji omogućuje pokretanje korisničkog koda na ploči. Korisnički kod se dodaje na način da se najprije kreira mapa sa svim izvornim datotekama. Mapa se kreira u direktoriju */vision_sdk/examples/tda2xx/src/alg_plugins* i naziva se proizvoljno. Mape u ovom radu nazvane su po nazivima filtara, a primjer takve i potrebnih datoteka nalazi se na slici 4.6. Na slici je primjer mape za kreiranje medijan algoritma, a istu stvar, tj. kreiranje mape sa svim izvornim datotekama, potrebno je napraviti i za preostala tri filtra (*moving average*, Gaussov i visokopropusni). Pri tome se nazivi mapa i datoteka također mijenjaju ovisno o nazivu filtra. U nastavku su navedeni nazivi datoteka po primjeru medijan filtra, a oni se također mijenjaju ovisno o filtru.



Sl. 4.6. Primjer mape algoritma s izvornim datotekama – medijan filtar

Nakon toga potrebno je u mapu na lokaciji */vision_sdk/include/link_api* dodati datoteku *algorithmLink_medianFilter.h*. Ova datoteka potrebna je za definiranje parametara koji povezuju *Algorithm link* modul s *VisionSDK API*-jem. Kako bi algoritam bio vidljiv, potrebno je i u datoteci na lokaciji */vision_sdk_/configs/tda2xx_evm_bios_all/uc_cfg.mk* dodati liniju *ALG_medianFilter=yes*. Za dodavanje algoritma u strukturu izgrađenog okruženja, u naredbenom retku (engl. *command prompt*) pozivaju se naredbe *gmake –s –j depend* i *gmake –s –j sbl_sd*. Provjera uspješnosti dodavanja algoritma izvodi se pozivom naredbe *gmake showconfig* nakon kojega bi naziv algoritma trebao biti vidljiv u ispisu naredbenog retka.

4.2.3. Kreiranje slučajeva upotrebe

Kako bi dodani algoritmi bili upotrebljivi, potrebno je dodati slučajeve upotrebe (engl. *usecase*). Najprije se kreira mapa u direktoriju */vision_sdk/examples/tda2xx/src/usecases* s tekstualnom datotekom koja definira veze koje će se koristiti. U ovom radu kreirana su 4 slučaja upotrebe – po jedan za svaki od implementiranih filtara (medijan, Gaussov, *moving average* i visokopropusni). Svi od navedenih slučajeva upotrebe algoritme izvršavaju na *DSP* procesoru, a

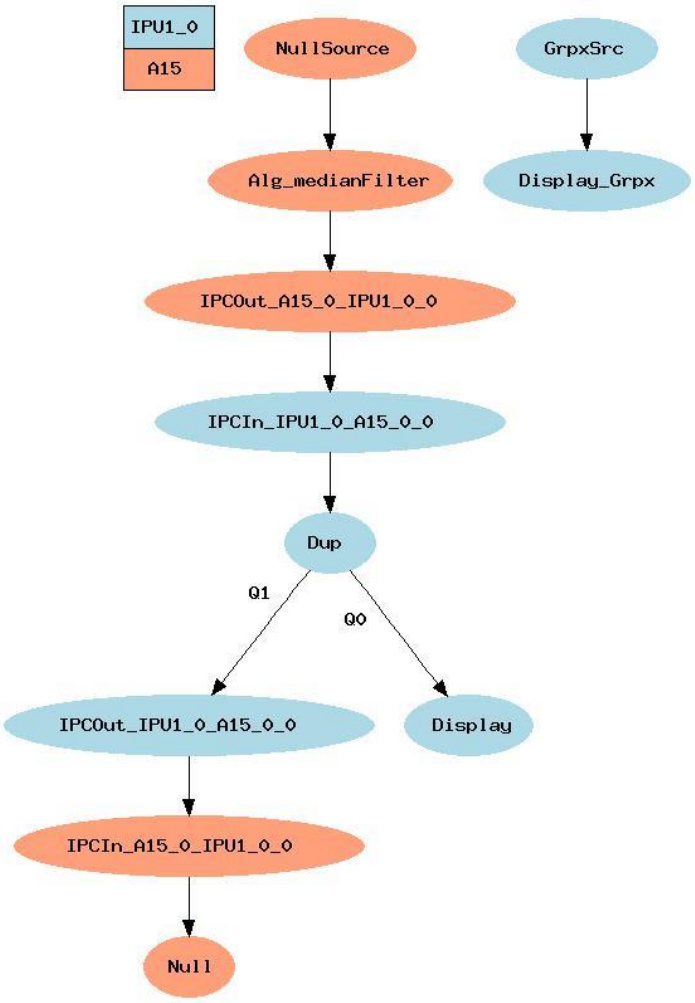
naknadno su kreirani slučajevi upotrebe s *A15* procesorima radi usporedbe i optimiziranja. Također su kreirani slučajevi s paralelnim procesorima, tj. istovremenom obradom jedne slike s dva procesora (*A15* i *DSP*). Usporedba brzine izvođenja na različitim procesorima nalazi se u petom poglavlju. Na slici 4.7. nalazi se primjer tekstualne datoteke jednog slučaja upotrebe, dok slika 4.8. prikazuje generiranu *.jpg* sliku s vezama navedenog slučaja upotrebe. Generiranje se odvija pozicioniranjem u direktorij */vision_sdk/tools/vision_sdk_usecase_gen/bin* u naredbenom retku te upotrebom naredbe *vsdk_win64.exe* kojoj se predaje putanja do direktorija i tekstualne datoteke unutar tog direktorija. U primjeru na slikama 4.7. i 4.8. prisutna je algoritamska veza za Gaussov filtar na *A15* procesoru. Algoritamska veza slijedi nakon *NullSource* veze koja omogućuje slanje slike s računala na ploču i zatim izvođenje operacija nad njom. Nakon algoritamske, slijedi *Dup* veza koja udvostručuje okvir koji se prenosi. U ovom slučaju omogućuje paralelni prikaz slike (veza *Display*) i slanje iste slike s ploče na računalo (veza *Null*).

```
UseCase: chains_gaussianFilter

NullSource (A15) -> Alg_gaussianFilter (A15) -> Dup -> Display
                                                Dup -> Null(A15)

GrpxSrc -> Display_Grpx
```

Sl. 4.7. Primjer *Links and chains usecase*-a teksturalne datoteke s A15 CPU-om



Sl. 4.8. Primjer *Links and chains usecase*-a s A15 CPU-om

Na isti način kao na slikama 4.7. i 4.8., kreiraju se i slučajevi upotrebe s *DSP* procesorom, ali i svim ostalima koje ploča sadrži (*M4*, *EVE*). Pri tome je potrebno promijeniti naziv procesora na način da bude odgovarajući te slijediti sve korake kao i za dodavanje algoritma za izvršavanje na procesoru *A15*. Procesore je moguće i kombinirati. U ovom radu kombinacija je odrađena na način da su se u algoritmu kreirala 2 spremnika (u *C* kodu *bufferY1* i *bufferY2*), pri čemu je u prvom bila prva polovica elemenata slike i obrađivala se na jednom *DSP* procesoru, a u drugom druga polovica koja se obrađivala paralelno na drugom *DSP* procesoru. Na slici 4.9. nalazi se izgled tekstualne datoteke s algoritmima koji se obrađuju paralelno na dva procesora, dok je na slici 4.10. primjer generirane *.jpg* slike istog slučaja upotrebe.

```
-> Alg_Filter_A (A15)-> Merge -> Sync -> Alg_AfterMerge (EVE1) -> Null (A15)
-> Alg_Filter_B (DSP2)-> Merge
```

Sl. 4.9. Primjer *Links and chains usecase*-a tekstualne datoteke kombinacije *A15* i *DSP CPU*-a

Sl. 4.10. Primjer *Links and chains usecase*-a kombinacije *A15* i *DSP CPU*-a

4.3. Provjera uspješnosti implementacije

Nakon kreiranja algoritama i slučajeva upotrebe (engl. *usecase*), programsko rješenje pokreće se na ploči. Najprije se postavlja statična *IP* adresa ploče, a zatim i povezuje računalo s pločom *ethernet* kablom. Nakon toga pokreću se naredbe za izgradnju (engl. *build*) kojima nastaju *MLO* i *AppImage* datoteke. Kreirane datoteke potrebno je prebaciti na *SD* karticu koja se koristi na

ploči. Slanje slika na ploču omogućuje se pokretanjem odgovarajućeg slučaja upotrebe iz izbornika slučajeva upotrebe u *tera term* softveru s podešenim odgovarajućim serijskim ulazom te brzinom prijenosa (engl. *baud rate*) iznosa 115200 signala u sekundi. Nakon toga potrebno je pozicionirati se u direktorij */vision_sdk/tools/network_tools/bin* u naredbenom retku te unijeti naredbu za slanje slike, *network_tx*, prikazanu na slici 4.11. Datoteka se na kraju naredbe zamjenjuje s imenom željene datoteke koju je potrebno slati i filtrirati, pri čemu je nužno da sadrži ekstenziji *.yuv* ili *.bin*.

```
network_tx --host_ip 192.168.10.1 --target_ip 192.168.10.2 --no_loop --files RAW_1280x720.yuv
```

Sl. 4.11. Naredba za slanje slike s računala na ploču

Za potrebe validacije korištena je i naredba za spremanje slike na računalo, *network_rx*. Validacija je obavljena na način da se u *Python* programskom jeziku napravila operacija oduzimanja slike filtrirane na osobnom računalu u *C* jeziku i slike filtrirane na ploči i spremljene pomoću naredbe na slici 4.12. Naziv datoteke na kraju naredbe moguće je zamijeniti proizvoljnim, ali mora imati ekstenziju *.bin* ili *.yuv*. Datoteke s tim ekstenzijama moguće je pregledati koristeći *yuvplayer*, program otvorenog koda pomoću kojega se mogu prikazivati slike i video signali spremljeni u *YUV* prostoru.

```
network_rx --host_ip 192.168.10.1 --target_ip 192.168.10.2 --port 7000 --files output.bin
```

Sl. 4.12. Naredba za spremanje slike s ploče na računalo

Ako je rezultat oduzimanja 0, razlike između slika na računalu i na ploči nema, tj. validacija je izvršena i implementacija je uspješna. Prije toga je nužno utvrditi da nema razlike niti između slike filtrirane ugrađenim (engl. *built in*) funkcijama filtriranja u *Python* jeziku i slika filtriranih u kreiranim korisničkim *C* funkcijama. *Python* ugrađene funkcije koje su korištene pri validaciji prikazane su na slici 4.13. Funkcija na liniji 1 predstavlja filtriranje *moving average* filtrom, na liniji 2 Gaussovim, 3 medijan i 4 visokopropusnim filtrom, gdje je *kernel* korisnički definirana maska. Parametar *src* u svakoj funkciji predstavlja izvornu, tj. originalnu, sliku s podešenom putanjom. *ksize* veličinu kernela koji se koristi prilikom filtriranja, a *sigmaX* i *sigmaY* parametrir koji određuju širinu Gaussove krivulje, tj. standardnu devijaciju. Parametar *ddepth* u funkciji za visokopropusno filtriranje predstavlja dubinu odredišta, tj. dubinu izlazne slike. U ovom je radu za parametar *ddepth* predana vrijednost -1 jer s tim iznosom ulazna i izlazna slika imaju jednaku dubinu.

Linija ***Kod***

```
1:      cv2.blur(src, ksize)
```

```
2:       cv2.GaussianBlur(src, ksize, sigmaX, sigmaY);
3:       cv2.medianBlur(src, ksize);
4:       cv2.filter2D(src, ddepth, kernel);
```

Sl. 4.13. *Python* ugrađene funkcije filtriranja

4.4. Optimizacija rješenja

Nakon uspješne validacije algoritamskog rješenja, izvršen je postupak optimizacije. Optimizacija je izvedena radi povećanja brzine izvođenja algoritama, odnosno radi povećanja efikasnosti istih. U ovom je radu osnovna podjela optimizacije na sklopovsku i programsku.

Sklopovska optimizacija odnosi se na raspoređivanje zadataka na različite resurse *SoC*-a i paralelizaciju poslova. Konkretnije, početni slučaj upotrebe u kojemu su zadaci izvršavani na *DSP* procesoru zamijenjen je slučajem upotrebe s *A15* procesorom, a nakon toga su kreirani slučajevi upotrebe u kojima dva *DSP* procesora zadatke obavljaju paralelno. Paralelizacija je izvedena na način da prvu polovicu slike obrađuje prvi *DSP* procesor, a drugu polovicu drugi *DSP* procesor. Rezultati testiranja i usporedbe brzine izvođenja na različitim procesorima prikazani su u petom poglavlju.

Programska optimizacija obuhvaća bilo koje metode modifikacije koda za poboljšanje kvalitete i učinkovitosti koda. Program se može optimizirati tako da postane manje veličine, brže se izvršava ili izvodi manje ulazno-izlaznih operacija. Osnovni zahtjev koji metode optimizacije trebaju zadovoljavati jest da optimizirani program mora imati iste izlaze i nuspojave kao i njegova neoptimizirana verzija. Međutim, ovaj se zahtjev može zanemariti u slučaju da se procijeni da je korist od optimizacije važnija. Neke od osnovnih tehnika softverske optimizacije korištene u ovom radu navedene su i objašnjene u nastavku.

Spajanje petlji (engl. *loop jamming* ili *loop fusion*) je tehnika u kojoj se spajaju, tj. kombiniraju dvije ili više petlji koje se izvršavaju nad istim skupom podataka, odnosno elemenata. Dobitak je u tome da se odbacuju nepotrebne iteracije u petljama i izvršava posao jednim prolaskom kroz skup podataka. Primjer petlji prije i nakon spajanja u ovom radu nalazi se na slikama 4.14. i 4.15. Pri tome je početna vrijednost iteriranja, tj. početna vrijednost varijable *i* (slika 4.15., linija 2) prilagođena petlji za alokaciju s kojom se ujedinila (slika 4.14., linija 2). Također, u originalnoj verziji koda se unutar *for* petlji odvijaju operacije koje na prikazanim slikama nisu navedene zbog veće jasnoće prikaza objašnjene metode optimizacije.

Linija Kod

```
1:   UInt8** PaddingMat= (UInt8**)Utils_memAlloc(HEAP_NAME, row*col,
     sizeof(UInt8*));
```

```
2:   for (i = 0; i < row; i++){
3:   PaddingMat [i] = (UInt8*)Utils_memAlloc(HEAP_NAME, col, sizeof(UInt8));
4:      }
5:   for (i = offset; i < row - offset; i++) {
6:     for (j = 0; j < col; j++){
7:     }
8:    }
```

Sl. 4.14. Dio *C* koda prije optimizacije tehnikom spajanje petlji

Linija Kod

```
1:   UInt8** PaddingMat= (UInt8**)Utils_memAlloc(HEAP_NAME, row*col,
     sizeof(UInt8*));
2:   for (i = 0; i < row; i++) {
3:      PaddingMat [i] = (UInt8*)Utils_memAlloc(HEAP_NAME,col,sizeof(UInt8));
4:     for (j = 0; j < col; j++) {
5:        }
6:   }
```

Sl. 4.15. Dio *C* koda nakon optimizacije tehnikom spajanje petlji

Tehnika „razmotavanje“ petlje (engl. *unrolling*) dokazuje činjenicu da manji broj linija koda ne znači nužno brže ili jednostavnije izvršavanje programa. Potpuno razvijanje petlje brzo je rješenje koje pri tome radi ispravno. Međutim, ova tehnika pogodna je uglavnom samo za manji broj elemenata. Za veliki broj elemenata ili u slučajevima kada je nepoznat broj iteracija koje će se izvršiti, razmotavanje petlje nije praktično. U ovom je radu ova tehnika upotrijebljena kod Gaussovog filtra. Točnije, primijenjena je za pristup elementima Gaussovog kernela za tri različite veličine: 3x3, 5x5 i 7x7. Broj elemenata dovoljno je mali da bi se metoda smatrala učinkovitom i optimalnom. Optimizacija ovom tehnikom izvedena je na način da se elementima Gaussovog kernela pristupa pojedinačno, „hardkodiranjem“, a ne prolaskom kroz redove i stupce kernela te pristupanjem elementima pomoću *for* petlje. Ovakvo razmotavanje petlje šteti čitljivosti koda, ali generalno ne stvara druge probleme. Dizajn ovakvog koda dio je loše programerske prakse, ali specifična situacija može zahtijevati upotrebu iste radi postizanja određenih ciljeva, najčešće brzine ili veličine. Konkretno u ovom slučaju, za obradu podataka tijekom upravljanja vozilom nužno je da bude u stvarnom vremenu (engl. *real time*), što znači da je brzina filtriranja slike u automotivu daleko bitnija od dizajna koda. Također, pri konkretnoj implementaciji na ploču namijenjenoj za vozilo unaprijed se definira najoptimalnija veličina kernela zbog čega je jednostavno upotrijebiti tehniku razmotavanja petlje. Primjer petlje prije i poslije optimizacije tehnikom razmotavanja petlje nalazi se na slikama 4.16. i 4.17. Navedeni primjer je za slučaj Gaussovog kernela veličine 3x3 elementa slike.

Linija Kod

```
1:   for (i = offset; i < (row – offset); i++{
2:      for(j = offset; j < (col – offset); j++{
```

```
3:          for (k = -offset; k < (offset + 1); k++){
4:            for (l = -offset; l < (offset + 1); l++){
5:               result = paddingMat[i + k][j + l] * kernelMat[k + offset][l +
                 offset];
6:               sum += result;
7:             }
8:          }
9:        matFiltered[i][j] = sum;
10:       sum = 0;
11:      }
12:    }
```

Sl. 4.16. Dio *C* koda prije optimizacije tehnikom razmotavanja petlje

Linija Kod

```
1:    for (i = offset; i < (row – offset); i++{
2:      for(j = offset; j < (col – offset); j++{
3:        sum += round(paddingMat[i-1][j-1] * kernelmat[0][0]);
4:        sum += (paddingMat[i-1][j] * kernelmat[0][1]);
5:        sum += (paddingMat[i-1][j+1] * kernelmat[0][2]);
6:        sum += (paddingMat[i][j-1] * kernelmat[1][0]);
7:        sum += (paddingMat[i][j] * kernelmat[1][1]);
8:        sum += (paddingMat[i][j+1] * kernelmat[1][2]);
9:        sum += (paddingMat[i+1][j-1] * kernelmat[2][0]);
10:       sum += (paddingMat[i+1][j] * kernelmat[2][1]);
11:       sum += (paddingMat[i+1][j+1] * kernelmat[2][2]);
12:      matFiltered[i][j] = sum;
13:      sum = 0;
14:     }
15:   }
```

Sl. 4.17. Dio *C* koda nakon optimizacije tehnikom razmotavanja petlje

Tehnika minimiziranja posla koji se obavlja unutar petlji jedna je od ključnih tehnika optimizacije. Ako postoji mogućnost da se dio izraza izračuna izvan petlje na način da se u petlji iskoristi samo rezultat, upravo to je potrebno i učiniti kako bi se ukupan broj računanja najčešće znatno smanjio. Ta je metoda dobra programerska praksa, a ponekad čak osim što ubrzava izvođenje, poboljšava i čitljivost koda. Na slikama 4.18. i 4.19. nalaze se primjeri iz ovoga rada prije i nakon upotrebe tehnike minimiziranja posla unutar petlje. U navedenom su primjeru računski izrazi (slika 4.18. , linije 3,4,5) zamijenjeni varijablama *a* i *b* u koje su spremljene iste računske operacije.

Linija Kod

```
1:    for (i = 0; i < kernel_size; i++{
2:      for(j = offset; j < kernel_size ; j++{
3:         double x = i – (kernel_size – 1) / 2.0;
4:         double x = i – (kernel_size – 1) / 2.0;
5:         gauss[i][j] = 1 * exp(((pow(x, 2) + pow(y, 2)) /
                        ((2 * pow(sigma, 2)))) * (-1));
6:        sum+= gauss[i][j];
7:      }
8:    }
```

Sl. 4.18. Dio *C* koda prije optimizacije tehnikom minimiziranja posla unutar petlje

Linija Kod

```
1:   int a = (kernel_size – 1) / 2.0;
2:   int b = 2 * pow(sigma, 2);
3:   for (i = 0; i < kernel_size; i++{
4:      for(j = offset; j < kernel_size ; j++{
5:         double x = i – a;
6:         double y = j – a;
7:         gauss[i][j] = 1 * exp(((pow(x, 2) + pow(y, 2)) / b) * (-1));
8:        sum+= gauss[i][j];
9:      }
10:  }
```

Sl. 4.19. Dio *C* koda nakon optimizacije tehnikom minimiziranja posla unutar petlje

Bržem izvođenju zadataka pridonosi i optimizacijska metoda kod koje se uvjeti redaju na način da onaj koji će najvjerojatnije biti istinit bude prvi. Ovaj princip odnosi se na *case* naredbu te *if else* uvjete. Primjer rasporeda uvjeta u ovome radu prikazan je na slici 4.20. U ovom je slučaju najprije potrebno definirati uvjet koji ispunjava najveći broj elemenata slike, tj. dio slike koji se nalazi između definiranih granica *offset*-a, odnosno između dodanih stupaca tijekom *padding*-a. Zatim je postavljen uvjet koji obuhvaća samo dodane stupce s lijeve strane slike u ovom primjeru. Pojam *padding* i *offset* detaljnije su objašnjeni u trećem poglavlju.

Linija Kod

```
1:   for (i = 0; i < row; i++{
2:      for(j = 0; j < col; j++{
3:         if (j >= offset && j < col – offset){
4:              PaddingMat[][] = mat[I – offset][j – offset];
5:         }
6:         else if (j >= 0 && j < offset){
7:              PaddingMat[][] = mat[i – offset][cntLeft - j];
8:         }
9:       }
10:    }
```

Sl. 4.20. Optimizirani raspored uvjeta

Prilikom optimizacije također je izmijenjena tehnika računanja sume kod *moving average* filtra. Nova tehnika bazira se na smanjenju broja računskih operacija zbrajanja na način da se prilikom prve iteracije zbrajanja svih elemenata unutar kernela u poseban spremnik sprema rezultat zbrajanja onih elemenata slike koji će se odbaciti u sljedećoj iteraciji, tj. koji će se od ukupne sume oduzeti. Ista stvar odvija se u svakoj sljedećoj iteraciji. Na slici 4.20. je primjer polja s maskom postavljenom na prvi skup elemenata pri čemu su osjenčani oni elementi koji se koriste i u sljedećoj iteraciji. Nadalje, razlici ukupne sume i navedenog spremnika dodaje se zbroj novih elemenata slike. Drugim riječima, izračunavanje konačne sume svodi se na oduzimanje elemenata prvog stupca prethodne iteracije od sume svih elemenata iste iteracije i zatim

dodavanje narednog stupca. Na slici 4.21. je crveno osjenčan stupac koji se oduzima od sume prve iteracije, a zeleno stupac koji se dodaje u drugoj iteraciji filtriranja – žuti skup podataka zajednički je objema iteracijama i upravo se ta činjenica iskorištava.

132	133	113	134	84	89	87
112	131	84	85	82	85	87
101	123	83	84	92	90	94
95	120	73	124	128	97	97
95	94	93	100	73	94	131
45	84	83	85	82	90	180
150	94	72	83	123	115	125

Sl. 4.20. Primjer zajedničkog skupa podataka u dvjema susjednim iteracijama pri *moving averegare* filtriranju

132	133	113	134	84	89	87
112	131	84	85	82	85	87
101	123	83	84	92	90	94
95	120	73	124	128	97	97
95	94	93	100	73	94	131
45	84	83	85	82	90	180
150	94	72	83	123	115	125

Sl. 4.21. Primjer pomoćnih spremnika u optimizaciji izračuna sume pri *moving average* filtriranju

Tablica 4.1. prikazuje rezultate testiranja brzine izvođenja prije i nakon upotrebe softverskih tehnika optimizacije. Testiranje je izvođeno nad filtrima s kernelima veličine 3x3 elemenata slike. Konačno optimizirani algoritmi implementirani su na ploču i nad njima se nadalje izvršavala sklopovska optimizacija s rezultatima navedenim u petom poglavlju.

Tablica 4.1. Tijek brzine izvođenja filtara na *DSP* procesoru pri različitim koracima optimizacije

TIP FILTRA	VRIJEME IZVRŠAVANJA[ms]	VREMENSKA RAZLIKA[ms]	OPTIMIZACIJSKA METODA
MEDIAN FILTAR	2754	-	Bez optimizacije
	2113	641	Minimiziranje posla u petlji
	1730	383	Optimizirani raspored uvjeta
	1484	246	Spajanje petlji
MOVING AVERAGE FILTAR	3242	-	Bez optimizacije
	2690	552	Minimiziranje posla u petlji
	1823	867	Novi način računanja sume
	1536	287	Optimizirani raspored uvjeta
	1243	293	Spajanje petlji
VISOKOPROPUSNI FILTAR	1931	-	Bez optimizacije
	1293	638	Optimizirani raspored uvjeta
	605	688	Spajanje petlji
GAUSSOV FILTAR	3677	-	Bez optimizacije
	3412	265	Minimiziranje posla u petlji
	3038	374	Optimizirani raspored uvjeta
	2839	199	Spajanje petlji
	1733	1106	Razmotavanje petlje

5. TESTIRANJE RADA IMPLEMENTIRANIH RJEŠENJA

Svaki od tri *SoC*-a koja sadrži *Alpha* ploča, pa tako i *SC* koji je korišten u ovom radu, sadrži nekoliko različitih procesorskih jezgri. Korisniku je omogućeno raspoređivanje zadataka između 2 *DSP*, dva *A15*, dva *M4* i četiri *EVE* procesora. Razlika između navedenih procesora je u radnim frekvencijama i namjeni. U ovom radu testirana je brzina izvođenja implementiranih algoritama na testnom skupu od deset slika u tri slučaja: izvršavanje zadataka na jednom *DSP* procesoru, na jednom *A15* procesoru i paralelno izvršavanje na dvama *DSP* procesorima. Testne slike mogu se vidjeti u prilogu P.5.1 na priloženom *CD*-u. Usporedba nije izvođena samo na temelju različitih procesora, nego i na temelju triju različitih rezolucija: 1280x720, 960x540 i 640x360 elemenata slike. Također, testiranje je izvedeno i za tri različite veličine kernela: 3x3, 5x5 i 7x7 elemenata slike. Kombinacije testiranja provedene su na sva četiri implementirana filtra. Nakon testiranja brzine izvođenja algoritama na deset različitih slika, izračunate su standardna devijacija i srednja vrijednost. Standardna devijacija računala se kako bi se definiralo prosječno srednje kvadratno odstupanje numeričkih vrijednosti rezultata brzine izvođenja od njihove aritmetičke sredine. Aritmetička sredina vrijednost je korištena pri usporedbi rezultata. Rezultati su navedeni i uspoređeni u nastavku.

5.1. Rezultati testiranja brzine izvođenja algoritma na *DSP* procesoru - *single threaded*

U tablicama 5.1. do 5.4. nalaze se rezultati testiranja brzine izvođenja implementiranih algoritama na *DSP* procesoru za tri različite rezolucije i tri različite veličine kernela za svaki filtar.

Tablica 5.1. Brzina izvođenja medijan filtra na *DSP* procesoru

Filtar	MEDIJAN FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	1561	975	540	5128	2960	1445	15914	8981	4051
Slika 2[ms]	1484	977	540	5213	2933	1445	15851	8979	4049
Slika 3[ms]	1485	892	457	5214	2934	1446	15852	8982	4048
Slika 4[ms]	1563	974	456	5131	2934	1445	15847	8980	4048
Slika 5[ms]	1567	978	541	5209	2953	1445	15926	9062	4132
Slika 6[ms]	1564	894	456	5127	3118	1445	15847	8980	4049

Slika 7[ms]	1563	975	455	5213	3128	1393	15848	9067	4132
Slika 8[ms]	1568	974	456	5211	3127	1363	15933	8980	4048
Slika 9[ms]	1568	975	456	5213	3119	1446	15851	8980	4049
Slika10[ms]	1563	974	450	5210	2931	1446	15842	8978	4049
Sr.vr. [ms]	1549	958	490	5187	3014	1432	15871	8997	4066
St.dev.	34	32	44	40	95	29	37	36	35

Tablica 5.2. Brzina izvođenja *moving average* filtra na *DSP* procesoru

Filtar	*MOVING AVERAGE* FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	1243	828	473	1231	823	386	1296	828	390
Slika 2[ms]	1247	828	474	1312	823	388	1343	748	389
Slika 3[ms]	1243	743	390	1227	741	387	1296	829	390
Slika 4[ms]	1243	745	472	1311	742	387	1212	830	389
Slika 5[ms]	1245	827	472	1226	736	387	1232	747	391
Slika 6[ms]	1324	827	474	1308	825	387	1294	748	390
Slika 7[ms]	1326	746	472	1231	825	389	1338	832	389
Slika 8[ms]	1330	745	388	1313	739	387	1215	748	390
Slika 9[ms]	1326	828	389	1316	741	388	1252	829	389
Slika10[ms]	1225	829	473	1232	741	388	1296	749	390
Sr.vr. [ms]	1285	795	448	1271	774	387	1277	789	390
St. dev.	43	43	41	44	43	1	47	43	1

Tablica 5.3. Brzina izvođenja visokopropusnog filtra na *DSP* procesoru

Filtar	VISOKOPROPUSNI FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	607	387	216	695	427	236	766	486	260
Slika 2[ms]	607	387	215	696	426	235	799	486	260
Slika 3[ms]	604	388	217	693	426	233	863	485	261
Slika 4[ms]	605	387	216	693	426	233	776	485	260

Slika 5[ms]	605	385	216	693	425	235	733	487	260
Slika 6[ms]	605	387	217	694	426	234	777	487	261
Slika 7[ms]	606	387	216	693	427	234	840	486	259
Slika 8[ms]	604	387	216	694	426	234	866	486	260
Slika 9[ms]	607	388	217	692	426	233	779	485	260
Slika10[ms]	605	385	216	693	425	234	830	488	261
Sr.vr. [ms]	606	387	216	694	426	234	806	486	260
St. dev.	1	1	1	1	1	1	40	1	1

Tablica 5.4. Brzina izvođenja Gaussovog filtra na *DSP* procesoru

Filtar	GAUSSOV FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	1726	1038	507	3762	2121	1071	6496	3659	1755
Slika 2[ms]	1735	1124	506	3713	2143	996	6449	3683	1764
Slika 3[ms]	1735	1125	507	3701	2145	1075	6446	3673	1677
Slika 4[ms]	1742	1124	507	3732	2148	1084	6465	3692	1767
Slika 5[ms]	1745	1122	507	3825	2155	1085	6478	3698	1771
Slika 6[ms]	1780	1039	506	3599	2123	1051	6563	3614	1735
Slika 7[ms]	1745	1125	507	3741	2155	1058	6562	3782	1772
Slika 8[ms]	1733	1124	509	3708	2148	1078	6530	3759	1747
Slika 9[ms]	1732	1123	506	3704	2162	993	6438	3759	1762
Slika10[ms]	1738	1041	506	3809	2147	999	6553	3771	1767
Sr.vr. [ms]	1741	1099	507	3729	2165	1053	6498	3709	1752
St.dev.	15	41	1	63	49	40	50	56	29

5.2. Rezultati testiranja brzine izvođenja algoritma na *A15* procesoru - *single threaded*

U tablicama 5.5. do 5.8. nalaze se rezultati testiranja brzine izvođenja implementiranih algoritama na *A15* procesoru za tri različite rezolucije i tri različite veličine kernela za svaki filtar.

Tablica 5.5. Brzina izvođenja medijan filtra na *A15* procesoru

Filtar	MEDIJAN FILTAR								
Procesor	*A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	876	547	298	4134	2333	1094	14256	8255	3650
Slika 2[ms]	961	550	299	4158	2347	1096	14268	8373	3664
Slika 3[ms]	957	547	296	4032	2409	1087	14022	8418	3638
Slika 4[ms]	952	545	297	4017	2405	1085	15249	8321	3627
Slika 5[ms]	860	539	293	3957	2283	1153	14020	8440	3583
Slika 6[ms]	872	545	297	4114	2326	1089	14004	8215	3647
Slika 7[ms]	874	546	297	4039	2330	1090	14386	8362	3723
Slika 8[ms]	880	551	299	4080	2347	1181	14294	8259	3739
Slika 9[ms]	965	548	297	4170	2350	1097	15386	8289	3663
Slika10[ms]	874	545	297	4124	2329	1091	14166	8636	3724
Sr.vr. [ms]	907	546	297	4083	2346	1106	14405	8357	3666
St. dev.	45	3	2	69	37	33	499	123	49

Tablica 5.6. Brzina izvođenja *moving average* filtra na *A15* procesoru

Filtar	*MOVING AVERAGE* FILTAR								
Procesor	*A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	454	309	193	452	323	199	489	418	200
Slika 2[ms]	457	308	194	452	325	200	491	424	202
Slika 3[ms]	452	311	192	453	322	198	574	438	200
Slika 4[ms]	456	311	194	454	323	200	574	412	201
Slika 5[ms]	454	311	192	456	324	198	570	432	202
Slika 6[ms]	455	310	193	456	323	200	487	418	200
Slika 7[ms]	453	310	193	457	324	198	489	446	202
Slika 8[ms]	452	318	192	454	324	200	489	425	201
Slika 9[ms]	458	308	192	458	325	199	491	413	201
Slika10[ms]	453	310	193	452	324	199	569	417	201
Sr.vr. [ms]	454	310	193	474	324	199	522	424	201
St. dev.	2	1	1	2	1	1	43	11	1

Tablica 5.7. Brzina izvođenja visokopropusnog filtra na *A15* procesoru

Filtar	VISOKOPROPUSNI FILTAR								
Procesor	*A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	343	239	148	445	290	177	526	483	198
Slika 2[ms]	342	240	150	541	290	176	522	485	197
Slika 3[ms]	344	241	148	447	290	178	526	480	198
Slika 4[ms]	342	240	149	449	290	177	524	479	198
Slika 5[ms]	342	240	149	448	290	176	526	482	197
Slika 6[ms]	344	240	148	447	291	177	526	473	198
Slika 7[ms]	342	240	150	448	292	175	525	464	198
Slika 8[ms]	342	240	147	447	291	177	527	473	198
Slika 9[ms]	344	239	149	446	290	178	526	482	199
Slika10[ms]	344	240	149	447	291	176	537	480	197
Sr.vr. [ms]	343	230	149	457	291	177	527	478	198
St. dev.	1	1	1	30	1	1	4	6	1

Tablica 5.8. Brzina izvođenja Gaussovog filtra na *A15* procesoru

Filtar	GAUSSOV FILTAR								
Procesor	*A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	639	411	210	1119	671	339	1827	1215	603
Slika 2[ms]	664	412	206	1138	675	343	1823	1301	596
Slika 3[ms]	647	498	209	1128	675	341	1874	1224	613
Slika 4[ms]	661	496	204	1198	664	335	1892	1269	678
Slika 5[ms]	669	496	203	1192	662	333	1880	1276	598
Slika 6[ms]	686	413	213	1204	669	340	1880	1336	620
Slika 7[ms]	670	414	203	1108	662	335	1867	1269	596
Slika 8[ms]	651	497	207	1121	670	337	1819	1194	599
Slika 9[ms]	656	412	209	1144	692	344	1889	1306	612
Slika10[ms]	659	380	205	1110	664	334	1890	1263	678
Sr.vr. [ms]	660	442	207	1146	670	338	1864	1265	619
St. dev.	13	49	3	38	9	4	29	44	32

5.3. Rezultati testiranja brzine izvođenja algoritma na dva paralelna *DSP* procesora

U tablicama 5.9. do 5.12. nalaze se rezultati testiranja brzine izvođenja implementiranih algoritama na paralelnim *DSP* procesorima za tri različite rezolucije i tri različite veličine kernela za svaki filtar.

Tablica 5.9. Brzina izvođenja medijan filtra na *DSP + A15* procesorima

Filtar	MEDIJAN FILTAR								
Procesor	*DSP + A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	902	521	263	4003	2214	962	13001	8123	3390
Slika 2[ms]	906	518	261	3989	2234	969	13019	8109	3385
Slika 3[ms]	906	520	263	3984	2232	968	13037	8108	3387
Slika 4[ms]	903	523	261	4011	2214	962	13032	8114	3396
Slika 5[ms]	908	520	257	4008	2229	974	13006	8123	3395
Slika 6[ms]	911	523	261	3985	2202	963	13001	8125	3378
Slika 7[ms]	912	529	260	4014	2201	975	13042	8126	3393
Slika 8[ms]	908	520	269	4002	2214	964	13008	8119	3378
Slika 9[ms]	906	521	258	4003	2221	971	13023	8111	3379
Slika10[ms]	911	528	260	4011	2207	971	13028	8115	3381
Sr.vr. [ms]	907	522	261	4001	2216	967	13019	8117	3386
St. dev.	3	3	3	11	11	5	14	6	7

Tablica 5.10. Brzina izvođenja *moving average* filtra na *DSP + A15* procesorima

Filtar	*MOVING AVERAGE* FILTAR								
Procesor	*DSP + A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	813	639	421	927	671	418	894	841	432
Slika 2[ms]	823	633	419	916	679	416	881	844	428
Slika 3[ms]	812	626	426	917	667	415	898	841	432
Slika 4[ms]	812	624	424	925	673	415	889	836	434

Slika 5[ms]	811	630	425	928	667	423	906	824	423
Slika 6[ms]	824	639	427	917	676	418	892	824	435
Slika 7[ms]	812	627	412	919	680	415	891	844	427
Slika 8[ms]	815	636	428	933	674	417	904	846	432
Slika 9[ms]	816	622	412	933	673	423	909	843	431
Slika10[ms]	823	621	424	918	680	422	896	839	433
Sr.vr. [ms]	816	629	421	923	674	418	896	832	430
St. dev.	5	6	6	6	5	3	8	8	3

Tablica 5.12. Brzina izvođenja visokopropusnog filtra na *DSP + A15* procesorima

Filtar	VISOKOPROPUSNI FILTAR								
Procesor	*DSP + A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	301	223	155	428	272	156	502	442	163
Slika 2[ms]	299	222	154	424	272	158	516	442	167
Slika 3[ms]	298	222	156	426	272	157	506	438	164
Slika 4[ms]	302	219	155	428	263	159	502	439	168
Slika 5[ms]	301	218	155	427	274	158	504	438	166
Slika 6[ms]	298	225	154	426	269	157	504	443	163
Slika 7[ms]	302	224	153	419	273	156	502	442	167
Slika 8[ms]	301	218	155	418	272	158	506	445	165
Slika 9[ms]	301	223	152	423	274	159	508	437	169
Slika10[ms]	298	225	155	417	273	156	516	438	168
Sr.vr. [ms]	300	221	154	423	271	157	506	440	166
St. dev.	2	3	1	4	3	1	5	3	2

Tablica 5.12. Brzina izvođenja Gaussovog filtra na *DSP + A15* procesorima

Filtar	GAUSSOV FILTAR								
Procesor	*DSP + A15*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	1390	843	431	2328	1442	621	3235	2368	1252
Slika 2[ms]	1398	842	429	2323	1456	624	3224	2352	1254

Slika 3[ms]	1372	843	428	2318	1442	621	3235	2362	1252
Slika 4[ms]	1398	838	436	2325	1450	626	3223	2367	1245
Slika 5[ms]	1381	839	433	2332	1450	619	3226	2354	1252
Slika 6[ms]	1369	837	437	2318	1442	625	3247	2358	1245
Slika 7[ms]	1362	839	433	2332	1450	616	3244	2362	1255
Slika 8[ms]	1369	843	438	2308	1453	619	3237	2368	1252
Slika 9[ms]	1396	842	432	2316	1443	618	3248	2362	1247
Slika10[ms]	1368	846	434	2308	1453	618	3234	2365	1253
Sr.vr. [ms]	1380	841	433	2320	1448	620	3235	2361	1250
St. dev.	13	3	3	8	5	3	9	5	3

5.1. Rezultati testiranja brzine izvođenja algoritma na osobnom računalu

U tablicama 5.13. do 5.16. nalaze se rezultati testiranja brzine izvođenja implementiranih algoritama na osobnom računalu za tri različite rezolucije i tri različite veličine kernela za svaki filtar.

Tablica 5.33. Brzina izvođenja medijan filtra na osobnom računalu

Filtar	MEDIJAN FILTAR								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	820	453	239	1947	1024	491	4776	2288	1160
Slika 2[ms]	805	462	233	1846	1003	485	4496	2352	1137
Slika 3[ms]	791	435	227	1851	1018	481	4499	2377	1166
Slika 4[ms]	794	438	227	1864	1029	483	4545	2362	1139
Slika 5[ms]	834	444	238	1841	1028	481	4473	2324	1154
Slika 6[ms]	787	456	236	1936	1020	492	4693	2277	1113
Slika 7[ms]	797	453	242	1889	1041	486	4529	2273	1153
Slika 8[ms]	812	432	231	1948	1023	495	4465	2345	1172
Slika 9[ms]	824	462	242	1852	1043	492	4592	2332	1123
Slika10[ms]	836	455	245	1856	1032	485	4724	2373	1135
Sr.vr. [ms]	810	449	236	1883	1026	487	4579	2330	1145
St.dev.	17	10	6	42	11	5	106	37	18

Tablica 5.14. Brzina izvođenja *moving average* filtra na osobnom računalu

Filtar	*MOVING AVERAGE* FILTAR

Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	713	357	189	730	404	190	868	431	192
Slika 2[ms]	724	378	183	734	442	208	848	450	206
Slika 3[ms]	725	404	202	733	440	201	868	418	192
Slika 4[ms]	735	404	203	729	419	203	856	436	194
Slika 5[ms]	715	399	205	739	439	205	833	459	197
Slika 6[ms]	728	388	187	718	427	209	863	414	204
Slika 7[ms]	724	376	199	745	437	197	852	429	202
Slika 8[ms]	719	394	197	722	445	212	869	437	192
Slika 9[ms]	737	406	205	747	413	204	846	436	206
Slika10[ms]	726	384	184	735	452	198	856	456	204
Sr.vr. [ms]	725	389	195	733	431	203	856	437	199
St. dev.	7	15	8	9	15	6	11	14	6

Tablica 5.15. Brzina izvođenja visokopropusnog filtra na osobnom računalu

Filtar	VISOKOPROPUSNI FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	702	337	169	675	371	183	712	422	205
Slika 2[ms]	655	372	176	640	427	190	785	473	219
Slika 3[ms]	665	393	177	688	424	186	819	463	222
Slika 4[ms]	672	397	178	688	386	184	783	457	223
Slika 5[ms]	681	375	166	658	394	178	738	487	235
Slika 6[ms]	682	362	175	656	434	190	797	465	207
Slika 7[ms]	700	383	183	675	438	185	784	442	215
Slika 8[ms]	696	378	169	642	391	187	765	452	230
Slika 9[ms]	672	357	176	688	384	199	809	475	207
Slika10[ms]	667	345	185	654	395	195	813	465	202
Sr.vr. [ms]	679	370	175	666	404	188	781	460	217
St. dev.	15	19	6	18	23	6	32	17	11

Tablica 5.16. Brzina izvođenja Gaussovog filtra na osobnom računalu

Filtar	GAUSSOV FILTAR								
Procesor	*DSP*								
Maska[px]	3x3			5x5			7x7		
Rez.[px]	1280x720	960x540	640x360	1280x720	960x540	640x360	1280x720	960x540	640x360
Slika 1[ms]	783	530	217	1379	789	395	1727	986	468
Slika 2[ms]	798	550	266	1445	837	359	1806	1069	491
Slika 3[ms]	768	592	230	1427	849	362	1829	1021	473
Slika 4[ms]	798	545	253	1398	838	378	1819	998	482
Slika 5[ms]	762	567	261	1387	792	371	1827	1013	495
Slika 6[ms]	768	548	260	1442	840	365	1783	1078	475
Slika 7[ms]	791	590	225	1435	785	389	1790	989	465
Slika 8[ms]	765	582	238	1396	852	394	1826	997	491
Slika 9[ms]	789	567	219	1438	791	362	1730	1022	490
Slika10[ms]	788	579	254	1440	803	390	1756	1008	482
Sr.vr. [ms]	781	565	242	1418	817	377	1789	1018	481
St.dev.	13	20	18	24	26	14	37	30	10

5.2. Usporedba dobivenih rezultata

Uzevši u obzir procesore na kojima su izvedeni implementirani algoritmi, najveću brzinu izvođenja daje *A15* procesor. U nekim je slučajevima riječ o dvostruko većoj brzini nego kod *DSP* procesora. Paralelno raspoređivanje zadataka na 2 *DSP* procesora rezultira nešto manjom brzinom od *A15*, ali znatno većom od slučaja obavljanja zadataka na samo jednom *DSP* procesoru, u jednom redu (engl. *single threaded*). Najbolji rezultati izvođenja algoritama na ploči uspoređeni su s brzinom izvođenja istih algoritama na osobnom računalu i uočeno je malo vremensko odstupanje. Riječ je o razlici do 35 ms, dok je prije programske i sklopovske optimizacije bilo riječ o nekoliko sekundi u korist osobnog računala. Veličina kernela najveću ulogu u brzini izvođenja ima kod medijan filtra. Velika ovisnost brzine o veličini kernela tog filtra leži u potrebi za sortiranjem elemenata unutar kernela te pronalaskom medijan vrijednosti unutar sortiranog niza, što je dugotrajan proces ako je kernel velik. Smanjenje veličine kernela sa 7x7 na 3x3 može ubrzati vrijeme izvođenja do čak 10 puta. Nasuprot tome, veličina kernela igra neznatnu ulogu kod *moving average* filtra upravo zbog pomoćnih spremnika dodanih tijekom optimizacije. Gaussov filtar, kao i medijan, znatno ovisi o veličini kernela. Smanjenjem

rezolucije također se može ubrzati izvođenje za oko 2 puta. Visokopropusni filtar, za razliku od Gaussovog i medijan filtra, u znatno manjoj mjeri ovisi o veličini kernela. Ovisno o slučaju, ponekad je potrebno „izvagati“ hoće li se optimalnije rješenje postići smanjenjem rezolucije ili smanjenjem veličine kernela. Generalno, najveća brzina izvođenja zabilježena je kod visokopropusnog filtra na *A15* procesoru veličine kernela 3x3 i rezolucije 640x360 elemenata slike te sadrži srednju vrijednost od 149 ms. Nasuprot tome, najviše vremena za izvođenje potrebno je medijan filtru na *DSP* procesoru, veličine kernela 7x7 i rezolucije 1280x720 elemenata slike, a srednja vrijednost pri tome iznosi 15871 ms.

6. ZAKLJUČAK

Prostorno filtriranje slike iznimno je bitan dio predobrade slike u automotiv industriji jer omogućuje sve daljnje korake obrade slike pomoću *ADAS* sustava kao što je npr. detekcija objekata. Sklop digitalnog fotoaparata ili nepovoljni vremenski uvjeti mogu uzrokovati šum, odnosno slučajne varijance na digitalnoj slici. One onemogućuju ispravnu obradu slike te ih je potrebno filtrirati. Ovisno o primjeni, filtriranje se može provoditi različitim filtrima s različitim parametrima. U ovom su radu na *ADAS Alpha* ploču implementirana četiri različita filtra u *C* programskom jeziku upotrebom *Vision SDK* programskog okruženja. Kreirano algoritamsko rješenje optimizirano je koristeći razne softverske metode. Utvrđeno je da je softverskim tehnikama spajanja petlji (engl. *loop jamming* ili *fusion*), minimiziranja posla unutar istih ili njihovog „razmatanja“ (engl. *unrolling*) ušteda na vremenu znatna. Sklopovska optimizacija odrađena je efikasnijim rasporedom zadataka na dostupne procesore. Ukupno je kreirano dvanaest testnih slučajeva kombiniranjem različitih filtara i tri slučaja procesora: *DSP*, *A15* i kombinacija dvaju *DSP* procesora. Testiranje je izvedeno na navedenim kombinacijama s dodatnim promjenjivim parametrima: veličinom kernela i rezolucijom. Testirane su tri različite veličine kernela: 3x3, 5x5 i 7x7 te tri različite rezolucije: 640x360, 960x540 i 1280x720 elemenata slike. Utvrđeno je da brzina izvođenja uvelike ovisi o veličini kernela kod medijan i Gaussovog filtra, dok nešto manje kod *moving average* i visokopropusnog. Smanjenje rezolucije također bitno smanjuje potrebno vrijeme izvođenja. Najbrže filtriranje postignuto je visokopropusnim filtrom, dok je najviše vremena potrebno za filtriranje medijan filtrom. Nasuprot *DSP* procesoru, *A15* procesor zahtijeva najmanje vremenskih resursa.

LITERATURA

[1] P. Patidar, M. Gupta, S. Srivastava, i A. K. Nagawat, „Image De-noising by Various Filters for Different Noise“, *Int. J. Comput. Appl.*, sv. 9, izd. 4, str. 45–50, stu. 2010, doi: 10.5120/1370-1846.

[2] „OpenCV #004 Common Types of Noise“, *Master Data Science*, svi. 16, 2019. http://datahacker.rs/opencv-common-types-of-noise/ (pristupljeno srp. 29, 2021).

[3] D. Vajak, „Alpha Board and VisionSDK - Reference Guide“, str. 91.

[4] S. Sadangi, S. Baraha, D. K. Satpathy, i P. K. Biswal, „FPGA implementation of spatial filtering techniques for 2D images“, u *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, svi. 2017, str. 1213–1217. doi: 10.1109/RTEICT.2017.8256791.

[5] N. R. Shaikh, „Smoothing of a Noisy Image Using Different Low Pass Filters“, sv. 5, izd. 2, str. 4, 2017.

[6] V. Viswanathan i R. Hussein, „Applications of Image Processing and Real-Time embedded Systems in Autonomous Cars: A Short Review“, str. 15, 2017.

[7] M. Vranješ, R. Grbić, predavanja iz kolegija Digitalna obrada slike i videa za autonomna vozila, studij Automobilsko računarstvo i komunikacije, „postupci za predobradu slike“ https://loomen.carnet.hr/pluginfile.php/2459745/mod_resource/content/1/3.%20Postupci%20za %20predobradu%20digitalne%20slike.pdf

[8] Computer Vision Academy, *opencv c++ - Lecture 5 - part 2 : padding types (zero, replicate, and mirror padding)*. Pristupljeno: srp. 22, 2021. [Na internetu Video]. Dostupno na: https://www.youtube.com/watch?v=h0c0Tt9wsLg

[9] R. C. Gonzalez, R. E. Woods, i B. R. Masters, „Digital Image Processing, Third Edition“, *J. Biomed. Opt.*, sv. 14, izd. 2, str. 029901, 2009, doi: 10.1117/1.3115362.

[10] V. Novoselac i S. Rimac-Drlje, „Svojstva i primjena medijana i aritmetičke sredine“, str. 17.

[11] „Data Augmentation techniques in Python | LaptrinhX“. https://laptrinhx.com/data-augmentation-techniques-in-python-1574246488/ (pristupljeno srp. 22, 2021).

[12] „ResearchGate | Find and share research“, *ResearchGate*. https://www.researchgate.net/ (pristupljeno srp. 22, 2021).

[13] Dwayne. Phillips „ImageProcessingC.pdf“, 2000.

[14] „Problems for pedestrians, local officers speak on safety - Opera News“. https://www.dailyadvent.com/news/412915c698278992e8f6286c81640d10-Problems-for-pedestrians-local-officers-speak-on-safety (pristupljeno srp. 22, 2021).

[15] „Institut RT-RK Osijek d.o.o.“, *Osijek Software City*. https://softwarecity.hr/tvrtka/rt-rk/ (pristupljeno srp. 23, 2021).

SAŽETAK

U ovom radu implementirani su i uspoređeni različiti tipovi prostornih filtara. Najprije su predstavljeni osnovni tipovi šuma koji stvaraju potrebu za filtriranjem, a to su: Gaussov, šum „soli i papra“ te kvantni. Nakon toga su predloženi načini suzbijanja istih različitim tipovima filtara te je dan teorijski osvrt na četiri prostorna filtra: medijan, *moving average*, Gaussov i visokopropusni. Razvijeni algoritmi implementirani su na realnu razvojnu ADAS platformu s prilagođenim C kodom za *Alpha* ploču i kreiranim slučajevima upotrebe za svaki filtar za 3 različita rasporeda procesora: DSP, A15 i kombinacija dvaju DSP procesora. Dodatne kombinacije filtara i procesora su filtri s različitim veličinama kernela i različitim rezolucijama. Svaka od navedenih kombinacija se testirala na testnom skupu od deset slika. Rezultati su uspoređeni te je utvrđeno da se najbrže izvodi visokopropusni filtar, a najsporije medijan. Ovisno o primjeni, može se koristiti veća rezolucija, ali ona, kao i povećanje veličine kernela, uzrokuje smanjenje brzine izvođenja.

ABSTRACT

In this paper were implemented and compared different types of spatial filters. Firstly, the basic types of noise, which create the need for spatial filters, are described. Common types of noise are: Gaussian, „salt and pepper“ and quantization noise. After that are named and described different types of spatial filters used for reducing noises in digital image. These filters are: median, moving average, Gaussian and highpass filter. The developed algorithms are implemented on a real *ADAS* development platform with custom *C* code for *Alpha* board and created usecases for each filter for three different *CPU*-s: *DSP*, *A15* and a combination of two *DSP CPU*-s in parallel. Aditional combinations are three different resolution and three different kernel sizes in each filter. Each of this combinations were tested on a test set of ten images. The results were compared and it was found that high pass filter is the fastest, and median the slowest. Depending on the application, a higher resolution can be used, but it, as well as increasing the kernel size, causes a decrease in execution speed.

ŽIVOTOPIS

Kristina Opačak rođena je 27.3.1998. godine u Stuttgartu, nakon čega 2001. godine s obitelji seli u Slavonski Brod. Tamo završava Osnovnu školu Bogoslava Šuleka, a nakon toga i opću gimnaziju Matije Mesić. U listopadu 2016. godine upisuje preddiplomski studij elektrotehnike na FERIT-u u Osijeku, a 2019. godine nastavlja obrazovanje na diplomskom studiju Automobilsko računarstvo i komunikacije.

_____

Potpis autora

PRILOZI

P.3.1. – Mapa s datotekama algoritama napisanih u *C* programskom jeziku

P.5.1. – Mapa testnim slikama