Implementacija algoritma za detekciju rubova u slici na realnu ADAS platformu

Ćorić, Dario

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

Sveučilišni diplomski studij računarstva

IMPLEMENTACIJA ALGORITMA ZA DETEKCIJU RUBOVA U SLICI NA REALNU ADAS PLATFORMU

Diplomski rad

Dario Ćorić

Osijek, 2020.

SADRŽAJ

1. UVOD

Napredak automobilske industrije u zadnjem desetljeću usmjeren je prema razvoju autonomnih vozila. Istraživanje iz 2015. godine [1] pokazuje da je 94% prometnih nesreća uzrokovano ljudskom greškom te da je većina grešaka uzrokovano raznim ometanjima, poput umora vozača, korištenja mobilnog telefona tijekom vožnje i slično. Tek 2% nesreća uzrokovano je mehaničkim kvarovima vozila te još 2% uvjetima okoline. Zbog toga je jedan od glavnih ciljeva automobilske industrije smanjiti vozačevu ulogu u prometu i naposljetku potpuno automatizirati vožnju.

Kako bi se vožnja automatizirala fokus se stavlja na razvoj naprednih sustava za pomoć pri vožnji (engl. *Advanced Driving Assistance System* - ADAS). ADAS sustavi koriste različite senzore za percepciju okoline te obrađuju dobivene podatke pomoću dostupne sklopovske i programske podrške. Na temelju obrađenih podataka sustav poduzima određene radnje poput prilagodbe brzine, upozoravanja vozača o napuštanju vozne trake i drugih.

Udruga inženjera automobilske industrije (engl. *Society of Automotive Engineers* – SAE) definira šest razina autonomnosti vožnje (0-5) [2]. Na nultoj razini mogu postojati određeni sustavi za upozorenje vozača no ne postoje sustavi za upravljanje vožnjom. Na prvoj razini postoje sustavi koji pomažu vozaču pri izvršavanju određenih osnovnih operacija, dok se na drugoj razini istovremeno može izvršavati više sustava za pomoć vozaču. Na navedene tri razine vozač mora biti potpuno angažiran te u svakom trenutku mora biti spreman preuzeti potpunu kontrolu nad vozilom. Treća razina omogućava potpunu automatizaciju vožnje u određenim uvjetima no vozač i dalje mora biti spreman preuzeti kontrolu u slučaju da sustav to zahtijeva. Na četvrtoj razini omogućena je potpuno autonomna vožnja u ograničenom geografskom području, npr. autonomna taksi vozila. Konačno, na petoj razini, sustav preuzima potpunu kontrolu nad svim aspektima vožnje u svim uvjetima te prisutnost vozača uopće nije potrebna.

Pred razvojem takvih sustava nalaze se brojna ograničenja i izazovi. Prva prepreka je osiguranje fizičkog prostora za ugradnju računalnog sustava u automobil. Sljedeći izazov je implementacija programskog rješenja koje može obraditi podatke dobivene od senzora u stvarnom vremenu, budući da kašnjenje u takvim sustavima može rezultirati smrtnim posljedicama. Pored toga, rad programskog rješenja ovisi i o signalu dobivenom od senzora, koji može ovisiti od vremenskih uvjeta i od drugih faktora okoline.

Jedan od glavnih senzora ADAS sustava je kamera za koju su implementirani različiti algoritmi poput algoritma za detekciju vozne trake, prometnih znakova, vozila i drugih objekata. Svi ti algoritmi u predobradi koriste algoritme za detekciju rubova kako bi se smanjila količina informacije koju glavni algoritam mora obraditi. Zbog toga je od iznimne važnosti implementirati algoritam za detekciju rubova koji odgovara zahtjevima rada u stvarnom vremenu.

Ostatak rada strukturiran je na sljedeći način; U drugom poglavlju opisani su česti pristupi problemu detekcije rubova, problem implementacije takvih rješenja na ugradbene sustave te je dan pregled recentnih radova u području. U trećem poglavlju opisani su ADAS ALPHA platforma i VisionSDK razvojno okruženje, implementirani algoritmi i njihova validacija. U četvrtom poglavlju predstavljene su metode i rezultati testiranja implementiranih rješenja.

2. PROBLEM DETEKCIJE RUBOVA NA SLICI

Detekcija rubova je grana računalnog vida koja podrazumijeva pronalazak dijelova digitalne slike na kojima postoji nagla promjena intenziteta svjetline. Preciznije, odnosi se na pronalazak diskontinuiteta na slici. Jedan je od temeljnih alata u obradi slike i računalnom vidu, naročito u područjima detekcije i ekstrakcije [3] značajki na slici, budući da diskontinuitet na slici često odgovara konturama objekata ili granicama između različitih segmenata objekta.

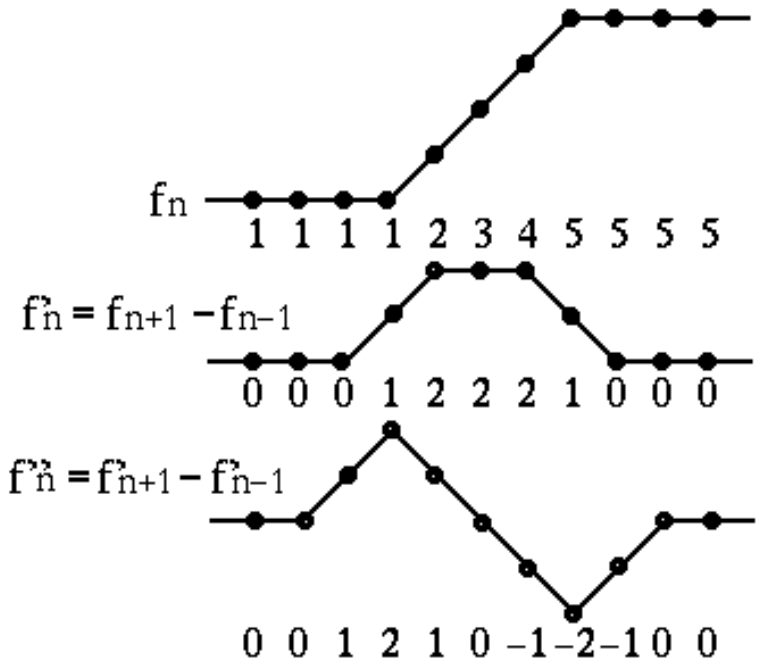
Kroz razvoj računarstva razvijene su brojne metode za detekciju rubova no većina ih se može svrstati u dvije kategorije – one temeljene na pretrazi slike za naglim promjenama svjetline (engl. *search-based*) i one temeljene na pronalasku nultočaka druge derivacije (engl. *zero crossing*). U metodama temeljenim na pretrazi koriste se aproksimacije prve derivacije intenziteta elemenata u okolini promatranog elementa slike [4] s ciljem pronalaska horizontalnog i vertikalnog gradijenta, tj. količine promjene intenziteta svjetline u susjedstvu promatranog elementa slike. Općenito, gradijent funkcije s dvije varijable je vektor čije su komponente dobivene prvom derivacijom u horizontalnom i vertikalnom smjeru, kao što je prikazano u izrazu (2.1.). U slučaju diskretnih vrijednosti, kakve su kod digitalnih slika, derivacija u elementu slike (x,y) može se približno odrediti kao razlika susjednih elemenata prema izrazima (2.2.) i (2.3.), gdje $I(x,y)$ predstavlja intenzitet promatranog elementa slike.

$$\nabla I = \left(\frac{\partial \mathrm{I}}{\partial x}, \frac{\partial \mathrm{I}}{\partial y}\right), \tag{2.1.}$$

$$\frac{\partial \mathrm{I(x,y)}}{\partial x} = \frac{I(x+1,y) - I(x-1,y)}{2}, \tag{2.2.}$$

$$\frac{\partial \mathrm{I(x,y)}}{\partial y} = \frac{I(x,y+1) - I(x,y-1)}{2}. \tag{2.3.}$$

S druge strane, metode temeljene na pronalasku nultočaka koriste aproksimaciju druge derivacije intenziteta slike, kako je prikazano na slici 2.1. [5] gdje f_n predstavlja vrijednost signala f u položaju n. Na slici su prikazane derivacije jednodimenzionalnog signala i vidljivo je da maksimum prve derivacije odgovara nultočki druge derivacije, odnosno prijelazu iz pozitivnog u negativni dio osi. Detektori iz ove skupine pronalaze rubove na takvim prijelazima, odnosno u dijelovima slike gdje su susjedni elementi suprotnog predznaka. Na taj način radi Laplaceov operator [6] koji je također opisan u trećem poglavlju.



Sl. 2.1. Prva i druga derivacija diskretnog signala

U radu iz 1986. [7] Canny je predložio novu metodu za detekciju rubova koja se može podijeliti u pet koraka: filtriranje pomoću Gaussovog filtra kako bi se uklonio šum sa slike, pronalazak gradijenata slike, primjena ne-maksimalnog suzbijanja. tj. suzbijanja rubova čija vrijednost nije najveća u usporedbi sa susjednim rubovima (engl. *non-maximum suppression*), kako ne bi bilo lažnih rubova, primjena dvostruke granice za određivanje potencijalnih rubova (engl. *double thresholding*) te praćenje ruba histerezom (engl. *edge tracking*). Posljednji korak podrazumijeva suzbijanje slabih rubova i onih rubova koji nisu pored jakih rubova.

Glavni problem implementacije detektora rubova na ugradbene računalne sustave, poput onih koji se nalaze u modernim automobilima, je osiguranje rada u stvarnom vremenu uz ograničenu radnu memoriju i procesor. Zbog toga su razvijene nove metode, često prilagodbe postojećih rješenja, kojima se nastoji omogućiti implementaciju detektora rubova u ugradbenim sustavima. Neki od radova u kojima su predložene takve metode opisane su u nastavku.

U radu [8], predstavljene su dvije metode pronalaska rubova. Prva metoda temelji se na pronalasku nizova u kojima se vrijednosti piksela mijenjaju uzlazno ili silazno. To se postiže usporedbom promatranog piksela s njegovim susjedima – ukoliko je njihova razlika veća od određene granice, niz će biti određen kao uzlazni ili silazni. Ukoliko se detektira promjena smjera promjene vrijednosti piksela (npr. iz silaznog u uzlazni) promatrani piksel biva označen

kao rubni. Druga predložena metoda je jednostavnija – promatra se samo razlika vrijednosti promatranog piksela i njegovog desnog i donjeg susjeda. Ukoliko je jedna od tih razlika veća od granice, piksel se smatra rubnim. Algoritmi su uspoređeni sa Sobelovim operatorom i Cannyjevim algoritmom. Kao mjere preciznosti detekcije izračunate su srednja kvadratna greška (engl. *mean square error,* MSE) i maksimalni omjer signala šuma (engl. *peak signal-to-noise ratio,* PSNR) prema izrazima (2.4.) i (2.5.).

$$MSE = \frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N}(I(x,y) - I'(x,y))^2\,, \tag{2.4.}$$

$$PSNR = 10\log(\frac{255^2}{MSE})\,. \tag{2.5.}$$

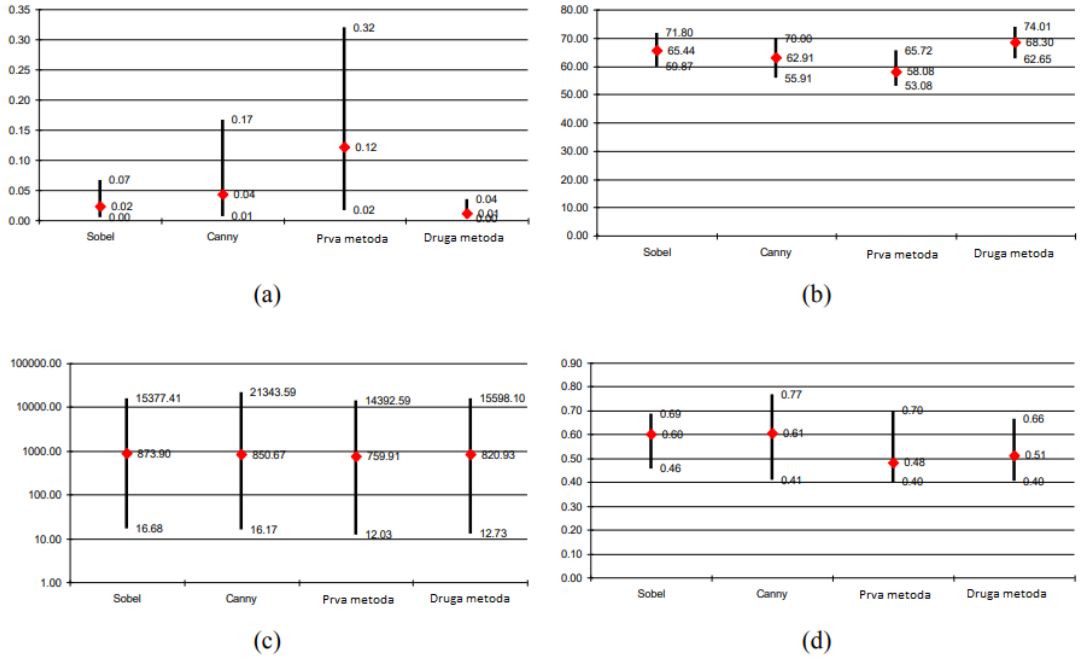
U izrazu (2.4.) M i N predstavljaju dimenzije slike, gdje $I'(x,y)$ predstavlja piksel iz slike označene temeljnom istinom (engl. *ground truth labeled image*) iz Berkeley skupa podataka za segmentaciju (engl. *Berkeley Segmentation Dataset,* BSDS) [9]. Također su kao mjerilo korišteni Baddeleyeva delta mjera [10] (engl. *Baddeley Delta Metric,* BDM) te F-mjera (engl. *F-measure*), koja je računata kao harmonijska sredina preciznosti (engl. *precision*) i osjetljivosti (engl. *recall*) kao što je prikazano u izrazima (2.6.), (2.7.), (2.8.). Preciznost je određena kao omjer točno pozitivnih elemenata (engl. *true postive,* t_p) i sume točno pozitivnih i lažno pozitivnih elemenata (engl. *false positive,* f_p) i predstavlja udio detektiranih rubova koji su zaista rubovi. Osjetljivost je određena kao omjer točno pozitivnih elemenata i sume točno pozitivnih i lažno negativnih elemenata (engl. *false negative,* f_n) i predstavlja postotak detektiranih stvarnih rubova.

$$precision = \frac{t_p}{t_p + f_p}\,, \tag{2.6.}$$

$$recall = \frac{t_p}{t_p + f_n}\,, \tag{2.7.}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}\,. \tag{2.8.}$$

Objema metodama, pri testiranju na Atmel i PIC uređajima izmjereno je manje vrijeme izvođenja u odnosu na Cannyjev i Sobelov algoritam. Slika 2.2. prikazuje rezultate mjerenja preciznosti novih metoda. Iako druga metoda pokazuje veću srednju kvadratnu grešku te obje metode imaju nižu F-vrijednost, za metode se može smatrati da dobro detektiraju rubove. Zbog manjeg vremena izvođenja pogodne su za ugradbene sustave u kojima je brzina izvođenja od većeg značaja nego preciznost.



Sl. 2.2. Rezultati mjerenja preciznosti algoritama: Sobel, Canny, prva predložena metoda, druga predložena metoda; a) MSE b) PSNR c) BDM d) F-vrijednost

U radu [11], predloženo je skraćivanje vremena izvođenja Cannyjevog algoritma pomoću podjele slike na blokove veličine $m \times m$ te klasifikacije piksela i blokova. Rješenje je uspješno implementirano na FPGA (engl. *Field Programmable Gate Array*) platformu. Slika se najprije dijeli na određeni broj nepreklapajućih blokova veličine n × n i uzima se konvolucijska maska veličine $L \times L$. Zatim se blokovima dodaje $(L + 1)/2$ piksela vrijednosti nula kako bi se izbjegli artifakti pri konvoluciji graničnih piksela blokova. Time blokovi dobivaju konačnu veličinu $m \times m$, pri čemu vrijedi da je $m = n + L + 1$. Prema metodi opisanoj u [12] svaki se piksel unutar pojedinog bloka zatim klasificira kao jednolični, teksturni ili rubni te se blokovi klasificiraju kao jednolični, jednolično-teksturni, teksturni, teksturno-rubni, srednji ili jaki rubni. Gornja granica Cannyjevog algoritma se određuje na temelju cijele slike te se prema njoj određuje postotak jakih rubova (P_1) u svim vrstama blokova. Koristeći skup podataka od 200 slika izračunata je srednja vrijednost P_1 za svaku vrstu bloka. Pokazano je da za jednolične i jednolično-teksturne blokove vrijedi $P_1 = 0$ i zbog toga nema potrebe tražiti rubove u tim blokovima, što smanjuje broj potrebnih operacija. Brzina izvođena je uspoređena između stolnog računala (Intel Core i7-945, 8192kB cache, 12GB RAM) i Xilinx SXT Virtex-5 FPGA platforme. Pokazano da je izvođenje brže na FPGA. Na slikama veličine 512×512 algoritam pronalazi rubove za 0,721ms što ga čini pogodnim za implementacije u stvarnom vremenu čak i za sadržaj u Full-HD rezoluciji.

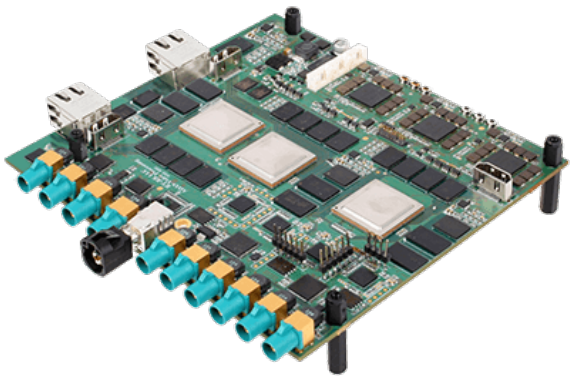
3. OPIS VLASTITE IMPLEMENTACIJE ALGORITAMA ZA DETEKCIJU RUBOVA U SLICI NA REALNU ADAS PLATFORMU

U ovom poglavlju opisani su razvojna platforma VisionSDK (engl. *Software Development Kit*) , korištena ADAS ploča, implementirani algoritmi te postupak njihove implementacije na ADAS ploču. U prvom potpoglavlju opisane su specifikacije ADAS razvojne ploče ALPHA, korištene tijekom diplomskog rada i razvojna platforma VisionSDK. U drugom potpoglavlju opisani su implementirani detektori. U trećem potpoglavlju opisan je postupak implementacije algoritama na ADAS ploču. Na kraju, četvrto potpoglavlje opisuje postupak provjere uspješnosti ugradnje rješenja u VisionSDK okruženje.

3.1. ADAS razvojna ploča ALPHA i razvojna platforma VisionSDK

3.1.1. ADAS razvojna ploča ALPHA

Razvojna ploča ALPHA [13] korištena tijekom diplomskog rada sastoji se od tri sustava na čipu (engl. *System on a Chip* – SoC) razvijenih od strane tvrtke *Texas Instruments* namijenjenih za različite primjene. Na samoj ploči postoji SoC za pogled oko automobila (engl. *Surround Camera*, SC), SoC za stereoskopski uskokutni prednji pogled, širokokutni prednji pogled i pogled u noćnim uvjetima (engl. *Front view camera near angle stereoscopic view, Front view camera wide angle, Night vision* – FFN) te SoC za spajanje (engl. *Fusion* - FUS). Ploča je prikazana na na slici 3.1. [13]



Sl. 3.1. ADAS ALPHA razvojna ploča

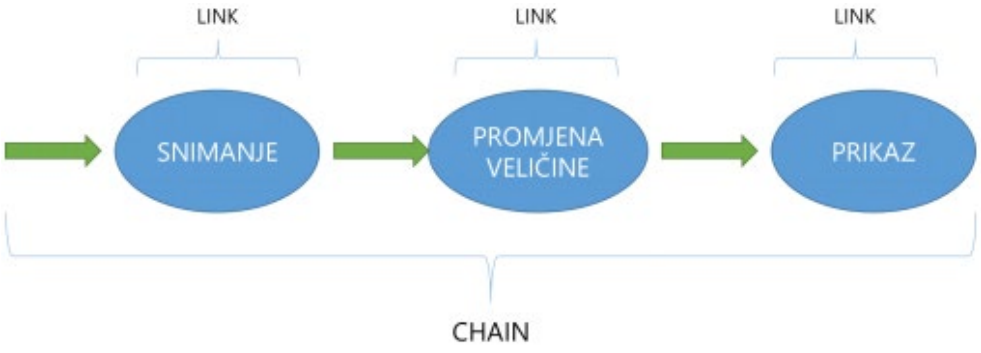
Svaki SoC sastoji se, između ostalog, od različitih jezgri koje rade na frekvencijama od 200MHz do 1150MHz po jezgri, 1,5GB DDR3 memorije, HDMI izlaza i ulaza za SD kartice.

Ploča posjeduje 10 konektora za kamere, od čega je 6 za SC SoC i 4 za FFN SoC. Svi SoC-ovi su u seriji vezani *ethernet* vezom, SC i FUS su povezani PCIe vezom te su FFN i FUS povezani video kanalom. Ulazi za SD kartice mogu se koristit za pokretanje SoC-a na koji su povezani. Način pokretanja bit će opisan kasnije u radu.

3.1.2. Razvojna platforma VisionSDK

VisionSDK je programski okvir koji omogućava programiranje ADAS SoC-ova. Korisnicima omogućava stvaranje različitih algoritama, slučaja upotrebe (engl. *usecase*), izvršavanje različitih algoritama na različitim procesorskim jedinicama te sadrži pokazne primjere. Zasnovan je na programskom okviru „karike i lanci“ (engl. *Links and Chains*), a njegovo programsko sučelje (engl. *Application Programming Interface* – API) zove se Link API. Svi direktoriji i datoteke vezane uz okruženje se nalaze u direktoriju *VISION_SDK_02_12_01_00* i sve putanje spomenute u nastavku rada imaju taj direktorij kao korijenski.

U *Links and Chains* okviru jedan lanac (engl. *chain*) sastoji se od više međusobno povezanih karika (engl. *link*). Grafički prikaz ove strukture prikazan je na slici 3.2. Svaki *link* predstavlja određeni algoritam (jedinicu obrade), hardversku vezu (npr. *Display link* odgovara prikazu) i slično.



Sl. 3.2. Grafički prikaz strukture *Links and Chains* okvira

Svaki *link* ima jednu ili više ulaznih i izlaznih konekcija (redova) i svaka konekcija u sebi može sadržavati jedan ili više logičkih kanala, kako je prikazano na slici 3.3.. Svaki kanal odgovara određenom međuspremniku pri čemu postoji više tipova međuspremnika kao što su međuspremnik video okvira, toka bita ili metapodataka. Međuspremnik video okvira može sadržavati podatke u YUV_422I_YUYV ili YUV_420SP_UV [14] formatima, koji su poduzorkovani formati iz YUV prostora boja.

Sl. 3.3. Link i njegove ulazne i izlazne konekcije

Kako ADAS ploča posjeduje *ethernet* ulaze, u okruženju postoje alati za mrežnu komunikaciju. Oni omogućuju slanje naredbi i video signala na ploču te primanje video signala s ploče. Dostupni alati su alat za upravljanje mrežom (engl. *Network Control Tool* - NCT), alat za primanje video signala (Network RX) te alat za slanje video signala (Network TX). NCT omogućava slanje naredbi i parametara definiranih od strane korisnika s računala na Alpha ploču. Network RX omogućava primanje MJPEG i RAW/YUV okvira s ploče na računalo a može se koristiti i za spremanje rezultata algoritama na računalo, dok *Network TX* omogućava slanje okvira s računala na ploču. Kako bi bilo moguće koristiti ih u *usecase*-u, potrebno NullSrc link kao izvorni (u slučaju Network TX) ili Null link kao odredišni (Network RX). Također je potrebno poznavati IP adresu ploče, koju je moguće statički postaviti.

3.2. Opis implementiranih algoritama

3.2.1. Sobelov i Prewitt operator

Pomoću Sobelovog operatora moguće je odrediti gradijent, odnosno usmjerenu promjenu intenziteta elemenata slike, u horizontalnom i vertikalnom smjeru te ukupne jakosti ruba prema izrazima (3.1.), (3.2.) i (3.3.). G_x i G_y predstavljaju gradijente u vodoravnom i vertikalnom smjeru, **A** predstavlja matricu koja sadrži elemente ulazne slike i znak * predstavlja operaciju konvolucije. Prema predznacima se može vidjeti kako pozitivan rezultat ukazuje na prelazak iz tamnijeg u svjetlije područje gledajući sliku prema desno po redovima i prema dolje po stupcima. Sukladno tome, negativan rezultat ukazuje na prelazak iz svjetlijeg u tamnije područje, odnosno prelazak iz područja manjeg u područje većeg intenziteta.

Prewitt operator funkcionira na isti način kao i Sobelov. Razliku čine konvolucijske matrice prikazane u izrazima (3.4.) i (3.5.).

$$G_{x(i,j)} = A_{(i,j)} * \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \tag{3.1.}$$

$$G_{y(i,j)} = A_{(i,j)} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}, \tag{3.2.}$$

$$G_{(i,j)} = \sqrt{{G_{x(i,j)}}^2 + {G_{y(i,j)}}^2}, \tag{3.3.}$$

$$G_{x(i,j)} = A_{(i,j)} * \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \tag{3.4.}$$

$$G_{y(i,j)} = A_{(i,j)} * \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}. \tag{3.5.}$$

3.2.2. Laplaceov operator

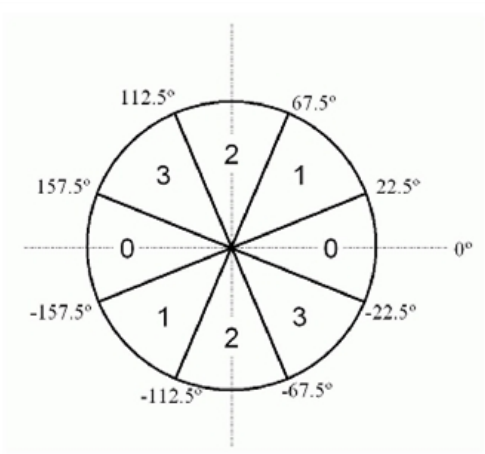
Laplaceov operator aproksimira drugu derivaciju intenziteta elemenata u okolini promanranog elementa slike i pronalazi rubove prema izrazu (3.6.). Vrijednosti dobivene primjenom ovog operatora predstavljaju brzinu promjene (engl. *rate of change*) intenziteta svjetline u susjedstvu promatranog elementa slike. U pravilu, dijelovi slike u kojima susjedni elementi imaju rezultantne vrijednosti suprotnog predznaka predstavljaju potencijalne rubne dijelove.

$$G_{(i,j)} = A_{(i,j)} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{3.6.}$$

3.2.3. Cannyjev algoritam

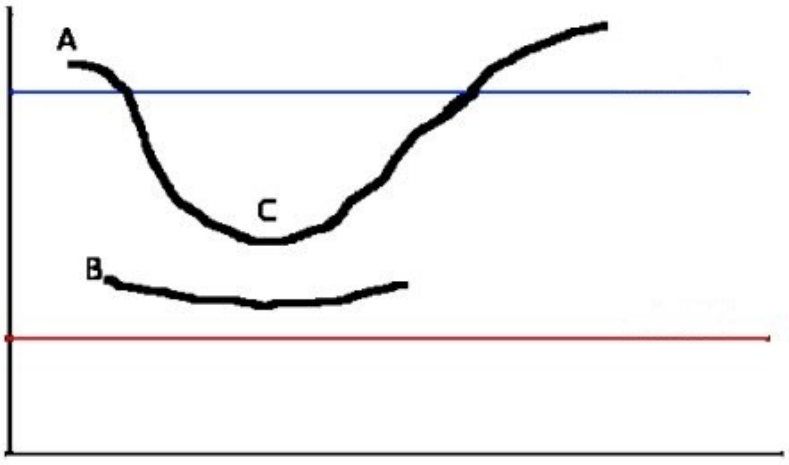
Cannyjev algoritam [7] može se opisati kao nadogradnja Sobel operatora čiji je cilj sačuvati jake rubove i odbaciti slabe. Time se dobiva binarna mapa rubova bez gradijenta u kojoj određeni element ili jest ili nije rub. To se postiže kroz 5 koraka – primjena Gaussovog filtra za uklanjanje šuma, pronalazak gradijenta prema izrazima (3.1.), (3.2.) i (3.3.), određivanje smjera gradijenta, ne-maksimalno suzbijanje, primjena dvojne granice i praćenje rubova. Nakon što je pronađen smjer gradijenta prema izrazu (3.7.), gdje θ predstavlja kut, potrebno ga je zaokružiti na jedan od 4 smjera u kojem piksel može imati susjeda. Način na koji se određuju smjerovi prikazan je na slici 3.4., gdje područja označena brojem 0 odgovaraju kutu od 0°, brojem 1 kutu od 45°, brojen 2 kutu od 90° te brojem 3 kutu od 135°.

$$\theta_{(i,j)} = atan(G_{y(i,j)}/G_{x(i,j)}). \tag{3.7.}$$



Sl. 3.4. Određivanje diskretnog smjera ruba

Pošto su određeni diskretni smjerovi ruba, moguće je primijeniti ne-maksimalno suzbijanje. Ono podrazumijeva usporedbu svakog piksela s njegovim susjedima i zadržavanje samo onih čija je jakost veća od jakosti njegovih susjeda. Ostali pikseli se brišu, tj. vrijednost im se postavlja na nulu. Ovim korakom postignuti su tanji rubovi koje je zatim moguće usporediti s gornjom i donjom granicom.. Svi rubovi čija je vrijednost iznad gornje granice se zadržavaju i smatraju se jakim rubovima, dok se svi oni čija je vrijednost ispod donje granice brišu. Rubovi koji se nalaze između dviju granica se smatraju slabim rubovima. Oni se zadržavaju ukoliko im je barem jedan od susjeda jaki rub ili su preko drugih slabih rubova vezani na jaki rub. Grafički prikaz zadnjeg koraka nalazi se na slici 3.5. [15]. Plava linija predstavlja gornju granicu a crvena donju. Rub A će biti zadržan jer mu je vrijednost iznad gornje granice. Rub B će biti obrisan jer nije povezan niti s jednim jakim rubom, a rub C će biti zadržan jer je povezan s rubom A.



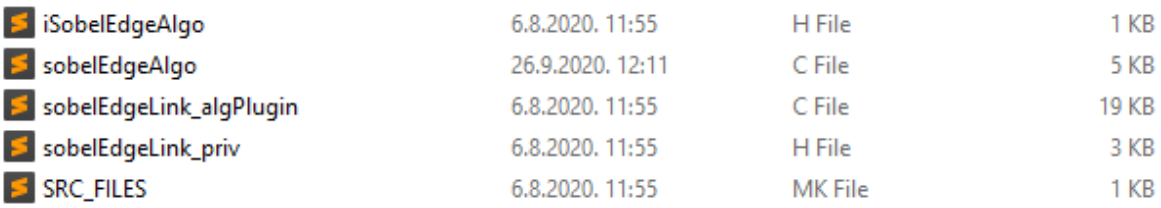
Sl. 3.5. Grafički prikaz praćenja ruba

3.3. Opis postupka implementacije

U prvom potpotpoglavlju opisani su koraci koje je potrebno napraviti kako bi algoritam bio dodan u VisionSDK okruženje. U drugom potpotpoglavlju opisana je i prikazana pseudokodom logika svakog implementiranog algoritma. Na kraju, u trećem potpotpoglavlju, opisan je postupak provjere uspješnosti implementacije te su dani primjeri izlaznih slika implementiranih algoritama.

3.3.1. Postupak dodavanja novog algoritma u VisionSDK

Kako bi se algoritam dodao u VisionSDK, najprije je potrebno kreirati mapu koja će sadržavati izvorne datoteke u direktoriju *\vision_sdk\examples\tda2xx\src\alg_plugins*. Kao primjer pokazano je dodavanje Sobelovog operatora. Na slici 3.6. prikazane su datoteke koje je potrebno kreirati u mapi. Iste datoteke je potrebno posebno kreirati i za Laplaceov i Prewitt operator te Cannyjev algoritam.



Sl. 3.6. Sadržaj mape *sobeledge*

Nakon kreiranja mape i popratnih datoteka potrebno je algoritam ugraditi u VisionSDK. Najprije je potrebno u mapi *\vision_sdk\include\link_api* dodati datoteku *algorithmLink_sobelEdge.h* koja definira potrebne parametre za povezivanje novododanog algoritamskog linka s VisionSDK API-jem. Osim toga, u datoteci *\vision_sdk\configs\tda2xx_evm_bios_all\uc_cfg.mk* potrebno je dodati liniju *ALG_sobeledge=yes* kako bi algoritam bio uzet u obzir za vrijeme izgradnje.

Naredbama *gmake –s –j depend* i *gmake –s –j sbl_sd* u upravljačkoj liniji (engl. *Command Prompt*) algoritam se dodaje u strukturu izgrađenog okruženja. Uspješnost dodavanja može se zatim provjeriti naredbom *gmake showconfig*. Naziv algoritma trebao bi biti vidljiv u ispisu kao na slici 3.7.

```
# Alg plugins included in build,
# ALG_autocalibration ALG_clr ALG_colortogray ALG_sobeledge ALG_laplaceedge ALG_canny ALG_dmaSwMs ALG_framecopy ALG_obje
ctdetection ALG_safe_framecopy ALG_surroundview
```

Sl. 3.7. Ispis ugrađenih algoritama

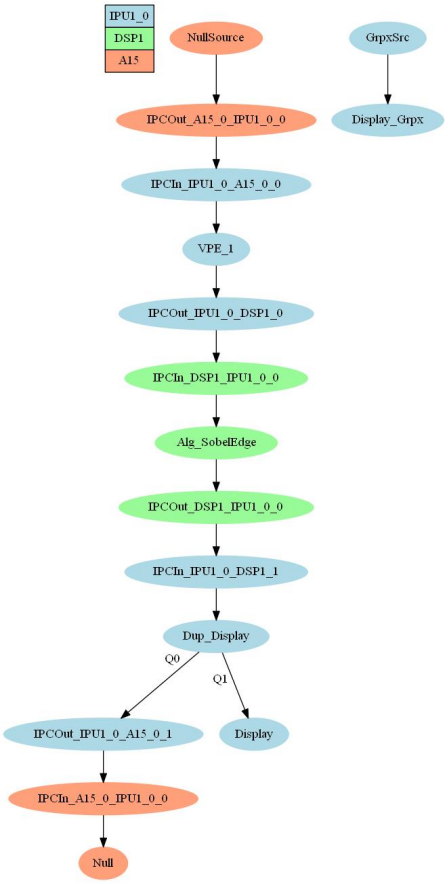
Kako bi bilo moguće testirati rad algoritma potrebno ga je dodati u *usecase*. Prvi korak stvaranja *usecase*-a je dodavanje nove mape u direktoriju *\vision_sdk\examples\tda2xx\src\usecases*. U njemu je potrebno dodati tekstualnu datoteku u kojoj je specificirano koji *linkovi* će biti korišteni. Na slici 3.8. prikazan je sadržaj datoteke u kojoj se algoritamski *link* za Sobelov operator dodaje u *usecase* koji će biti korišten tijekom testiranja. Također su prisutni NullSrc i Null *linkovi* koji su dodani za potrebe testiranja. *Dup* link udvostručuje okvir koji se prenosi te u ovom slučaju omogućava istovremeno prikazivanje obrađene slike i slanje iste slike s ploče na računalo. Nakon toga potrebno je generirati *usecase*. Za to je potrebno pozicionirati se u direktorij *\vision_sdk\tools\vision_sdk_usecase_gen\bin* u upravljačkoj liniji i iskoristiti naredbu *vsdk_win64.exe* koja kao argumente uzima putanju do direktorija te do tekstualne datoteke unutar istog direktorija. Time nastaju slika *usecase*-a, poput one na slici 3.9. i datoteke u kojima su definirani svi linkovi u *usecase*-u.

```
UseCase: chains_oneFileOneSobel

NullSource (A15) -> VPE_1 -> Alg_SobelEdge (DSP1) ->  Dup_Display
Dup_Display -> Null (A15)
Dup_Display -> Display

GrpxSrc -> Display_Grpx
```

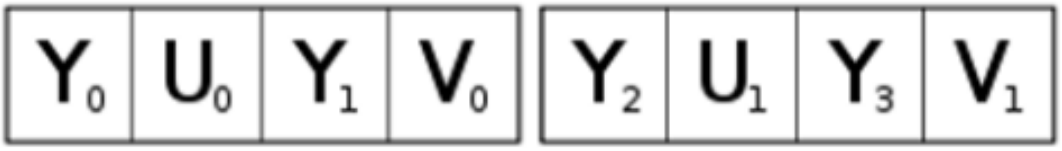
Sl. 3.8. Primjer *usecase*-a



Sl. 3.9. Slika generiranog *usecase*-a

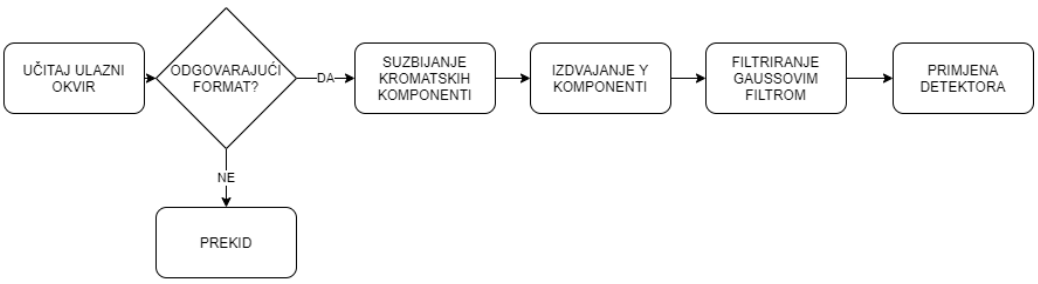
3.3.2. Implementacija algoritama na ADAS ploču

Na početku drugog poglavlja navedeno je kako se rub može definirati kao nagla promjena svjetline na slici. U YUV prostoru boja, s kojim je omogućen rad u VisionSDK, Y komponenta sadrži informacije o svjetlini te će se obrada izvršavati isključivo na toj komponenti. Druge dvije komponente, U i V sadrže informacije o boji. Kao takve nisu od značaja i potrebno ih je prigušiti. Od dva oblika poduzorkovanja koja VisionSDK podržava za implementaciju je odabran YUV_422I_YUYV. Slika 3.10. prikazuje zapis tog formata. Može se vidjeti kako su komponente isprepletene. tj. da nakon svake Y komponente slijedi U ili V komponenta.



Sl. 3.10. Zapis YUV_422I_YUYV formata

Svaki implementirani algoritam može se prikazati dijagramom na slici 3.11.. Prvi koraci u implementaciji svih algoritama su provjera formata boja te suzbijanje U i V komponenti. Prema [16], njihove vrijednosti je potrebno postaviti na 128 što rezultira time da nema boje, tj. da će slika biti prikazana u nijansama sive boje (engl. *grayscale*) odnosno od crne do bijele. Pri tome crna boja predstavlja nepostojanje ruba na tom dijelu slike a bijela predstavlja najjači mogući rub. Kako se obrada ne bi vršila direktno nad ulaznim podacima, paralelno sa suzbijanjem U i V komponenti vrijednosti Y komponenti su spremljene u pomoćno polje.



Sl. 3.11. Općeniti dijagram implementacije algoritama

Prvobitno je memorija za polje zauzimana uobičajenim funkcijama u C jeziku (malloc, calloc), no to je dovelo do pojave ispisa o nedostatku memorije tijekom pokretanja algoritma na ploči. Taj problem je riješen koristeći funkcije ugrađene u VisionSDK - Utils_memAlloc i Utils_memFree. Funkcija Utils_memAlloc pruža mogućnost zauzimanja memorije iz jedne od

postojećih hrpi (engl. *heap*), pri čemu kao parametre prima naziv polja za koje korisnik želi zauzeti memoriju, količinu memorije u byte-ima te veličinu jednog elementa u byte-ima. Nakon što su Y komponente spremljene, moguće je prijeći na sljedeći korak koji je također zajednički svim algoritmima, a to je primjena Gaussovog filtra. [17]

U prvobitnoj implementaciji Sobelovog operatora primijenjene su formule (3.1.), (3.2.) i (3.3.) bez dodatnih operacija i rješenje je testirano. U usporedbi s rezultatom dobivenim iz Sobelovog operatora koji postoji u biblioteci OpenCV [18] primijećeno je kako postoje znatne razlike u jakosti rubova. Pretpostavljeno je kako do toga dolazi zbog prelijevanja vrijednosti van raspona (engl. *overflow*). Kako bi se otklonio problem uzeta su u obzir dva rješenja – skaliranje izračunatih jakosti rubova u rasponu 0-255 ili postavljanje vrijednosti onih rubova koji se prelijevaju na 255. Obje metode su isprobane te je zbog jednostavnosti izvedbe kao konačno rješenje primijenjena druga metoda. Postupak je isti za Prewitt operator, osim što su korištene drukčije konvolucijske matrice. Pseudokod implementacije oba operatora nalazi se na slici 3.12., gdje je $Gx(i,j)$ horizontalni gradijent, $Gy(i,j)$ vertikalni gradijent te je $G(i,j)$ ukupna jakost ruba u promatranom elementu slike. Programski kodovi operatora mogu se pronaći na priloženom CD-u u mapama Prilog P.3.1. i Prilog P.3.2.

Linija ***Pseudokod***

```
1:      Za svaki element slike:
2:          Izračunaj Gx(i,j)
3:          Izračunaj Gy(i,j)
4:          Izračunaj G(i,j)
5:          Ako G(i,j) > 255
6:              G(i,j) = 255
7:          Spremi G u ulazni spremnik
8:      Kraj
```

Sl. 3.12.Pseudokod Sobel i Prewitt operatora

Implementacija Laplaceovog operatora napravljena je na sličan način. Najprije je implementirana konvolucija prema izrazu (3.6.) te su promotreni rezultati. Kao i kod Sobelovog operatora, primijećena su odstupanja od OpenCV rješenja. Usporedbom vrijednosti elemenata izlaznih slika vlastitog i postojećeg operatora zaključeno je kako je potrebno pronaći elemente čija je vrijednost manja od nule i postaviti ih na nulu kako bi svi elementi bili u valjanom rasponu. Pseudokod implementacije nalazi se na slici 3.13., gdje $G(i,j)$ predstavlja jakost ruba promatranog elementa. Programski kod operatora može se pronaći na priloženom DVD-u u mapi Prilog P.3.3.

Linija Pseudokod

```
1:      Za svaki element slike:
4:          Izračunaj G(i,j)
5:          Ako G(i,j) < 0
6:              G(i,j) = 0
7:          Spremi G(i,j) u ulazni spremnik
8:      Kraj
```

Sl. 3.13.Pseudokod Laplaceovog opeatora

Naposljetku je bilo potrebno implementirati i Cannyjev algoritam. Kako je početni korak nakon filtriranja pronalazak gradijenta, za taj korak iskorišten je prethodno implementirani Sobelov operator. Uz gradijent je u izračun dodan i pronalazak smjerova gradijenta i određivanje diskretnih vrijednosti kutova prema slici 3.4.. Nakon toga je bilo potrebno pronaći lokalne maksimume. U ovom koraku svaki element slike uspoređen je sa dva susjedna elementa u ovisnosti od smjera gradijenta u promatranom elementu. Na primjer, u slučaju kuta od 0° odnosno horizontalnog smjera gradijenta usporedba se obavila s elementima u istom retku i susjednim stupcima (*i, j-1*) i (*i, j+1*), pri čemu su (*i,j*) koordinate promatranog elementa. Nakon toga su vrijednosti elemenata uspoređeni s donjom i gornjom granicom na način da su najprije elementi čija je vrijednost iznad gornje granice (jaki rubovi) zadržani i postavljeni na 255 te su elementi čija je vrijednost manja od donje granice suzbijeni i vrijednost im je postavljena na 0. Pri obradi elemenata čija se vrijednost nalazi između dviju granica (slabi rubovi) prvobitno su uzeti u obzir samo elementi koji su direktni susjedi jakim rubovima a ostali su bili suzbijeni. Takvo rješenje je uspoređeno sa izlazom Cannyjevog detektora koji postoji u biblioteci OpenCV te je primijećena znatna razlika u broju zadržanih rubova. Konkretno, vlastiti algoritam je zadržavao manje rubova. Pretpostavljeno je kako su pronađeni rubovi dobri i da je potrebno povezati jake rubove preko svih slabih rubova smještenih između njih. Problem je riješen tako što je u slučaju pronađenog jakog ruba implementirana provjera vrijednoti susjednih elemenata.. U slučaju pronalaska susjednog elementa čija je vrijednosti između donje i gornje granice, vrijednost tog ruba se postavlja na 255. Pseudokod implementacije prikazan je na slici 3.14., gdje funkcija *pronadjiSobelGradijente()*odgovara pseudokodu Sobel opeatora na slici 3.12., *G*(*i,j*) predstavlja promatrani element slike, *donjaGranica* i *gornjaGranica* predstavljaju granice Cannyjevog algoritma s kojima se uspoređuju promatrani elementi te *susjedniElement* predstavlja sve susjede promatranog elementa. Programski kod algoritma može se pronaći na priloženom CD-u u mapi Prilog P.3.4.

Linija Pseudokod

```
1:    pronadjiSobelGradijente()
2:    Za svaki element slike:
3:        Ako kut == 0
4:            Ako G(i,j) <= G(i+1,j) ili G(i,j) <= G(i-1,j)
5:                G(i,j)=0
6:        Ako kut == 45
7:            Ako G(i,j) <= G(i+1,j+1) ili G(i,j) <= G(i-1,j-1)
8:                G(i,j)=0
9:        Ako kut == 90
10:           Ako G(i,j) <= G(i,j+1) ili G(i,j) <= G(i,j-1)
11:               G(i,j)=0
12:       Ako kut == 135
13:           Ako G(i,j) <= G(i+1,j-1) ili G(i,j) <= G(i-1,j+1)
14:   Za svaki element slike:
15:       Ako G(i,j) >= gornjaGranica
16:           G(i,j) = 255
17:           ako susjed >= donjaGranica i susjed < gornjaGranica
18                susjed = 255
19:       Ako G(i,j) >= donjaGranica
20:           G(i,j) = 0
21:   Za svaki element slike:
22:       Ako G(i,j) >= donjaGranica i G(i,j) < gornjaGranica
23:           G(i,j) = 0
24:   Kraj
```

Sl. 3.14.Pseudokod implementacije Cannyjevog algoritma

3.4. Provjera uspješnosti implementacije

Nakon što su kreirani algoritmi i *usecase*-ovi u koje su oni uključeni, potrebno je dovršiti izgradnju programskog rješenja i pokrenuti ga na ploči.

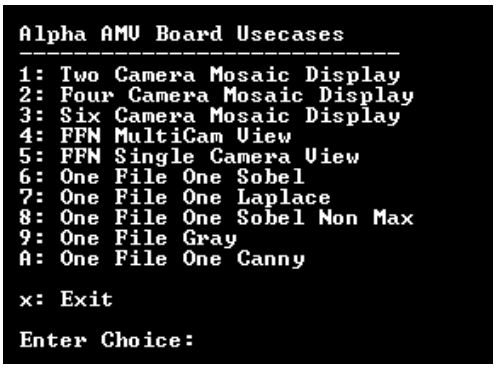
Najprije je potrebno omogućiti postavljanje statične IP adrese ploče te postaviti samu adresu u datoteci *vision_sdk\src\main_app\tda2xx\cfg\NDK_config.cfg*. Da bi promjene u mrežnim postavkama bile primijenjene, potrebno je izgraditi mrežne postavke naredbom u upravljačkoj liniji prikazanoj na slici 3.15.. Uz to je potrebno postaviti statičnu IP adresu računala i povezati računalo i ploču *ethernet* kablom.

```
C:\VISION_SDK_02_12_01_00\ti_components\networking\nsp_gmacsw_4_15_00_00>xdc --xdcpath="C:/VISION_SDK_02_12_01_00/ti_com
ponents/os_tools/bios_6_46_00_23/packages" -P packages/ti/nsp/drv/
```

Sl. 3.15.Naredba za izgradnju mrežnih postavki

Nakon toga je potrebno ponoviti postupak izgradnje opisan u prethodnom poglavlju. Izgradnjom nastaju datoteke MLO i AppImage koje je potrebno prebaciti na SD karticu, a karticu ubaciti u utor povezan s odgovarajućim SoC-om.

Da bi se slike poslale s računala na ploču, najprije je potrebno pokrenuti odgovarajući *usecase* iz izbornika prikazanog na slici 3.16. Zatim je potrebno pozicionirati se u direktorij *\vision_sdk\tools\network_tools\bin* u upravljačkoj liniji te unijeti naredbu *network_tx*, kao naslici 3.17., koja kao argument uzima naziv slike u YUV prostoru boja i šalje ju na ploču. U drugom prozoru upravljačke linije potrebno je omogućiti prihvaćanje obrađene slike i spremanje iste na računalo naredbom *network_rx*, kao na slici 3.18. Spremljenu sliku moguće je pregledati koristeći yuvplayer, program otvorenog koda koji omogućuje prikazivanje slika i video signala spremljenih u YUV prostoru. Na slici 3.20. prikazan je primjer izlaza implementiranih algoritama.



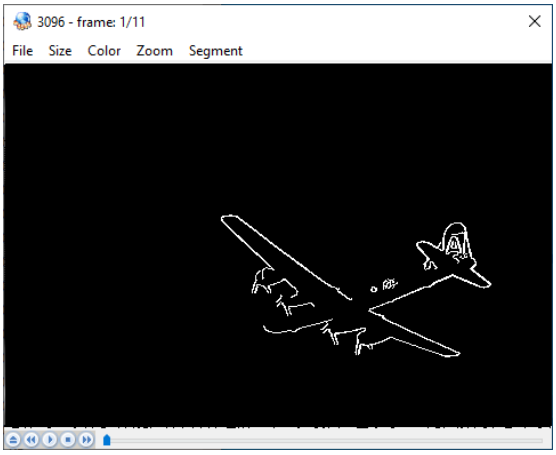
Sl. 3.16. Izbornik

```
c:\VISION_SDK_02_12_01_00\vision_sdk\tools\network_tools\bin>network_tx --host_ip 192.168.10.1
 --target_ip 192.168.10.2 --no_loop --files 3096.yuv
```

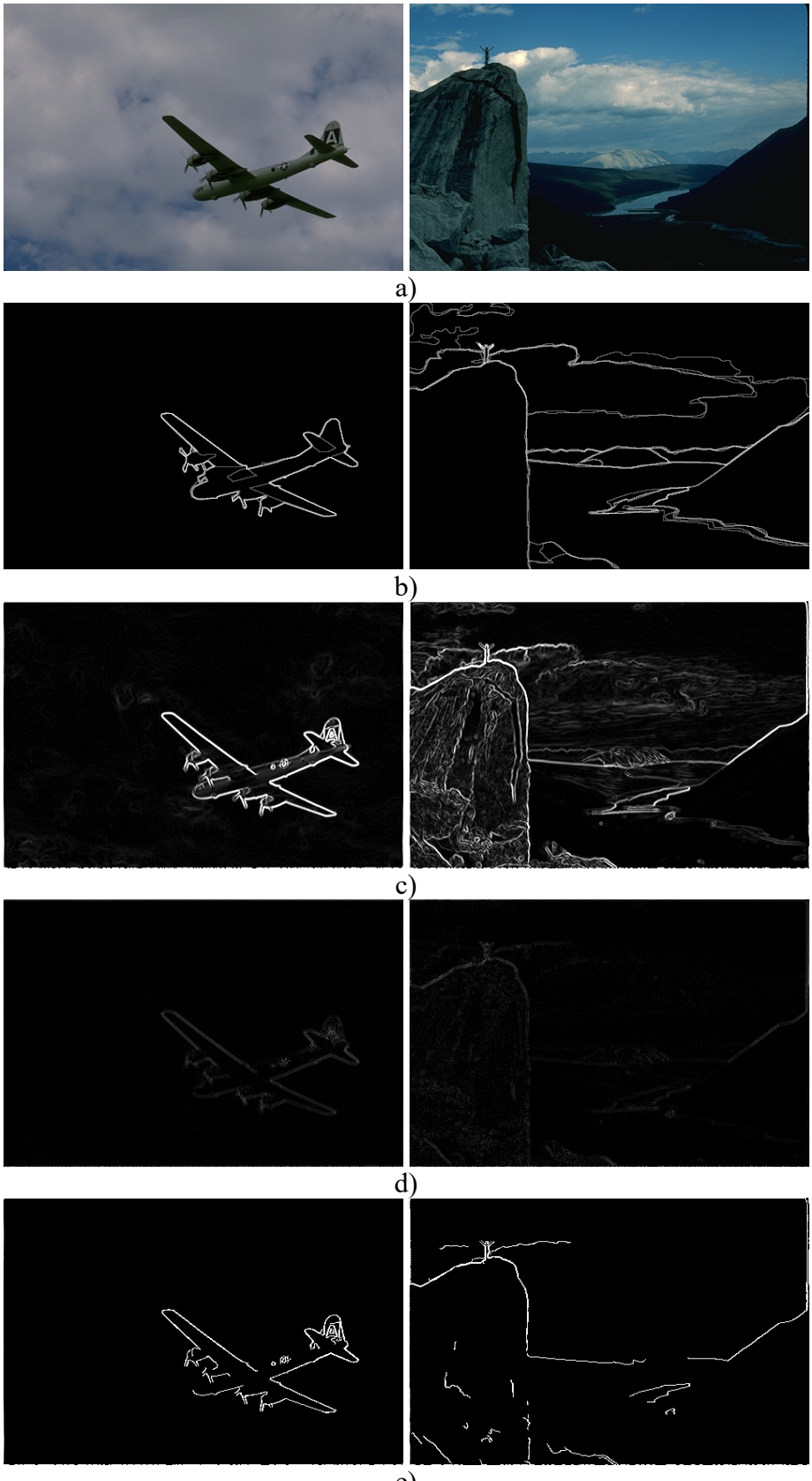
Sl. 3.17. Naredba za slanje slike putem mreže

```
c:\VISION_SDK_02_12_01_00\vision_sdk\tools\network_tools\bin>network_rx --host_ip 192.168.10.1
 --target_ip 192.168.10.2 --port 7000 --files 3096.bin
```

Sl. 3.18. Naredba za primanje izlazne slike putem mreže



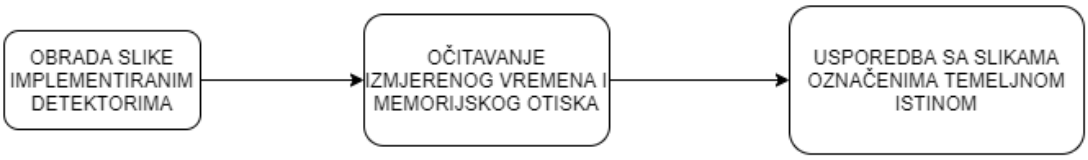
Sl. 3.19. Izlazna slika prikazana u *yuvplayer-u*



Sl. 3.20. a) izvorna slika b) *ground-truth* slika c) Sobel d) Laplace e) Canny

4. TESTIRANJE RADA IMPLEMENTIRANIH RJEŠENJA

U ovom poglavlju opisani su postupak i rezultati testiranja algoritama na ADAS ploči i na stolnom računalu. Korišteni su Berkeley [9] i Kitti [19] skupovi podataka. Za obje platforme i oba skupa podataka mjereno je vrijeme izvođenja po okviru i memorijski otisak. Izlazne slike algoritama primijenjenih na slike iz Berkeley skupa podataka su uspoređene s odgovarajućim *ground-truth* slikam tako što je pronađen broj elemenata slike koji se razlikuju te je izračunata srednja suma apsolutnih razlika. Kako slike iz Kitti skupa podataka nemaju odgovarajuće *ground-truth* slike, za usporedbu su korištene izlazne slike postojećih algoritama u biblioteci OpenCV. Osim na ADAS ploči, testiran je i rad algoritama implementiranih u OpenCV-u i to na stolnom računalu čije su specifikacije dane u drugom potpoglavlju. Postupak testiranja prikazan je dijagramom na slici 4.1.



Sl. 4.1. Dijagram postupka testiranja

4.1. Testiranje na ADAS ALPHA ploči

U ovom potpoglavlju opisano je testiranje detektora implementiranih na ADAS ploču. Za testiranje rada algoritama korišten je Berkeley skup podataka [9] koji sadrži 100 slika raznolikog sadržaja i odgovarajućih *ground-truth* slika. Izdvojeno je 77 slika rezolucije i orijentacije (481×321), koje su zbog nemogućnosti slanja signala neparnih dimenzija putem NullSrc linka skalirane na rezoluciju 480×320 te je format boja pretvoren u NV12 (ekvivalent formata YUV_420SP_UV kojeg NullSrc link zahtijeva) koristeći FFmpeg. Također je korišten Kitti skup podataka koji sadrži izdvojene kadrove četiri snimka vožnje. Slike su rezolucije 640x480, ali je iz svake izrezana cjelina veličine 480x180 na kojoj se nalazi sadržaj vezan uz promet, poput voznih traka, pješačkih prijelaza, prometnih znakova i drugih vozila.. Primjer izvorne slike i izrezane cjeline nalazi se na slici 4.2. Format boja izdvojenih slika također je pretvoren u NV12 zbog rada s NullSrc linkom. Korištene slike mogu se pronaći na priloženom CD-u u mapi Prilog P.4.1. i Prilog P.4.2.

Sl. 4.2. Primjer izdvajanja relevantnog sadržaja slike

Ciljevi testiranja bili su usporedba *ground-truth* slika i izlaznih slika postojećih detektora s izlaznim slikama implementiranih detektora te provjera brzine izvođenja detektora i njihovog memorijskog otiska. Za usporedbu slika izračunati su srednja suma apsolutnih razlika i ukupan broj piksela koji se razlikuju dok je za mjerenje brzine izvođenja korištena ugrađena funkcija u VisionSDK. Srednja suma apsolutnih razlika određena je izrazom (4.1.), gdje n predstavlja ukupan broj elemenata koji su različiti na uspoređenim slikama, $A(i)$ predstavlja element iz izlazne slike detektora te $A'(i)$ predstavlja element *ground-truth* slike, odnosno izlazne slike postojećeg detektora.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|A(i) - A'(i)|\,. \tag{4.1.}$$

4.1.1. Rezultati testiranja na Berkeley skupu podataka

U tablicama 4.1. i 4.2. nalazi se pregled srednje sume apsolutnih razlika i ukupnog broja različitih elemenata slika za svih 77 slika iz Berkeley skupa podataka.

Tablica 4.1. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (ADAS) (1/2)

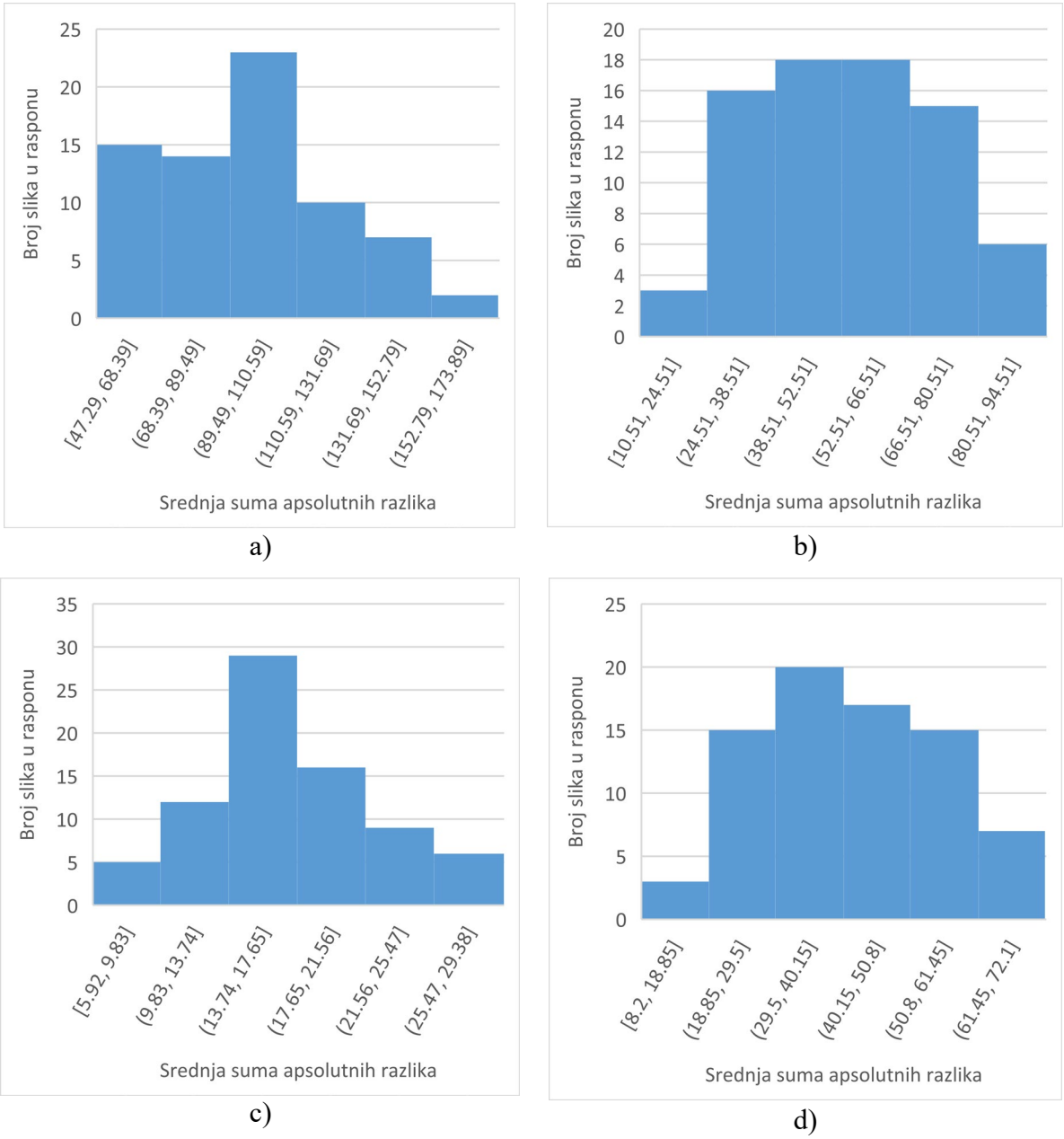
	Sobel		Prewitt		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
3096	10.51	139390	8.2	139427	64.31	8963	5.92	109832
8023	41.63	147363	31.53	148111	97.15	18036	9.24	132440
12084	61.09	142805	46.8	143634	108.3	44368	18.89	138356
14037	26.73	124008	21.54	124417	56.03	25098	13.4	110645
16077	54.9	144557	42.4	145302	105	35333	17.12	137309
19021	57.07	140504	44.41	140942	99.32	41268	18.56	133152
21077	49.24	142634	38.55	143337	108	32552	14.96	134833
24077	66.58	139403	54.78	139414	101.2	60155	21.99	129901
37073	28.04	145970	21.97	146907	70.15	36959	14.34	133160
38082	59.2	142767	43.89	143462	77.43	36795	16.11	140476
38092	56.66	139860	43.21	140115	93.96	39786	20.3	131166
41033	36.76	145219	27.95	146389	79.9	21473	12.2	135895
41069	79.87	142718	59.26	143173	149.9	34918	21.63	135608
42049	26.72	130792	22.02	130767	82.65	28421	14.02	110810
43074	33.24	145227	25.36	145852	86.11	20374	11.06	127719
45096	20.46	114230	17.05	114563	60.3	24073	13.11	99344
58060	90.14	130405	72.08	130215	141.05	53825	29.34	121922
62096	52.99	140241	41.19	140980	130.19	30256	16.01	126685
65033	73.03	140849	55.73	141241	105.98	47457	21.58	137990
66053	51.09	144647	38.72	145297	77.83	39891	17.36	136467
69020	53.14	144205	39.72	145128	69.43	31625	16.77	139978
69040	77.68	142408	59.43	142605	125.9	48728	19.29	137588
76053	42.12	146217	32.3	146879	58.08	36851	14.21	138560
85048	68.21	145368	52.72	145817	92.91	57654	22.62	139376
86016	76.66	144230	56.45	144997	114.15	48871	27.06	137913
86068	64.62	145276	36.46	143612	77.24	24476	14.73	137956
87046	70.03	141473	49.58	145784	122.88	34183	17.02	134964
97033	40.15	146733	54.5	141828	131.78	43134	18.99	133517
103070	68.33	129491	30.72	147269	65.66	31325	13.27	132708
105025	25.37	145009	51.67	129892	121.25	34867	19.41	122219
106024	68.19	142441	19.68	145254	57.84	17220	9.51	121219
108005	77.62	143274	53.36	142695	125.78	39113	16.94	137110
108070	49.41	135732	60.19	143587	135.77	42558	22.15	136853
108082	43.16	144846	46.82	130823	115.59	29917	16.69	122778
109053	67.14	141917	33.87	145347	75.63	41615	17.08	133450
119082	49.62	145063	53.51	142077	107.92	53122	21.55	136828
123074	30.61	135608	37.58	145782	92.46	23442	13.14	134771
126007	65.39	142838	29.77	132555	92.38	32045	15.88	120576
130026	84.38	143344	49.83	143540	139.68	33300	17.19	135007
134035	23.07	144105	67.9	143158	110.25	70284	25.54	137985
143090	49.74	142683	18.04	144266	66.06	22737	10.96	122597
145086	41.67	144958	39.5	142707	96.25	38575	18.75	131470

Tablica 4.2. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (ADAS) (2/2)

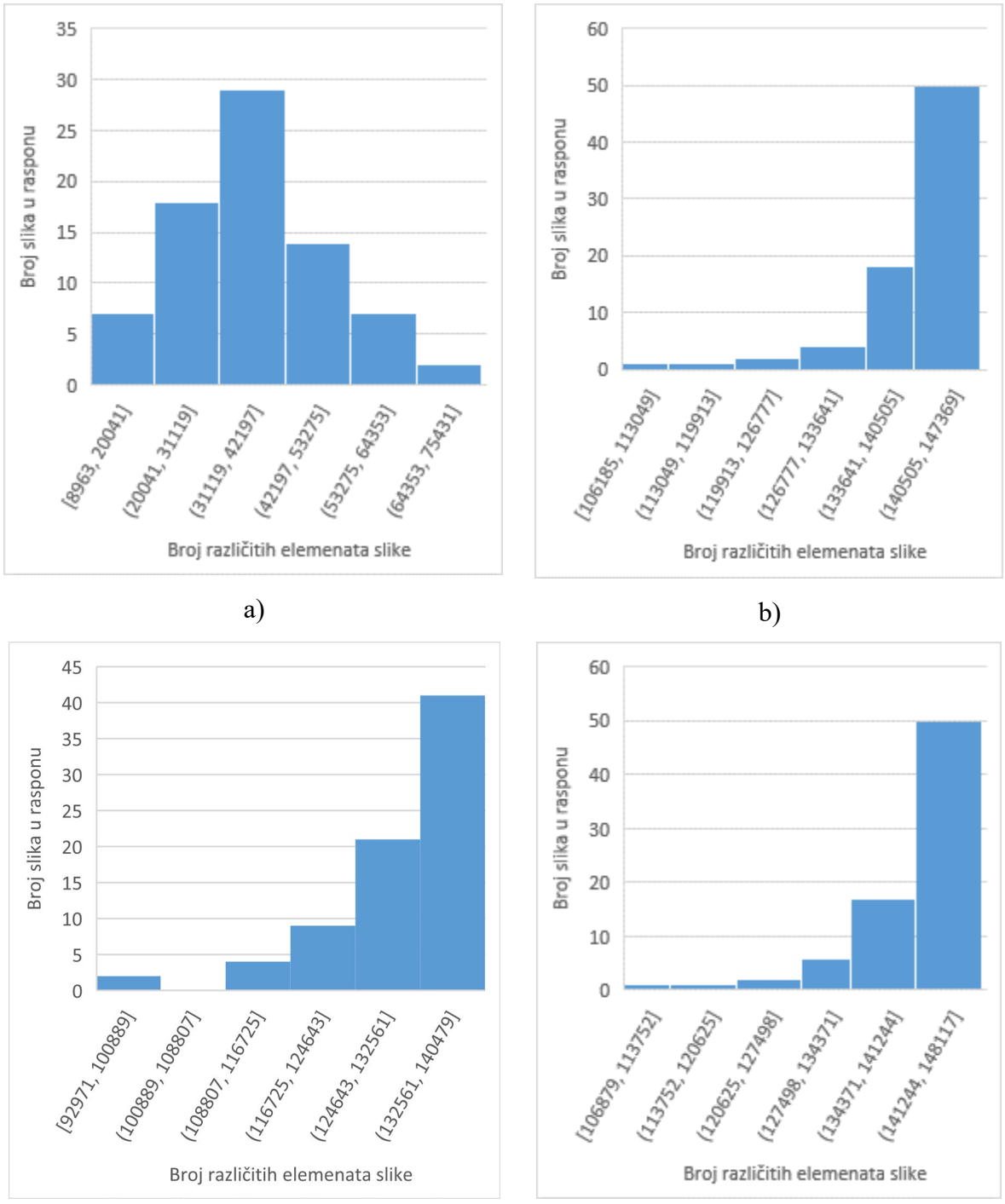
	Sobel		Prewitt		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
147091	80.21	141989	33.09	145314	96.17	36123	15.83	129795
148089	60.93	140320	63.72	142004	101.36	75427	26.96	139723
156065	57.38	140290	45.44	141005	108.04	38064	18.24	134902
157055	71.47	135255	44.9	140631	87.24	55024	20.45	133724
159008	49.68	135215	55.74	135622	106.46	48332	19.09	129308
160068	42.71	146127	39.54	135122	117.13	36433	16.97	117309
163085	25.49	106185	33.45	146867	79.94	39711	17.33	133764
167062	43.43	146467	19.62	106879	67.4	15616	9.14	92971
170057	89.35	141225	34.32	146943	60.39	54881	19.04	137413
175043	56.39	140098	67.55	141594	173.88	36828	18.41	136772
182053	37.02	143539	44.82	139903	112.12	50995	23.22	126469
196073	39.63	145818	26.74	145065	86.62	10693	10.95	140007
197017	55.65	137445	43.99	137725	86.02	23612	14.39	132207
216081	40.9	143968	32.51	144356	92.91	50786	24.01	129614
219090	53.89	145698	42.92	145875	107.88	40835	14.69	131414
220075	63.79	142095	48.24	142436	100.36	28115	15.44	131820
223061	81.19	141464	63.66	141868	161.43	46288	16.41	130741
229036	73.37	143723	55.23	144230	139.65	31398	22.72	136496
236037	28.22	131722	30.24	146446	142.77	26577	17.45	137259
241004	63.87	136138	22.4	132490	67.89	41572	14.62	120517
241048	65.09	144676	48.01	136829	98.9	57558	19.55	130102
253027	26.22	138115	54.82	144511	110.1	27031	27.33	135420
253055	26.56	146113	20.89	138634	61.33	14755	13.8	123040
260058	94.47	143938	19.99	146863	47.29	62638	8.14	131588
291000	43.74	139641	71.13	144023	127.78	37485	29.16	137449
295087	25.38	147132	33.71	139967	85.43	25424	18.09	127868
296007	30.88	139388	19.85	147806	51.16	22704	11.99	132692
296059	25.68	142601	23.55	140317	65.46	25429	12.94	128756
299086	32.77	141787	20.09	143450	65.45	17620	12.61	126517
300091	85.22	142509	24.61	142873	100.13	32352	10.12	129489
304034	51.69	134295	64.35	142830	168.98	42208	17.54	136120
306005	79.37	122999	39.83	134717	89.97	44343	17.32	125116
351093	43.49	145371	61.37	123036	144.76	35153	25.41	114554
361010	56.45	142635	34.55	146115	89.11	46000	15.79	137113
385039	38.49	141654	44.73	143148	103.63	29917	20.76	133197

Na slici 4.3. grafički je prikazana raspodjela srednjih suma apsolutnih razlika između *ground-truth* slika i izlaznih slika pojedinih detektora, dok je na slici 4.4. grafičkii prikazana raspodjela ukupnog broja različitih piksela po slikama. Vidljivo je kako rezultati Cannyjevog algoritma imaju najveću srednju sumu apsolutnih razlika u odnosu na *ground-truth*, no i kako je ukupan broj različitih piksela manji nego za Sobelov, Prewitt i Laplaceov operator. Uzrok tome

je binarna priroda izlazne slike Cannyjevog algoritma pikseli koji nisu rubovi imaju vrijednost 0 dok rubni pikseli imaju vrijednost 255 te zbog toga dolazi do značajne sume čak i pri manjem broju razlika. Pored toga, korištene *ground-truth* slike nisu binarne tako da čak i u slučaju podudaranja pronađenog ruba i ruba na *ground-truth* slici postoji mogućnost da su pikseli različite vrijednosti. S druge strane, rezultati Sobelovog, Prewitt i Laplaceovog operatora imaju manju srednju sumu apsolutnih razlika, ali uz to imaju i velik broj razlika. Na izlaznim slikama sva tri detektora nalazi se značajan broj slabijih rubova i zbog toga je broj razlika velik ali su srednje sume apsolutnih razlike manje u odnosu na Cannyjev algoritam.



Sl. 4.3. Srednja suma apsolutnih razlika u odnosu na *ground-truth* slike za a) Canny b) Sobel c) Laplace d) Prewitt



Sl. 4.4. Broj različitih piksela u odnosu na *ground-truth* slike za a) Canny b) Sobel c) Laplace d) Prewitt

Nadalje, u tablicama (4.3. i 4.4.) prikazana su vremena izvođenja i memorijski otisak algoritama. U stupcima označenima s *t* nalaze se vremena izvođenja u milisekundama dok se u stupcima označenim s *m* nalaze memorijski otisci u kilobyte-ima.

Tablica 4.3. Brzina izvođenja i memorijski otisak (ADAS) (1/2)

	Sobel		Laplace		Canny		Prewitt	
naziv slike	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]
3096	69	300	29	300	360	604.04	69	300
8023	71	300	29	300	405	604.04	70	300
12084	71	300	30	300	406	604.04	70	300
14037	67	300	29	300	350	604.04	67	300
16077	71	300	29	300	394	604.04	70	300
19021	70	300	29	300	396	604.04	70	300
21077	71	300	30	300	385	604.04	70	300
24077	71	300	29	300	401	604.04	70	300
37073	71	300	29	300	401	604.04	70	300
38082	70	300	29	300	401	604.04	70	300
38092	70	300	29	300	390	604.04	70	300
41033	71	300	29	300	403	604.04	71	300
41069	71	300	29	300	412	604.04	70	300
42049	68	300	29	300	349	604.04	68	300
43074	71	300	29	300	388	604.04	70	300
45096	66	300	30	300	325	604.04	66	300
58060	69	300	29	300	377	604.04	69	300
62096	71	300	29	300	393	604.04	70	300
65033	70	300	30	300	398	604.04	70	300
66053	70	300	29	300	403	604.04	70	300
69020	71	300	29	300	409	604.04	71	300
69040	71	300	29	300	404	604.04	71	300
76053	71	300	29	300	402	604.04	71	300
85048	71	300	29	300	411	604.04	71	300
86016	71	300	29	300	414	604.04	70	300
86068	70	300	29	300	405	604.04	71	300
87046	71	300	29	300	409	604.04	71	300
97033	70	300	29	300	403	604.04	70	300
103070	70	300	29	300	404	604.04	70	300
105025	69	300	29	300	381	604.04	69	300
106024	70	300	29	300	389	604.04	70	300
108005	71	300	29	300	405	604.04	71	300
108070	71	300	29	300	411	604.04	71	300
108082	69	300	29	300	378	604.04	69	300
109053	71	300	29	300	394	604.04	70	300
119082	70	300	29	300	385	604.04	70	300
123074	71	300	29	300	407	604.04	71	300
126007	69	300	29	300	365	604.04	68	300
130026	71	300	29	300	408	604.04	71	300
134035	70	300	29	300	410	604.04	71	300
143090	70	300	29	300	379	604.04	70	300
145086	70	300	29	300	398	604.04	71	300

Tablica 4.4. Vrijeme izvođenja i memorijski otisak (ADAS) (2/2)

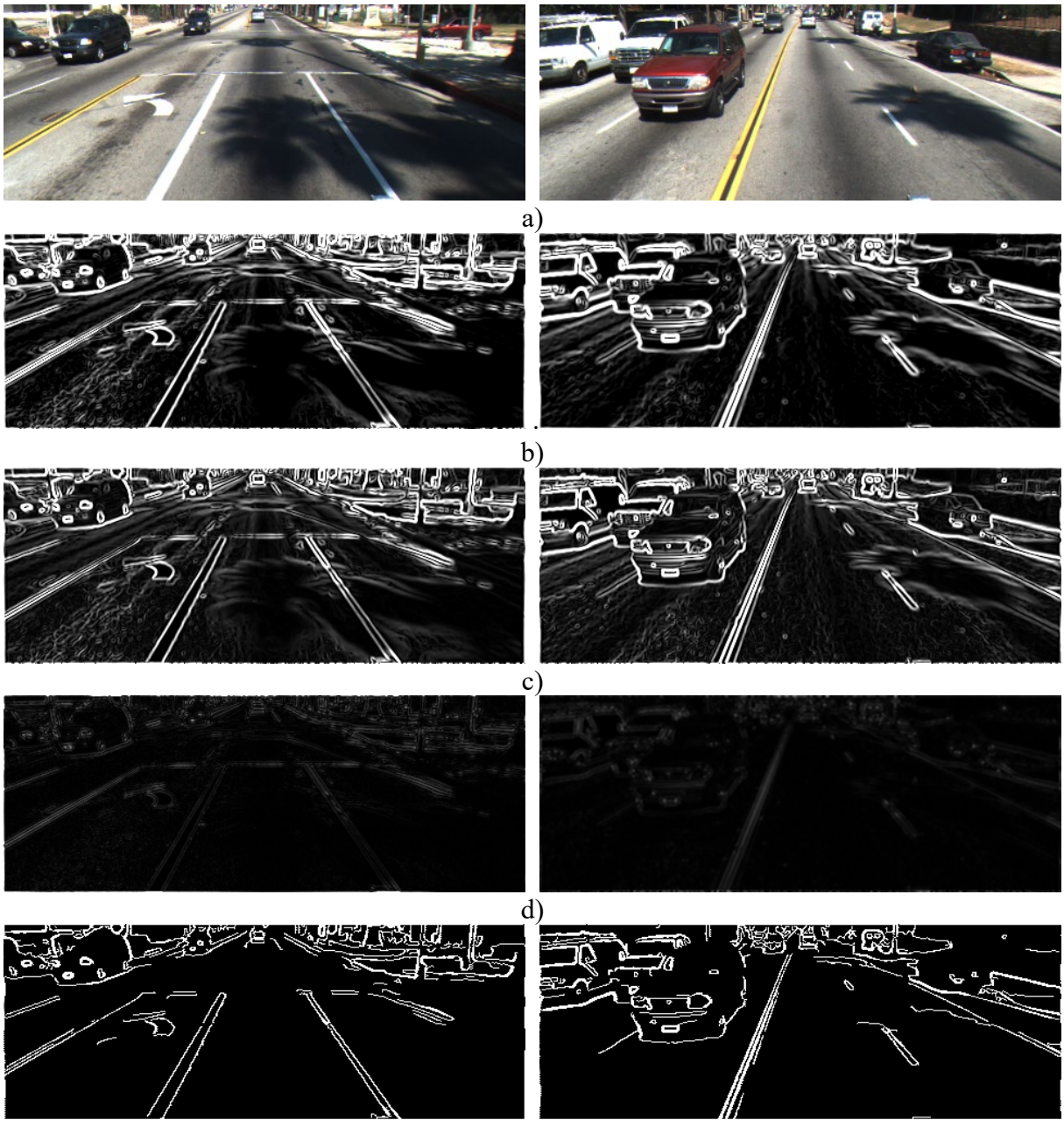
	Sobel		Laplace		Canny		Prewitt	
naziv slike	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]
147091	71	300	29	300	395	604.04	71	300
148089	71	300	29	300	397	604.04	71	300
156065	71	300	29	300	399	604.04	70	300
157055	71	300	29	300	388	604.04	71	300
159008	70	300	29	300	390	604.04	69	300
160068	69	300	29	300	372	604.04	69	300
163085	71	300	29	300	396	604.04	70	300
167062	65	300	29	300	317	604.04	69	300
170057	71	300	29	300	402	604.04	70	300
175043	71	300	29	300	415	604.04	71	300
182053	70	300	29	300	379	604.04	70	300
196073	71	300	29	300	403	604.04	70	300
197017	71	300	30	300	401	604.04	71	300
216081	70	300	29	300	382	604.04	69	300
219090	70	300	29	300	389	604.04	70	300
220075	71	300	29	300	397	604.04	70	300
223061	70	300	29	300	406	604.04	70	300
229036	70	300	29	300	406	604.04	71	300
236037	71	300	29	300	414	604.04	70	300
241004	69	300	29	300	370	604.04	68	300
241048	70	300	29	300	392	604.04	69	300
253027	71	300	30	300	406	604.04	71	300
253055	70	300	29	300	376	604.04	69	300
260058	71	300	29	300	397	604.04	71	300
291000	71	300	29	300	412	604.04	71	300
295087	70	300	29	300	384	604.04	70	300
296007	71	300	29	300	396	604.04	70	300
296059	70	300	29	300	386	604.04	70	300
299086	70	300	29	300	387	604.04	70	300
300091	71	300	29	300	390	604.04	71	300
304034	71	300	29	300	413	604.04	71	300
306005	69	300	29	300	380	604.04	69	300
351093	68	300	29	300	365	604.04	67	300
361010	71	300	29	300	402	604.04	70	300
385039	71	300	29	300	398	604.04	70	300

Iz tablica se može vidjeti kako Cannyjev algoritam ima najdulje vrijeme izvođenja i dvostruko veći memorijski otisak u odnosu na ostale algoritme. Razlog većem memorijskom otisku je potreba za spremanjem smjerova gradijenta za korištenje u koraku nemaksimalnog suzbijanja. Osim što je vrijeme izvođenja najdulje, ima i najveću varijaciju. Na varijaciju utječe posljednji

korak algoritma, odnosno usporedba s donjom i gornjom granicom, čije je vrijeme izvođenja ovisno o broju elemenata koje je potrebno povezati.

4.1.2. Rezultati testiranja na Kitti skupu podataka

Na slikama izdvojenim iz Kitti skupa podataka prikazani su pojedini kadrovi video zapisa snimljenog kamerom postavljenom na prednju stranu automobila u vožnji. Primjer izvornih slika i izlaznih slika detektora može se vidjeti na slici 4.5..



Sl. 4.5. a) izvorna slika b) Sobel c) Prewitt d) Laplace e) Canny

U tablici 4.5. prikazane su srednje sume apsolutnih razlika te ukupni brojevi različitih elemenata slike za svaku korištenu sliku iz Kitti skupa podataka. Izlazne slike Sobelovog, Laplaceovog i Cannyjevog detektora implementiranih na ADAS ploči uspoređene su s izlaznim slikama istih detektora koji postoje u OpenCV biblioteci. Budući da Prewitt nije implementiran u samom OpenCV-u, njegova izlazna slika uspoređena je s vlastitom implementacijom napisanom koristeći funkcionalnosti OpenCV-a.

Tablica 4.5. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (ADAS)

	Sobel		Prewitt		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
road_0	27.28	82034	17.73	80356	64.65	14970	7.21	72723
road_1	25.06	81106	16.95	80462	60.05	13555	6.91	72981
road_2	24.93	81503	16.91	80774	62.18	12820	6.53	73001
road_3	25.71	82659	16.18	81165	63.17	12214	6.38	73180
road_4	28.05	82400	18.1	81688	62.47	14329	6.98	75759
road_5	27.52	80533	20.45	80836	69.51	19172	7.76	75993
road_6	25.5	82745	16.62	80680	68.3	11684	6.45	72779
road_7	26.01	79006	20.16	80699	55.34	18095	7.6	75640
road_8	27.61	82082	18.46	81230	71.34	15770	6.98	75618
road_9	35.59	82639	18.49	81038	75.32	14293	7.07	75695

Iz tablice se može vidjeti kako postoji razlika između izlaznih slika detektora na ploči i detektora u OpenCV-u. Do razlike dolazi zbog toga što su za testiranje na ADAS ploči korištene slike čiji je format pretvoren u NV12 dok su u OpenCV-u korištene slike čiji je format boja pretvoren u *grayscale* tijekom učitavanja na početku algoritma iz razloga što pri učitavanju slike u OpenCV-u nije podržan NV12.

Cannyjev algoritam ima srednju sumu apsolutnih razlika u rasponu od 55,34 do 75,32. S obzirom da izlazna slika Cannyjevog algoritma treba imati elemente vrijednosti 0 ili 255, takav rezultat nije očekivan. Uspoređene su vrijednosti elemenata izlaznih slika iz vlastitog algoritma i algoritma iz OpenCV-a te je primijećeno kako u slikama iz OpenCV-a postoji znatan broj elemenata čije vrijednosti nisu 0 ni 255 i zbog toga je srednja suma u prethodno navedenom rasponu.

U tablici 4.6. prikazana su vremena izvođenja i memorijski otisak za pojedinačne slike. Vidljivo je kako su vremena izvođenja manja u odnosu na slike iz Berkeley skupa, što je i očekivano zbog manje rezolucije slika iz Kitti skupa podataka.

Tablica 4.6. Vrijeme izvođenja i memorijski otisak (ADAS)

	Sobel		Laplace		Canny		Prewitt	
slika	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]	t [ms]	m [kB]
road_0	40	168.79	17	168.79	227	337.54	40	168.79
road_1	40	168.79	17	168.79	228	337.54	40	168.79
road_2	40	168.79	17	168.79	226	337.54	40	168.79
road_3	40	168.79	17	168.79	228	337.54	40	168.79
road_4	40	168.79	17	168.79	229	337.54	40	168.79
road_5	40	168.79	17	168.79	229	337.54	40	168.79
road_6	40	168.79	17	168.79	226	337.54	40	168.79
road_7	40	168.79	17	168.79	227	337.54	40	168.79
road_8	40	168.79	17	168.79	228	337.54	40	168.79
road_9	40	168.79	17	168.79	227	337.54	40	168.79

4.2. Testiranje u OpenCV biblioteci

Rad algoritama implementiranih u OpenCV-u testiran je na stolnom računala čije se specifikacije nalaze u tablici 4.7..

Tablica 4.7. Specifikacija računala

Procesor	Intel Core i7-6700
RAM	Kingston DDR4 16GB 1066MHz
HDD	Seagate Barracuda ST1000DM003 1TB
Grafička kartica	Intel HD Graphics 530
Matična kartica	Gigabyte GA-B150M-HD3
Operacijski sustav	Windows 10

4.2.1. Rezultati testiranja na Berkeley skupu podataka

U tablicama 4.8. i 4.9. nalazi se pregled srednje sume apsolutnih razlika i ukupnog broja različitih elemenata slika za svih 77 slika iz Berkeley skupa podataka.

Tablica 4.8. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (OpenCV) (1/2)

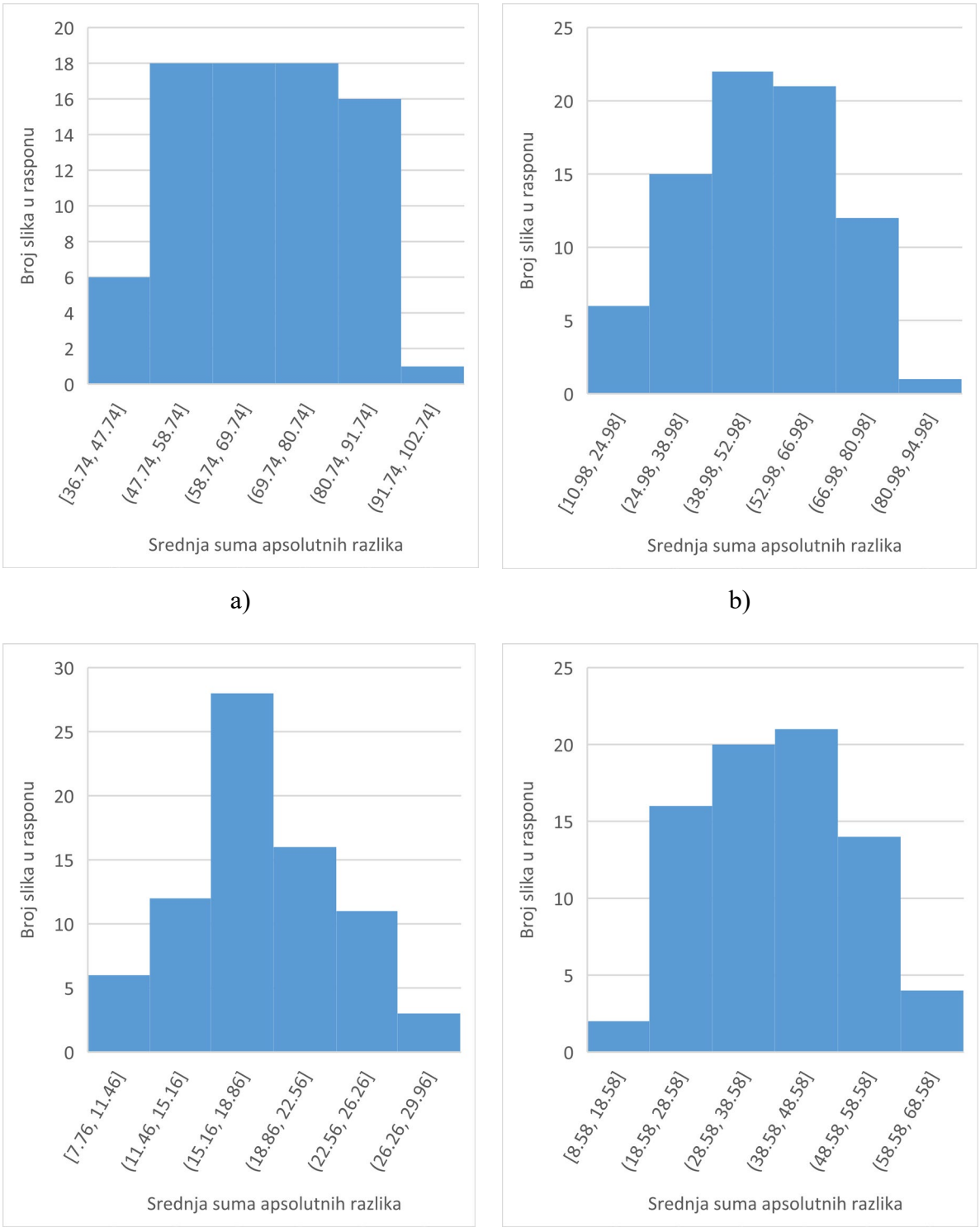
	Sobel		Prewitt		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
3096	10.98	131964	8.58	131493	48.6	11500	7.76	69791
8023	42.22	151683	32.17	151584	67.14	24343	9.92	93043
12084	54.6	150842	41.55	150713	69.85	56571	21.12	104965
14037	26.49	125307	21.48	124129	48.42	27716	16.62	79206
16077	53.19	150554	40.87	150313	70.64	48159	18.69	103296
19021	53.11	144485	41.33	144104	68.96	49392	20.03	100542
21077	48	146298	37.45	145980	78.26	41572	15.83	98674
24077	64.78	143610	52.66	143171	79.57	70856	25.22	105287
37073	26.42	148839	20.81	148209	56.91	42099	17.65	101393
38082	50.01	151697	37.44	151594	44.84	41522	15.4	109647
38092	50.69	142342	38.6	142396	62.92	46800	21.68	98700
41033	35.62	151278	27.37	150993	59.32	28030	13.56	99884
41069	67.09	151561	49.8	151437	74.14	48384	16.13	107150
42049	31.04	105915	25.48	105221	65.98	32534	23.21	65362
43074	29.45	149879	22.6	149598	51.85	22863	11.59	92197
45096	22.32	101368	18.61	100867	51.04	26780	19.26	63243
58060	81.15	136208	62.97	135823	94.55	69396	23.55	95483
62096	54.85	135054	42.95	134667	84.54	44527	16.78	88725
65033	61.33	149750	46.59	149497	64.58	57537	21.04	108585
66053	48.64	150571	37.08	150275	55.87	47119	18.45	104841
69020	47.99	151674	36.32	151570	50.19	38610	16.62	109614
69040	67.04	151935	50.49	151843	73.23	63292	16.66	107557
76053	41.07	150097	31.5	149860	46.61	41401	15.83	104046
85048	66.97	150812	51.96	150624	73.73	72280	24.31	114030
86016	63.67	152148	47.85	152042	68.33	56717	23.01	112953
86068	42.41	150857	31.84	150611	46.65	30709	13.7	101914
87046	61.92	151533	47.3	151409	81.19	46374	17.68	101494
97033	64.81	148526	49.87	148172	84.34	58261	17.58	102660
103070	39.57	151178	30.39	150967	49.36	36728	15.39	98065
105025	62.05	135402	47.03	135165	74.37	46386	17.56	94872
106024	26.67	143516	20.69	143193	47.26	20490	11.53	85015
108005	62.76	151690	48.31	151576	80.06	50838	16.24	103284
108070	69.61	151496	53.41	151306	82.32	55715	18.68	105837
108082	58.42	134426	45.56	133975	74.54	40640	16.66	88638
109053	40.57	149796	31.81	149485	56.08	48249	19.49	102112
119082	64.77	147591	51.7	147224	83.35	61994	24.25	104128
123074	47.94	151666	36.45	151512	59.95	31835	13.71	98897
126007	37.55	123757	30.17	122690	65.5	36103	20.02	81322
130026	60.33	148747	45.75	148470	81.12	46986	15.33	99405
134035	79.55	151048	62.15	150702	84.1	84458	24.71	113290
143090	23.62	142478	18.49	141967	57.57	25201	14.17	84079
145086	45.59	148139	36.52	147612	69.34	45448	18.51	98899
147091	39.93	145244	31.41	144663	73.01	41477	17.49	95212

Tablica 4.9. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (OpenCV) (2/2)

	Sobel		Prewitt		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
148089	70.76	150634	54.7	150398	78.16	85184	27.72	117091
156065	52.6	148368	39.3	148005	62.81	50199	16.83	104714
157055	51.68	148259	39.8	147961	65.18	64414	22.12	106057
159008	67.67	144244	52.3	143860	73.1	64307	19.19	100811
160068	51.21	130001	39.92	129669	84.44	44473	19.5	83556
163085	42.24	150831	33.46	150573	62.32	49805	20	102986
167062	24.79	101372	19.74	99940	47.92	19322	11.42	58793
170057	40.72	151738	32.29	151636	50.22	58991	22.16	109949
175043	80.88	151846	60.48	151778	83.49	64259	14.89	104901
182053	57.88	130743	45.28	129528	82.2	59291	27.94	90796
196073	31.35	151400	23.21	151127	56.75	14768	9.76	103137
197017	37.19	149801	28.5	149502	60.34	30360	15.92	95793
216081	49.81	144034	39.05	143736	69.19	58722	26.86	104324
219090	40.18	143808	31.71	143247	80.04	35532	16.58	92837
220075	54.72	149752	43.16	149549	73.87	50884	17.01	97781
223061	56.62	150007	42.26	149886	84.65	42100	12.51	97885
229036	72.77	151149	56.39	150945	88.45	62660	20.81	106443
236037	68.26	151770	51.29	151643	78.24	47658	15.14	102821
241004	26.69	133096	21.64	132381	56.17	30855	17.97	86135
241048	60.47	141356	45.88	141147	73.05	52872	19.75	101708
253027	60.92	150952	51.01	150695	87.2	65655	25.3	107911
253055	25.95	138525	20.98	137984	52.9	29885	17.42	89384
260058	26.34	150421	20.08	150015	36.74	17863	8.46	95701
291000	78.48	151844	58.55	151740	80.14	78668	24.55	113842
295087	38.52	144262	30.07	143490	58.45	42693	19.3	98442
296007	24.64	148884	19.63	148315	44.61	27562	14.04	97626
296059	28.23	144229	29.61	142970	49.21	26252	14.99	93183
299086	24.2	146126	19.27	145610	51.37	28856	15	92908
300091	30.12	146706	22.87	146111	65.75	22282	10.65	89920
304034	80	151093	59.97	150953	87.96	55161	19.21	93183
306005	49.43	135120	37.92	134693	61.28	51172	24.23	85396
351093	73.77	122219	56.5	121530	89.49	58299	18.62	102028
361010	41.97	151055	33.02	150870	68.69	41136	23.07	100253
385039	55.88	142551	44.5	142174	81.36	55501	23.07	100253

Na slici 4.6. grafički je raspodjela srednjih suma apsolutnih razlika između *ground-truth* slika i izlaznih slika pojedinih detektora, dok je na slici 4.7. grafički prikazana raspodjela ukupnog broja različitih piksela po slikama. Vidljivo je kako i u ovom slučaju Cannyjev algoritam ima najmanji broj razlika no najveću srednju sumu apsolutnih razlika dok ostali algoritmi imaju veći broj razlika, ali ukupno manju srednju sumu. Razlika u rezultatima u odnosu na izlaze algoritama

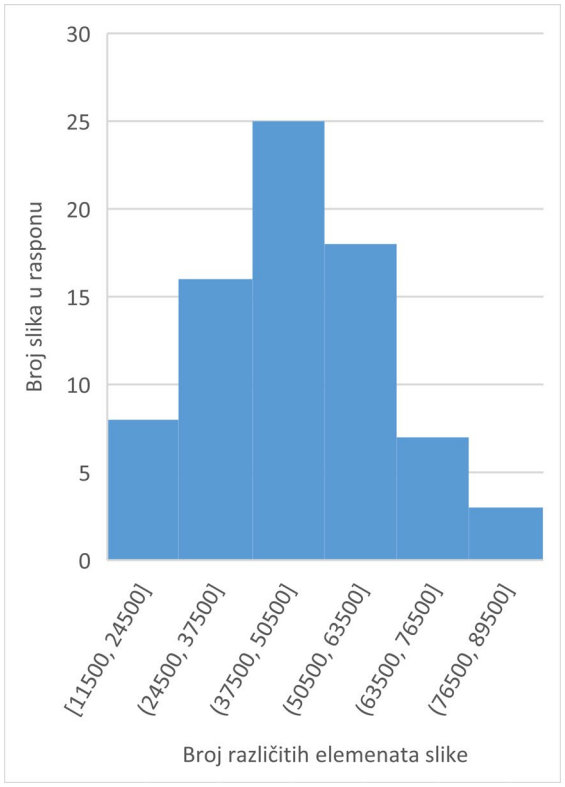
implementiranih na ADAS ploču nastala je zbog razlike u ulaznim vrijednostima, kao što je objašnjeno u prošlom potpoglavlju u slučaju slika iz Kitti skupa podataka.



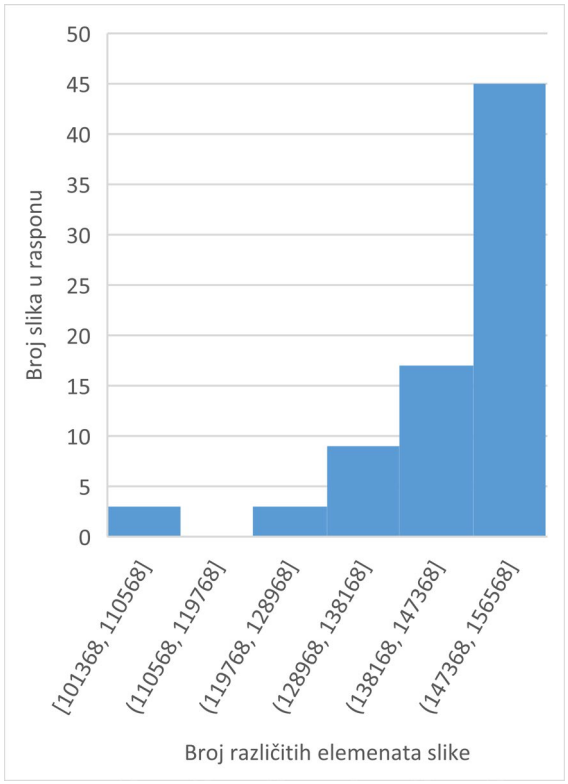
a) b)

c) d)

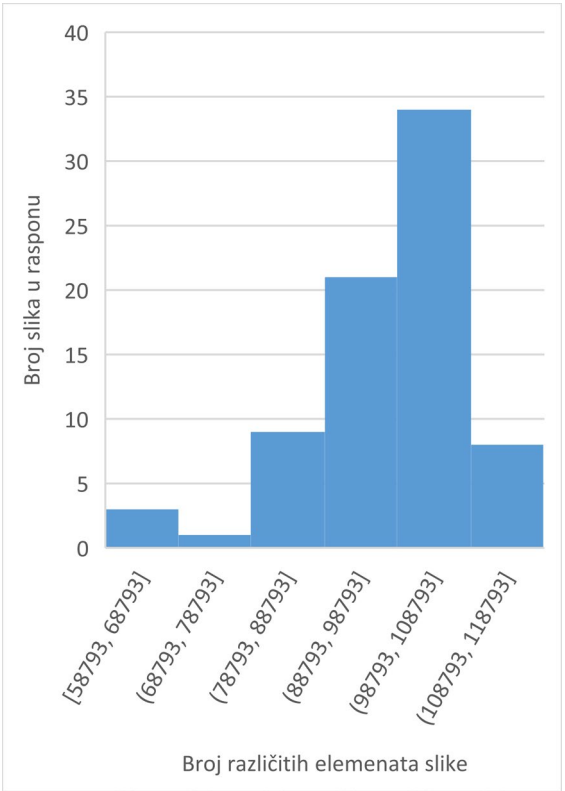
Sl. 4.6. Srednja suma apsolutnih razlika u odnosu na *ground-truth* slike za a) Canny b) Sobel c) Laplace d) Prewitt



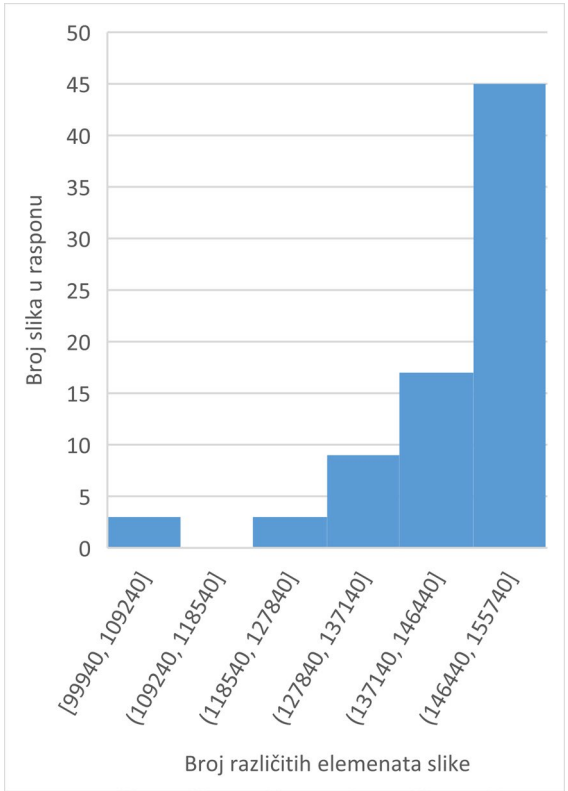
a)



b)



c)



d)

Sl. 4.7. Broj različitih piksela u odnosu na *ground-truth* slike za a) Canny b) Sobel c) Laplace d) Prewitt

U tablicama 4.10. i 4.11. prikazana su vremena izvođenja i memorijski otisak algoritama. U stupcima označenima s *t* nalaze se vremena izvođenja u milisekundama dok se u stupcima označenim s *m* nalaze memorijski otisci u megabyte-ima.

Tablica 4.10. Vrijeme izvođenja i memorijski otisak (OpenCV) (1/2)

	Sobel		Laplace		Canny		Prewitt	
slika	t [ms]	m [MB]	t [ms]	m [MB]	t [ms]	m [MB]	t [ms]	m [MB]
3096	103	6.5	50	6.5	123	6.8	113	6.5
8023	106	6.5	59	6.5	127	6.8	125	6.5
12084	108	6.5	50	6.5	127	6.8	108	6.5
14037	118	6.5	48	6.5	122	6.8	108	6.5
16077	116	6.5	55	6.5	125	6.8	110	6.5
19021	105	6.5	48	6.5	134	6.8	108	6.5
21077	119	6.5	48	6.5	127	6.8	131	6.5
24077	110	6.5	51	6.5	129	6.8	107	6.5
37073	106	6.5	50	6.5	127	6.8	106	6.5
38082	103	6.5	51	6.5	125	6.8	124	6.5
38092	104	6.5	50	6.5	127	6.8	113	6.5
41033	114	6.5	49	6.5	126	6.8	111	6.5
41069	106	6.5	52	6.5	127	6.8	106	6.5
42049	110	6.5	51	6.5	121	6.8	111	6.5
43074	114	6.5	50	6.5	124	6.8	107	6.5
45096	110	6.5	49	6.5	119	6.8	126	6.5
58060	108	6.5	50	6.5	148	6.8	115	6.5
62096	117	6.5	48	6.5	135	6.8	108	6.5
65033	108	6.5	51	6.5	128	6.8	114	6.5
66053	115	6.5	49	6.5	126	6.8	109	6.5
69020	109	6.5	48	6.5	126	6.8	111	6.5
69040	105	6.5	55	6.5	128	6.8	110	6.5
76053	108	6.5	52	6.5	139	6.8	109	6.5
85048	110	6.5	48	6.5	130	6.8	109	6.5
86016	108	6.5	52	6.5	135	6.8	106	6.5
86068	115	6.5	51	6.5	123	6.8	113	6.5
87046	113	6.5	49	6.5	129	6.8	108	6.5
97033	105	6.5	50	6.5	125	6.8	110	6.5
103070	109	6.5	53	6.5	123	6.8	109	6.5
105025	112	6.5	52	6.5	127	6.8	110	6.5
106024	116	6.5	48	6.5	127	6.8	111	6.5
108005	119	6.5	49	6.5	135	6.8	116	6.5
108070	111	6.5	49	6.5	132	6.8	109	6.5
108082	109	6.5	50	6.5	131	6.8	107	6.5
109053	107	6.5	48	6.5	128	6.8	113	6.5
119082	116	6.5	58	6.5	129	6.8	118	6.5
143090	122	6.5	51	6.5	127	6.8	109	6.5

Tablica 4.11. Vrijeme izvođenja i memorijski otisak (OpenCV) (2/2)

	Sobel		Laplace		Canny		Prewitt	
slika	t [ms]	m [MB]	t [ms]	m [MB]	t [ms]	m [MB]	t [ms]	m [MB]
145086	107	6.5	51	6.5	130	6.8	109	6.5
147091	106	6.5	56	6.5	121	6.8	110	6.5
148089	120	6.5	51	6.5	126	6.8	116	6.5
156065	108	6.5	52	6.5	127	6.8	111	6.5
157055	107	6.5	48	6.5	122	6.8	113	6.5
159008	117	6.5	49	6.5	124	6.8	106	6.5
160068	107	6.5	49	6.5	124	6.8	109	6.5
163085	108	6.5	51	6.5	121	6.8	107	6.5
167062	108	6.5	49	6.5	117	6.8	110	6.5
170057	108	6.5	52	6.5	127	6.8	117	6.5
175043	110	6.5	55	6.5	129	6.8	110	6.5
182053	106	6.5	49	6.5	125	6.8	107	6.5
196073	107	6.5	50	6.5	117	6.8	122	6.5
197017	114	6.5	56	6.5	121	6.8	108	6.5
216081	111	6.5	50	6.5	122	6.8	132	6.5
219090	111	6.5	49	6.5	127	6.8	106	6.5
220075	108	6.5	49	6.5	123	6.8	107	6.5
223061	107	6.5	48	6.5	121	6.8	106	6.5
229036	107	6.5	50	6.5	129	6.8	109	6.5
236037	108	6.5	49	6.5	126	6.8	110	6.5
241004	124	6.5	58	6.5	119	6.8	117	6.5
241048	109	6.5	51	6.5	122	6.8	109	6.5
253027	109	6.5	56	6.5	125	6.8	106	6.5
253055	120	6.5	52	6.5	116	6.8	108	6.5
260058	107	6.5	48	6.5	121	6.8	108	6.5
291000	108	6.5	53	6.5	126	6.8	110	6.5
295087	109	6.5	51	6.5	120	6.8	115	6.5
296007	106	6.5	50	6.5	130	6.8	107	6.5
296059	106	6.5	48	6.5	120	6.8	109	6.5
299086	111	6.5	54	6.5	118	6.8	115	6.5
300091	111	6.5	48	6.5	119	6.8	107	6.5
304034	107	6.5	49	6.5	128	6.8	110	6.5
306005	107	6.5	48	6.5	122	6.8	105	6.5
351093	107	6.5	49	6.5	125	6.8	106	6.5
361010	114	6.5	49	6.5	125	6.8	115	6.5
385039	113	6.5	50	6.5	125	6.8	117	6.5

Iz tablica se može vidjeti kako u ovom slučaju Cannyjev algoritam ima razmjerno kraće vrijeme izvođenja u odnosu na vrijeme izvođenja na ADAS ploči, no ipak dulje od ostalih algoritama. Također se može vidjeti kako je vrijeme izvođenja Sobelovog, Laplaceovog i Prewitt detektora dulje od vremena izvođenja istih detektora na ADAS ploči.

4.2.2. Rezultati testiranja na Kitti skupu podataka

U tablici 4.12. prikazane su srednje sume apsolutnih razlika te ukupni brojevi različitih elemenata slike za svaku korištenu sliku iz Kitti skupa podataka. Kako Prewitt operator nije implementiran u samom OpenCV-u, na tom algoritmu nije obavljeno mjerenje u ovom setu. Uočeno je kako Sobelov operator pokazuje najveći broj različitih piksela. Detaljnijom usporedbom izlaznih slika zaključeno je kako se najveće razlike u vrijednostima izlaznih slika OpenCV i vlastite implementacije pojavljuju u prvim i zadnjim retcima i stupcima matrica slika. Do te razlike dolazi zbog toga što ti redovi i stupci nisu obrađivani u vlastitoj implementaciji. Pred toga, u ostalim dijelovima izlaznih slika moguće je pronaći značajan broj elemenata čije se vrijednosti razlikuju za 1 kao rezultat različitog zaokruživanja vrijednosti nakon izračuna. Zbog tih elemenata Sobelov operator pokazuje najmanju srednju sumu apsolutnih razlika. Između izlaznih slika postojećeg i vlastitog Laplaceovog operatora razlike su pronađene samo u prvim i zadnjim stupcima i redovima iz istog razloga kao i u slučaju Sobelovog operatora. Nadalje, uočen je i značajan broj različitih elemenata između postojeće i vlastite implementacije Cannyjevog algoritma. Kao i u slučaju testiranja na ADAS ploči, značajan dio tih razlika nastaje zbog elemenata izlazne slike postojećeg rješenja čije se vrijednosti razlikuju od 0 i 255. Ostatak nastaje zbog lažno detektiranih rubova, nedetektiranih stvarnih rubova i zbog rubova na izlaznim slikama vlastitog rješenja čija je širina veća od jednog elementa.

Tablica 4.12. Srednja suma apsolutnih razlika i broj različitih elemenata za sve slike (OpenCV)

	Sobel		Canny		Laplace	
slika	MAE	Broj razlika	MAE	Broj razlika	MAE	Broj razlika
road_0	10.7	18252	44.07	12562	98.66	1316
road_1	11.16	17909	46.38	13407	97.26	1316
road_2	10.93	18432	44.38	13816	101.43	1316
road_3	13.31	15465	44.81	12198	104.61	1316
road_4	10.78	17755	45.62	13042	102.76	1316
road_5	9.49	23914	42.47	18881	105.45	1316
road_6	11.95	14838	40.07	10627	98.96	1316
road_7	8.16	26580	45.69	19839	103.34	1316
road_8	10.56	19342	46.7	14775	98.03	1315
road_9	10.64	18319	52.2	15462	103.25	1315

U tablici 4.13. prikazani su vrijeme izvođenja i memorijski otisak algoritama implementiranih u OpenCV-u. Kao i za Berkeley skup podataka, vremena izvođenja Laplaceovog, Sobelovog i Prewitt opretatora je dulje u OpenCV implementaciji nego u implementaciji na ploči, dok je vrijeme izvođenja Cannyjevog algoritma razmjerno kraće.

Tablica 4.13. Vrijeme izvođenja i memorijski otisak (OpenCV)

	Sobel		Laplace		Canny		Prewitt	
slika	t [ms]	m [kB]	t [ms]	m [MB]	t [ms]	m [MB]	t [ms]	m [kB]
road_0	62	6.1	30	6.1	68	6.2	64	6.1
road_1	64	6.1	29	6.1	69	6.2	65	6.1
road_2	61	6.1	29	6.1	73	6.2	61	6.1
road_3	69	6.1	28	6.1	70	6.2	66	6.1
road_4	63	6.1	30	6.1	72	6.2	67	6.1
road_5	62	6.1	30	6.1	74	6.2	73	6.1
road_6	61	6.1	32	6.1	71	6.2	60	6.1
road_7	61	6.1	31	6.1	69	6.2	60	6.1
road_8	60	6.1	29	6.1	72	6.2	65	6.1
road_9	72	6.1	28	6.1	71	6.2	60	6.1

5. ZAKLJUČAK

Detekcija rubova je od velike važnosti u razvoju algoritama za detekciju objekata u ADAS sustavima. Izlazna slika detektora rubova lakša je za obradu zbog manje količine infomacije koju algoritam mora obraditi i zbog toga postoji potreba za implementacijom detektora rubova na ugradbeni računalni sustav koji može biti ugrađen u automobil.

U ovom diplomskom radu prikazana je implementacija Sobelovog, Laplaceovog i Prewitt operatora te Cannyjevog algoritma za detekciju rubova na realnu ADAS platformu. Algoritmi su prvobitno implementirani koristeći funkcionalnosti biblioteke OpenCV. Rješenja su uspoređivana s postojećim rješenjima te su zatim implementirana na ADAS ploču. Testiranje rješenja provedeno je koristeći Berkeley i Kitti skupove podataka. Testovi su obavljeni na rješenjima na ploči i na rješenjima u OpenCV-u.

Testirana je točnost pronalaska rubova u odnosu na *ground-truth* slike u slučaju Berkeley skupa podataka, odnosno na izlazne slike postojećih detektora u OpenCV-u u slučaju Kitti skupa podataka. U slučaju testiranja koristeći slike iz Berkeley skupa podataka na ploči, pokazano je kako izlazne slike Cannyjevog algoritma imaju najmanji broj različitih elemenata dok izlazne slike Laplaceovog operatora imaju najmanju srednju sumu apsolutnih razlika.

Provedenim testiranjem također je pokazano kako Sobel, Laplace i Prewitt operatori imaju kraće vrijeme izvođenja na ploči nego na računalu, dok je vrijeme izvođenja Cannyjevog algoritma dulje na ploči. Nadalje, Implementirani Cannyjev algoritam, čija je izlazna slika najpogodnija za daljnje korištenje u detekciji objekata na ploči ima vrijeme izvođenja u rasponu od 350ms do 414ms na slikama iz Berkeley skupa podataka, odnosno od 227ms do 229ms na slikama iz Kitti skupa podataka. Kao takav nije pogodan za primjenu u stvarnom vremenu. Postoji prostor za nadogradnju rješenja, poput podjele ulazne slike na više segmenata koji bi paralelno bili obrađeni.

LITERATURA

[1] S. Singh, „Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey“, *Traffic Saf. Facts - Crash Stats*, Art. izd. DOT HS 812 506, ožu. 2018, Pristupljeno: ruj. 17, 2020. [Na internetu]. Dostupno na: https://trid.trb.org/view/1507603.

[2] „SAE International Releases Updated Visual Chart for Its “Levels of Driving Automation” Standard for Self-Driving Vehicles“. https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles (pristupljeno ruj. 17, 2020).

[3] S. E. Umbaugh, J. Snyder, i E. A. Fedorovskaya, „Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools, Second Edition“, *J Electron. Imaging*, 2011, doi: 10.1117/1.3628179.

[4] „ImageGradients.pdf“. Pristupljeno: ruj. 10, 2020. [Na internetu]. Dostupno na: https://www.cs.umd.edu/~djacobs/CMSC426/ImageGradients.pdf.

[5] „The Laplace Operator“. http://fourier.eng.hmc.edu/e161/lectures/gradient/node7.html (pristupljeno ruj. 18, 2020).

[6] L. J. van Vliet, I. T. Young, i G. L. Beckers, „A nonlinear laplace operator as edge detector in noisy images“, *Comput. Vis. Graph. Image Process.*, sv. 45, izd. 2, str. 167–195, velj. 1989, doi: 10.1016/0734-189X(89)90131-X.

[7] J. Canny, „A Computational Approach to Edge Detection“, *IEEE Trans. Pattern Anal. Mach. Intell.*, sv. 8, izd. 6, str. 679–698, lip. 1986, doi: 10.1109/TPAMI.1986.4767851.

[8] S. Usama, K. Bukhari, R. Brad, i C. Bălă – Zamfirescu, „Fast Edge Detection Algorithm for Embedded Systems“, *Stud. Inform. Control*, sv. 23, izd. 2, lip. 2014, doi: 10.24846/v23i2y201404.

[9] „The Berkeley Segmentation Dataset and Benchmark“. https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/ (pristupljeno kol. 13, 2020).

[10] A. J. Baddeley, *An Error Metric for Binary Images*. 1992.

[11] Q. Xu, S. Varadarajan, C. Chakrabarti, i L. J. Karam, „A Distributed Canny Edge Detector: Algorithm and FPGA Implementation“, *IEEE Trans. Image Process.*, sv. 23, izd. 7, str. 2944–2960, srp. 2014, doi: 10.1109/TIP.2014.2311656.

[12] J. K. Su i R. M. Mersereau, „Post-processing for artifact reduction in JPEG-compressed images“, u *1995 International Conference on Acoustics, Speech, and Signal Processing*, svi. 1995, sv. 4, str. 2363–2366 sv.4, doi: 10.1109/ICASSP.1995.479967.

[13] „TDA4 | TDA3 | TDA2| ADAS | Design & development | Automotive Processors | TI.com“. https://www.ti.com/processors/automotive-processors/tdax-adas-socs/design-development.html (pristupljeno ruj. 28, 2020).

[14] „YUV - VideoLAN Wiki“. https://wiki.videolan.org/YUV (pristupljeno ruj. 02, 2020).

[15] „OpenCV: Canny Edge Detection“. https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html (pristupljeno ruj. 14, 2020).

[16] Naturalthoughts, „Back To Basics : YUV to RGB conversion (Color Space)“, *Natural Thoughts*, lip. 25, 2009. http://thinknaturally.blogspot.com/2009/06/back-to-basics-yuv-to-rgb-conversion.html (pristupljeno ruj. 04, 2020).

[17] M. Nixon, *Feature Extraction & Image Processing - 2nd Edition*. Elsevier, 2008.

[18] „About“. https://opencv.org/about/ (pristupljeno ruj. 12, 2020).

[19] „The KITTI Vision Benchmark Suite“. http://www.cvlibs.net/datasets/kitti/ (pristupljeno ruj. 18, 2020).

SAŽETAK

U ovom diplomskom radu opisana je implementacija Sobelovog, Prewitt i Laplaceovog operatora te Cannyjevog algoritma za detekciju rubova na realnu ADAS platformu. Detektori su najprije napisani u programskom jeziku C++ koristeći funkcionalnosti biblioteke OpenCV te su zatim preneseni u VisionSDK okruženje u programskom jeziku C i implementirani na ADAS ploču. Za testiranje rješenja korišteni su Berkeley i Kitti skupovi podataka. Svojstva implementiranih detektora evaulirana su kroz vrijeme izvođenja, memorijski otisak te kroz usporedbu izlaznih slika detektora sa slikama označenima temeljnom istinom u slučaju Berkeley skupa, odnosno izlaznim slikama postojećih detektora u OpenCV-u u slučaju Kitti skupa podataka. Pokazano je kako Cannyjev algoritam ima najdulje vrijeme izvođenja na ploči i računalu te najveći memorijski otisak dok Laplaceov operator ima najkraće vrijeme izvođenja. Također je pokazano da izlazne slike Cannyjevog algoritma na ADAS ploči imaju najmanje različitih elemenata u odnosu na slike označene temeljnom istinom odnosno izlazne slike detektora iz OpenCV-a.

Ključne riječi: *detekcija rubova, ADAS, VisionSDK, Sobel, Prewitt, Laplace, Canny*

Implementation of image edge detection algorithm on a real ADAS platform

ABSTRACT

This master's thesis describes the implementation of the Sobel, Prewitt and Laplace operators and the Canny edge detection algorithm onto a real ADAS platform. The detectors were first written in C++ using OpenCV functionalities and then transfered into the VisionSDK environment in the C programming language and implemented on the ADAS board. The Berkeley and Ktti datasets were used during testing. The performance of the solutions was evalueated through execution time, memory imprint and comparing the detectors' output images with ground truth labeled images for images from the Berkeley and the output images of existing OpenCV detectors for images from the Kitti dataset. It was shown that the Canny algorithm's execution time was the longest on both the board and computer and that it had the largest memory imprint, while the Laplace operator had the shortest execution time. It was also shown that the output images of the Canny algorithm on the ADAS board has the least total different pixels compared to ground-truth labeled images and OpenCV detector output images.

Keywords: *edge detection, ADAS, VisionSDK, Sobel, Prewitt, Laplace, Canny*

ŽIVOTOPIS

Dario Ćorić rođen je 1.3.1996.g. u Dubrovniku. Godine 2014. završava Opću gimnaziju KŠC „Sveti Pavao“ u Zenici. Iste godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija koji završava 2018. godine te potom upisuje diplomski studij računarstva.

_____

Potpis

PRILOZI

- P.3.1. Programski kod Sobelovog operatora
- P.3.2. Programski kod Prewitt operetora
- P.3.3. Programski kod Laplaceovog operatora
- P.3.4. Programski kod Cannyjevog algoritma
- P.4.1. Slike iz Berkeley skupa podataka korištene u procesu testiranja algoritama
- P.4.2. Slike iz Kitti skupa podataka korištene u procesu testiranja algoritama