FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt iz predmeta Računalni vid
Ak. god. 2020/21

Rubni operatori

Autor: Fran Pugelnik

prosinac, 2020

Sadržaj

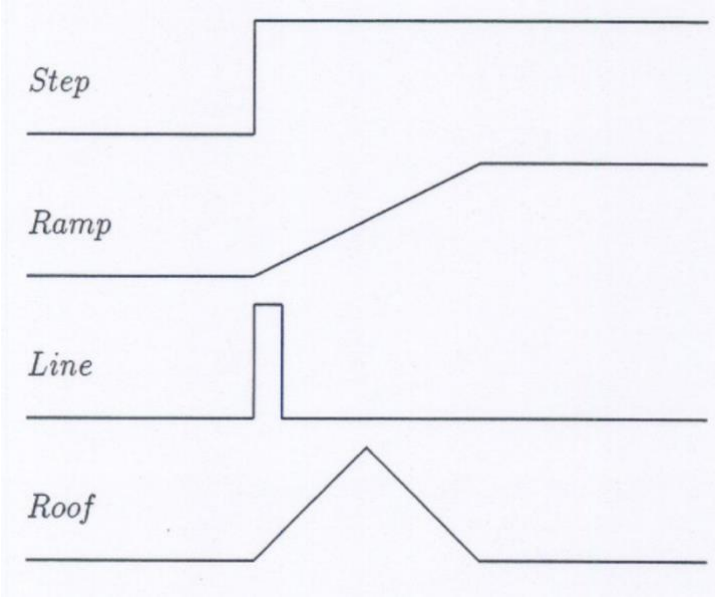
1. Projektni zadatak

1.1 Teorijska podloga i problematika zadatka

U ranim koracima većine algoritma računalnog vida pokušavamo izlučiti nekakve zadatku relevantne značajke za koje želimo da nose određena svojstva. Među takve značajke spadaju rubovi unutar slike koji su nam veoma korisni kod segmentacije i sličnih operacija.
Prvo definirajmo što su rubovi i njihova svojstva te teoretski kako ih razlikovati od ostatka slikovnih jedinica (piksela) u slici. Rubovi se tipično karakteriziraju tako da su oni granica između dvaju razina ili površina slike što znači da se na području rubova događa prijelaz iz jednog u drugo područje i to za sobom povlači uočljivu promjenu intenziteta slike. Intenzitet slikovnog elementa nam predstavlja osvijetljenost tog elementa ili možda bolje reći svjetlinu u tom malom području i generalno se kreće u rangu od 0 do 255. U našem slučaju treba naglasiti da promatramo slike koje će biti u grayscale formatu, to jest nisu u boji i pikseli su u biti razine sive boje.
Vratimo se sada na promjenu intenziteta koja nam definira rubove, dakle kada u slici uočimo diskontinuitet intenziteta slike ili prve derivacije istog on može biti jedan od četiri moguće opcije. Jedan od oblika u kojem se diskontinuiteti[1] mogu pojaviti je line (linijski) diskontinuitet koji je okarakteriziran brzom i iznenadnom promjenom intenziteta bilo povećanjem ili smanjenjem, a zatim je popraćen naglom promjenom u suprotnom sjeru dok se ne vrati na izvorni iznos. Druga vrsta diskontinuiteta je step (koračni) diskontinuitet gdje se vrijednost intenzitet također iznenadno i brzo promjeni u određenom smjeru, ali za razliku od linijskog intenziteta ovaj se ne vraća na izvornu vrijednost već ostaje na novom iznosu, mogli bi reći da prelazi na drugu razinu ili površinu.
Nažalost ovakvi oblici se uglavnom ne jeavljaju u realnim situacijama već su više idealni oblici, kod slika češće uočavamo preostala dva oblika ramp(rampa) i roof (krovni). Ramp diskontinuitet je definitran tako da se promjena intenziteta vrši postepeno iz jedne prema drugoj vrijednosti intenziteta i ona je kao što se da uočiti realna verzija idealnijeg step diskontinuiteta. Četvrti oblik koji se pojavljuje je roof diskontinuitet i njega karakterizira to da liči obrisu krova, dakle kreće postepeno iz početne razine intenziteta prema novoj razini te kada ju dostigne se

vreća postepeno u suprotnom smjeru prema početnom iznosu. Radi lakše vizualizacije ovih prijelaza referirat ćemo se na Slika 1.

Također postoji i mogućnost kombiniranja ova četiri oblika kojim dobivano razne varijacije, ali to ne predstavlja prevelik problem. Ono što bi eventualno moglo predstavljati poteškoću je kada u slici imamo prisutan visok šum ili neuobičajene oblike koji odbijaju svjetlo na različite načine. Dakle generalno fokusirati ćemo se na step i ramp oblike diskontinuiteta, međutim opisani



Slika 1. oblici diskontinuiteta

koncepti nisu primjenjivi samo na te oblike nego i generalno. Dakle zašto su nam rubovi toliko važni na slici, pa jednostavno oni često predstavljaju promjenu površine, a te različite površine često odgovaraju različitim segmentima koje tražimo.

Valjalo bi spomenuti da je prisutnost šuma u slikama značajan problem kod detekcije rubova i da ćemo radi njega morati imati dodatne korake u postupku kako nebi dobili krive odzive. Koraci kod algoritama detekcije rubova će dakle biti prvo obavljamo filtriranje kako bi se riješili početnog šuma i uglavnom koristimo gausovo zaglađivanje[1] u tom koraku koje nam daje zamućenu sliku, ali manju količinu šuma. Naravno treba biti umjeren sa gausovima zaglađivanjem kako ne bi izgubili informacije o rubovima. Drugi korak je generalno naglašavanje područje koje su imale snažan odziv na neki od naših rubnih operatora i treći korak se fokusira na detekciju rubova, to jest traži one odzive koji bi uistinu predstavljali rubove, a ne krive odzive. U svrhu toga najčešće koristimo prag iznad kojeg rješenja uzimamo, a ispod kojeg zanemarujemo.

Kada govorimo o detekciji rubova gradijent nam je izuzetno važan jer nam u suštini on govori gdje se nalazi rub. Gradijent nam generalno govori o promjeni neke funkcije što u biti često

korelira sa rubom jer tražimo promjenu intenziteta. Za naše slike koje su dvodimenzionalne koristili bi gradijent definiran ovako

$$G[f(x,y)] = \begin{bmatrix} Gx \\ Gy \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Ovaj gradijent ima dva važna svojstva, prvo vektor koji dobijemo izračunom ovog gradijenta ukaziuje nam na smjer promjene funkcije i amplitudu te promjene koju računamo kao

$$G[f(x,y)] = \sqrt{Gx^2 + Gy^2}$$

gdje nam Gx i Gy predstavljau parcijalnu derivaciju u smjeru x i y. Često se kod magnitude koristi i samo zbroj apsolutnih vrijednosti umjesto korijena ili ponekad samo maksimalna vrijednost oba iznosa.

$$G[f(x,y)] = |Gx| + |Gy|$$

Na kraju ostaje nam izračunati smjer te promjene to jest kut koji dobijamo ovako

$$\alpha(x,y) = \tan^{-1}\left(\frac{Gy}{Gx}\right)$$

Pošto mi radimo sa diskretnim skupom podataka, a i radi računske jednostavnosti javlja se potreba za aproksimacijom gore navedenih gradijenata koju ćemo definirati ovako

$$Gx = f[i, j+1] - f[i,j]$$

te za Gy

$$Gy = f[i,j] - f[i+1,j]$$

Ove aproksimacije možemo matrično prikazati na ovaj način

$$Gx = \begin{matrix} -1 & 1 \end{matrix} \qquad Gx = \begin{matrix} 1 \\ -1 \end{matrix}$$

No problem koji se javlja kada ovako računamo aproksimaciju gradijenta je da nam se pozicije rezultata ne poklapaju za Gx rezultat se nalazi na $[\mathrm{i}, \mathrm{j} + \frac{1}{2}]$, a za Gy rezultat se nalazi na na $[\mathrm{i} + \frac{1}{2}, \mathrm{j}]$. Rješenje ovog problema je korištenje sljedećih matrica dimenzija 2x2

$$Gx = \begin{matrix} -1 & 1 \\ -1 & 1 \end{matrix} \qquad Gx = \begin{matrix} 1 & 1 \\ -1 & -1 \end{matrix}$$

No onda nam je rezultat na položaju $[\mathrm{i} + \frac{1}{2}, \mathrm{j} + \frac{1}{2}]$ što znači između piksela, to također možemo riješiti, ovaj put korištenjem matrice 3x3 no detaljnije o tome u sljedećem poglavlju.

1.2 Robertsov, sobelov i perwittov operator

Robertsov križni operator[1][3] je prvi koji ćemo obraditi i on radi aproksimaciju korištenjem matrice dimenzija 2x2

$$G\big[f[i,j]\big] = |f[i,j] - f[i+1,j+1]| + |f[i+1] - f[i,j+1]|$$

što ako koristimo konvolucijske maske možemo zapisati ovako

$$G\big[f[i,j]\big] = |Gx| + |Gy|$$

pa zatim aproksimirati matricama

$$Gx = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \qquad Gx = \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$$

Kao što smo prije rekli matrice dimenzija 2x2 imaju nezgodnu posljedicu što nam daju rješenje između piksela točnije na polovini udaljenosti među njima.

Sobelov operator[1][3] na drugu ruku koristi matrice dimenzija 3x3 koje nam daju rezultat na loakcijama piksela, ali gubimo po jedan piksel sa svake strane. Matrica veličine 3x3 računa gradijent za centralni element, drugim riječima uzima okolne elemente (osmero susjedstvo) i na temelju njih aproksimira gradijent u točci, no to znači da kod ruba slike ne možemo uzeti element koji je van slike već prvi element koji imamo u slici. Dakle na svakom rubu izgubit ćemo po jedan red/stupac piksela i zato koristimo proširenje kako bi dobili sliku iste dimenzije i to nam neće predstavljati značajan problem. Kod Robertsovog operatora na drugu ruku matrice su veličine 2x2 pa u biti računamo gradijent „između“ piksela i također gubimo piksele, ali upola manje njih nego kod 3x3 matrica. Sobelov kompasni operator je magnituda gradijenta

$$M = \sqrt{{s_x}^2 + {s_y}^2}$$

gdje su parcijalne derivacije

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$

$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4)$$

sa zadanom konstantom c=2. Nadalje to možemo aproksimirati matricama

$$s_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad s_y = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Iz ove dvije matrice možemo primijetiti da će Sobelov operator stavljati naglasak na elemnte bliže središtu, to se može zaključiti i po konstanti c koja utječe samo na elemente koji su u četvero susjedstvu srednjeg elementa.

Konačno recimo nešto o Prewittovom kompasnom rubnom operatoru[1][3], on je poseban slučaj Sobelovog operatora koji koristi konstantu c = 1 umjesto 2 i time ne stavlja naglasak na elemente bliže središtu matrice. Matrice za ovaj oblik opertora su dakle

$$s_x = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \qquad s_y = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

1.3 Kirschov operator

Kirschov kompasni rubni operator[2][4][5] funkcionira na malo drugačijem principu od prethodno navedenih operatora, iako on još uvijek koristi matrice dimenzija 3x3 i obavlja konvoluciju te matrice se razlikuju od prethodnih. Prvo navedimo te matrice pa ćemo pojasniti ideju iza kirschovog operatora

$$N = \begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix} \quad NW = \begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix} \quad W = \begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix} \quad SW = \begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$S = \begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix} \quad SE = \begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix} \quad E = \begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix} \quad NE = \begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

Kao što se da uočiti iz predočenog imamo osam matrica naprema prethodnih dvije kod Robertsovog, Sobelovog i Prewittovog operatora.

Funkcionalnost dodatnih matrica je procjena smjera u točki, tu procjenu smjera dobivamo tako da gledamo odzive za svih 8 konvolucija i tražimo najveći među njima. Taj odziv nam onda govori koji je kut orijentacije najizgledniji, zato smo matrice označili kao smjerove kod kompasa: sjever, sjeverozapad, zapad, itd. i oni kao što se da zaključiti razlikuju po koracima od 45°.

Prisjetimo se kod Sobelovog i Prewittovog operatora smo također mogli dokučiti smjer promjene funkciji ili bolje rečeno intenziteta i to smo radili pomoću arkus tangensa parcijalnih derivacija po y i x. Međutim ovdje imamo dvije prednosti, odmah dobijemo traženu orijentaciju kada izaberemo maksimalni odziv i rješenja su nam u sva četiri kvadranta za razliku od dva kod arkusa tangensa. Negativna stvar je to što imamo manju granulaciju nego kod računanja arkusa tangensa.

1.4 Konceptualno rješenje zadatka

Zadanu temu ćemo obraditi uz pomoć jezika Python. Python je odabran radi jednostavnosti rukovanja podatcima i biblioteka koje možemo koristiti unutar njega, a koje već imaju implementirane neke od potrebnih algoritama, poput Sobelovog operatora. Algoritme koji su ugrađeni u neke biblioteke iskoristit ćemo gdje god možemo radi usporedbe sa rješenjima koje smo sami implementirali.

Recimo onda nešto o alatima koje ćemo koristiti u sklopu ovoga zadatka, prvo ukratko nešto o OpenCv-u. OpenCv je biblioteka koja je specijalizirana i visoko optimizirana za korištenje u području računalnog vida sa naglaskom na uporabu u „real-time“ aplikacijama. Unutar nje implementiran nam je algoritam Sobelovog rubnog operatora[6] i Cannyev operator[8] koji ćemo koristiti u ovome radu. Cannyev operator koristit ćemo kao ground truth u zadatku i njega nismo posebno teorijski opisivali jer u ovom slučaju to nije potrebno, ali ćemo reći da ima dobre rezultate kod detekcije i jedan je od najčešće korištenih algoritama za detekciju rubova. Unutar OpenCv-a postoje još funkcije za učitavanje i prikaz slika koje ćemo često koristiti, sve ove podatke moguće je proučiti na stranicama dokumentacije[8].

Prije opisa postupka još ćemo naglasiti i biblioteku Numpy koja nam služi za baratanje matričnim tipovima podataka i dobar dio OpenCv-a koristit implementacije iz iste, za primjer recimo spremanje slike ili slika se odvija u matričnom zapisu pomoću tipa podataka iz Numpya..

Što se tiče postupka rješavanja zadataka on teče ovako. Prvo uzimamo danu sliku i učitavamo je sa dane lokacije na disku. Nakon toga tu sliku moramo pretvoriti u prethodno opisani grayscale format jer se on u većini slučajeva koristit kod baratanja sa podatcima slike. Kada imamo sliku u grayscale formatu tu sliku prije nego što damo funkciji Robertsa, Sobela, Prewitta ili Kirscha „zaglađujemo“ Gaussovim filterom kako bi eliminirali početni šum sa slike. Zaglađenu sliku zatim obrađujemo zadanim operatorima, bilo ugrađenim ili onima koje smo sami implementirali, te rješenja istih ispisujemo to jest prikazujemo i uspoređujemo kako bi vidjeli razliku između dvaju implementacija. Poanta je da bi ugrađene i naše implementacije trebale imati sličan ili identičan rezultat.

Konačno usporedili bi vremensku izvedbu implementiranih rješenja i ugrađenih rješenja. Kod ovog koraka vrlo je vjerojatno da će naše rješenje imati lošiju vremensku složenost prvenstveno iz razloga što Python nije jezik orijentiran na performanse, a funkciju implementirane unutar

OpenCv-a implementirane su većinom u C++ ponekad i u asembleru, a oboje imaju znatno veće performanse od Pythona.

2. Postupak rješavanja zadatka

2.1 Učitavanje podataka i početno filtriranje

2.1.1 Učitavanje slike, pretvorba u grayscale i prikaz

Učitavanje slika vršimo funkcijom cv2.imread() ugrađenom u openCv

```
img_directory = "<direktorij slike>"
lena_photo = cv2.imread(img_directory, cv2.IMREAD_GRAYSCALE)
```

ona vraća podatak tipa ndarrar koji je definiran unutar Numpya, to je u biti više dimenzijski niz podataka to jest možemo ga zamisliti kao n dimenzijsku matricu. Parametar cv2.IMREAD_GRAYSCALE definira kako ćemo preslikati podatke iz ulazne slike u novi ndarray. Slike kojima baratamo su generalno u boji što znači da imamo 3 kanala R, B i G koji reprezentiraju crvenu, plavu i zelenu nijansu. Svaki kanal je većinom predstavljen sa 8 bitnim „unsigned“ iliti pozitivnim integerom koji ima vrijednosti intenziteta u skupu od 0 do 255, dakle svaki piksel može imati jednu od 255^3 boja. Kada govorimo o preslikavanju iz tri kanala boje u jedan kanal funkcija cv2.IMREAD_GRAYSCALE[10] taj prijelaz računa kao:

$$dst(I) = 0.114 * src(I).B + 0.587 * src(I).G + 0.299 * src(I).R$$

Znači svaki od tri kanala pomnožen je nekom težinom, a zbroj tih težina je jednak jedinici, u našem slučaju kanal plave boje pomnožen je faktorom 0.114, kanal zelene boje sa faktorom 0.587 i kanal crvene boje sa faktorom 0.299. Kao primjer dat ćemo sljedeću sliku, originalan izgled i izgled nakon transformacije u grayscale.

Slika 3. slika u boji, prije transformacije

Slika 2. slika u grayscale formatu, nakon transformacije

2.1.2 Gaussov filter

Nakon transformacije slike u grayscale format, sliku treba zagladiti kako bi se riješili šuma koji bi nam negativno utjecao na detekciju rubova to zaglađivanje radimo uz pomoć Gaussovog filtera[8] koji je implementiran unutar OpenCv-a. Funkcija glasi ovako:

```
lena_blurred = cv2.GaussianBlur(lena_photo, (3, 3), 0, 0)
```

Mi funkciji kao parametar predajemo sliku i veličinu matrice, u našem slučaju 3x3, te standardnu devijaciju u smjeru x i y. Jedini zahtjev koji kod parametara mora biti ispunjen je da veličina matrice budu neparni brojevi i logično veći ili jednaki jedinici. Rezultati su ovakvi, iz slike 3. dobivamo:

Slika 4. slika nakon korištenja gausovog filera

2.2 Primjena Rubnih operatora

2.2.1 Robertsov, Sobelov, Prewittov operator

Robertsov operator nije implementiran unutar OpenCv-a tako da ga ne možemo direktno usporediti sa našom implementacijom, ali to i nije pretjerano bitno jer samo želimo prikazati kako ovi operatori izgledaju i kakvi su im izlazi, a ne jesu li bolji ili lošiji od ugrađene implementacije. Za početak kod izgleda ovako:

```
kernelx_roberts = np.array([[1,0],[0,-1]])
kernely_roberts = np.array([[0,-1],[1,0]])
for i in range(0, lena_blurred_padded.shape[0] - 2):
  for j in range(0, lena_blurred_padded.shape[1] - 2):
    result[i, j] = np.sum(np.multiply(v, kernelx_roberts))
```

Znači imamo ugniježđenu petlju u kojoj vanjska petlja prolazi po y osi (povećanjem parametra i, y os se pomiče u negativnom smjeru, lijevi koordinatni sustav), a unutarnja petlja po x osi (j). Kako bi to zornije ilustrirali jednostavnije rečeno naša matrica se pomiče s lijeva u desno jedan po jedan red, te obavljamo konvoluciju piksela slike na kojima se nalazimo u danom koraku i elemenata matrice. Matrica korištena kod Robertsovog operatora definirana je u uvodnom poglavlju.
Ovdje koristimo dvije funkcije uključene unutar Numpya, np.multiply i np.sum. Funkcija multiply obavlja množenje elemenata matrica na istim pozicijama, tako zvano „element-wise“ množenje, a sum obavlja sumu svih elemenata u matrici što nam daje konačan odziv. Odzivi gore prikazani su u smjeru x osi, nakon računanja ih odziva moramo još izračunati odzive u smjeru y osi uz pomoć matrice „kernely_roberts“ i ta dva odziva zbrojiti kako bi dobili konačan rezultat:

```
roberts = robertsx + robertsy
```

Kod ostalih operatora postupak je identičan sa jednom malom razlikom,

```
lena_blurred_padded[(i + 1):(i + 3), (j + 1):(j + 3)]
```

se mijenja u

```
lena_blurred_padded[i:(i + 3), j:(j + 3)]
```

radi povećanja dimenzija matrica sa 2x2 na 3x3 i naravno koristimo drugačije matrice, kako ne bi sve ponavljali u sljedeća dva reda samo ćemo navesti te matrice :

```
#sobelov kompasni rubni operator
kernelx_sobel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
kernely_sobel = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
#perwittov kompasni rubni operator
kernelx_perwitt = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
kernely_perwitt = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
```

2.2.2 Kirschov operator

Kod Kirschovog operatora postupak je identičan, ali također postoje dvije male razlike. Kirschov operator ima 8 matrica koje su zadane ovako:

```
N = np.array([[-3,-3,5],[-3,0,5],[-3,-3,5]])
NW = np.array([[-3,5,5],[-3,0,5],[-3,-3,-3]])
W = np.array([[5,5,5],[-3,0,-3],[-3,-3,-3]])
SW = np.array([[5,5,-3],[5,0,-3],[-3,-3,-3]])
S = np.array([[5,-3,-3],[5,0,-3],[5,-3,-3]])
SE = np.array([[-3,-3,-3],[5,0,-3],[5,5,-3]])
E = np.array([[-3,-3,-3],[-3,0,-3],[5,5,5]])
NE = np.array([[-3,-3,-3],[-3,0,5],[-3,5,5]])
```

Druga razlika s obzirom na prethodne operatore je da ove rezultate ne zbrajamo već tražimo maksimalan odziv medu njih osam, taj odziv nam onda daje informaciju o rubu i o orijentaciji ruba u toj točki.

3. Ispitivanje rješenja

3.1 Ispitna baza

Ispitni podatci to jest slike su uzete iz primjera koji su dani na službenom git repozitoriju OpenCv-a[7], kod možemo isprobati na bilo kojoj slici u tom repozitoriju i generalno na bilo kojoj slici, ali ja ovdje koristim samo jednu radi jednostavnosti prezentacije i pošto nema potreba za uspoređivanjem ikakvih performansi ovih operatora na različitim skupovima. Slika koja je korištena u tekstu rada je lena.jpg, dimenzije slike su 512x512 piksela i može ju se pronaći u priloženim materijalima te na git repozitoriju navedenom maloprije.

3.2 Rezultati primjene operatora i analiza rezultata

Slika 5. Robertsov operator

Slika 6. Cannyev operator

Slika7. Sobelov operator, OpenCv

Slika 7. Sobelov operator

Slika 7. Kirschov operator za matricu N

Slika 7. Prewittov operator

Slika 13. Kirschov operator za matricu NW

Slika 13. Kirschov operator za matricu W

Slika 13. Kirschov operator za matricu SW

Slika 13. Kirschov operator za matricu S

Slika 13. Kirschov operator za matricu SE

Slika 13. Kirschov operator za matricu E

Slika 14. Kirschov operator za matricu NE

Slike 5., 7., 9. i 10. su vlastite implementacije i radi usporedbe se vidi da su ručna implementacija Sobelovog operatora (slika 7.) i opencv implementacija (slika 8.) identične. Također vidimo da Canny za pragove 90 i 90*3(prema naputcima u dokumentaciji) daju jasne linije rubova, dok ostali operatori daju „zamrljane rubove“ to ne dobijemo baš crte kao što bi to željeli. Rješenje tog problema bi bilo da primijenimo prag na dobivene slike onda bi dobili uže rubove. Prikazani rezultati Robertsova, Sobelova i Prewittova operatora su dobiveni tako da su im vrijednosti dobivene prolazom matrice Gx i Gy zbrojene. Rezultati Kirschowog operatora prikazanog na slikama dobiveni su prolazom odgovarajućih matrica koje su naglašene ispod slika.

Na kraju ćemo se samo kratko osvrnuti na brzinu izvođenja ovih odsječaka koda. Svi operatori koji su ručno implementirani, u jednom prolazu, po x osi ili za matricu N (Kirsch) imaju trajanje u rangu 1.6 do 1.7 sekundi gdje Robertsov algoritam najčešće ima najkraće izvođenje što nije začuđujuće jer ipak imamo manje računskih operacija.

Naspram naših implementiranih operatora, izvođenje Sobelovog operatora koji je jedini implementiran unutar OpenCv-a traje u prosjeku 0.003 sekunde što je veoma značajna razlika, ali i kao što smo na početku rekli i očekivana.

4. Zaključak

Pogledom na ove slike i usporedbom možemo zaključiti da korištenjem Cannyevog operatora (za koji smo rekli da bi bio ground truth) dobivamo konkretne linije rubova dok kod drugih operatora uočavamo da rezultati nisu tako binarni već su rubovi više „razmazani“ no kao što smo rekli primjenom praga bi se to moglo do neke mjere riješiti. Također ako želimo informaciju o orijentaciji rubova možemo ju dobiti izračunavanjem kuta kako je opisano u uvodnom odjeljku, no tu se javlja jedan problem pošto koristimo arkus tangens ne možemo reprezentirati sve kutove. Kako bi riješili ovaj problem koristimo Kirschov operator koji može razlikovati sve kutove, ali je ograničen na samo njih osam.

5. Literatura

[1] roberts, prewitt, sobel i generalno o detekciji rubova(više teorija) - Jain Kasturi Schunck - Machine Vision (str. 140 nadalje)
https://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.htm
[2]Rafael C. Gonzales, Richard E. Woods - Digital image processing - ISBN 10: 1-292-22304-9
[3] roberts, prewitt, sobel (detaljnije o algoritmu) -- Nixon, Aguado Feature extraction and image processing
[4] nešto o kirschovom operatoru pošto se ne pojavljuje njegov opis u prethodna dva izvora- https://www.researchgate.net/publication/346055342_Image_steganography_based_on_Kirsch_edge_detection
[5] još dodatno jedan rad o kirschu -- Moving object detection based on kirsch operator combined with Optical Flow -- https://ieeexplore.ieee.org/document/5476045
[6] sobel operator i perwitt operator (prewitt se obavlja korištenjem sobel funkcije, ali sa drukčijim parametrima) --https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
[7] baza podataka - https://github.com/opencv/opencv/tree/master/samples/data
[8] OpenCv dokumentacija - https://docs.opencv.org/master/
[9] numpy dokumentacija - https://numpy.org/doc/
[10] IMREAD_GRAYSCALE (BGR2Gray) konverzija - https://docs.opencv.org/master/dc/d38/group__gapi__colorconvert.html#ga3e8fd8197ab16811caf9b31cb1e08a05