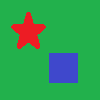
Sistemi za digitalnu obradu signala

Izvještaj

Radio: Nemanja Čenić, 1161/19

Za algoritam za detekciju ivica izabran je Laplasov algoritam. Kao pripremu za Laplasov algoritam potrebno je prvo odraditi *grayscale* slike, kao i *Gaussian* *blur*. Primjeri će u nastavku biti pisani u python kodu, radi lakšeg razumjevanja. Takodje, biće korištena sljedeća slika za primjere (dimenzija 100x100).



Slika 1.

**-Grayscale:**

Grayscale se realizuje tako što se svaka komponenta piksela pomnoži odgovarajućim brojem, zatim se saberu i iste vrijednosti dodjele RGB komponentama piksela. Računanje grayscale se, u mome slučaju, realizovalo primjenom sljedeće formule:

gray = 0.299\*R + 0.587\*G + 0.114\*B

gdje su RGB odgovarajuće red, green i blue komponente piksela. Zatim se radi sljedeće:

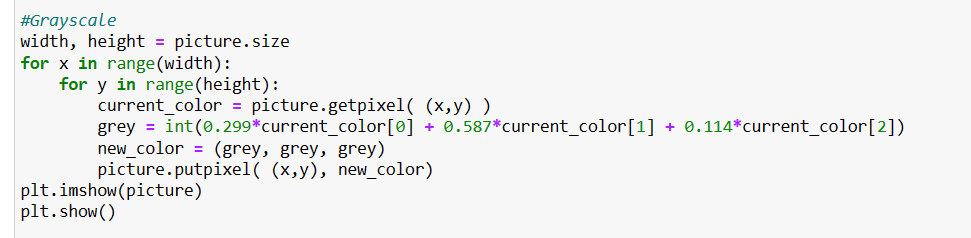
R = gray,

G = gray,

B = gray,

svakoj komponenti piksela se dodjeli ista vrijednost.

Primjer koda je dat na sljedećoj slici.

Slika 2: Grayscale

Pozivom picture.getpixel((x,y)) dobijamo piksel (current\_color) na x, y poziciji u slici. Zatim se računaju gray komponente primjenom prethodno objašnjene tehnike. Pozivom picture.putpixel((x,y), new\_color) postavljamo novi pixel (new\_color) na x, y poziciju.

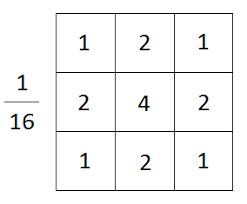
Rezultat ovog algoritma je prikazan na slici ispod.



Slika 3.

**- Gaussian blur**

Gaussian blur je algoritam za zamagljivanje slike. Potreban je da bi se uklonili šumovi sa slike, da bi Laplasov algoritam što bolje detektovao ivice. Funkcioniše tako što kernel matricu, dimenzija 3x3, prevlačimo preko slike, zatim se računaju novi pikseli. Kernel matrica za Gausov algoritam izgleda kao na sljedećoj slici:

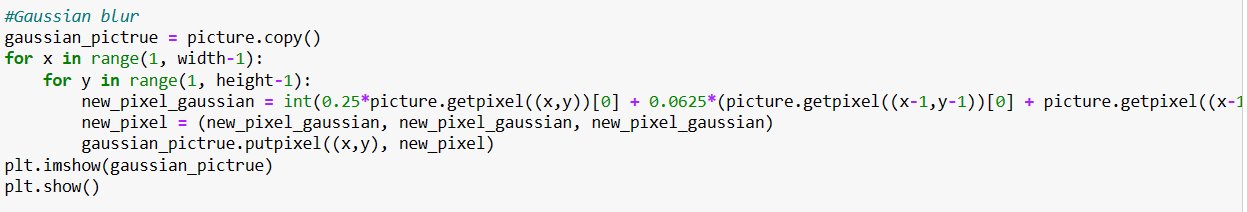


Slika 4.

Matrica sa slike 4 se prevlači preko slike i novi pikseli se računaju na sljedeći način.

new\_pixel\_gaussian = 0.25\*pixel(x,y) + 0.0625\*(pixel(x-1,y-1) + pixel(x-1,y+1) + pixel(x+1,y-1) + pixel(x+1,y+1)) + 0.125\*(pixel(x,y-1) + pixel(x,y+1) + picture.getpixel((x-1,y))[0] + pixel(x+1,y))

Gdje pixel(x,y) označava pixel na x, y poziciji na slici. Zatim se taj novi piksel smješta u piksel koji se nalazi u središtu matrice. Primjer koda je dat na sljedećoj slici.

Slika 5.

Kod za računanje novog piksela nije uspio stati na sliku 5 ali prati logiku koja je data prethodnom jednačinom.

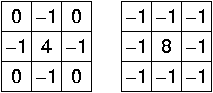
Rezultat ovog algoritma je dat na sljedećoj slici.



Slika 6.

**- Laplasov algoritam**

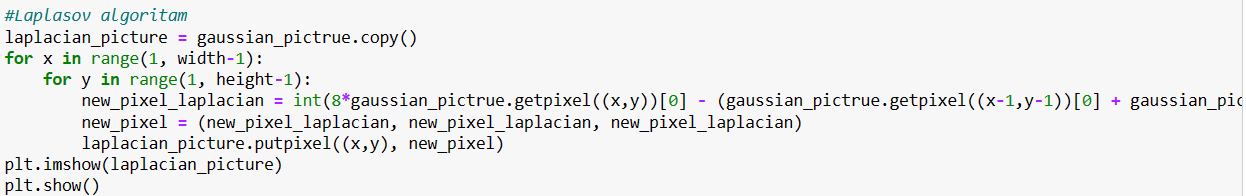
Laplasov algoritam radi na sličan način kao i Gaussian blur. Razlika je u korištenoj kernel matrici. Postoje 2 vrste kernel matrice za ovaj algoritam i prikazane su na sljedećoj slici.



Slika 7.

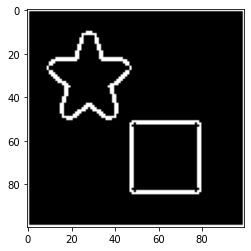
Kernel matrica koja je korištena je desna matrica na slici 7.

Primjer koda Laplasovog algoritma je dat na sljedećoj slici.

Slika 8.

Takodje, kod za računanje novog piksela nije uspio stati na sliku 8.

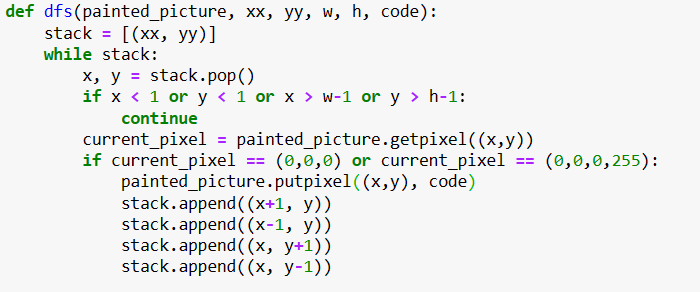
Rezultat ovog algoritma je dat na sljedećoj slici.



Slika 9.

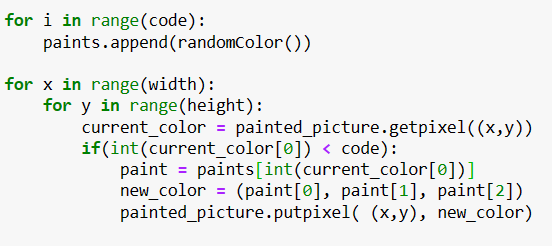
Nakon Laplasovog algoritma za detekciju ivica slijedi kodovanje i bojenje piksela. Pikseli koje se nalaze u istoj konturi se koduju istim kodom, dok se pikseli sa istim kodom boje sa istom bojom. Izabrani algoritam za bojenje slike je Flood Fill, koji koristi DFS algoritam. DFS algoritam se poziva rekurzivno sve dok se svi pikseli u konturi ne koduju istim kodom.

Primjer koda za DFS algoritam je dat na sljedećoj slici:

Slika 10.

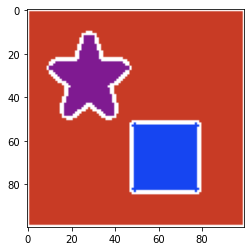
Pikseli se prvo boje u jednom pravcu, pa ukoliko se naiđe na ivicu, krene se bojati u drugom smijeru (ivica je piksel svjetlije boje). U stek se smještaju pikseli iste konture i koduju se sa vrijednosti „code“. Slika izgleda indentično kao i slika nakon detekcije ivica (slika 9).

Nakon kodovanja slike slijedi bojenje. Primjer koda za bojenje slike dat je na slici ispod:



Slika 11.

U listi paints nalazi se „code-1“ slučajno izabranih boja. Zatim se selektuje odgovarajuća boja tako što trenutni piksel odgovara indeksu boje u toj listi (tako će pikseli sa kodom 1 dobiti prvu boju u listi, itd...). Rezultat ovog algoritma je dat na sljedećoj slici.



Slika 12.

**Performanse**

Prvo ćemo uporediti perfomanse izvršavanja koda u Python-u i u C programskom jeziku (koji se izvršava na računaru, ne na ADSP ploči). Slika je ista kao iz primjera (slika 1), kao i rezultati algoritama.

|  |  |  |
| --- | --- | --- |
|  | Python | C |
| Grayscale | 0.237 s | 0.001 s |
| Detekcija ivica | 1.074 s | 0.023 s |
| Kodovanje | 0.139 s | 0.050 s |
| Bojenje | 0.219 s | 0.001 s |

Ogromna je razlika u brzini izvršavanja.

**- Perfomanse bez optimizacije na ADSP ploči**

|  |  |
| --- | --- |
| Grayscale | 1\_413\_210 |
| Detekcija ivica | 23\_604\_096 |
| Kodovanje | 144\_895\_957 |
| Bojenje | 3\_007\_604 |

Brzina izvršavanja programa je 2 min.

**- Perfomanse sa optimizacijama na ADSP ploči**

|  |  |
| --- | --- |
| Grayscale | 965\_782 |
| Detekcija ivica | 18\_624\_648 |
| Kodovanje | 144\_270\_828 |
| Bojenje | 2\_535\_477 |

Za optimizaciju korištene su kompalerske optimizacije -O(100%), kao i odgovarajuće pragma direktive (#pragma SIMD\_for).

Za kodovanje malo je rezultat neočekivan i možda ne realan, jer u poređenju sa učitavanjem slike, koji traje nešto manje od milion ciklusa, kodovanje bi se trebalo 4 puta brže izvršavati, tako da ne znam objasniti taj rezultat.

Ukoliko pogledamo ostale rezultate vidimo da se primjenom kompajlerskih optimizacija dobijaju bolji rezultati.

Ukoliko uporedimo ta vremena izvršavanja i vremena izvršavanja na računaru, ovo je znatno sporije (zapravo ni ne možemo uporediti).