Pretvaranje fotografije u bojanku

Barišić, Renata

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

Sveučilišni studij

PRETVARANJE FOTOGRAFIJE U BOJANKU

Završni rad

Renata Barišić

Osijek, 2022.

SADRŽAJ

1. UVOD

U ovom završnom radu obradit će se problem pretvorbe fotografije u bojanku pomoću tehnologije detekcije rubova. Za početak, u drugom poglavlju, objasnit će se pojam detekcije rubova te navesti najsuvremenije metode provedbe te tehnologije. Zatim će, u trećem poglavlju, biti predstavljen programski jezik *Python* te korištena biblioteka – *OpenCV*. Treće se poglavlje sastoji od tri potpoglavlja. U prvom će potpoglavlju biti objašnjene i uspoređene različite metode detekcije rubova te će se izabrati najprikladnija vrsta za ovaj zadatak. U drugom potpoglavlju će biti objašnjeno kako i zašto je potrebno pripremiti fotografiju prije primjene algoritma detekcije rubova – jer se detekcija rubova se obavlja na zamućenoj fotografiji crno-bijele boje. U trećem potpoglavlju bit će navedeno kako poboljšati rad algoritma uz korištenje *trackbara* i kako povećati detektirane rubove te obrnuti crne i bijele boje na fotografiji kako bi se dobio klasičan izgled bojanke. Četvrto poglavlje prikazivat će eksperimentalne rezultate napisanog koda te će se sastojati od tri potpoglavlja. U prvom će se potpoglavlju usporediti rezultati primjene različitih algoritama detekcije rubova. U drugom potpoglavlju bit će prikazan i objašnjen rad *trackbara*, te kako njegova upotreba utječe na rezultate detekcije. U trećem i posljednjem potpoglavlju usporedit će se rezultat detekcije na različitim fotografijama te odrediti kako vrsta fotografije utječe na parametre koje predajemo funkciji za detekciju.

1.1. Zadatak završnog rada

U ovom završnom radu zadatak je proučiti *OpenCV* biblioteku, s naglaskom na dio o detekciji rubova. Zatim je potrebno detektirane rubove fotografije naglasiti, odnosno povećati, te maknuti boje iz fotografije kako bi nastala crno-bijela bojanka. Za rješavanje navedenog zadatka koristit će se *Python* te *OpenCV* biblioteka.

2. DETEKCIJA RUBOVA

Detekcija rubova je tehnika obrade fotografija kojom se prepoznaju rubovi, odnosno granice između različitih područja na fotografiji. Rubovi su zapravo samo nagle promjene u intenzitetu piksela unutar fotografije. Oni se detektiraju pronalaskom takvih naglih promjena među susjednim pikselima fotografije. U nastavku ovog poglavlja prikazat će se najsuvremenije arhitekture, metode i načini detekcije rubova.

DexiNed (engl. *Dense Extreme Inception Network for Edge Detection*) je arhitektura detekcije rubova koja se koristi strojnim učenjem te koja uči detektirati rubove bez potrebe za unaprijed uvježbanim parametrima, prema [1]. *DexiNed* nadmašuje ostale algoritme kada se u svima primjenjuje isti skup podataka. Ona se također dobro generalizira na nove skupove podataka bez ikakvog finog podešavanja. Uz to, rubovi koje ona detektira su vidljivo oštriji i tanji spram većine drugih arhitektura detekcije rubova.

RCF (engl. *Richer Convolutional Features*) je arhitektura koja za točnu detekciju rubova koristi bogatije konvolucijske značajke, kao što ime predlaže, te konvolucijske neuronske mreže, prema [2]. Ona enkapsulira sve semantičke značajke i značajke sitnih detalja jer koristi sve konvolucijske značajke za predviđanje gdje će se rubovi nalaziti. *RCF* arhitektura je točna i učinkovita, pa je zbog toga primjenjiva u drugim područjima računalnog vida, poput segmentacije slike.

BDCN (engl. *Bi-Directional Cascade Network*) je arhitektura detekcije rubova koja sliku rastavlja na slojeve, prema [3], te se svaki sloj promatra zasebno umjesto da se promatraju kao cjelina. Na ovaj se način na svaki sloj primjenjuju različiti parametri pri prepoznavanju rubova jer se slojevi razlikuju po intenzitetu. Intenzitet piksela se dakle gleda na razini sloja, a ne na razini cjelokupne slike, što daje preciznije rezultate. Ovakva arhitektura je primjenjiva u područjima segmentacije slike, procjene optičkog toka te prepoznavanja objekata.

CATS (engl. *Context-Aware Tracing Strategy*) je strategija za oštru detekciju rubova s detektorima dubokih rubova, prema [4]. Temelji se na dubokom učenju i ne-miješanju konvolucijskih značajki. Ona je jednostavna, ali učinkovita te rješava problem nejasnoća lokalizacije za moderne detektore dubokih rubova i omogućuje im postizanje oštrih rubova iz neobrađenih predviđanja rubova, uz znatno poboljšanu preciznost lokalizacije.

Na slici 2.1. mogu se vidjeti rezultati detekcije rubova navedenih arhitektura treniranih na istim fotografijama iz *BIPED* skupa podataka.

Slika 2.1. *Rezultati detekcije rubova najsuvremenijih arhitektura treniranih na BIPED skupu podataka [1]*

Također je bitno spomenuti kako se metode detekcije rubova mogu implementirati i hardverski. Dosad navedene metode se uglavnom implementiraju softverski i zbog toga su sporije, prema [5]. Za povećanje brzine obrade i rezultate u stvarnom vremenu potrebna je namjenska hardverska implementacija. Za to je prikladna *FPGA* (engl. *Field Programmable Gate Array*) platforma. [6] potvrđuje navedenu pretpostavku i pokazuje na primjeru *Sobel Edge Detection* metode kako je ona brža na *FPGA* implementaciji, nego u svojim *C* i *C++* inačicama. Unatoč tomu, u ovom će se radu problem pretvorbe fotografije u bojanku riješiti softverski.

Najsuvremenije softverske metode uglavnom se oslanjaju na duboko učenje. No postoje brojne metode koje ne koriste duboko učenje, a daju zadovoljavajuće rezultate – poput *Canny* i *Sobel*

Edge Detection metoda koje će se detaljnije obraditi u nastavku ovog rada. U većini slučajeva se za detekciju rubova ne koristi duboko učenje jer su klasične metode, poput navedenih, puno brže, skoro jednako precizne, razumljivije i jednostavnije pri uklanjanju pogrešaka. Također, iako u ovom radu detektirani rubovi predstavljaju i konačni rezultat – bojanku, u većini je slučajeva detekcija rubova samo međukorak prije daljnje obrade podataka, npr. segmentacije slike, prepoznavanja objekata ili ekstrakcije podataka, što je još jedan od razloga zašto korak detekcije rubova nije potrebno kompliciratи upotrebom strojnog učenja. Zato će se u ovom radu riješiti problem pretvorbe fotografije u bojanku bez uporabe strojnog učenja.

3. DETEKCIJA RUBOVA U *PYTHON*-U I *OPENCV*-U

Za rješavanje problema ovog završnog rada koristit će se programski jezik *Python*. *Python* je programski jezik visoke razine i opće namjene. On podržava više paradigmi programiranja, što uključuje strukturirano – osobito proceduralno, objektno-orijentirano te funkcionalno programiranje. *Python* dan danas rangira kao jedan od najpopularnijih programskih jezika i to ne bez razloga. Prema [7], on je kvalitetan, produktivan, portabilan, modularan i ugodan za korištenje. *Python* se smatra kvalitetnim zbog jednostavne čitljivosti programskog koda, produktivan je jer smanjuje količinu napisanog koda potrebnog za postizanje rezultata, spram nekih drugih programskih jezika, portabilan je jer se program pisan u *Pythonu* može pokrenuti neovisno o operacijskom sustavu, a njegova modularnost se postiže korištenjem raznih biblioteka programskih funkcija koje se mogu iskoristiti za proširenje funkcionalnosti. Baš zbog toga je *Python* idealan jezik za rješenje zadatka ovog rada – koristit će se sve prednosti ovog programskog jezika te proširenje funkcionalnosti bibliotekom programskih funkcija *OpenCV*.

OpenCV (engl. *Open Source Computer Vision Library*) je biblioteka koja sadrži programske funkcije većinom vezane za strojno učenje i računalni vid u stvarnom vremenu. Prema [8], ona sadrži preko 2500 optimiziranih algoritama što uključuje opsežan skup klasičnih i najsuvremenijih algoritama računalnog vida i strojnog učenja. U ovom će se radu koristiti dva glavna algoritma te brojne pomoćne funkcije sadržane u biblioteci. *OpenCV* biblioteka je višeplatformska – podržava *Windows, Linux, Android* i *Mac OS* operacijske sustave, i besplatna je za korištenje pod otvorenim kodom. Izvorno je napisana u programskom jeziku *C++*, no postoje sučelja i za programske jezike *C, Python, Java, Matlab* i *Octave*. Kao što je već navedeno, u ovom će se radu koristiti *Python* inačica biblioteke.

3.1. Odabir algoritma detekcije

U *OpenCV* biblioteci, koja je korištena u ovom radu, postoje dva algoritma za detekciju rubova – *Sobel Edge Detection* i *Canny Edge Detection*. U nastavku ovog poglavlja, opisat će se i usporediti dva navedena algoritma te odabrati prikladniji za rješenje problema kojim se ovaj rad bavi.

Sobel Edge Detection algoritam je jedan od najpoznatijih algoritama za detekciju rubova fotografije. Ovaj algoritam detektira rubove koji su obilježeni naglim promjenama intenziteta piksela. Ako se intenzitet piksela prikaže kao funkcija, te se gleda prva derivacija funkcije intenziteta, porast intenziteta postaje još vidljiviji. Dakle, rubovi se mogu detektirati u područjima gdje je gradijent veći od određene granične vrijednosti. Osim toga, nagla promjena derivacije otkrit

će i promjenu intenziteta piksela. Derivacija se može aproksimirati koristeći jezgru dimenzija 3×3. Koristi se jedna jezgra za otkrivanje naglih promjena intenziteta piksela u smjeru x-osi – horizontalna jezgra, a druga u smjeru y-osi – vertikalna jezgra. Detekcija rubova ovim algoritmom može se provesti u oba smjera – koristeći obje jezgre, ili u jednom smjeru koristeći samo jednu od navedenih jezgri – tada dobivamo konačne slike u kojima su naglašeni samo horizontalni, odnosno samo vertikalni rubovi. Ova će funkcionalnost biti prikazana u četvrtom poglavlju.

Canny Edge Detection algoritam je također jedan od najpoznatijih algoritama za detekciju rubova. Rad ovog algoritma može se podijeliti u tri koraka:

1. izračunavanje gradijenta intenziteta slike
2. suzbijanje lažnih rubova
3. određivanje praga histereze.

U prvom se koraku slika predaje *Sobel Edge Detection* algoritmu pomoću kojeg se računa veličina gradijenta intenziteta slike te smjer svakog piksela. Smjer gradijenta se zaokružuje na najbliži kut od 45 stupnjeva. Koristi se varijanta *Sobel Edge Detection* algoritma u oba smjera.

U drugom koraku algoritam koristi tehniku zvanu ne-maksimalno potiskivanje rubova kako bi filtrirao neželjene piksele – one koji zapravo ne predstavljaju rub. To se postiže tako što se svaki piksel uspoređuje sa svojim susjednim pikselima, u pozitivnom i negativnom smjeru gradijenta. Ako je veličina gradijenta trenutnog piksela veća od susjednih piksela, ona ostaje nepromijenjena. U suprotnom se ona postavlja na nulu. Sada je broj rubova puno manji jer se više ne uzimaju u obzir brojni „lažni“ rubovi.

U trećem i posljednjem koraku algoritma, veličine gradijenata intenziteta se uspoređuju s dvije granične vrijednosti – donjom i gornjom graničnom vrijednosti, koje dodatno filtriraju koji će se rubovi uzeti u obzir. Donja granična vrijednost označava veličinu gradijenta intenziteta ispod koje se rubovi neće uzimati u obzir. Gornja granična vrijednost označava veličinu gradijenta intenziteta iznad koje se rubovi smatraju „snažnima“ te se svakako uzimaju u obzir. Svi pikseli sa veličinom gradijenta intenziteta između donje i gornje granične vrijednosti pripadaju „slabim“ rubovima te će se oni uzeti u obzir jedino ako su povezani s nekim od „snažnih“ rubova.

Pozivanjem funkcije *Canny()* programu se predaju samo izvorna slika te donja i gornja granična vrijednost. Sve navedene korake program sam obavlja u pozadini te se korisnik ne mora brinuti o njima.

Iz opisanog je vidljivo kako je *Canny Edge Detection* bolji izbor za rješenje zadanog problema, jer on unutar sebe koristi *Sobel Edge Detection* algoritam uz dodatna poboljšanja.

3.2. Priprema fotografije

Prije same upotrebe algoritma, potrebno je pripremiti fotografiju kako bi se mogao primijeniti odabrani algoritam na nju, odnosno kako bi se postigao željeni rezultat. Fotografiju je prvo potrebno učitati, a to se postiže funkcijom *imread()* sadržanom u biblioteci *OpenCV*. Funkciji *imread()* predaje se ime datoteke, odnosno relativnu putanju do datoteke ako se ona ne nalazi u istoj mapi kao i program. Najbolji način za unos imena datoteke jest pomoću funkcije *input()*. Na taj način svakim pokretanjem programa može se unijeti drugo ime datoteke jer se podatci ne ugrađuju izravno u izvorni kod programa.

Slijedeći korak je zamutiti fotografiju. Ovaj je korak nužan za pravilan rad programa i urednije rezultate jer zamućenje fotografije izglađuje njezine piksele, što čini promjene intenziteta među pikselima znatno manjim, te se stoga ne detektiraju rubovi tamo gdje nije potrebno. U *OpenCV* biblioteci postoje razne funkcije za zamućivanje fotografija, a za ovaj rad odabrana je *GaussianBlur()*. Njoj se osim izvorne datoteke predaju veličina jezgre koja će se koristiti, odnosno broj susjednih piksela koji će se uzeti u obzir.

Osim toga potrebno je i ukloniti boje iz fotografije, odnosno učiniti ju crno-bijelom. Razlog tomu je što uklanjanje boja iz fotografije pojednostavljuje algoritam i smanjuje računske zahtjeve, prema [9]. Uklanjanjem boja iz fotografije potrebno je uspoređivati samo intenzitet piksela istih boja, a ne i njihovu boju. Osim toga, detekcija rubova na fotografijama crno-bijele boje je dobro istraženo područje, dok detekcija rubova obojenih fotografija, osim što je slabije istražena, daje i lošije rezultate, prema [10]. Naime, i uz različite algoritme uklanjanja boje, i uz različite algoritme detekcije rubova, detekcija rubova na crno-bijelim fotografijama je uvijek davala bolje rezultate nego detekcija rubova na obojenim fotografijama. Uklanjanje boje iz fotografije vrši se funkcijom *cvtColor()*. Nakon ovih koraka fotografija je spremna za primjenu odabranog algoritma.

3.3. Usavršavanje rada

Nakon odabira željenog algoritma detekcije rubova i pripreme fotografije za rad može se primijeniti algoritam na fotografiju. No, postoji još par detalja na koje treba obratiti pozornost kako bi se dobio optimalan rezultat.

Za početak, kao što je već navedeno, funkciji *Canny()* potrebno je, uz fotografiju, predati donju i gornju graničnu vrijednost veličine gradijenta intenziteta. Te vrijednosti se u pravilu ne mogu

unaprijed znati, niti će one imati isti iznos za sve fotografije pa ih nema smisla upisivati ili na bilo koji način izračunavati u kodu. Njih je potrebno naknadno odrediti nakon što je vidljiv rezultat primjene algoritma na fotografiju, a to će se postići ugradnjom *trackbara* u program.

Trackbar je također značajka biblioteke *OpenCV*. To je traka za podešavanje vrijednosti koja se koristi kao dinamičan način unosa podataka. Korištenjem ove značajke postavlja se neku vrijednost na željenu pomicanjem oznake na traci, bez potrebe za unosom podataka sa tipkovnice. Postoje slučajevi kada je potrebno pokrenuti program s različitom vrijednošću ili parametrom i eksperimentirati kako bi se pronašla optimalna vrijednost parametra. Korištenje *trackbara* zamjenjuje uređivanje vrijednosti varijable u izvornom kodu i ponavljano pokretanje programa, te unos podataka s tipkovnice koji također može biti spor i zamoran ako postoji veći broj varijabli čiju je vrijednost potrebno mijenjati. *Trackbar* može ili pozvati funkciju ili ažurirati vrijednost varijable na temelju položaja oznake na traci. Vidljivo je da je *trackbar* savršeno rješenje za navedeni problem.

Kako bi se dodao *trackbar* u program, prvo je potrebno stvoriti prozor na kojem će se on nalaziti. Prozor se stvara funkcijom *namedWindow()* kojoj se predaje proizvoljno ime prozora. *Trackbar* se stvara funkcijom *createTrackbar()* kojoj se predaju: proizvoljan naziv, prozor na kojem će se nalaziti, minimalna i maksimalna vrijednost koja se može postići te funkcija koja se poziva svakim pomakom oznake na traci. Za rješenje problema unošenja donje i gornje granične vrijednosti potrebna su dva *trackbara*. Također je potrebno provjeravati poziciju oznake na obje trake. One se provjeravaju funkcijom *getTrackbarPos()* te spremaju u varijable koje se zatim predaju funkciji *Canny()*. Ovaj se postupak ponavlja u beskonačnoj petlji sve dok nije postignut zadovoljavajuć rezultat, a rezultat se ažurira u stvarnom vremenu, odnosno na ekranu je moguće vidjeti promjene na fotografiji u istom trenutku kada je pomaknuta oznaka na traci. Na slici 3.1. prikazan je izgled dva *trackbara* korištena u programu, vidljiv je naziv prozora „*Canny Edge Detection*“ te nazivi oba *trackbara* – „*lower*“ predstavlja donju, a „*upper*“ gornju graničnu vrijednost veličine gradijenta intenziteta. Također se mogu vidjeti i minimalna i maksimalna vrijednost na koju se one mogu postaviti te one iznose 0 i 255.

Slika 3.1. *Izgled trackbarova*

Postoje još dva problema koja je potrebno riješiti, a to su podebljavanje rubova i invertiranje boja. Rubovi se podebljavaju funkcijom *dilate()* kojoj je uz željenu fotografiju potrebno predati proizvoljnu jezgru pomoću koje će se podebljati rubovi te broj iteracija podebljavanja. U *OpenCV* biblioteci postoje razni ugrađeni oblici jezgre, kao što su pravokutna, eliptična te jezgra u obliku križa. U ovom će se slučaju za podebljavanje rubova koristiti eliptična jezgra jer daje najuredniji izgled rubova. Eliptična jezgra se stvara predajom elementa MORPH_ELLIPSE i proizvoljne veličine jezgre funkciji *getStructuringElement().* Nakon što su rubovi podebljani na željenu veličinu, boja se jednostavno invertira funkcijom *bitwise_not()* na željenoj fotografiji. Na ovaj se način postigao željeni izgled klasične bojanke.

4. EKSPERIMENTALNI REZULTATI

U ovom će se poglavlju testirati rad napisanog programa na konkretnim fotografijama. U prvom će potpoglavlju biti uspoređeni *Sobel* i *Canny Edge Detection* algoritmi, te posebno tri varijante *Sobel Edge Detection* algoritma, iz čega će biti vidljivo zašto je *Canny Edge Detection* algoritam odabran za ostatak rada. U drugom će se potpoglavlju uz pomoć jedne fotografije i *Canny Edge Detection* algoritma uspoređivati rezultat pri promjeni donje i gornje granične vrijednosti veličine gradijenta intenziteta pomicanjem oznake na *trackbaru*. U trećem i posljednjem će potpoglavlju biti primijenjen *Canny Edge Detection* algoritam na šest testnih fotografija kako bi se otkrilo koje vrijednosti donje i gornje granične vrijednosti veličine gradijenta intenziteta su potrebne kod različitih vrsta fotografija. Koriste se ekstremni slučajevi kako bi razlika bila zamjetnija: koristit će se tri fotografije prirode – to su fotografije visoke frekvencije s puno sitnih detalja, te tri jednostavna crteža s niskom frekvencijom, odnosno s velikim jednobojnim plohama i malo detalja.

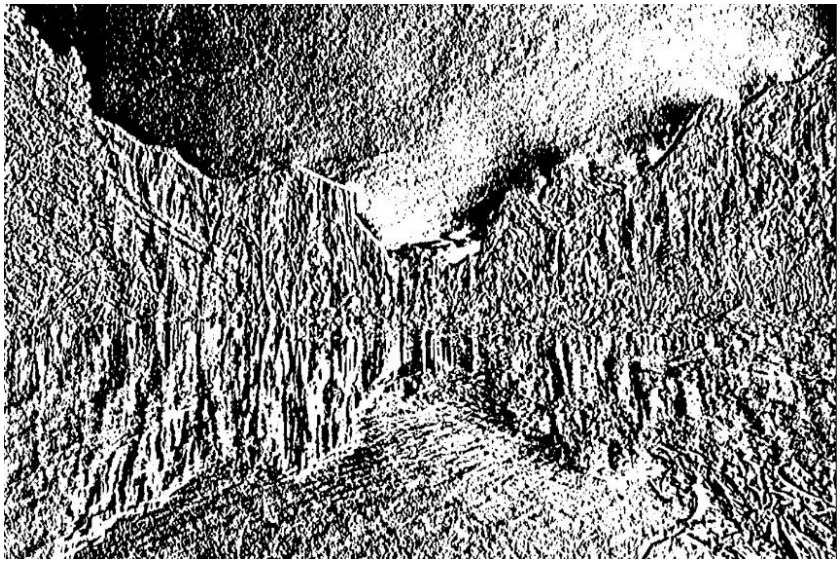
4.1. Usporedba različitih algoritama detekcije rubova

Kao što je navedeno u trećem poglavlju ovog rada, u *OpenCV* biblioteci postoje dva algoritma za detekciju rubova – *Sobel* i *Canny Edge Detection*, te *Sobel Edge Detection* uz to ima i tri različite varijante. U nastavku ovog poglavlja bit će isprobana sva četiri algoritma na istoj fotografiji, prikazanoj na slici 4.1.

Slika 4.1. *Fotografija jezera [12]*

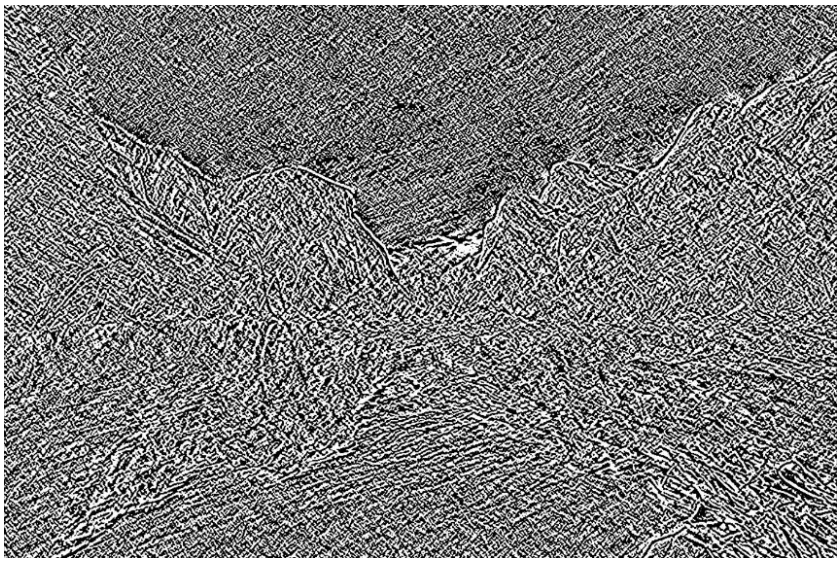
Slike 4.2.-4.5. prikazivat će rezultate navedenih algoritama. U sva tri slučaja koja se koriste *Sobel Edge Detection* algoritmom, funkcija će biti pozvana s istim parametrima osim što će se u prvom

slučaju koristiti samo horizontalna jezgra, u drugom slučaju samo vertikalna jezgra, a u trećem slučaju koristit će se obje jezgre.

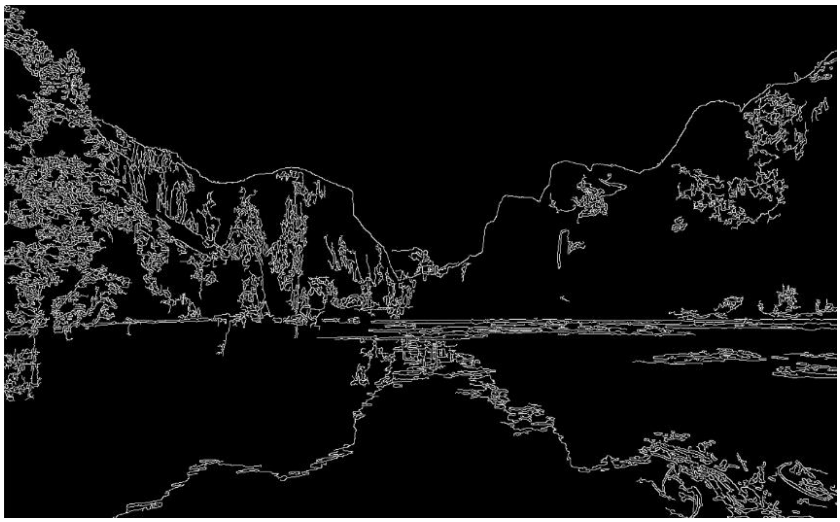


Slika 4.2. *Fotografija jezera nakon primjene Sobel Edge Detection algoritma u smjeru x-osi*

Slika 4.3. *Fotografija jezera nakon primjene Sobel Edge Detection algoritma u smjeru y-osi*



Slika 4.4. *Fotografija jezera nakon primjene Sobel Edge Detection algoritma u smjeru obje osi*



Slika 4.5. *Fotografija jezera nakon primjene Canny Edge Detection algoritma*

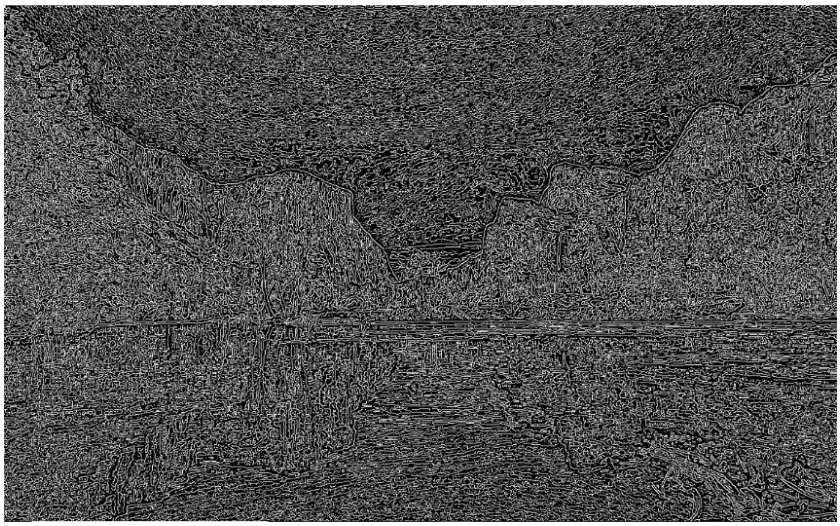
Prikazane slike potvrđuju funkcionalnost opisanu u trećem poglavlju – na slici 4.2. je vidljivo kako su naglašeni samo vertikalni rubovi kada se koristi jezgra u smjeru x-osi. Isto tako, na slici 4.3. je vidljivo kako su naglašeni samo horizontalni rubovi kada je korištena jezgra u smjeru y-osi. Na slici 4.4 prikazani su svi rubovi pošto su korištene obje jezgre. Na slici 4.5. su također prikazani svi rubovi pošto *Canny Edge Detection* algoritam unutar sebe koristi *Sobel Edge Detection* algoritam u oba smjera, samo što su u ovom slučaju ti rubovi daljnje profiltrirani te se prikazuju samo oni koji čine glavne rubove fotografije. Rezultati su potvrdili sva predviđanja iz trećeg

poglavlja, te je iz njih vidljivo zašto je *Canny Edge Detection* algoritam prikladniji za rješavanje problema ovog završnog rada od *Sobel Edge Detection* algoritma.

4.2. Rad *trackbara*

U ovom potpoglavlju koristit će se slika iz prethodnog potpoglavlja – slika 4.1., te će se na njoj prikazati utjecaj promjene graničnih vrijednosti veličine gradijenta intenziteta korištenjem *trackbarova* na konačnu fotografiju.

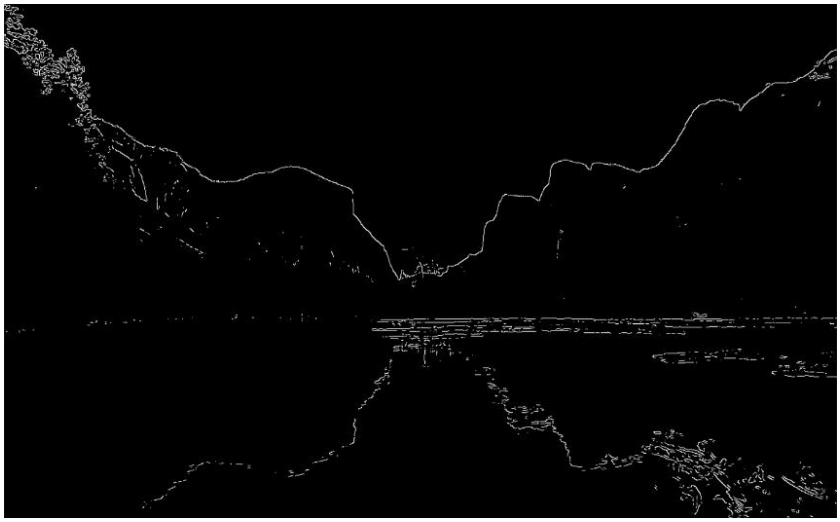
U trećem poglavlju je rečeno kako donja granična vrijednost označava veličinu gradijenta intenziteta ispod koje se rubovi neće uzimati u obzir, a gornja granična vrijednost označava veličinu gradijenta intenziteta iznad koje se rubovi smatraju „snažnima“ te se svakako uzimaju u obzir. Kada se program pokrene, obje granične vrijednosti su u početnom trenutku u nuli. Kada je donja granična vrijednost u nuli, to znači da nema rubova koji se neće uzimati u obzir. A kada je gornja granična vrijednost u nuli, to znači da se svi rubovi smatraju „snažnima“, te se svi uzimaju u obzir za konačnu sliku. Rezultat *Canny()* funkcije s parametrima 0,0 prikazuje sve rubove fotografije te se može vidjeti na slici 4.6.



Slika 4.6. *Fotografija jezera nakon primjene Canny Edge Detection algoritma s parametrima 0,0*

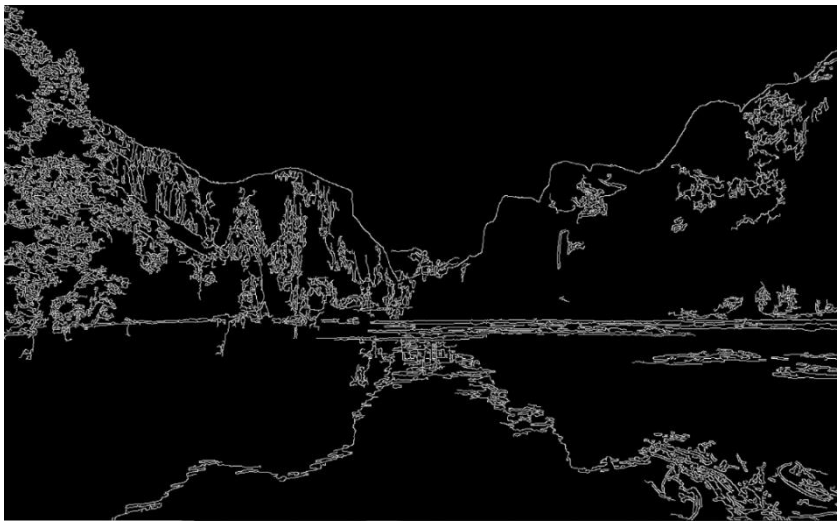
Maksimalna vrijednost graničnih vrijednosti veličine gradijenta intenziteta je 255. Kada se obje granične vrijednosti postave na maksimalnu, dobiva se rezultat prikazan slikom 4.7. Vidljivo je kako je broj rubova znatno manji jer se sada ne uzimaju u obzir rubovi s veličinom gradijenta intenziteta manjom od 255, te se smatraju „snažnima“ jedino ako je njihova vrijednost iznosom jednaka maksimalnoj. Pošto su obje granične vrijednosti iste iznosom, kao i u prvom slučaju, ne

postoji raspon vrijednosti između donje i gornje granične vrijednosti u kojom bi se mogli nalaziti „slabi“ rubovi.



Slika 4.7. *Fotografija jezera nakon primjene Canny Edge Detection algoritma s parametrima 255,255*

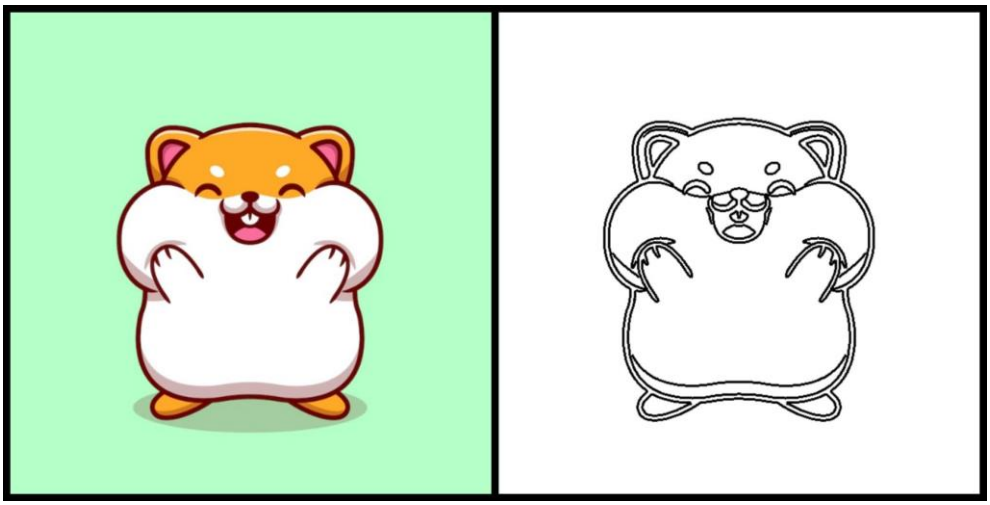
Ono što je potrebno napraviti za optimalan rezultat je postupno mijenjati vrijednosti na *trackbarovima* obraćajući pozornost na to kako se mijenja rezultat. Svi pikseli sa veličinom gradijenta intenziteta između donje i gornje granične vrijednosti pripadaju „slabim“ rubovima te će se oni uzeti u obzir jedino ako su povezani s nekim od „snažnih“ rubova.



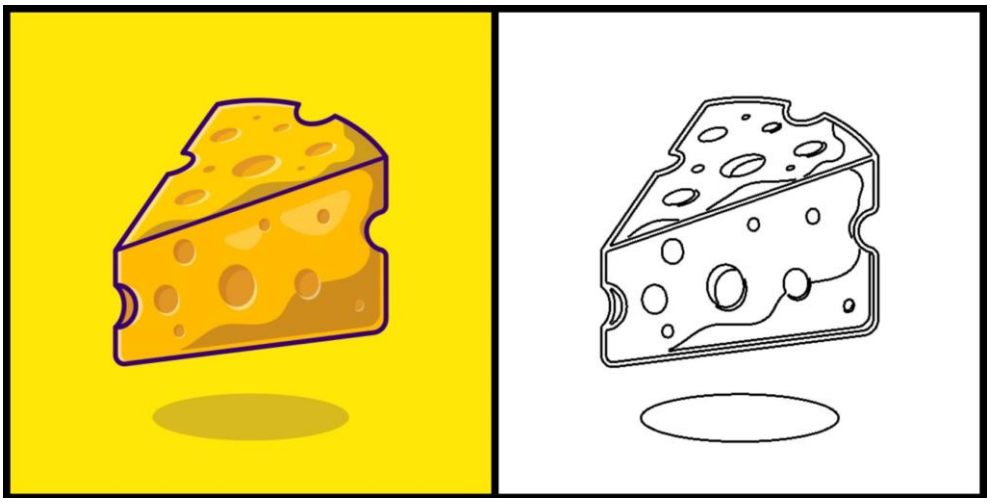
Slika 4.8. *Fotografija jezera nakon primjene Canny Edge Detection algoritma s parametrima 10, 245*

4.3. Usporedba različitih tipova fotografija

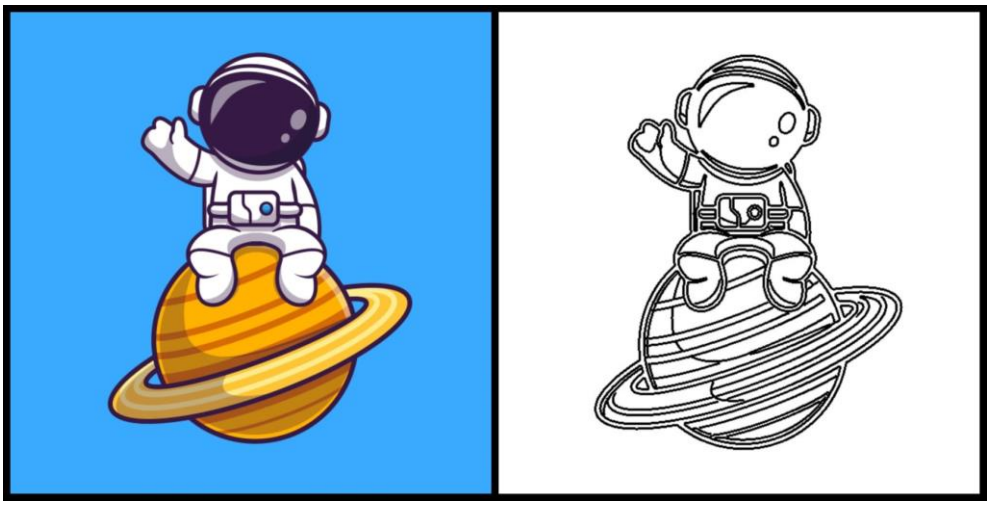
U ovom će se potpoglavlju prikazati razlike u radu algoritma na različitim vrstama fotografija. Kao rezultat bit će prikazane bojanke nakon obavljanja svih koraka programa, odnosno nakon podebljavanja rubova i invertiranja boja, za razliku od prethodna dva potpoglavlja gdje su bili prikazani rezultati funkcija detekcija rubova, a ne cjelokupnog programa. Slike 4.9.-4.11. prikazuju rezultate programa na jednostavnim crtežima, dok slike 4.12.-4.14. prikazuju rezultate programa na fotografijama prirode.



Slika 4.9. *Crtež hrčka [13] prije i poslije primjene algoritma Canny Edge Detection*

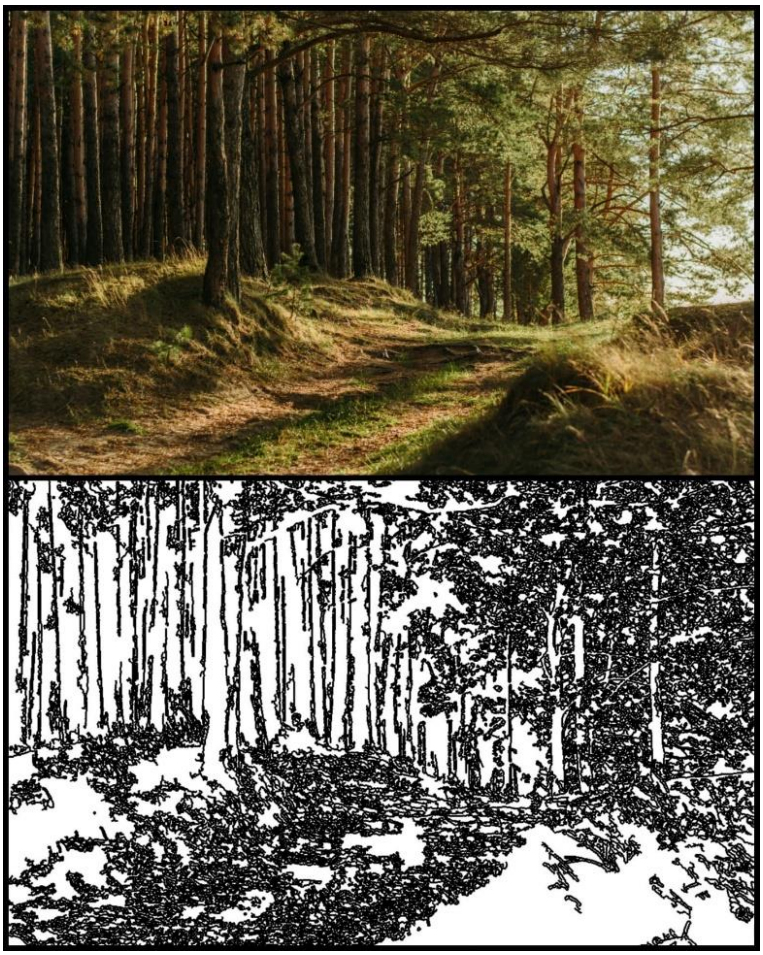


Slika 4.10. *Crtež sira [14] prije i poslije primjene algoritma Canny Edge Detection*



Slika 4.11. *Crtež astronauta [15] prije i poslije primjene algoritma Canny Edge Detection*

Slika 4.12. *Fotografija stabla [16] prije i poslije primjene algoritma Canny Edge Detection*



Slika 4.13. *Fotografija šume [17] prije i poslije primjene algoritma Canny Edge Detection*

Slika 4.14. *Fotografija ceste [18] prije i poslije primjene algoritma Canny Edge Detection*

Tablica 4.1. *Vrijednosti gornjeg i donjeg praga korištene pri primjeni Canny Edge Detection algoritma na slikama 4.9.-4.14.*

Fotografija	Vrijednost donjeg praga	Vrijednost gornjeg praga
Slika 4.9.	15	176
Slika 4.10.	7	140
Slika 4.11.	8	138
Slika 4.12.	0	255
Slika 4.13.	0	255
Slika 4.14.	0	255

Iz tablice 4.1. se da primijetiti kako su vrijednosti donjeg i gornjeg praga kod prve tri fotografije, odnosno kod jednostavnih crteža, vrlo niski. Prag ispod kojeg se rubovi ne uzimaju u obzir vrlo rano eliminira nepotrebni višak rubova, odnosno „lažne“ rubove. Gornji prag također nije potrebno

visoko postaviti jer su svi preostali rubovi u pravilu iste veličine intenziteta gradijenta. Povećavanje gornjih pragova na vrijednosti veće od navedenih skoro pa ni ne mijenja rezultate, ali njegovim namještanjem može se odrediti razina detalja koju se želi zadržati odnosno izbaciti iz konačne bojanke – što je iznos gornjeg praga veći, razina detalja je manja.

Kod druge tri fotografije, odnosno kod fotografija prirode, situacija je drugačija. Vidljivo je kako ovi rezultati nisu toliko precizni kao prethodni – nazire se oblik glavnih dijelova fotografija, no rubovi nisu toliko točni i uredni, a razlog tomu je to što fotografije prirode imaju puno više detalja. Kada bi čovjek uzeo detektirati rubove na ovim fotografijama, rezultati bi bili drugačiji, a i precizniji, sve zbog ljudskog predznanja o objektima koji se pojavljuju na fotografijama. Čovjek bi znao raspoznati što je rub objekta, a što npr. sjena, što računalo ne zna. Računalo samo prepoznaje one piksele s velikim razlikama u intenzitetu i ne zna odrediti što je drvo, što list, a što sjena ili samo nekakva razlika u bojama. Najidealniji rezultati za ovakve fotografije su se pokazali kada je donja granica na nuli, a gornja na 255. Tada konačnu sliku čine samo „najsnažniji“ rubovi te „slabi“ rubovi koji ih popunjavaju. Pošto je donja granica na nuli, ne postoje rubovi koji se nipošto ne uzimaju u obzir, već se svi rubovi s veličinom gradijenta intenziteta manjom od 255 smatraju „slabim“ te se koriste samo ako nadopunjavaju „snažne“ rubove.

5. ZAKLJUČAK

U ovom je radu napravljena pretvorba fotografije u bojanku koristeći tehnologiju detekcije rubova. Za početak su predstavljene najaktualnije metode rješenja navedenog problema. Zatim su objašnjene konkretne tehnologije koje se koriste u radu – programski jezik *Python* i biblioteka *OpenCV*. Uspoređena su dva algoritma detekcije rubova iz *OpenCV* biblioteke – *Sobel Edge Detection* i *Canny Edge Detection*, te je *Canny Edge Detection* izabran kao prikladniji. Objašnjeno je kako se i zašto fotografija mora pripremiti prije primjene algoritma detekcije, odnosno kako joj se moraju ukloniti boje te kako se mora zamutiti da bi se dobili najoptimalniji rezultati. Program je usavršen dodavanjem značajke *trackbar* koji omogućuje dinamičan način unosa podataka, umjesto uređivanja vrijednosti varijabli u izvornom kodu i ponavljanog pokretanja programa ili unosa podataka s tipkovnice. Nakon primjene algoritma i ugađanja rezultata pomoću *trackbara*, rad je dovršen podebljavanjem detektiranih rubova te invertiranjem boja kako bi se dobio klasičan izgled bojanke, odnosno bijela pozadina sa crnim rubovima. Rad programa testiran je na nekolicini fotografija, a eksperimentalni rezultati potvrdili su sve teorijske pretpostavke navedene u radu. Naravno, napisani program nije savršen te je u eksperimentalnim rezultatima uočen prostor za napredak. Rad napisanog programa mogao bi se poboljšati korištenjem strojnog učenja gdje bi računalo, osim uspoređivanja piksela, učilo raspoznavati objekte na fotografiji, pa bi tada rezultati bili sličniji onom što ljudsko oko vidi.

LITERATURA

[1] X. Soria, A. Sappa, P. Humanante i A. Arbarinia, „Dense Extreme Inception Network for Edge Detection“, *Elsevier*, pro. 2021, doi:10.48550/arXiv.2112.02250.

[2] Y. Liu, MM. Cheng, X. Hu, K. Wang i X. Bai, „Richer Convolutional Features for Edge Detection“, *IEEE TPAMI*, kol. 2019, doi:10.48550/arXiv.1612.02103

[3] J. He, S. Zhang, M. Yang, Y. Shan i T. Huang, „BCDN: Bi-Directional Cascade Network for Perceptual Edge Detection“, *IEEE TPAMI*, sij. 2022, doi:10.1109/TPAMI.2020.3007074

[4] L. Huan, N. Xue, X. Zheng, W. He, J. Gong i GS. Xia, „Unmixing Convolutional Features for Crisp Edge Detection“, *IEEE TPAMI*, lip. 2021, doi:10.48550/arXiv.2011.09808

[6] L. Sharma i P. Dahiya, „State-of-the-Art Image Edge Detection Techniques“, *The IUP Journal of Telecommunications*, svi. 2019, https://ssrn.com/abstract=3383898

[6] Z. Gho, W. Xu i Z. Chai, „Image Edge Detection Based on FPGA“, *IEEE*, ruj. 2010, doi: 10.1109/DCABES.2010.39

[7] T. Šantić, „Programski jezik Python“, 2017., urn:nbn:hr:200:374473

[8] About, OpenCV, 2022, dostupno na: https://opencv.org/about/ [15.6.2022.]

[9] C. Kanan i G. W. Cottrell „Color-to-Grayscale: Does the Method Matter in Image Recognition?“, *PLOS ONE*, sij. 2012, doi:10.1371/journal.pone.0029740.

[10] I. Ahmad, I. Moon i S. J. Shin, „Color-to-grayscale algorithms effect on edge detection — A comparative study“, *IEEE*, tra. 2018, doi:10.23919/ELINFOCOM.2018.8330719.

[11] Edge Detection Using OpenCV, Big Vision LLC, 2022, dostupno na: https://learnopencv.com/edge-detection-using-opencv/ [13.6.2022.]

[12] S. Leonardi, Yosemite fire falls, Unsplash, 2022, dostupno na: https://unsplash.com/photos/_2mLg07Yn6s [13.6.2022.]

[13] Catalyststuff, Cute hamster holding the cheek cartoon illustration, Freepik, dostupno na: https://www.freepik.com/free-vector/cute-hamster-holding-cheek-cartoon-illustration_13037994.htm#query=cartoon&position=22&from_view=search [15.6.2022.]

[14] Catalyststuff, Cheese cartoon icon illustration, Freepik, dostupno na: https://www.freepik.com/free-vector/cheese-cartoon-icon-illustration_12567364.htm#query=cartoon%20food&position=49&from_view=search [15.6.2022.]

[15] Catalyststuff, Astronaut sitting on planet and waving hand cartoon vector icon illustration, Freepik, dostupno na: https://www.freepik.com/free-vector/astronaut-sitting-planet-waving-hand-cartoon-vector-icon-illustration-science-technology-icon-concept-isolated-premium-vector-flat-cartoon-style_16953182.htm?query=cartoon [15.6.2022,]

[16] Veeterzy, Tree in forest of plants, Unsplash, 2016, dostupno na: https://unsplash.com/photos/sMQiL_2v4vs [15.6.2022.]

[17] I. Iriser, Unsplash, 2018, dostupno na: https://unsplash.com/photos/2Y4dE8sdhlc [15.6.2022.]

[18] J. Towner, Beam of light on a forest road, Unsplash, 2016, dostupno na: https://unsplash.com/photos/3Kv48NS4WUU [15.6.2022.]

PRETVARANJE FOTOGRAFIJE U BOJANKU

SAŽETAK

Cilj ovog završnog rada bio je uz *OpenCV* biblioteku napisati program u programskom jeziku *Python* koji predanu fotografiju pretvara u bojanku koristeći tehnologiju detekcije rubova. Za rješenje navedenog problema korišten je ugrađeni algoritam *Canny Edge Detection* te razne pomoćne *Python* i *OpenCV* funkcije za poboljšanje rada navedenog algoritma. Uz pomoć tih funkcija bilo je potrebno naglasiti, odnosno povećati detektirane rubove fotografije, te maknuti boje iz nje i invertirati ih kako bi nastala crno-bijela bojanka. Pomoćne funkcije su također omogućile dodavanje *trackbara* u program kako bi se omogućio dinamičan način unosa podataka, odnosno kako bi se izbjeglo uređivanje vrijednosti varijabli u izvornom kodu i ponavljano pokretanje programa, te unos podataka s tipkovnice. Rad algoritma uspoređen je s jednostavnijim, *Sobel Edge Detection* algoritmom, te je uspoređen i na različitim vrstama fotografija. Uspješnost algoritma ovisila je o vrsti fotografije koja mu je predana, iz čega se da zaključiti da ova tehnologija ima još prostora za napredak.

Ključne riječi: bojanka, detekcija rubova, *OpenCV*, *Python*

CONVERTING A PHOTOGRAPH INTO A COLORING BOOK

ABSTRACT

The goal of this paper was to write a program in the Python programming language using the OpenCV library, which turns a submitted photograph into a coloring book using edge detection technology. To solve this problem, the built-in Canny Edge Detection algorithm was used, as well as various auxiliary Python and OpenCV functions to improve the performance of this algorithm. With the help of these functions, the detected edges of the photograph had to be emphasized, i.e., increased, and the colors from it had to be removed and inverted to create a black and white coloring book. Auxiliary functions also allowed the addition of a trackbar to the program to enable entering data in a dynamic way, i.e., to avoid editing the values of variables in the source code and re-running the program and entering data from the keyboard. The performance of the algorithm was compared with the simpler, Sobel Edge Detection algorithm, and was also compared on different types of photographs. The success of the algorithm depended on the type of photograph submitted to it, from which it can be concluded that this technology still has room for improvement.

Key words: coloring book, edge detection, OpenCV, Python

ŽIVOTOPIS

Autor ovog završnog rada, Renata Barišić, rođena je 7.3.2001. u Osijeku. U Osijeku je pohađala osnovnu školu „Mladost“ te srednju školu – III. gimnaziju Osijek. Za vrijeme pisanja ovog rada, studira na trećoj godini preddiplomskog sveučilišnog studija računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.

_____

Potpis autora