Pronalaženje i označavanje kontura objektima na slici

Požgaj, Mislav

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

ZAVRŠNI RAD

Mislav Požgaj

Zagreb, 2022.

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

ZAVRŠNI RAD

Voditelj rada:
Dr. sc. Tomislav Stipančić, dipl. ing.

Student:
Mislav Požgaj

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tokom studija i navedenu literaturu.

Zahvaljujem se mentoru dr.sc. Tomislavu Stipančiću na mentorstvu i savjetima tokom izrade završnog rada.

Također, zahvaljujem se obitelji i prijateljima na podršci tijekom cijelog preddiplomskog studija.

-Mislav Požgaj

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove: **proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika**

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Mislav Požgaj** JMBAG: **0035213874**

Naslov rada na hrvatskom jeziku: **Pronalaženje i označavanje kontura objektima na slici**

Naslov rada na engleskom jeziku: **Finding and marking the contours of objects in an image**

Opis zadatka:

Pronalaženje kontura (rubova) objekata na slikama važan je početni korak kod mnogih zadaća računalnog vida. Između ostaloga, konture kao granice objekata predstavljaju temelj za pronalaženje, lokalizaciju, segmentaciju i praćenje.

U radu je potrebno:

- objasniti značenje kontura i rubova kao nositelja informacija na slikama,
- odrediti i objasniti korake koji vode ka računalnom prepoznavanju kontura objektima na slici,
- implementirati korake za prepoznavanje kontura u računalni program,
- usporediti nekoliko metoda za određivanje kontura,
- analizirati dobre i loše strane odabrane metode.

Razvijeno programsko rješenje je potrebno temeljiti na OpenCV biblioteci implementiranoj kroz Python programski jezik. Model je potrebno eksperimentalno evaluirati koristeći vizijski sustav u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

Popis oznaka:

Oznaka:	Jedinica:	Opis:
g	-	- Vrijednost piksela
t	s	-Vrijeme izvršavanja programa
x	-	- Varijabla koordinate
y	-	- Varijabla koordinate
G	-	- Gaussova funkcija filtera
$\mathbf{G}_{(i,j)}$	-	- Gradijent rubova
$\mathbf{G}_{x(i,j)}$	-	- Gradijent rubova u horizontalnom smjeru
$\mathbf{G}_{y(i,j)}$	-	- Gradijent rubova u horizontalnom smjeru
$\mathbf{I}$	-	-Matrica slike kod Sobelovog operatora
P_1	%	- Visoki prag
P_2	%	- Niski prag
T	-	- Prag pozadine i objekta
σ	-	- Standardna devijacija
θ	◦	- Kut smjera gradijenta

SAŽETAK

Računalni vid predstavlja jedno od najbrže rastućih područja umjetne inteligencije. Temelj računalnog vida predstavlja prepoznavanje kontura objekata sa slike. Cilj ovog rada je predstaviti dva najčešće korištena algoritma za prepoznavanje kontura, a to su Cannyjev algoritam i thresholding metoda. Kroz rad dana je teorijska osnova oba algoritma te je izrađena programska aplikacija za usporedbu navedenih metoda. Aplikacija je izrađena pomoću Python programskog jezika i OpenCV biblioteke. Usporedba metoda vrši se na temelju broja pronađenih kontura i vremenu potrebnom da se program izvrši. Uz to, analizirane su prednosti i nedostatci odabranih metoda.

Ključne riječi: pronalaženje kontura, Canny algoritam, thresholding, OpenCV

SUMMARY

Computer vision is one of the fastest growing areas of artificial intelligence. The basis of computer vision is the recognition of object contours in images. The aim of this paper is to present the two most commonly used contour recognition algorithms, namely the Canny algorithm and the thresholding method. The paper outlines the theoretical basis of both algorithms and the development of a software application for comparing the above methods. The application was created using the Python programming language and the OpenCV library. The methods are compared based on the number of contours found and the time required to execute the program. Additionally, the advantages and disadvantages of the selected methods are analysed.

Key words: contour detection, Canny algorithm, thresholding, OpenCV

1. UVOD

Vid predstavlja glavno osjetilo ljudi preko kojeg dobivamo većinu informacija iz okoline. Već u šezdesetim godinama prošlog stoljeća javlja se ideja implementacije vida u računala. Tako nastaje jedna od disciplina umjetne inteligencije pod nazivom računalni vid. Kako bi računala bila autonomna nije dovoljno da samo vide sliku, nego da su sposobna izvući sve potrebne informacije iz nje. Razvojem računalnog vida, uz primjenu standardne računalne opreme i programa, omogućeno je računalima da obavljaju jednostavnije, ali i sve više složenih zadataka. Ljudi su svakodnevno okruženi proizvodima i tehnologijama koji dobivenih uz pomoć računalnog vida, kao što su otključavanje pametnih uređaja pomoću prepoznavanja lica ili pomoću otiska prsta, detekcija pokreta u sigurnosnim sustavima, praćenje proizvodnih procesa i još mnogo drugih. Temelj većine nabrojenih zadataka je prepoznavanje kontura sa slika i video unosa što će u ovom radu biti obrađeno te će biti pružen primjer aplikacije koja prepoznaje konture sa unesenih slika pomoću dvije različite metode pripreme slike.

2. TEORIJSKA OSNOVA DETEKCIJE KONTURA

2.1. Računalni vid

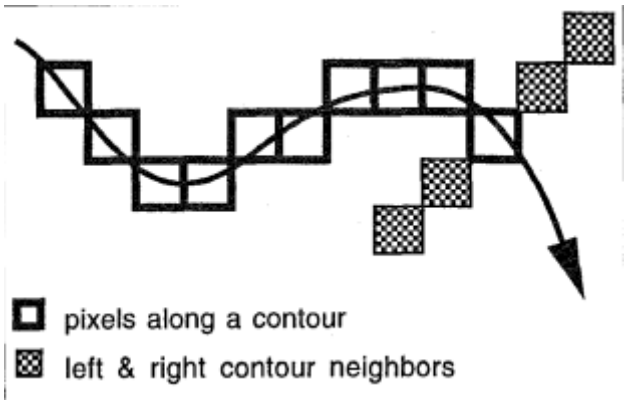
Računalni vid jedna je od disciplina umjetne inteligencije. Da bi postigli razvijen sustav, nije dovoljno da računalo vidi sliku nego da ima sposobnost izvući sve bitne informacije iz nje. Iz tog razloga računalni vid se često povezuje sa strojnim učenjem kako bi računalo moglo razumjeti ono što vidi. Postoje dva stajališta s kojih se promatra računalni vid. Prvo stajalište je biološko te je njemu cilj rekonstruirati ljudski vidni sustav na računalima s ciljem boljeg razumijevanja navedenog sustava. Drugo stajalište je inženjersko, a ono predstavlja želju inženjeru da se vidni sustav čovjeka rekreira na računalu s ciljem davanja mogućnosti računalu da obavlja samostalno zadatke koje može i čovjek. Oba stajališta usko su povezana jer biološki pristup daje inženjerima ideju kako konstruirati algoritam za računala, a često, dobiveni algoritam daje biološkom pristupu dodatna saznanja kako vidni sustav ljudi funkcionira.[1.]

Ideja računalnog vida javlja se već u šezdesetim godinama prošlog stoljeća, a prva primjena se javlja u sedamdesetim godinama. Ocem računalnog vida smatra se Larry Roberts koji obrađuje temu mogućnosti dobivanja trodimenzionalnih geometrijskih informacija iz dvodimenzionalnih slika blokova. Kasnije mnogi znanstvenici otkrivaju potrebu uzimanja slika iz stvarnog svijeta te se tada javlja mogućnost jednostavnih zadataka kao što su detekcija kontura i segmentacija slika. Od tada razvoj računalnog vida uključuje mnoge algoritme koji omogućuju širok spektar primjena radi čega je usko povezan s ostalim područjima kao što su obrada slika, određivanje poza objekata u 3D-u i računalna grafika.

Računalni vid predstavlja izrazito zanimljivo, ali i komplicirano područje istraživanja. Razvoj tehnologije omogućuje veće mogućnosti računalnog vida, ali se i dalje suočava sa jednim glavnim problemom. Problem je usporedba računalnog vida sa ljudskim vidnim sustavom. Razlika između njih je najviše očita u primjeru prepoznavanja lica. Ljudi mogu pamtiti ogromnu količinu lica i prepoznati ih u uvjetima lošeg osvjetljenja, bez obzira na izraz lica i kut gledanja. Računala imaju problem sa svakim od tih uvjetima te im je potrebna velika količina procesorske i memorijske snage da to naprave, pogotovo u stvarnom vremenu.

2.2. Definicija kontura

Koncept kontura nema određenu matematičku formulaciju nego proizlazi iz iskustva ljudi te kao takva predstavlja granicu i obrub oblika određenog objekta. Cilj pronalaženja i označavanje kontura je ekstrakcija krivulja koje predstavljaju oblik objekta. Uz pojam kontura javljaju se još dva koncepta: koncept granica i koncept rubova.[2.] Granice su u sklopu računalnog vida definirane kao linije na slici koje predstavljaju oblik objekta kod kojih dolazi do promjene vlasništva nad pikselima od jedne površine ili objekta ka drugoj.[3.] Koncept te ideje prikazana je na Slika 1.



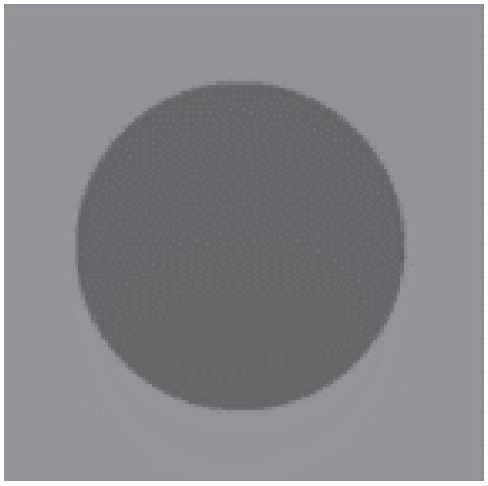
Slika 1. Granica konture i susjednih površina [4.]

Često se granice direktno povezuju sa konturama s ciljem podjele na zanimljive regije unutar slike, ali algoritmi detekcije kontura ne mogu garantirani da će oni producirati zatvorene granice. To se događa kad je koncept granice i konture sličan, ali ne i isti.

Koncept rubova predstavlja nižu razinu razlikovanja koja se temelji na karakteristikama slike. Najbitnije karakteristike na kojima se temelji prepoznavanje rubova su svjetlina piksela i njihova boja. Kod ovakvog pristupa detekcija rubova predstavlja prvi korak ka detekciji kontura.

Postoje četiri klasična pristupa detekciji kontura, a to su: detekcija na temelju razlike osvjetljenja, detekcija na temelju razlika tekstura, detekcija na temelju percepcijske grupacije i detekcija na temelju iluzorne konture.[3.] Slika 2. prikazuje nam prepoznavanje kontura na temelju različitih

osvjetljenja pojedinih dijelova i ona proizlazi iz jasnih granica između dvije različito osvjetljene plohe.



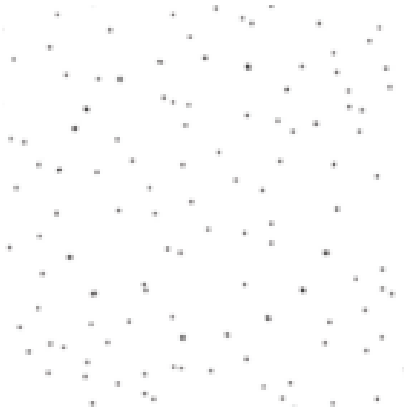
Slika 2. Prepoznavanje kontura na temelju različitog osvjetljenja [2.]

Na Slika 3. možemo vidjeti drugi koncept prepoznavanja kontura i to na temelju različitih tekstura dva različita objekta. Kao i kod prvog primjera i ovdje su granice i konture različitih objekata jasno vidljive te ih čak i preko jednostavnijeg algoritma pronalaženja kontura možemo pronaći.

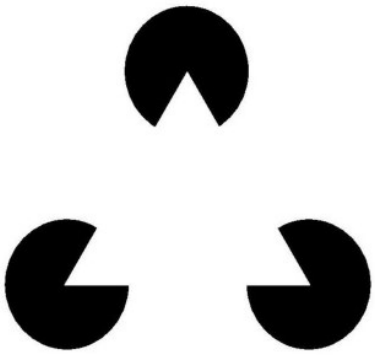
Slika 3. Prepoznavanje kontura na temelju različite teksture [2.]

Slika 4. i Slika 5. nam prikazuju preostala dva pristupa detekciji kontura. Slika 4. prikazuje nam detekciju na temelju percepcijske grupacije, a Slika 5. prikazuje detekciju na temelju iluzorne granice. Ljudskom oku lako je prepoznati gdje bi se kontura trebala iscrtati dok računalo nema tu mogućnost. Oba pristupa su mnogo kompleksnija od prethodna dva te se detekcija kontura temelji na globalnom konceptu koji se dobiva iz slike. Takav način prepoznavanja kontura je mnogo

složeniji i zahtjeva puno razrađenije algoritme pronalaženja kontura te se uz to koriste i algoritmi dubokog učenja i umjetne inteligencije.



Slika 4. Detekcija na temelju percepcijske grupacije[2.]



Slika 5. Detekcija na temelju iluzorne konture [5.]

U ovom radu biti će obrađen način detekcija kontura na temelju promatranja piksela i lokalnom prepoznavanju kontura na temelju prepoznavanja rubova i granica za čije će se potrebe koristiti Canny algoritam i metoda thresholdinga uz pomoć Pythona i OpenCV-a.

2.3. Važnost i primjena detekcija kontura

Konture i rubovi predstavljaju jedne od glavnih nositelja informacija na slikama kod računalnog vida. Pomoću njih na slici prepoznajemo objekte te ih po potrebi možemo izdvojiti, dobiti informacije o veličini objekata i njihovom položaju. Količina dobivenih informacija pomoću kontura i rubova omogućuje nam izradu mnogo zanimljivih i korisnih aplikacija. Neke od tih aplikacija su detekcija pokreta, prepoznavanje ostavljenih predmeta, segmentacija pozadine i prvog plana te prepoznavanje lica. Detekcija pokreta detektira promjene položaja praćenog objekta naspram okoline. Najveću primjenu dobiva u sklopu autonomnih sigurnosnih sustava, praćenju prometa i detekciju objekata tokom sportskih događanja.[6.] Prepoznavanje ostavljenih predmeta razvija se u zadnjih nekoliko godina. Njegova važnost je najviše vidljiva u sigurnosnim sustavima jer se svaki ostavljeni predmet smatra opasnim. Temelji se na detekciji konture predmeta te označavanju istog s ciljem izdvajanja iz okoline.[7.] Segmentacija pozadine se koristi kada je potrebno zamijeniti pozadinu sa nekom drugom te je pristup sa detekcijom kontura jedan od najjednostavnijih za te potrebe. Tokom prepoznavanja lica dobivanje kontura predstavlja prvi korak u razvoju algoritma. Osim navedenih primjena aplikacija detekcija kontura vidljiva je u strojarskoj industriji gdje se s ciljem automatizacije implementira u različite procese i sustave.

2.4. Korištene aplikacije i resursi

2.4.1. Python programski jezik

Python je interpretativan, interaktivan i objektno orijentiran programski jezik. Python predstavlja programski jezik koji služi za rješavanje velikog niza različitih problema. Interpretativan je što znači da se kod izvršava direktno pomoću interpretera pa radi toga nema potrebe za kompiliranjem prije izvršavanja. Python se koristi modulima, iznimkama i dinamičkim povezivanjem. Dolazi sa velikom bazom podataka koja uključuje područja poput obrade nizova podataka, internetske protokole, programsko inženjerstvo i primjenu sučelja operativnih sustava.[8.]

U ovom radu korišten je Python 3.9. Razlog je to što je besplatan, ima jednostavnu sintaksu i pruža veliki pristup modulima otvorenog koda. Pristup modulima otvorenog koda glavni je razlog te nam omogućuje pristup mnogim modulima koji se koriste pri strojnom učenju te je radi toga i radi mogućnosti integracije s drugim programskim jezicima jedan od najkorištenijih programskih jezika za strojno učenje.

2.4.2. OpenCV

OpenCV (Open Source Computer Vision) je biblioteka otvorenog tipa koja se koristi za računalni vid i strojno učenje. OpenCV je jedna od najrasprostranjenijih biblioteka za strojno učenje i računalni vid te je razvijena s ciljem pružanja zajedničke infrastrukture za računalni vid. Sadrži 2500 algoritama koji uključuju klasične, ali i napredne algoritme za računalni vid i strojno učenje.[9.] Za ovaj rad odabran je OpenCV radi izrazito velike funkcionalnosti i mogućnosti za prepoznavanje kontura. Posjeduje sve potrebne funkcije za detekciju kontura na slikama koje će biti detaljnije objašnjene u nadolazećem poglavlju 4. Analiza koda.

2.4.3. Ostali korišteni moduli

2.4.3.1. NumPy

NumPy je modul otvorenog koda koji omogućuje numeričke operacije i rad s poljima unutar Pythona. Odabran je za ovaj rad zbog jednostavnosti korištenja i mogućnosti prikaza slika i drugih binarnih tokova kao polja realnih brojeva.

2.4.3.2. Tkinter

Tkinter predstavlja standardno grafičko korisničko sučelje (GUI). Korišten je za izradu jednostavnog grafičkog sučelja aplikacije za detekciju kontura pomoću OpenCV-a te za olakšani grafički prikaz rezultata.

2.4.3.3. Pillow

Pillow (Python Imaging Library) je biblioteka otvorenog tipa za Python koji omogućuje otvaranje i rad sa slikama unutar Pythona. U ovom radu korišten je za grafičko prikazivanje rezultata dobivenih detekcijom kontura pomoću OpenCV-a.

2.4.3.4. TimeIt

Modul TimeIt koristimo radi mjerenja vremena potrebnog za izvršavanje koda. Pomoću njega možemo uspoređivati vremena potrebna za izvršavanje različitih metoda prepoznavanja kontura.

3. PRONALAŽENJE KONTURA I ALGORITMI

Segmentacija je proces dijeljenja digitalne slike na segmente tj. setove piksela. Cilj segmentacije je pojednostaviti prikaz slike radi lakšeg analiziranja. Segmentacija predstavlja temelj pronalaženja kontura. Postoje mnoge metode odvajanja objekta kojeg promatramo od pozadine, ali u ovom radu će biti obrađene dvije: metoda pomoću Cannyjevog algoritma i metoda thresholdinga.

3.1. Cannyjev algoritam

Cannyjev algoritam predstavlja jedan od najpoznatijih i najviše korištenih algoritama detekcije rubova. Predstavljen je 1986. godine u radu J. Cannya koji je uz razvijeni algoritam napisao i teorijsku osnovu detekcije rubova u kojoj je objasnio zašto algoritam radi. Cannyjev algoritam smatra se nadmoćnim nad ostalim algoritmima detekcija rubova radi njegove visoke standardne preciznosti, velikog omjera ulaznih signala prema šumu slike i radi boljih performansi u sistemima koji rade u stvarnom vremenu.[10.]

Canny se temelji na tri temeljna kriterija performansi:

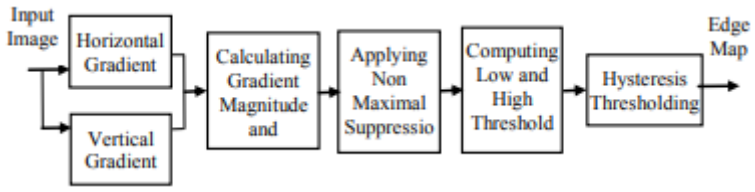
1.) Dobra detekcija. Mora postojati mala mogućnost da se na slici ne označe stvarni rubovi i mala mogućnost da se označe lažni rubovi. To nam govori da tijekom algoritma moramo maksimizirati SNR (*eng. signal-to-noise ratio*).[11.]

2.) Dobra lokalizacija. Rubovi koji su označeni pomoću algoritma moraju biti što bliži stvarnim rubovima[11.].

3.) Mora postojati samo jedan odaziv po rubu. Ovaj uvjet je zapravo uključen u prvi uvjet, ali radi matematičkog pojašnjenja napisan je posebno.[11.]

3.1.1. Koraci Cannyjevog algoritma

Cannyjev algoritam sastoji se od pet koraka: uklanjanje šuma pomoću Gaussovog filtera, računanje gradijenta slike, primjena ne-maksimalnog suzbijanja, izračun visokog i niskog praga i određivanje praga histerezom.



Slika 6. Grafički prikaz redoslijeda Cannyjevog algoritma[10.]

Uklanjanje šuma pomoću Gaussovog filtera važan je korak detekcije rubova jer je detekcija rubova osjetljiva na šum. Pomoću Gaussovog filtera uklanjamo neželjene informacije sa slike i reduciramo šum. Time dobivamo masku koju slažemo na originalnu sliku. Sama maska nosi puno manje informacija od originalne slike pa je time i algoritam brži.[10.] Gaussova funkcija $G(x, y)$ može se opisati slijedećom jednadžbom:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left[\frac{x^2+y^2}{2\sigma^2}\right] \tag{1}$$

Pri čemu su x i y varijable dok σ predstavlja standardnu devijaciju.[12.]

Računanje gradijenta slike slijedeći je korak te se pomoću gradijenta slike određuje snaga rubova. Gradijent predstavlja usmjerenu promjenu intenziteta elementa slike u horizontalnom odnosno x smjeru i vertikalnom odnosno y smjeru. Ako matricu elemenata ulazne slike označimo sa $\mathbf{I}$, a operator konvolucije sa * dobijemo slijedeće izraze za gradijente. Gradijent u horizontalnom smjeru označen je sa $\mathbf{G}_x$ te njegov izraz glasi:

$$\mathbf{G}_{x(i,j)} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{I}_{(i,j)} \tag{2}$$

Gradijent u vertikalnom smjeru označen je sa $\mathbf{G}_y$ te njegov izraz glasi:

$$\mathbf{G}_{y(i,j)} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{I}_{(i,j)} \tag{3}$$

Rezultat u svakoj točki na slici dobiva se slijedećom jednadžbom.

$$\mathbf{G}_{(i,j)} = \sqrt{\mathbf{G}_{x(i,j)}{}^2 + \mathbf{G}_{y(i,j)}{}^2} \tag{4}$$

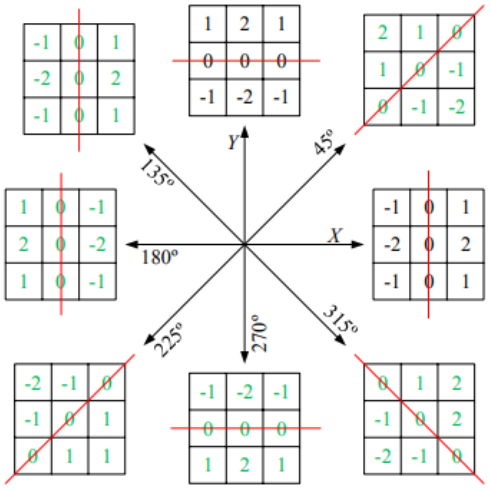
Ili jednostavnijim izrazom:

$$\mathbf{G}_{(i,j)} = \left|\mathbf{G}_{x(i,j)}\right| + \left|\mathbf{G}_{y(i,j)}\right| \tag{5}$$

Pozitivan rezultat nam ukazuje na prelazak iz tamnijeg područja u svijetlije ako sliku gledamo po redovima sa lijeva na desno i po stupcima od gore prema dolje dok nam negativan rezultat ukazuje na prelazak iz svjetlijeg područja u tamno. Time se određuje prelazak iz područja većeg u područje manjeg intenziteta i suprotno. Još je jedino potrebno odrediti kut θ u jednom od mogućih smjerova susjeda promatranog piksela. To radimo pomoću slijedećeg izraza.

$$\theta(x,y) = \arctan\left[\frac{\mathbf{G}_{y(i,j)}}{\mathbf{G}_{x(i,j)}}\right] \tag{6}$$

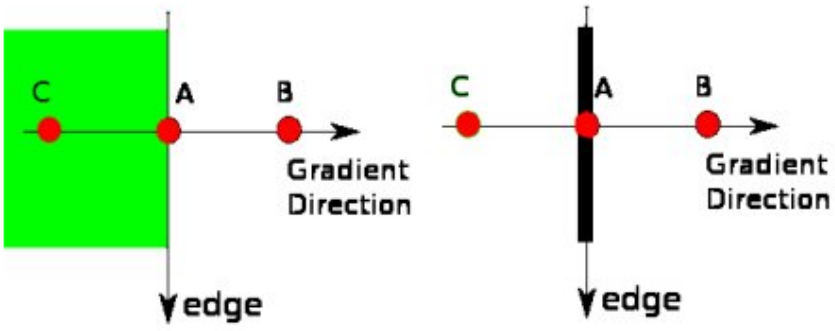
Kut dobiven uz pomoć jednadžbe (6) potrebno je pridružiti jednom od osam smjerova u kojem svaki piksel može imati susjeda na način prikazan na Slika 7. Time dobivamo vrijednost i smjer gradijenta svakog dijela slike.



Slika 7. Smjerovi kuta Sobelovog operatora[13.]

Nakon izračunatog gradijenta moramo primijeniti ne-maksimalnu supresiju. Ne-maksimalna supresija služi za uklanjanje svih piksela koji možda ne predstavljaju rub. To se vrši usporedbom susjednih piksela i provjerom koji od odabranih piksela ima veću jakost. Ako piksel nema najveću

lokalnu vrijednost njegova vrijednost se postavlja na nulu. Slika 8. prikazuje princip ovog koraka te način određivanja jakog ruba.

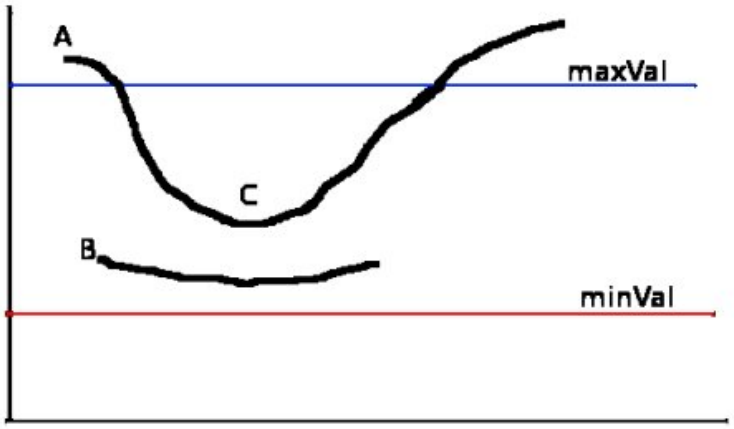


Slika 8. Ne-maksimalna supresija piksela[14.]

Rezultat koji dobijemo nakon ovog koraka je slika sa tankim rubovima[14.].

Slijedeći korak je izračun visokog i niskog praga piksela. Oni se računaju pomoću histograma gradijenta slike. Visoki prag određuje se kao postotak P_1 koji odgovara zbroju piksela koji se smatraju jakim rubovima. Postotak P_1 odgovara normalnoj razdiobi. Tada je niski prag označen sa P_2 koji je jednak (1-P_1). Uobičajene vrijednosti P_1 i P_2 su obično 20% i 40%.[10.]

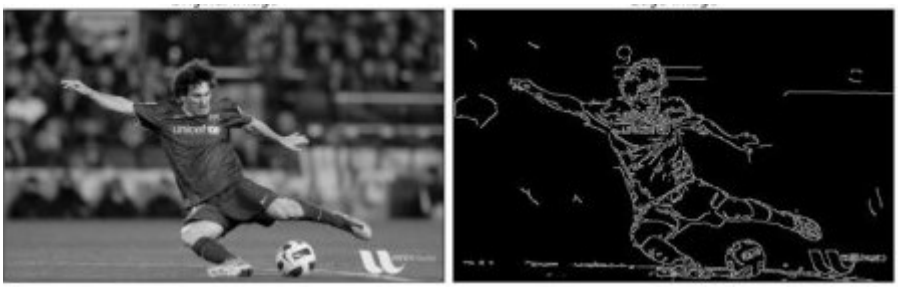
Posljednji korak Cannyjevog algoritma je određivanje praga histerezom. U ovom koraku provjera se da li piksel predstavlja jaki rub ili slabi rub. Jakim rubom se smatraju pikseli čija je vrijednost gradijenta veća od visokog praga dobivenog u prethodnom koraku, a slabim rubom ako je vrijednost gradijenta između niskog i visokog praga. Jaki rubovi se odmah iscrtavaju na slici dok se slabi iscrtavaju samo ako im je jaki rub susjed. Slika 9. prikazuje način određivanja jakosti rubova. Kako je rub A iznad linije visokog praga odmah se smatra rubom. Rub B je između vrijednosti visokog i niskog praga, ali nije povezan sa rubom A koji je jaki rub pa se zato odbacuje. Iako je rub C također između visokog i niskog praga, on je povezan sa rubom A pa se zato označava kao rub.



Slika 9. Primjer određivanja jakih i slabih rubova[14.]

Nakon ovog koraka dodatno se uklanja šum sa slike te dobijemo mapu samo jakih rubova te završni rezultat Cannyjevog algoritma.

U ovom radu je korišten Canny algoritam u sklopu OpenCV-a. U OpenCV-u cijeli algoritam sveden je na jednu naredbu cv.canny(). Ukupan rezultat Canny algoritma prikazan je na Slika 10.



Slika 10. Rezultat Canny algoritma[14.]

3.2. Metoda thresholdinga

Metoda thresholdinga je popularna metoda segmentacije slike koja se često koristi u računalnom vidu kao prvi korak detekcije kontura. Temelj ove metode je pretpostavka da promatrani objekt i pozadina imaju različitu konstantu refleksije i apsorpcije svjetla na površini. Pomoću toga možemo odrediti određeni prag koji će odvajati te dvije površine.[15.] Osnovni algoritam thresholdinga pretvara ulaznu sliku u binarnu. Potrebno je provjeriti sve piksele slike i pomoću algoritma pretvoriti sliku u binarnu. Osnovni algoritam opisan je slijedećim jednadžbama u kojima je T prag koji odabiremo, $g(i, j)$=1 pozadina, a $g(i, j)$=0 objekti.[16.]

$$g(i,j) = 1\ za\ f(i,j) \geq T \tag{7}$$

$$g(i,j) = 0\ za\ f(i,j) < T \tag{8}$$

Kako se vidi iz jednadžbi (7) i (8) jedan od glavnih problema kod binarne segmentacije predstavlja dobro određeni prag. Određivanje praga radi se pomoću analize oblika histograma. U određivanju praga kod binarnog thresholdinga koristi se histogram prvog reda te on predstavlja relativnu frekvenciju svjetline točaka slike. Prag se očitava pomoću minimuma u histogramu pri čemu je bitno da postoje pikseli sa manjom vrijednosti svjetline od određenog praga. Greške se javljaju jer je histogram izlomljen pa je potrebno to ispraviti izglađivanjem histograma ili aproksimacijom krivulje na histogram.[16.] Često je slučaj da jedan određeni prag nije zadovoljavajući za cijelu sliku, odnosno često nije dovoljno odrediti samo globalni prag, nego se moraju odrediti lokalni pragovi. Razlog tome su nepravilnosti u osvjetljenju slike, izražene sjene i niz drugih faktora. Problem se također javlja ako se objekti dodiruju jer onda ova metoda ne može odrediti granicu između objekata. Postoji još mnogo metoda thresholdinga, ali za potrebe ovog rada koristi se binarni thresholding. Poput Canny algoritma i ova metoda ima ugrađenu naredbu u OpenCV-u koja glasi cv.threshold(). Funkcija može sadržavati pet različitih argumenata tipova thresholdinga, ali u ovom će se radu koristiti samo jedan, a to je cv.THRESH_BINARY. Naredba cv.THRESH_BINARY se koristi jer je za korištene metode dobivanja kontura potrebna binarna razlika u bojama, odnosno slika mora biti crno-bijela.

4. ANALIZA KODA

U sklopu ovog rada napravljena je jednostavna aplikacija za detekciju kontura. Kako je prije napomenuto korišten je Python programski jezik te dodatni moduli koji omogućuju obradu i prikaz slike. U ovom poglavlju biti će objašnjene sve funkcije koje su korištene tokom izrade aplikacije. Slika 11. će služiti kao originalna slika koja služi kao primjer primjene dijelova koda kroz ovo poglavlje.

Slika 11. Primjer za primjenu koda[17.]

Prvi korak u izradi koda je uvoz svih potrebnih modula kako je prikazano na Slika 12. te definirati funkciju Tk() koja služi za prevađanje naredbi Tkintera pomoću Tcl/Tk interpretera.

```
from tkinter import *
from tkinter import filedialog
import cv2 as cv
import numpy
from PIL import Image
from PIL import ImageTk
import timeit

window=Tk()
```

Slika 12. Uvezeni moduli

U tom koraku definirani su moduli OpenCV-a i Numpy koji služe za detekciju kontura i Tkinter i Pillow pomoću kojih se rezultati prikazuju te modul TimeIt pomoću kojeg mjerimo vrijeme izvršavanja programa.

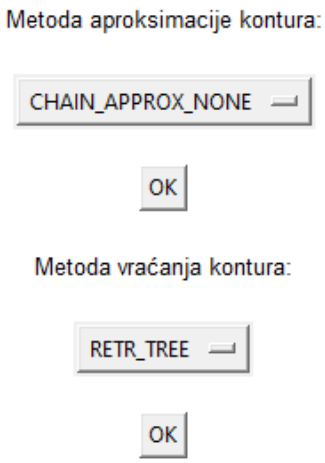
U slijedećem koraku definiramo funkciju koja omogućuje odabir datoteke s bilo koje lokacije na računalu. To je omogućeno pomoću funkcije Tkintera askopenfile() čiji argumenti sadrže početno mjesto pretraživanja, naziv prozora koji se otvara i vrstu datoteka koja se pretražuje. Kako je nama potrebno otvoriti sliku, traženi formati su .jpg, .jpeg i .png. Također, funkcija sprema varijablu lokacije kako bi se kasnije mogla koristiti u otvaranju slike u OpenCV-u.

```
s=0
def browse():
    slika = filedialog.askopenfilename(initialdir = "/",
                                       title = "Select a File",
                                       filetypes = (("Pictures",
                                                     "*.jpeg*",),("Pictures",
                                                     "*.jpg*",),("Pictures",
                                                     "*.png*",),
                                                    ("all files",
                                                    "*.*")))
    global s
    s=slika.replace('/','\\\\')
```

Slika 13. Funkcija za odabir datoteke

U ovom koraku kreiramo dva padajuća izbornika, prvi za odabir metode aproksimacije kontura koje su CHAIN_APPROX_NONE i CHAIN_APPROX_SIMPLE i drugi za metode vraćanja kontura koje su RETR_TREE, RETR_LIST, RETR_EXTERNAL i RETR_CCOMP. Svaka od tih metoda biti će detaljnije obrađena u nastavku ovog poglavlja. Kako imamo veliki broj mogućih kombinacija funkcija i dvije metode pripremanja slike za prepoznavanje kontura odabran je ovakav način izrade programa radi lakšeg snalaženja kod usporedbe rezultata. Slika 14. prikazuje kod te rezultat koji se dobiva pozivanjem.

```
#Padajući izbornik za metodu aproksimacije kontura
z=0
Mode=["CHAIN_APPROX_NONE","CHAIN_APPROX_SIMPLE"]
variable = StringVar(window)
variable.set(Mode[0]) # prva prikazana vrijednost
Pad = OptionMenu(window, variable, *Mode)
##Provjera koji gumb je odabran
def ok():
    global z
    if variable.get()=="CHAIN_APPROX_NONE":
        z=0
    else:
        z=1
button3=Button(window,text="OK",command=ok)

#Padajući izbornik za metode vraćanja kontura
q=0
Mode_1=["RETR_TREE","RETR_LIST","RETR_EXTERNAL","RETR_CCOMP"]
variable_1 = StringVar(window)
variable_1.set(Mode_1[0]) #prva prikazana vrijednost
Pad_1 = OptionMenu(window, variable_1, *Mode_1)
#Provjera koji gumb je odabran
def ok_1():
    global q
    if variable_1.get()=="RETR_TREE":
        q=0
    elif variable_1.get()=="RETR_LIST":
        q=1
    elif variable_1.get()=="RETR_EXTERNAL":
        q=2
    else:
        q=3
button4=Button(window,text="OK",command=ok_1)
```

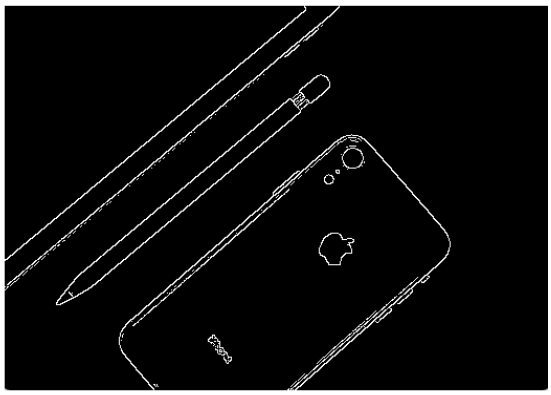


Slika 14. Kreiranje padajućeg izbornika za odabir metode i način određivanja kontura i rezultat koda

Slijedeći dio koda služi za učitavanje slika u OpenCV, te postupke kojima pripremamo sliku za pronalaženje kontura. Za pripremu slike koristimo Cannyjev algoritam. Prvo definiramo novu funkciju koja se poziva kad odaberemo gumb u aplikaciji. Tada ova funkcija stvara novi prozor za prikazivanje rezultata. Naredbom cv.imread() učitavamo prethodno odabranu sliku koju ćemo koristiti u obradi. Ako umjesto varijable koja definira odabranu sliku u funkciju stavimo 0 ona će uključiti kameru računala i koristiti snimljenu sliku u obradi. Nadalje, odabranu sliku potrebno je skalirati kako bi se mogla prikazati uz dobivene rezultate. To radimo pomoću definirane funkcije rescaleFrame. Unosom željene vrijednosti, skaliranje dimenzija slike, skaliraju se tako da ostane jednaki omjer kao kod originalne slike. Kako je većinu vremena potrebno smanjiti dimenzije slike, definirana funkcija vraća skaliranu verziju originalne slike pomoću naredbe cv.resize() čiji su argumenti novo dobivene dimenzije, te metoda interpolacije cv.INTER_AREA. Funkcija cv.INTER_AREA najpogodnija je metoda za smanjivanje slike jer zadržava omjere u kojima su pikseli postavljeni. Kako bi proces detekcije Canny rubova bio optimalan potrebno je učitanu sliku

pretvoriti u grayscale format. OpenCV slike učitava u BGR (*eng. blue, green, red)* formatu pa je potrebno napraviti pretvorbu u grayscale format. To radimo pomoću naredbe cv.cvtColor() čiji su argumenti slika koju želimo obraditi i promjena formata. Promjena formata definirana je naredbom cv.COLOR_BGR2GRAY. Slijedeće, sliku u grayscale formatu učitavamo u funkciju cv.canny(). Kako je već prije spomenuto funkcija cv.canny() obavlja detekciju Canny rubova unutar OpenCV-a. Njezini argumenti su slika na kojoj želimo pronaći rubove, minimalna vrijednost ruba i maksimalna vrijednost ruba. U cv.canny() funkciji standardna veličina kernela kod Gaussovog filtera je 5x5, a σ je 2[18.]. Ako je šum prisutan na slici u velikoj mjeri moguće je prije Cannyjevog algoritma iskoristiti funkciju cv.GaussianBlur() u kojoj odabiremo veličinu kernela i σ. Funkcija cv.GaussianBlur() dodatno smanjuje šum, ali se mogu izgubiti slabije vidljivi rubovi kod prevelike obrade slike. U ovom koraku također započinjemo mjerenje vremena potrebnog za izvršavanje operacije prepoznavanja kontura. Slika 15. prikazuje kod i njegov rezultat.

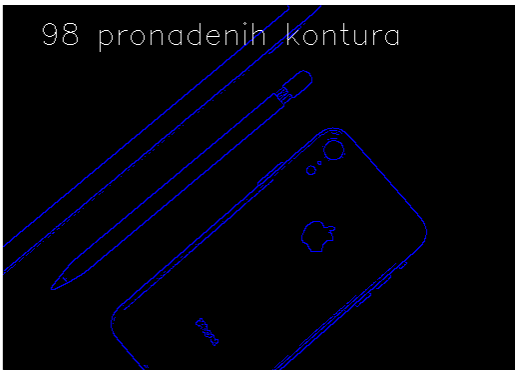
```
def opencv_1():
    #Otvaranje novog prozora za prikaz rezultata
    window_2=Toplevel(window)
    window_2.title("Rezultati")
    window_2.geometry()
    window_2.resizable()
    #Globalna varijabla za panele
    global panelA
    global panelB
    global panelC
    #Učitavanje slike pomoću OpenCVa
    img=cv.imread(s)
    #Funkcija za scaling slike
    def rescaleFrame(frame,scale=1):
        width=int(frame.shape[1]*scale)
        height =int(frame.shape[0]*scale)
        dimensions=(width,height)
        return cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)
    resized=rescaleFrame(img)
    #Pretvorba u Grayscale
    gray=cv.cvtColor(resized,cv.COLOR_BGR2GRAY)
    #blur=cv.GaussianBlur(gray,(5,5),1.4)
    start = timeit.default_timer()
    canny= cv.Canny(gray,100,200)
```



Slika 15. Kod pripreme slike za prepoznavanje rubova i rezultat koda

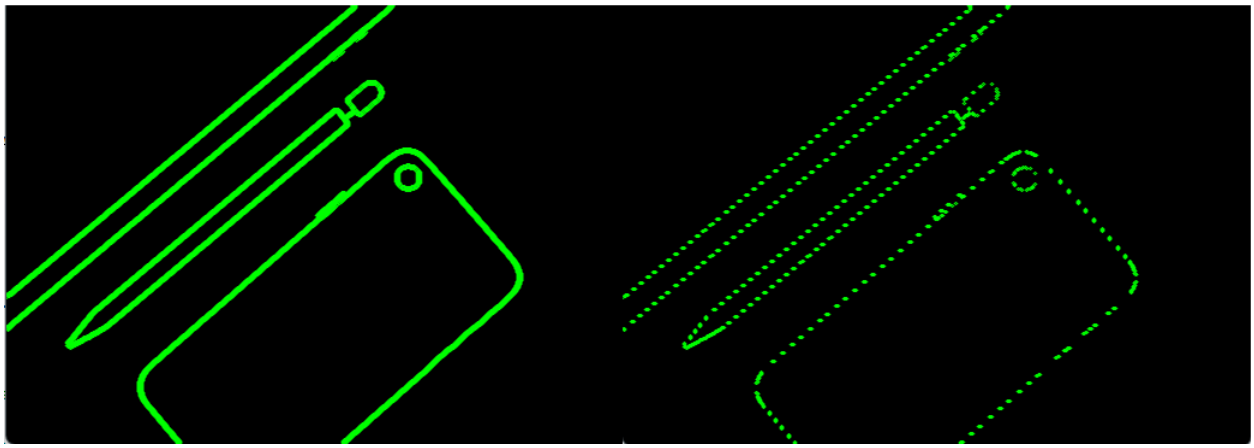
Kod prikazan na Slika 16. poziva funkciju cv.findCountours() koja služi za određivanje kontura sa slike te sadrži slijedeće argumente. Prvi argument određuje koju sliku koristimo kod prepoznavanju kontura, drugi određuje metode vraćanja kontura koje su RETR_TREE, RETR_LIST, RETR_EXTERNAL i RETR_CCOMP i posljednji argument određuje metodu aproksimacije kontura koja može biti CHAIN_APPROX_NONE ili CHAIN_APPROX_SIMPLE. Kod prikazan predstavlja samo jedan slučaj koji je odabran u aplikaciji.

```
if z==0 and q==0:
    contours, hierarchies=cv.findContours(canny,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
    y=(255,0,0)
print(f"LIST: {hierarchies}")
#Pretvorba u PIL format radi prikazivanja u programu
img1=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
img1=Image.fromarray(img1)
img2=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
img2=cv.drawContours(img2,contours,-1,y)
font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(img2,(f'{len(contours)}'+' pronadenih kontura'),(40,50),font,1,(0,0,0),1)
#Stvaranje crne slike dimenzija redimenzionirane slike
h, w, c = resized.shape
blank = 0* numpy.ones(shape=(h, w, c), dtype=numpy.uint8)
img3=cv.drawContours(blank,contours,-1,y)
cv.putText(img3,(f'{len(contours)}'+' pronadenih kontura'),(40,50),font,1,(255,255,255),1)
img2=Image.fromarray(img2)
img3=Image.fromarray(img3)
img1=ImageTk.PhotoImage(img1)
img2=ImageTk.PhotoImage(img2)
img3=ImageTk.PhotoImage(img3)
stop = timeit.default_timer()
execution_time = stop - start
cv.waitKey(0)
```



Slika 16. Kod prepoznavanja kontura i rezultat

Funkcija cv.findCountours() vraća dvije vrijednosti, prva je contours koja nam daje listu svih kontura na slici koja je prikazana kao NumPy matrica sa koordinatama (x, y) dok je druga hierarchies koja nam vraća Numpy matricu koja nam definira hijerarhijske odnose kontura na slici.[19.] Ona je za svaku konturu definirana kao matrica 4x1 u kojem prva vrijednost predstavlja slijedeću konturu sa jednakom hijerarhijskom vrijednošću, druga predstavlja prethodnu konturu sa jednakom hijerarhijskom vrijednošću, treća predstavlja konturu koja je prva podređena odabranoj i četvrta predstavlja konturu kojoj je trenutna podređena. Vrijednosti unutar matrice hijerarhija ovisi o metodi vraćanja kontura. Naredba RETR_LIST ne stvara odnose podređene i nadređene konture pa je radi toga treća i četvrta vrijednost matrice hijerarhija jednaka -1. Prve dvije vrijednosti ovise o redoslijedu pronalaženja kontura. Naredba RETR_EXTERNAL vraća samo konture koje se smatraju najvišom razinom hijerarhije dok ostale zanemaruje te ih ne iscrtava. RETR_CCOMP vraća sve konture sa slike te primjenjuje hijerarhiju na dva nivoa, sve vanjske konture postaju konture hijerarhijskog nivoa jedan dok unutarnje konture postaju konture hijerarhijskog nivoa dva. Posljednja naredba RETR_TREE također vraća sve konture, ali hijerarhijski nivoi nisu limitirani na jedan i dva nivoa, nego svaka kontura ima svoj hijerarhijski sustav. Metode aproksimacija kontura CHAIN_APPROX_NONE vraća sve točke kontura, dok CHAIN_APPROX_SIMPLE pokušava komprimirati ravne linije u početne i završne točke te je radi toga brža od metode CHAIN_APPROX_NONE. Razlika između njih u većini slučajeva nije vidljiva na gotovom iscrtavanju kontura jer naredba cv.drawContours() spaja točke koje su dobivene naredbom CHAIN_APPROX_SIMPLE. Slika 17. prikazuje razliku između te dvije metode s obzirom na broj generiranih točaka na slici. Lijevi primjer prikazuje rezultate sa CHAIN_APPROX_NONE, a desni sa CHAIN_APPROX_SIMPLE. Iz usporedbe jasno je vidljivo da je količina točaka, a i time količina podataka na desnoj slici značajno manja.



Slika 17. Usporedba CHAIN_APPROX_NONE i CHAIN_APPROX_SIMPLE

Funkcija cv.drawCountours() sadrži argumente odabira slike na kojoj će se konture iscrtati, prethodno dobiven argument contours, vrijednost -1, što označava da je potrebno ispisati sve konture, te boju i debljinu iscrtanih linija . Još jedna važna naredba je cv.putText() koja nam omogućuje ispisivanje broja pronađenih kontura na slici. To nam je važno radi lakšeg snalaženja u usporedbi različitih metoda pronalaženja kontura. U toj funkciji definiramo sliku na kojoj želimo ispisati tekst, što želimo ispisati, mjesto na kojem se ispisuje i font. Naredba cv.cvtColor(cv.BGR2RGB) služi za pretvorbu slike iz formata BGR koji se koristi u OpenCV-u u format RGB koji se koristi u Pillow formatu, koji se u ovom radu koristi za prikaz rezultata. Image.fromarray() koristimo za učitavanje slike u Pillow format te ju pomoću naredbe ImageTk.PhotoImage() dodajemo u GUI kreiran pomoću Tkintera. U ovom djelu koda također zaustavljamo mjerenje vremena koje ćemo koristiti za usporedbu metoda.

Slika 18. prikazuje nam dio koda koji služi za postavljanje slika u novom prozoru te nam omogućuje istovremeno otvaranje više prozora sa rezultatima. Također, ovdje je definiran novi prozor koji na prikazuje proteklo vrijeme izvršavanja programa.

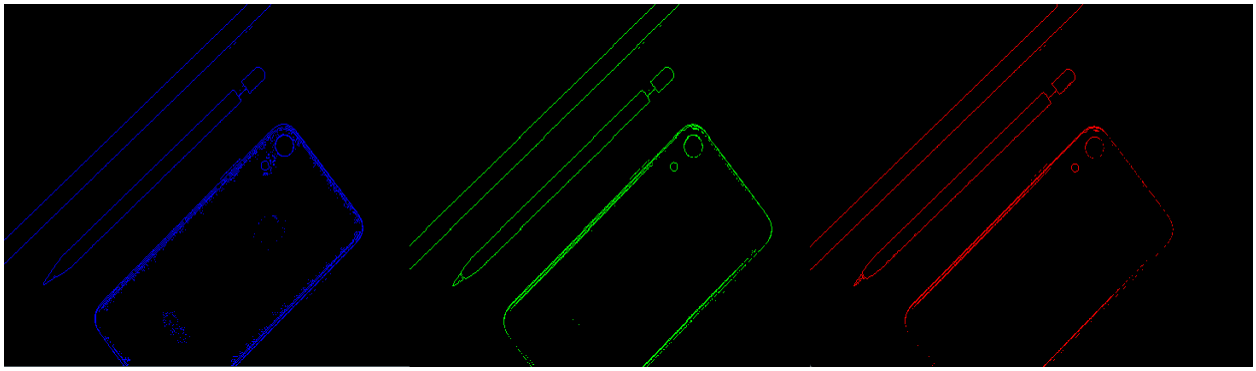
```
if panelA is None or panelB is None or panelC is None:
    # Prvi panel sprema orgionalnu sliku
    panelA = Label(window_2,image=img1)
    panelA.image = img1
    panelA.pack(side="top", padx=10, pady=10)
    # Drugi sprema dobivene konture na orgionalnu sliku
    panelB = Label(window_2,image=img2)
    panelB.image = img2
    panelB.pack(side="right", padx=10, pady=10)
    # Treci sprema konture na crnoj pozadini
    panelC = Label(window_2,image=img3)
    panelC.image = img3
    panelC.pack(side="left", padx=10, pady=10)

else:
    # Update panela
    panelA.configure(image=img1)
    panelB.configure(image=img2)
    panelA.image = img1
    panelB.image = img2
    panelC.configure(image=img3)
    panelC.image = img3
panelA=None
panelB=None
panelC=None
window_4=Toplevel(window)
window_4.title("Vrijeme proteklo tokom Canny metode")
window_4.geometry()
window_4.resizable()
print(str(execution_time))
Label4=Label(window_4,text=("Program izvršen u:"+str(execution_time)+ " s"),fg="black",bg="white",font=("Arial", 10))
Label4.pack(padx=10, pady=10)
```

Slika 18. Postavljanje rezultata u novi prozor

Ovo predstavlja zadnji korak prilikom izrade programa pomoću Cannyjevog algoritma. Slijedeće će biti prikazan dio koda koji predstavlja pronalaženje kontura uz pomoć thresholding metode.

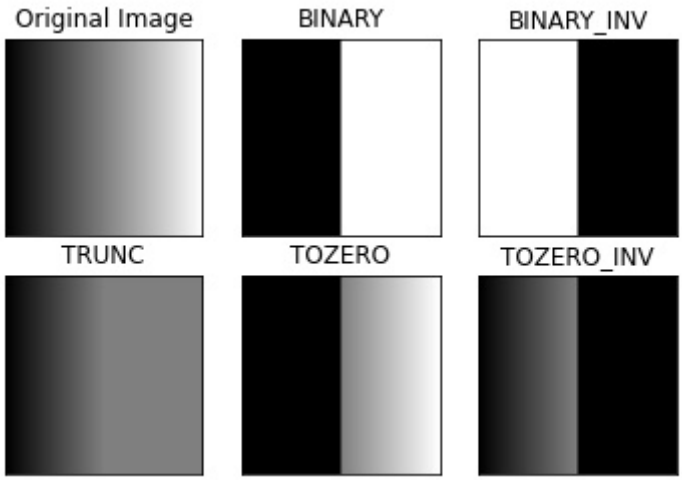
Kako je bio slučaj i kod djela koda za detekciju kontura pomoću Canny algoritma, tako i u ovom djelu definiramo novu funkciju koja se pokreće kad se u aplikaciji odabere gumb za metodu thresholdinga. Prvi dio koda služi nam za učitavanje slike u OpenCV. To se također radi preko naredbe cv.imread(). Argument ove funkcije ponovno je prethodno odabrana slika koju želimo koristiti u obradi. Sliku je potrebno skalirati pa je taj zadatak definiran funkcijom rescaleFrame(), koja skalira sliku sa zadržavanjem potrebnog omjera duljine i širine. U metodi thresholdinga, originalnu sliku potrebno je pretvoriti u grayscale format. Kako je format slike u OpenCV-u BGR koristimo naredbu cv.COLOR_BGR2GRAY te time dobivamo sliku koju možemo koristiti u thresholding metodi. Ovaj korak vrlo je bitan jer funkcija cv.threshold() ne podržava unos slika koje imaju više od jednog kanala boja. Moguće je koristiti sliku pretvorenu u sliku prikazanu samo preko plave, zelene ili crvene boje no to nam ne daje zadovoljavajuće rezultate. Kako je prikazano na Slika 19. vidimo usporedbu rezultata obrade slike ovisno o načinu prikazivanja. Lijeva slika prikazuje sliku dobivenu pomoću plavog kanala boje, srednja pomoću zelenog kanala i desna

pomoću crvenog. Sa slika jasno je vidljivo da nam ovaj način prikazivanja nije prikladan za traženje kontura jer konture ili nisu potpune ili je prikazan nepotreban šum sa slike.



Slika 19. Usporedba rezultata korištenja jednog kanala boja kod thresholding metode

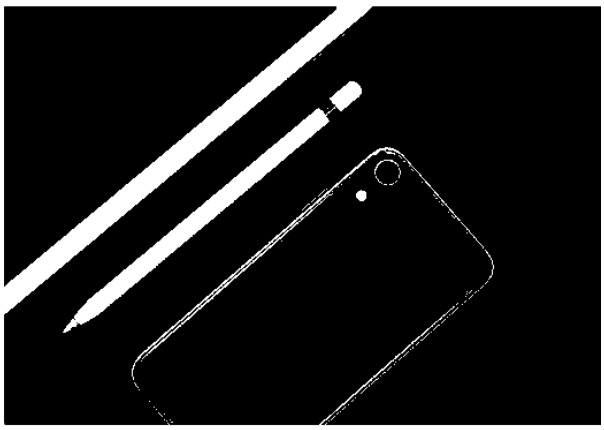
Iz tog razloga koristimo grayscale format slike koji nam vraća najoptimalnije rezultate. U funkciju cv.threshold(), osim argumenta željene slike, potrebno je odrediti prag kojim ćemo provjeravati piksele, vrijednost na koju će se postavljati pikseli koji prelaze prag te tip thresholdinga koji želimo. U ovom radu određeni prag je 150, maksimalna vrijednost na koju se pikseli postavljaju 255 koja odgovara bijeloj boji, a tip thresholdinga je cv.THRESH_BINARY jer nam je potrebno dobiti rezultat s jasno vidljivim razlikama u odvajanju objekta. OpenCV sadrži još nekoliko metoda thresholdinga koji su prikazani na Slika 20..



Slika 20. Tipovi thresholdinga u OpenCV-u[20.]

Definiranjem te funkcije dobivamo kod i rezultate prikazane na Slika 21. iz koje možemo vidjeti pretvorbu originalne slike u binarnu. Ovdje ponovno počinjemo mjeriti vrijeme potrebno za izvršavanje koda.

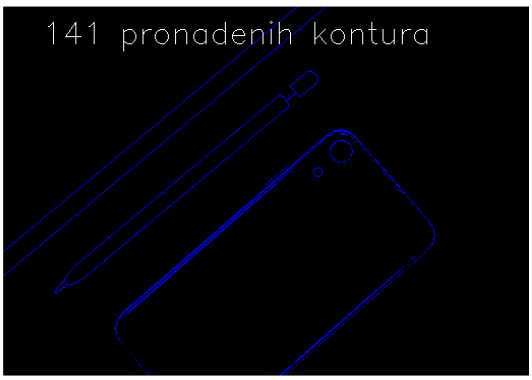
```
def opencv_2():
    #Novi prozor za prikaz rezultata
    window_3=Toplevel(window)
    window_3.title("Rezultati thresholdinga")
    window_3.geometry()
    window_3.resizable()
    global panelA
    global panelB
    global panelC
    #Učitavanje slike
    img=cv.imread(s)
    #Skaliranje slike
    def rescaleFrame(frame,scale=1):
        width=int(frame.shape[1]*scale)
        height =int(frame.shape[0]*scale)
        dimensions=(width,height)
        return cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)
    resized=rescaleFrame(img)
    start_1 = timeit.default_timer()
    gray=cv.cvtColor(resized,cv.COLOR_BGR2GRAY)
    ret,thresh =cv.threshold(gray,150,255,cv.THRESH_BINARY)
```



Slika 21. Kod i rezultati pripreme slike pomoću thresholding metode

Slijedeći dio koda isti je kao kod Canny algoritma samo što umjesto Canny algoritma koristimo pripremljenu sliku pomoću thresholding metode. Ponovno je moguće odabrati dva tipa aproksimacije kontura i četiri tipa metoda vraćanja kontura. U ovom poglavlju biti će prikazan kod i rezultati za jednu odabranu metodu. Slika 22. prikazuje nam kod i rezultate pronalaženja kontura pomoću thresholding metode te dio koda zaslužan za prikaz originalne slike i rezultata u novom prozoru. Također, na kraju ovog dijela zaustavljamo mjerenje vremena koje je proteklo tijekom izvršavanja programa.

```
if z==0 and q==0:
    contours, hierarchies=cv.findContours(thresh,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
    y=(255,0,0)
img1=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
img1=Image.fromarray(img1)
img2=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
img2=cv.drawContours(img2,contours,-1,y)
font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(img2,(f'{len(contours)}'+' pronadenih kontura'),(40,50),font,1,(0,0,0),1)
h, w, c = resized.shape
blank = 0* numpy.ones(shape=(h, w, c), dtype=numpy.uint8)#Generiranje prazne slike
img3=cv.drawContours(blank,contours,-1,y)
cv.putText(img3,(f'{len(contours)}'+' pronadenih kontura'),(40,50),font,1,(255,255,255),1)
img3=Image.fromarray(img3)#PIL format
img2=Image.fromarray(img2)
img1=ImageTk.PhotoImage(img1)
img2=ImageTk.PhotoImage(img2)
img3=ImageTk.PhotoImage(img3)
stop_1 = timeit.default_timer()
execution_time_1 = stop_1 - start_1
```



Slika 22. Kod i rezultati prepoznavanja kontura pomoću thresholding metode

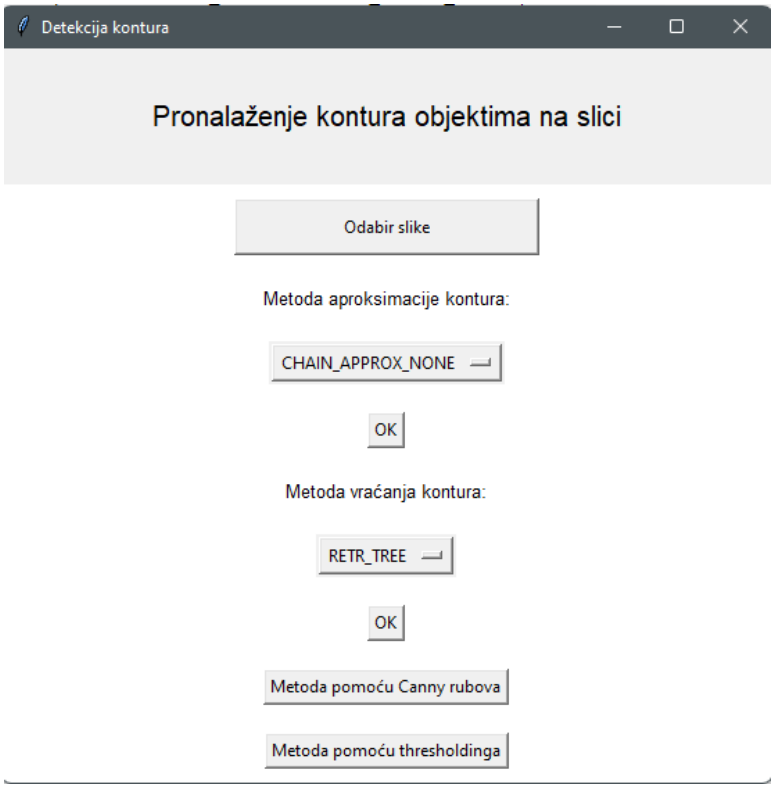
Dijelovi koda koji nisu detaljnije objašnjeni služe za prikaz slika na grafičkom sučelju pomoću Tkinter-a i Pillow-a.

Posljednji dio koda služi za definiciju grafičkih elemenata koji se prikazuju na jednostavnom sučelju koji omogućuju jednostavnu usporedbu različitih metoda. Ostatak koda prikazan je na Slika 23.

```
#Naslov prozora
window.title('Detekcija kontura')
#Veličina prozora
window.geometry()
#Pozadinska boja
window.config(background="white")

#Naslov programa
Label1=Label(window,text="Pronalaženje kontura objektima na slici",width=50,height=4,fg="black",font=("Arial", 15))
# Odabir metode
Label2=Label(window,text="Metoda pronalaženja kontura:",fg="black",bg="white",font=("Arial", 10))
#Odabir metode vraćanja
Label3=Label(window,text="Metoda vraćanja kontura:",fg="black",bg="white",font=("Arial", 10))

#Browse gumb
Button1=Button(window,text="Odabir slike",command=browse,height=2, width=30)
#gumb za canny metodu
Button2=Button(window,text="Metoda pomoću Canny rubova",command=opencv_1)
#gumb za thresholding metodu
Button3=Button(window,text="Metoda pomoću thresholdinga",command=opencv_2)

panelA=None
panelB=None
panelC=None

#Raspored elemenata na programu
Label1.pack(side="top")
Button1.pack(padx=10, pady=10)
Label2.pack(padx=10, pady=10)
Pad.pack(padx=10, pady=10)
button3.pack(padx=10, pady=10)
Label3.pack(padx=10, pady=10)
Pad_1.pack(padx=10, pady=10)
button4.pack(padx=10, pady=10)
Button2.pack(padx=10, pady=10)
Button3.pack(padx=10, pady=10)
window.mainloop()
```

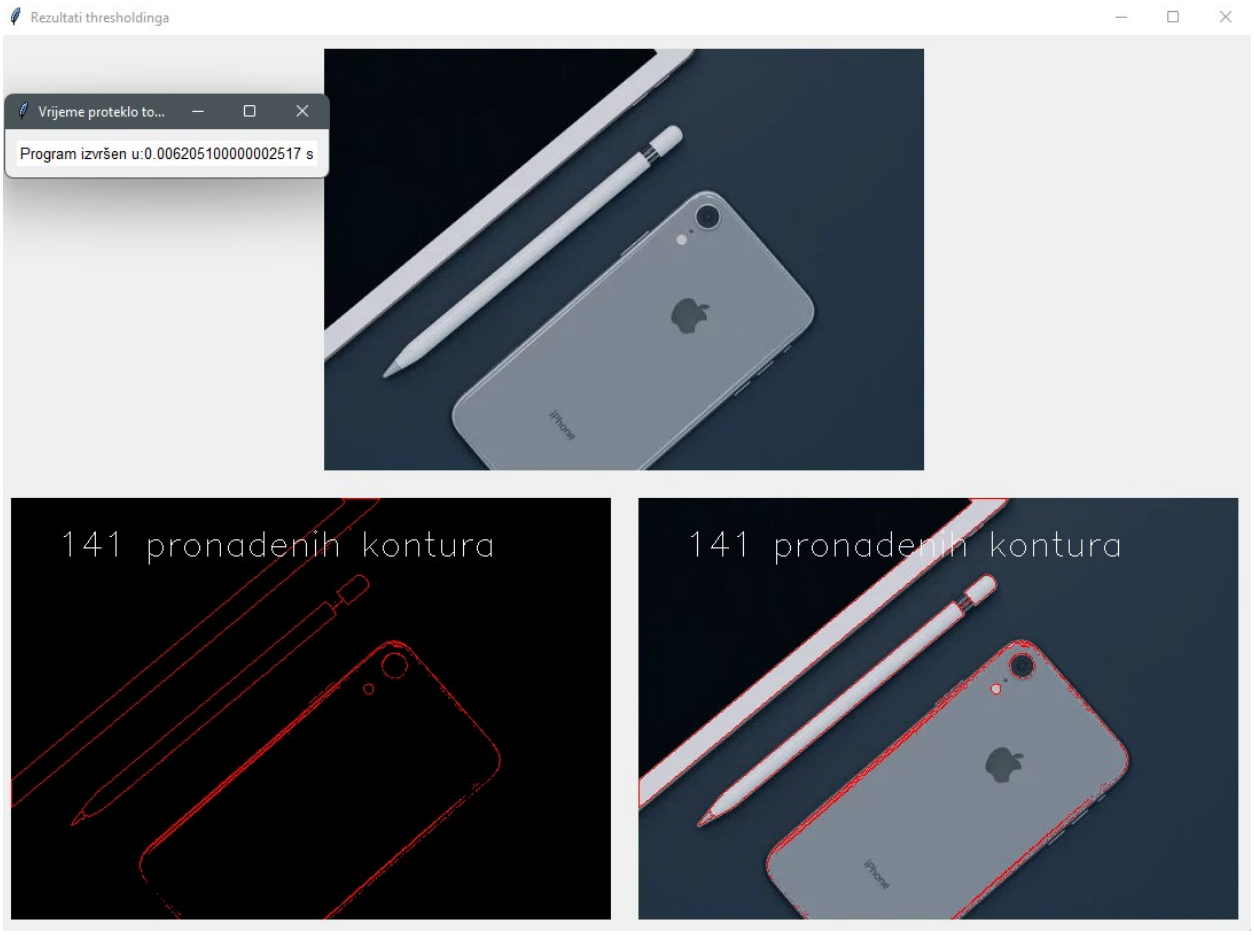
Slika 22. Ostatak koda

Time dovršavamo analizu koda. Na Slika 23., Slika 24. i Slika 25. prikazani su rezultati izvršavanja koda u cjelini. Slika 23. Prikazuje nam grafičko sučelje programa u kojem se vrši odabir slike i metoda, Slika 24. prikazuje rezultate dobivene pomoću Canny metode, a Slika 25. prikazuje rezultate dobivene pomoću thresholding metode.



Slika 23. Grafičko sučelje

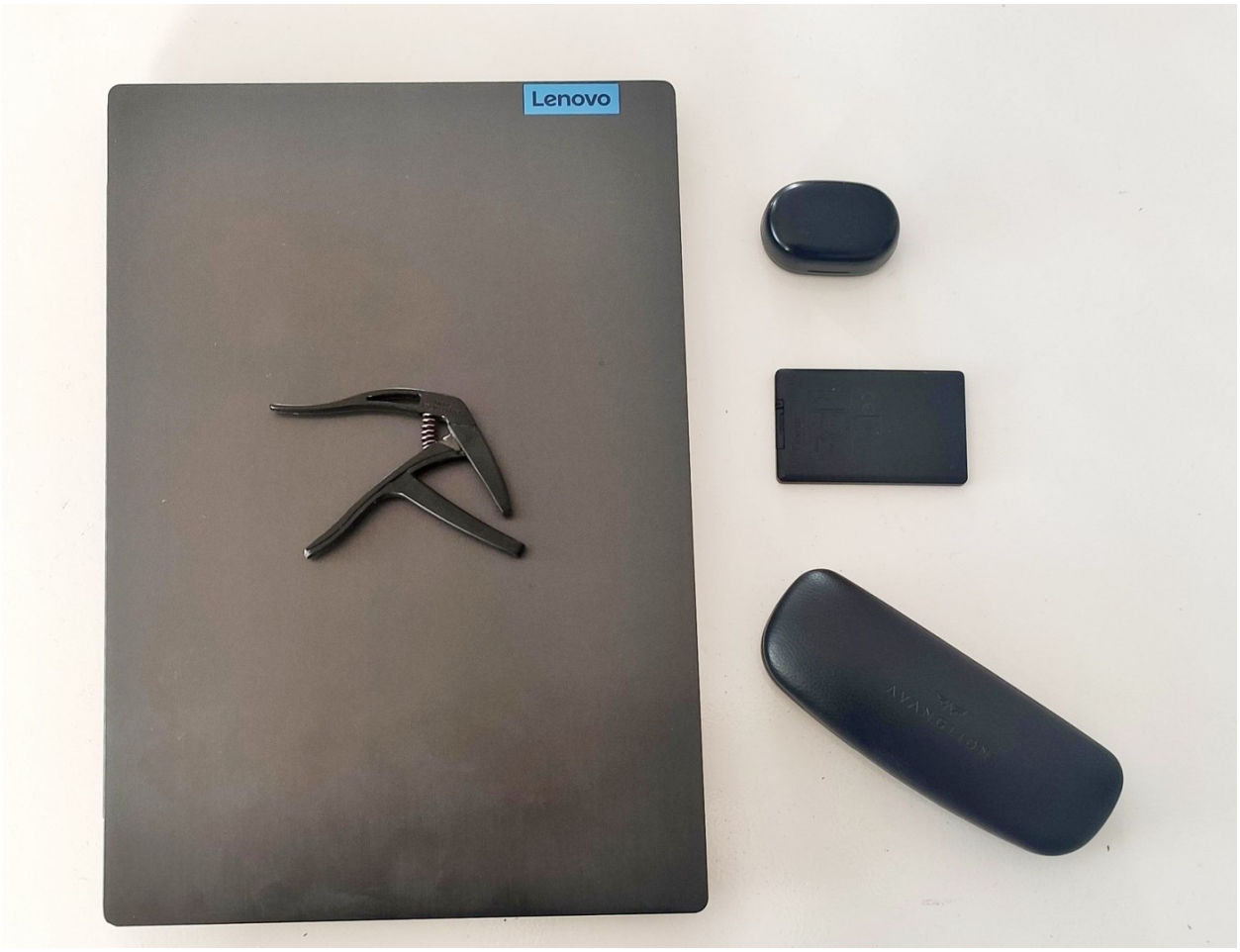
Slika 24. Rezultati Canny metode



Slika 25. Rezultati thresholding metode

5. REZULTATI

Pomoću izrađene aplikacije uspoređujemo metode pronalaženja kontura. Usporedba će se vršiti na temelju broja pronađenih kontura odnosno o sposobnosti razlikovanja stvarnih kontura na slici i vremenu koje je potrebno da se program izvrši. Slika 26. će se koristiti kao originalna slika te će uz pomoć nje biti testirane sve metode aproksimacije i označavanja kontura. Vrijeme se mjeri pomoću ugrađenog Python modula TimeIt. Radi lakše usporedbe svih metoda koristi se slika sa izraženim kontrastom između pozadine i predmeta.



Slika 26. Originalna slika za usporedbu metoda

Prvo će biti prikazani rezultati svih metoda dobivenih pomoću Canny rubova, a zatim pomoću thresholding metode.

5.1. Usporedba prema kriteriju broja pronađenih kontura

Na Slika 27. su prikazani rezultati programa za Canny rubove sa CHAIN_APPROX_NONE aproksimacijom i svim metodama hijerarhijske strukture.

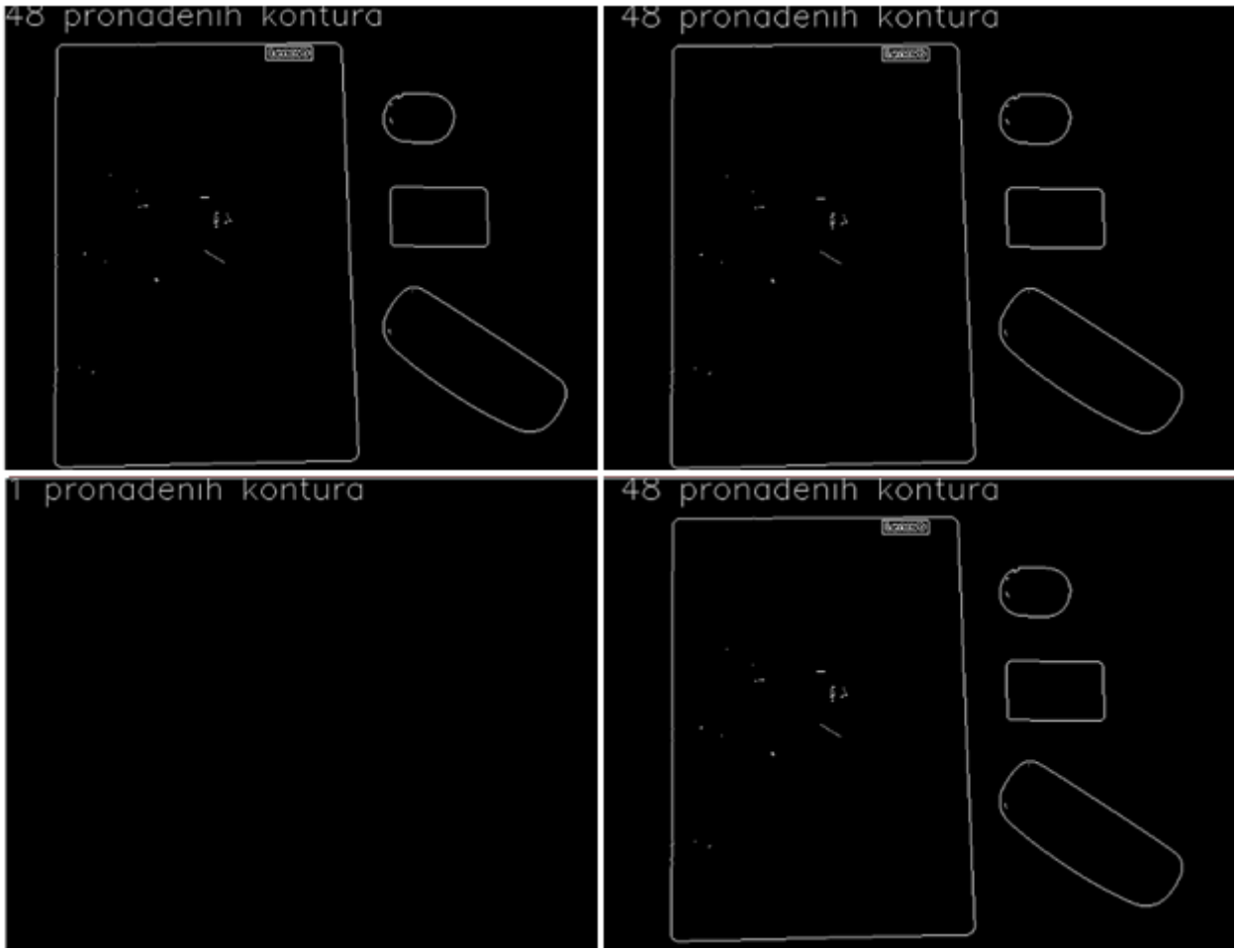
Slika 27. Canny rubovi pomoću CHAIN_APPROX_NONE i svih hijerarhijskih metoda

Gornja lijeva slika prikazuje rezultate dobivene pomoću RETR_TREE metode hijerarhija, gornja desna pomoću RETR_LIST, donja lijeva pomoću RETR_EXTERNAL i donja desna pomoću RETR_CCOMP. Iz ove usporedbe vidimo da nam program vraća slična rješenja ovisno o odabranoj metodi. Metoda RETR_EXTERNAL vraća manji broj kontura jer ona označava samo konture koje smatra najvišom hijerarhijskom razinom. Kako je vidljivo na slici prepoznate su samo vanjske konture dok su ostale zanemarene.

Slijedeća slika prikazuje rezultate pomoću CHAIN_APPROX_SIMPLE metode aproksimacije sa svim metodama hijerarhijske strukture.

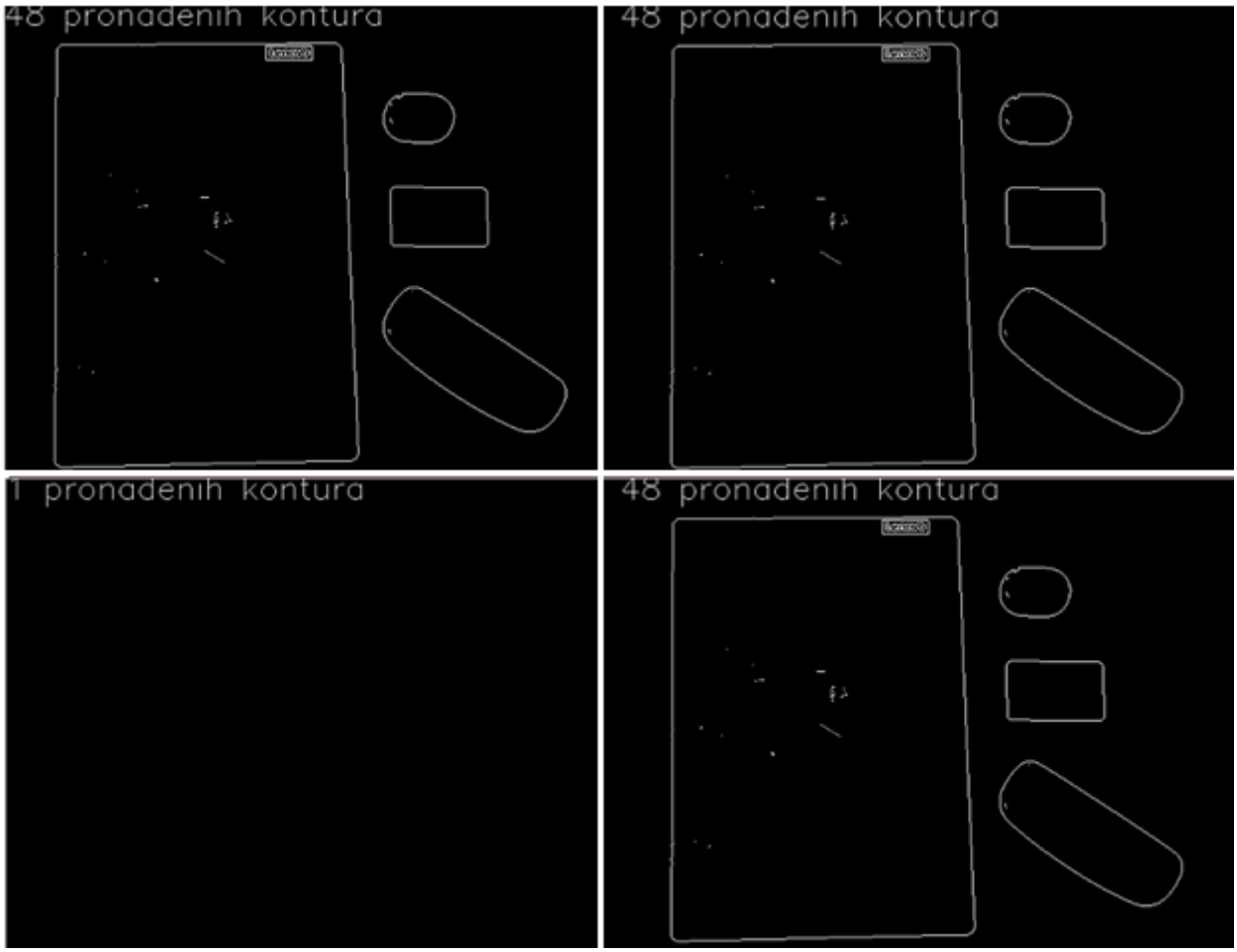
Slika 28. Canny rubovi pomoću CHAIN_APPROX_SIMPLE i svih hijerarhijskih metoda

Gornja lijeva slika prikazuje rezultate dobivene pomoću RETR_TREE metode hijerarhija, gornja desna pomoću RETR_LIST, donja lijeva pomoću RETR_EXTERNAL i donja desna pomoću RETR_CCOMP. Isti broj dobivenih kontura pomoću CHAIN_APPROX_SIMPLE metode i CHAIN_APPROX_NONE metode bio je očekivan jer kako je prije spomenuto, naredba cv.drawContours() spaja sve dobivene točke te na taj način prikazuje dobivene konture. Razlika između ovih metoda očituje se u nešto kraćem vremenu CHAIN_APPROX_SIMPLE metode jer računalo radi sa manje točaka. Kako su na originalnoj slici objekti jednostavni razlika nije izrazito zamjetna, ali kod kompliciranijih predmeta jest. Slika 29. prikazuje rezultate dobivene primjenom thresholding metode s CHAIN_APPROX_NONE metodom aproksimacije pomoću svih metoda hijerarhijskih struktura.



Slika 29. Thresholding pomoću CHAIN_APPROX_NONE i svih hijerarhijskih metoda

Gornja lijeva slika prikazuje rezultate dobivene pomoću RETR_TREE metode hijerarhija, gornja desna pomoću RETR_LIST, donja lijeva pomoću RETR_EXTERNAL i donja desna pomoću RETR_CCOMP. Kako je vidljivo sa slike metoda thresholdinga prepoznaje manje kontura od metode pomoću Canny rubova. Također, vidljivo je da ne može prepoznati predmet koji se nalazi na drugom predmetu radi male razlike u osvjetljenju i boji. Na tome primjeru možemo primijetiti najveći nedostatak metode thresholdinga, a to je nemogućnost detekcije kontura kad ne postoji razlika osvjetljenja i boja. Osim toga sa slike se vidi da metoda RETR_EXTERNAL vraća samo jednu konturu i to na samom vrhu slike. To se događa jer, kako je već prije spomenuto, naredba RETR_EXTERNAL vraća samo konture koje smatra najvišom hijerarhijskom razinom što je u našem slučaju samo jedna kontura na vrhu slike. Slika 30. prikazuje thresholding metodu uz CHAIN_APPROX_SIMPLE metodu aproksimacije i sve metode hijerarhija kontura.

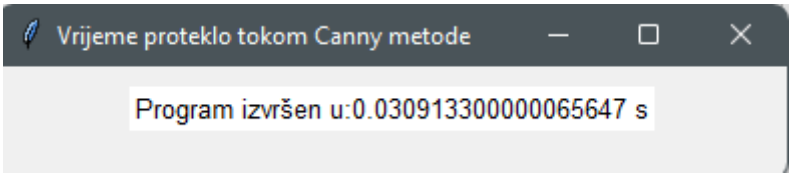


Slika 30. Thresholding pomoću CHAIN_APPROX_SIMPLE i svih hijerarhijskih metoda

Gornja lijeva slika prikazuje rezultate dobivene pomoću RETR_TREE metode hijerarhija, gornja desna pomoću RETR_LIST, donja lijeva pomoću RETR_EXTERNAL i donja desna pomoću RETR_CCOMP. Korištenjem CHAIN_APPROX_SIMPLE metode aproksimacije dobivamo isti broj kontura kao i korištenjem metode CHAIN_APPROX_NONE. Kako je spomenuto i kod metode pomoću Canny rubova, razlog je opcija cv.drawContours() koja spaja sve dobivene točke. Razlika između metoda uočljiva je kod vremena potrebnog za izvršavanje programa.

5.2. Usporedba prema kriteriju vremena izvršavanja programa

Vrijeme u programu se mjeri pomoću ugrađenog modula TimeIt te se prikazuje u novom prozoru kako je prikazano na Slika 31.



Slika 31. Prikaz vremena izvršavanja programa

Radi lakše usporedbe vremena svih metoda prikazana su u Tablica 1.. Vrijeme izvršavanja označeno je sa *t* dok je mjerna jedinica koja se koristi sekunda. Sva vremena dobivena su pomoću izrađene aplikacije i izmjerena za istu sliku koja je bila korištena za usporedbu prema broju pronađenih kontura.

Vrijeme izvršavanja programa			
			Vrijeme izvršavanja, *t*
Canny metoda	CHAIN_APPROX_NONE	RETR_TREE	0,024920 s
		RETR_LIST	0,018886 s
		RETR_EXTERNAL	0,012448 s
		RETR_CCOMP	0,019512 s
	CHAIN_APPROX_SIMPLE	RETR_TREE	0,023544 s
		RETR_LIST	0,018788 s
		RETR_EXTERNAL	0,010272 s
		RETR_CCOMP	0,019241 s

Thresholding metoda	CHAIN_APPROX_NONE	RETR_TREE	0,012609 s
		RETR_LIST	0,007833 s
		RETR_EXTERNAL	0,007209 s
		RETR_CCOMP	0,008079 s
	CHAIN_APPROX_SIMPLE	RETR_TREE	0,011684 s
		RETR_LIST	0,007818 s
		RETR_EXTERNAL	0,007119 s
		RETR_CCOMP	0,007884 s

Tablica 1. Vrijeme izvršavanja programa

Iz dobivenih rezultata možemo vidjeti da je potrebno vrijeme za izvršavanje Canny metode skoro deset puta veće od metode thresholdinga. Također, kod obje metode vidljivo je da korištenjem CHAIN_APPROX_SIMPLE metode aproksimacije dobivamo kraća vremena izvršavanja programa od metode aproksimacije CHAIN_APPROX_NONE. Kako je već prije spomenuto razlog tome je manji broj pronađenih točaka koje se koriste za aproksimaciju. Korištenjem različitih metoda hijerarhijske strukture također dobivamo razlike u vremenu izvršavanja. Korištenjem metode RETR_TREE potrebno je najviše vremena. Razlog tome jer što navedena metoda prikazuje sve konture te svakoj pridružuje vlastiti hijerarhijski sustav što zahtjeva najviše vremena. Metoda RETR_CCOMP slijedeća je prema potrebnom vremenu izvršavanja. RETR_CCOMP također vraća sve konture, ali radi samo dvostupanjsku hijerarhiju pa joj je vrijeme izvršavanja kraće od RETR_TREE. Slijedeća po potrebnom vremenu izvršavanja je metoda RETR_LIST. Ona također vraća sve konture, ali ne radi hijerarhijski sustav pa je njeno vrijeme izvršavanja nešto kraće od prethodno navedenih metoda. RETR_EXTERNAL predstavlja najbržu metodu jer ona ne vraća sve konture niti radi hijerarhijski sustav. Kako ona prikazuje samo

konture najviše hijerarhije vrijeme izvršavanja je najkraće, ali kao nedostatak ne prikazuje sve konture.

5.3. Ukupni rezultati

Tablica 2. prikazuje ukupne rezultate, odnosno broj pronađenih kontura i vrijeme potrebno za izvršavanje svih metoda. Iz tablice jasno je vidljivo da Canny metoda vraća više kontura od metode thresholdinga no zato joj je vrijeme izvršavanja dulje.

Ukupni rezultati				
Metoda			Vrijeme izvršavanja, *t*	Broj kontura
Canny metoda	CHAIN_APPROX_NONE	RETR_TREE	0,024920 s	64
		RETR_LIST	0,018886 s	64
		RETR_EXTERNAL	0,012448 s	12
		RETR_CCOMP	0,019512 s	64
	CHAIN_APPROX_SIMPLE	RETR_TREE	0,023544 s	64
		RETR_LIST	0,018788 s	64
		RETR_EXTERNAL	0,010272 s	12
		RETR_CCOMP	0,019241 s	64

Thresholding metoda	CHAIN_APPROX_NONE	RETR_TREE	0,012609 s	48
		RETR_LIST	0,007833 s	48
		RETR_EXTERNAL	0,007209 s	1
		RETR_CCOMP	0,008079 s	48
	CHAIN_APPROX_SIMPLE	RETR_TREE	0,011684 s	48
		RETR_LIST	0,007818 s	48
		RETR_EXTERNAL	0,007119 s	1
		RETR_CCOMP	0,007884 s	48

Tablica 2. Ukupni rezultati

Promatranjem dobivenih rezultata uočavamo prednost Canny metode kod pronalaženja kontura u uvjetima slabijeg osvjetljenja, odnosno kod manje razlike između boja predmeta i pozadine. Metoda thresholdinga pogodnija nam je kada je potrebno pronaći konture na dobro osvijetljenoj slici sa jasnim kontrastom predmeta dok nam je Canny pogodniji za slabije uvjete osvjetljenja i kad vrijeme izvršavanja nije najbitnije.

5.4. Nedostaci metoda

Na slijedećem primjeru biti će prikazani neki od nedostataka pojedinih metoda. Kako je već spomenuto glavni nedostatak metode thresholdinga je nemogućnost pravilnog određivanja kontura kada su predmet promatranja i pozadina slično osvjetljeni i kada im se boje ne razlikuju. Taj problem se javlja radi određenog praga koji se koristi kod pretvorbe slike u binarni oblik. Slika 32. prikazuje navedeni problem. Lijeva slika predstavlja originalnu sliku dok desna slika predstavlja sliku sa iscrtanim rubovima.

Slika 32. Nedostatak thresholding metode

Sa Slika 32. jasno je vidljivo da prepoznavanje kontura nije izvršeno zadovoljavajuće, tj. kao kontura prepoznat je samo odbljesak dok predmet nije prikazan kao odvojeni predmet. Canny metoda kod ove slike daje bolje rezultate te skoro potpuno prepoznaje cijeli predmet u uvjetima lošeg osvjetljenja kako je prikazano na Slika 33.. Ta metoda također nije potpuno točna, ali pomoću nje možemo dobiti preciznije rezultate u uvjetima lošeg osvjetljenja.

Slika 33. Canny metoda kod lošeg osvjetljenja

Glavni nedostatak Canny metode predstavlja šum na slici. Iako je u algoritam Canny funkcije unutar OpenCV-a uključen korak uklanjanja šuma pomoću Gaussovog filtera, često to nije dovoljno. Algoritam sa slike prikazuje dodatan šum koji nam je nebitan za prikaz slike. Šum se može dodatno smanjiti unutar OpenCV-a pomoću naredbe cv.GaussianBlur() koja, prije Canny algoritma, dodatno smanjuje šum na slici. Slika 34. prikazuje primjer dodatnog šuma na slici te njegovo smanjenje primjenom funkcije cv.GaussianBlur().

Slika 34. Primjena cv.GaussianBlur() funkcije

Srednja slika prikazuje nam rezultate obrade slike bez primjene funkcije cv.GaussianBlur() dok desna prikazuje rezultate s primjenom funkcije cv.GaussianBlur(). Iz ovog primjera jasno je vidljivo da primjenom funkcije smanjujemo dodatan šum sa slike te nam ona omogućuje kvalitetniji prikaz traženog predmeta promatranja. Nedostatak korištenja funkcije nam predstavlja gubitak određenih kontura koje su bitne za ukupan oblik konture predmeta promatranja.

ZAKLJUČAK

U ovom radu pružen je pregled metoda pronalaženja i označavanja kontura sa slika. Korištenjem Python programskog jezika i biblioteke OpenCV izrađena je programska aplikacija koja omogućuje pronalaženje kontura na slikama. U teorijskom djelu objašnjene su neke od najviše korištenih metoda koje se koriste za pronalaženje kontura, a to su Canny algoritam i thresholding metoda. Na primjeru nekoliko slika uspoređene su obje metode te uočene određene prednosti i mane svake metode. Cannyjev algoritam omogućuje nam preciznije pronalaženje kontura kod uvjeta slabijeg osvjetljenja, ali mu je potrebno više vremena za izvršavanje. Uz to, osjetljiv je na šum na slici. Thresholding metoda prepoznaje manji broj kontura, ali se zato brže izvršava. Pogodna je za korištenje u uvjetima adekvatnog osvjetljenja i kada je jasna razlika između boja promatranog predmeta i pozadine.

Detekcija kontura predstavlja podlogu mnogih operacija računalnog vida te bez nje danas ne bi imali mnoge aplikacije koje koristimo svakodnevno. OpenCV nam pruža jednostavnu implementaciju tih operacija u veliki niz aplikacija, od onih koje koristimo svakodnevno do specijaliziranih koji se koriste u strojarskoj i drugim industrijama.

Literatura

1. Huang, T. (1996-11-19). Vandoni, Carlo, E (ed.). *Computer Vision : Evolution And Promise* (PDF). 19th CERN School of Computing. Geneva: CERN. pp. 21–25. doi:10.5170/CERN-1996-008.21

2. Gong, XY., Su, H., Xu, D. *et al.* An Overview of Contour Detection Approaches. *Int. J. Autom. Comput.* **15,** 656–672 (2018). https://doi.org/10.1007/s11633-018-1117-z

3. D.R. Martin, C.C. Fowlkes, and J. Malik, "Learning to Detect Natural Image Boundaries Using Local Brightness, Color and Texture Cues," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 26, no. 5, pp. 530-549, May 2004

4. S. Crawford-Hines and C. Anderson, "Machine-learned contours to assist boundary tracing tasks," *1998 IEEE Southwest Symposium on Image Analysis and Interpretation (Cat. No.98EX165)*, 1998, pp. 229-231, doi: 10.1109/IAI.1998.666890.

5.Percepcijska organizacija: https://getwww.uni-paderborn.de/research/completed/perceptual_organization,pristupljeno 25.1.2022.

6. S. Parveen and J. Shah, "A Motion Detection System in Python and Opencv," *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, 2021, pp. 1378-1382, doi: 10.1109/ICICV50876.2021.9388404.

7. Neelam Dwivedi, Dushyant Kumar Singh, Dharmender Singh Kushwaha,An Approach for Unattended Object Detection through Contour Formation using Background Subtraction,Procedia Computer Science,Volume 171,2020,Pages 1979-1988,ISSN 1877-0509,

8. Python: https://docs.python.org/3/faq/general.html#what-is-python, pristupljeno 25.1.2022.

9. OpenCV :https://opencv.org/about/, pristupljeno 25.1.2022

10. K. B. Pawar and S. L. Nalbalwar, "Distributed canny edge detection algorithm using morphological filter," *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2016, pp. 1523-1527, doi: 10.1109/RTEICT.2016.7808087.

11. J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

12. Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001.

13. Y. Zhang, X. Han, H. Zhang and L. Zhao, "Edge detection algorithm of image fusion based on improved Sobel operator," 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC), 2017, pp. 457-461, doi: 10.1109/ITOEC.2017.8122336.

14.OpenCV, Canny: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html, pristupljeno 25.1.2022.

15. Qingmao Hu, Z. Hou and W. L. Nowinski, "Supervised range-constrained thresholding," in *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 228-240, Jan. 2006, doi: 10.1109/TIP.2005.860348.

16. Petar Ćurković, dipl. ing. Tomislav Stipančić, dipl. ing. Segmentacija slike: https://slideplayer.gr/slide/14907603/, pristupljeno 26.1.2022.

17. OpenCV: https://learnopencv.com/contour-detection-using-opencv-python-c/#Finding-and-Drawing-Contours-using-OpenCV, pristupljeno 26.1.2022.

18. Gary Bradski, Adrian Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, "O'Reilly Media, Inc.", 24. ruj 2008

19.OpenCV konture:
https://docs.opencv.org/4.x/d3/d05/tutorial_py_table_of_contents_contours.html,
pristupljeno 26.1.2022.

20. OpenCV thresholding: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html, pristupljeno 26.1.2022.

PRILOG

I. Kod – Prepoznavanje_kontura.py

```
from tkinter import *
from tkinter import filedialog
import cv2 as cv
import numpy
from PIL import Image
from PIL import ImageTk
import timeit
window=Tk()
s=0
def browse():
    slika = filedialog.askopenfilename(initialdir = "/",
                                       title = "Select a File",
                                       filetypes = (("Pictures",
                                                     "*.jpeg*",),("Pictures",
                                                     "*.jpg*",),("Pictures",
                                                     "*.png*",),
                                                    ("all files",
                                                    "*.*")))
    global s
    s=slika.replace('/','\\\\')

z=0
Mode=["CHAIN_APPROX_NONE","CHAIN_APPROX_SIMPLE"]
variable = StringVar(window)
variable.set(Mode[0])
Pad = OptionMenu(window, variable, *Mode)
def ok():
    global z
    if variable.get()=="CHAIN_APPROX_NONE":
        z=0
    else:
        z=1
button3=Button(window,text="OK",command=ok)

q=0
Mode_1=["RETR_TREE","RETR_LIST","RETR_EXTERNAL","RETR_CCOMP"]
variable_1 = StringVar(window)
variable_1.set(Mode_1[0])
Pad_1 = OptionMenu(window, variable_1, *Mode_1)
def ok_1():
    global q
    if variable_1.get()=="RETR_TREE":
        q=0
```

```
    elif variable_1.get()=="RETR_LIST":
        q=1
    elif variable_1.get()=="RETR_EXTERNAL":
        q=2
    else:
        q=3
button4=Button(window,text="OK",command=ok_1)
def opencv_1():
    window_2=Toplevel(window)
    window_2.title("Rezultati Canny")
    window_2.geometry()
    window_2.resizable()
    global panelA
    global panelB
    global panelC
    img=cv.imread(s)
    def rescaleFrame(frame,scale=0.3):
        width=int(frame.shape[1]*scale)
        height =int(frame.shape[0]*scale)
        dimensions=(width,height)
        return cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)
    resized=rescaleFrame(img)
    gray=cv.cvtColor(resized,cv.COLOR_BGR2GRAY)
    #blur=cv.GaussianBlur(gray,(5,5),1.4)
    start = timeit.default_timer()
    canny= cv.Canny(gray,100,200)

    if z==0 and q==0:
        contours,
hierarchies=cv.findContours(canny,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
        y=(255,0,0)
    elif z==0 and q==1:
        contours,
hierarchies=cv.findContours(canny,cv.RETR_LIST,cv.CHAIN_APPROX_NONE)
        y=(255,0,0)
    elif z==0 and q==2:
        contours,
hierarchies=cv.findContours(canny,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_NONE)
        y=(255,0,0)
    elif z==1 and q==3:
        contours, hierarchies=cv.findContours(canny,cv.RETR_CCOMP,
cv.CHAIN_APPROX_NONE)
        y=(255,0,0)
    elif z==1 and q==0:
        contours, hierarchies=cv.findContours(canny,cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,0,0)
    elif z==1 and q==1:
```

```
        contours, hierarchies=cv.findContours(canny,cv.RETR_LIST,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,0,0)
    elif z==1 and q==2:
        contours, hierarchies=cv.findContours(canny,cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,0,0)
    else:
        contours, hierarchies=cv.findContours(canny,cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,0,0)
    print(f"LIST: {hierarchies}")
    img1=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
    img1=Image.fromarray(img1)
    img2=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
    img2=cv.drawContours(img2,contours,-1,y,1)
    font = cv.FONT_HERSHEY_SIMPLEX
    cv.putText(img2,(f'{len(contours)}'+' pronadenih
kontura'),(0,20),font,1,(255,255,255),1)
    h, w, c = resized.shape
    blank = 0* numpy.ones(shape=(h, w, c), dtype=numpy.uint8)
    img3=cv.drawContours(blank,contours,-1,y)
    cv.putText(img3,(f'{len(contours)}'+' pronadenih
kontura'),(0,20),font,1,(255,255,255),1)
    img2=Image.fromarray(img2)
    img3=Image.fromarray(img3)
    img1=ImageTk.PhotoImage(img1)
    img2=ImageTk.PhotoImage(img2)
    img3=ImageTk.PhotoImage(img3)
    stop = timeit.default_timer()
    execution_time = stop - start
    cv.waitKey(0)

    if panelA is None or panelB is None or panelC is None:
        panelA = Label(window_2,image=img1)
        panelA.image = img1
        panelA.pack(side="top", padx=10, pady=10)
        panelB = Label(window_2,image=img2)
        panelB.image = img2
        panelB.pack(side="right", padx=10, pady=10)
        panelC = Label(window_2,image=img3)
        panelC.image = img3
        panelC.pack(side="left", padx=10, pady=10)

    else:
        panelA.configure(image=img1)
        panelB.configure(image=img2)
        panelA.image = img1
        panelB.image = img2
```

```
        panelC.configure(image=img3)
        panelC.image = img3
    panelA=None
    panelB=None
    panelC=None
    window_4=Toplevel(window)
    window_4.title("Vrijeme proteklo tokom Canny metode")
    window_4.geometry()
    window_4.resizable()
    print(str(execution_time))
    Label4=Label(window_4,text=("Program izvršen u:"+str(execution_time)+ "
            s"),fg="black",bg="white",font=("Arial", 10))
    Label4.pack(padx=10, pady=10)
def opencv_2():
    window_3=Toplevel(window)
    window_3.title("Rezultati thresholdinga")
    window_3.geometry()
    window_3.resizable()
    global panelA
    global panelB
    global panelC
    img=cv.imread(s)
    def rescaleFrame(frame,scale=0.1):
        width=int(frame.shape[1]*scale)
        height =int(frame.shape[0]*scale)
        dimensions=(width,height)
        return cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)
    resized=rescaleFrame(img)
    start_1 = timeit.default_timer()
    gray=cv.cvtColor(resized,cv.COLOR_BGR2GRAY)
    ret,thresh =cv.threshold(gray,140,255,cv.THRESH_BINARY)
    if z==0 and q==0:
        contours,
hierarchies=cv.findContours(thresh,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)
        y=(255,255,255)
    elif z==0 and q==1:
        contours,
hierarchies=cv.findContours(thresh,cv.RETR_LIST,cv.CHAIN_APPROX_NONE)
        y=(255,255,255)
    elif z==0 and q==2:
        contours,
hierarchies=cv.findContours(thresh,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_NONE)
        y=(255,255,255)
    elif z==0 and q==3:
        contours,
hierarchies=cv.findContours(thresh,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)
        y=(255,255,255)
    elif z==1 and q==0:
```

```
        contours, hierarchies=cv.findContours(thresh,cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,255,255)
    elif z==1 and q==1:
        contours, hierarchies=cv.findContours(thresh,cv.RETR_LIST,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,255,255)
    elif z==1 and q==2:
        contours, hierarchies=cv.findContours(thresh,cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,255,255)
    else:
        contours, hierarchies=cv.findContours(thresh,cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE)
        y=(255,255,255)
    img1=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
    img1=Image.fromarray(img1)
    img2=cv.cvtColor(resized,cv.COLOR_BGR2RGB)
    img2=cv.drawContours(img2,contours,-1,y,2)
    font = cv.FONT_HERSHEY_SIMPLEX
    cv.putText(img2,(f'{len(contours)}'+' pronadenih
kontura'),(0,20),font,1,(255,255,255),1)
    h, w, c = resized.shape
    blank = 0* numpy.ones(shape=(h, w, c), dtype=numpy.uint8
    img3=cv.drawContours(blank,contours,-1,y)
    cv.putText(img3,(f'{len(contours)}'+' pronadenih
kontura'),(0,20),font,1,(255,255,255),1)
    img3=Image.fromarray(img3)
    img2=Image.fromarray(img2)
    img1=ImageTk.PhotoImage(img1)
    img2=ImageTk.PhotoImage(img2)
    img3=ImageTk.PhotoImage(img3)
    stop_1 = timeit.default_timer()
    execution_time_1 = stop_1 - start_1
    cv.waitKey(0)
    if panelA is None or panelB is None or panelC is None:
        panelA = Label(window_3,image=img1)
        panelA.image = img1
        panelA.pack(side="top", padx=10, pady=10)
        panelB = Label(window_3,image=img2)
        panelB.image = img2
        panelB.pack(side="right", padx=10, pady=10)
        panelC = Label(window_3,image=img3)
        panelC.image = img3
        panelC.pack(side="left", padx=10, pady=10)

    else:
        panelA.configure(image=img1)
        panelB.configure(image=img2)
```

```
        panelA.image = img1
        panelB.image = img2
        panelC.configure(image=img3)
        panelC.image = img3
    panelA=None
    panelB=None
    panelC=None
    window_5=Toplevel(window)
    window_5.title("Vrijeme proteklo tokom thresholding metode")
    window_5.geometry()
    window_5.resizable()
    print(str(execution_time_1))
    Label5=Label(window_5,text=("Program izvršen u:"+str(execution_time_1)+ "
s"),fg="black",bg="white",font=("Arial", 10))
    Label5.pack(padx=10, pady=10)
window.title('Detekcija kontura')
window.geometry()
window.config(background="white")

Label1=Label(window,text="Pronalaženje kontura objektima na
slici",width=50,height=4,fg="black",font=("Arial", 15))
Label2=Label(window,text="Metoda aproksimacije
kontura:",fg="black",bg="white",font=("Arial", 10))
Label3=Label(window,text="Metoda vraćanja
kontura:",fg="black",bg="white",font=("Arial", 10))

Button1=Button(window,text="Odabir slike",command=browse,height=2, width=30)
Button2=Button(window,text="Metoda pomoću Canny rubova",command=opencv_1)
Button3=Button(window,text="Metoda pomoću thresholdinga",command=opencv_2)

panelA=None
panelB=None
panelC=None

Label1.pack(side="top")
Button1.pack(padx=10, pady=10)
Label2.pack(padx=10, pady=10)
Pad.pack(padx=10, pady=10)
button3.pack(padx=10, pady=10)
Label3.pack(padx=10, pady=10)
Pad_1.pack(padx=10, pady=10)
button4.pack(padx=10, pady=10)
Button2.pack(padx=10, pady=10)
Button3.pack(padx=10, pady=10)
window.mainloop()
```