Programska aplikacija za detekciju objekata temeljem boje i oblika

Batinović, Daria

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Daria Batinović

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Doc. dr. sc. Tomislav Stipančić, dipl. ing. Daria Batinović

Zagreb, 2021.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru doc. dr. sc. Tomislavu Stipančiću na ukazanom povjerenju i pruženoj pomoći tijekom izrade rada.

Zahvaljujem svojoj obitelji i prijateljima na razumijevanju i ogromnoj podršci tijekom studiranja koja mi je, uz Božju pomoć, bila izvor snage i ustrajnosti u učenju.

Posebno hvala mojoj sestri Ani koja je od početka bila moj veliki oslonac, a zajedno s Krešom uvijek bila vjetar u leđa.

Daria Batinović

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 / 21 - 6 / 1**	
Ur.broj: **15 - 1703 - 21 -**	

ZAVRŠNI ZADATAK

Student: **Daria Batinović** JMBAG: **0035212466**

Naslov rada na hrvatskom jeziku: **Programska aplikacija za detekciju objekata temeljem boje i oblika**

Naslov rada na engleskom jeziku: **Software application for object detection based on color and shape**

Opis zadatka:

Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatsko prepoznavanje objekata koristeći vizijske senzore smještene u stvarnu okolinu. Prepoznavanje je moguće temeljiti na različitim značajkama objekata te na različitim metodama iz područja strojnog vida.

U okviru završnog rada je potrebno:

- proučiti metode i algoritme za detekciju objekata u stvarnom vremenu,
- upoznati se s OpenCV programskom bibliotekom otvorenog koda te koristiti prikladne metode u svrhu prepoznavanja objekata na temelju boje i oblika,
- analizirati učinkovitost razvijenog modela u ovisnosti o trenutnim svjetlosnim uvjetima u prostoriji.

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:	Datum predaje rada:	Predviđeni datumi obrane:
6. svibnja 2021.	**2. rok (izvanredni):** 5. srpnja 2021. **3. rok:** 23. rujna 2021.	**2. rok (izvanredni):** 9.7.2021. **3. rok:** 27.9. – 1.10.2021.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
T	-	Prag objekta i pozadine
g	-	Vrijednost piksela
$\boldsymbol{G}$	-	Gradijent rubova
$\boldsymbol{G}_{\mathrm{x}}$	-	Gradijent u vodoravnom smjeru
$\boldsymbol{G}_{\mathrm{y}}$	-	Gradijent u vertikalnom smjeru
$\boldsymbol{A}$	-	Matrica koja sadrži elemente ulazne slike
θ	°	Kut smjera gradijenta
ρ	m	Udaljenost pravca od ishodišta
α	°	Nagib normale pravca s obzirom na x os
x, y	-	Koordinate sustava

SAŽETAK

Otkrivanje i detekcija objekata važan je zadatak u mnogim aplikacijama za računalni vid. Takve aplikacije postigle su ogroman napredak zahvaljujući razvoju novih algoritama umjetne inteligencije i većom dostupnošću potrebne računalne učinkovitosti pa se primjenjuju za sve više zadataka detekcije i prepoznavanja. Cilj ovog završnog rada je, koristeći prikladne metode i algoritme iz područja strojnog vida, izraditi programsku aplikaciju za detekciju objekata temeljem boje i oblika. Razvijeno programsko rješenje temelji se na OpenCV biblioteci implementiranoj kroz Python programski jezik, a na koncu se analizira učinkovitost razvijenog modela u ovisnosti o trenutnim svjetlosnim uvjetima.

Ključne riječi: detekcija objekata, boja, oblik, OpenCV, Python

SUMMARY

Object recognition and detection is an important task in many computer vision applications.

Such applications have made tremendous progress thanks to the development of new artificial intelligence algorithms and better availability of the required computational efficiency. Therefore, they are being applied to more and more detection and recognition tasks. The aim od this paper is to develop an android application for object detection, based on color and shape, using appropriate methods and algorithms from the field of machine vision. The developed software solution is based on the OpenCV library implemented through the Python programming language. Finally, the efficiency of the developed model is analyzed depending on the current lighting conditions.

Key words: object detection, color, shape, OpenCV, Python

1. UVOD

Od najranijih početaka ljudske civilizacije čovjekovo je djelovanje bilo usmjereno tehnološkom napretku koji se danas posebno očituje u razvoju računalnih znanosti. Skupu takvih znanosti pripada i računalni vid, odnosno područje umjetne inteligencije čiji je glavni cilj izrada umjetnih sustava koji dohvaćaju i obrađuju informacije iz slike. Automatska detekcija i prepoznavanje objekata na digitalnim slikama predstavlja najveće izazove s kojima se trenutno suočava područje računalnog vida. Ipak, zahvaljujući algoritmima i metodama za detekciju koji su razvijeni ili na kojima se još uvijek radi, svjedoci smo brojnih mogućnosti koje nam donosi računalni vid, a koje polako postaju naša svakodnevnica: autonomna vozila, mobilni telefoni koji se otključavaju uz pomoć algoritama za prepoznavanje lica, rekonstrukcija slika iz projekcija (CT), nadzor i mjerenje proizvodnih procesa, očitavanje otisaka prstiju i mnoge druge.

1.1. Motivacija

Razlikovati jabuku od banane, crveni automobil od plavog ili žensku osobu od muške, za čovjeka je vrlo jednostavno. Međutim, za računalo je taj zadatak prilično težak i obuhvaća puno metoda i algoritama umjetne inteligencije. Vrhunske tehnološke tvrtke kao što su Amazon, Google, Microsoft i Facebook ulažu milijarde dolara u istraživanje računalnog vida kako bi poboljšale i olakšale korištenje svojih aplikacija. Motiviran trenutnim istraživanjima, ovaj rad obrađuje identifikaciju objekata temeljem boje i oblika na dvodimenzionalnoj slici. Upravo su ta dva atributa, boja i oblik, fokus svakog sustava za prepoznavanje objekata. Baš zbog toga u ovom radu razvijena je programska aplikacija koja će na temelju ta dva atributa prepoznavati i razlikovati objekte.

1.2. Struktura rada

Rad je strukturiran na način da je prvo dana teorijska osnova u kojoj su opisani najvažniji pojmovi za zadanu detekciju. Nakon toga je prikazana izrada same programske aplikacije te su na kraju prikazani eksperimentalni rezultati i izveden je zaključak.

U drugom poglavlju dane su kratke teorijske osnove pojmova koji su potrebni za shvaćanje programske aplikacije.

U trećem i četvrtom poglavlju opisane su detekcija temeljem boje i detekcija oblika te su naglašene one metode koje se koriste u izradi same aplikacije. Za svaki pojam dana je definicija, osnovni princip i područja primjene.

U petom poglavlju opisuje se izrada tražene programske aplikacije u programskom jeziku Python koristeći OpenCV biblioteku. Poglavlje je podijeljeno na tri cjeline: prvo se detektiraju samo oblici, potom se algoritmu dodaje i detekcija boja što rezultira detekcijom objekata

temeljem boje i oblika. Analiziraju se učinkovitost i efikasnost aplikacije u smanjenim svjetlosnim uvjetima u prostoriji. U svrhu provjere valjanosti programske aplikacije prikazani su rezultati detekcije temeljem boje i oblika na slici sa savršenim oblicima te na slici smanjene kvalitete. Na kraju su analizirani rezultati razvijenog modela u tamnijim svjetlosnim uvjetima u prostoriji i detekcija objekata u stvarnom vremenu.

Peto poglavlje ujedno je i zaključak samog završnog rada.

2. TEORIJSKA OSNOVA RADA

2.1. Strojni vid

Iako se smatraju istom tehnologijom, računalni vid i strojni vid različiti su termini koji se koriste za tehnologije koje se poklapaju. Računalni vid u širokom pogledu odnosi se na snimanje i automatizaciju analize slike s naglaskom na funkciju analize slike u širokom rasponu teorijskih i praktičnih primjena. Strojni vid tradicionalno se odnosi na korištenje računalnog vida u industrijskoj ili praktičnoj primjeni, gdje je potrebno izvršiti određene funkcije ovisno o analizi koja je odrađena od strane vizualnog sustava.[1]

2.2. Računalni vid

Računalni vid je područje računarske znanosti koje razvija teorijske i algoritamske temelje pomoću kojih se korisna informacija o svijetu može automatski izlučiti i analizirati i to iz slika, skupa slika ili iz slijeda slika uporabom računala opće namjene ili specijaliziranog računala. Računalni vid daje mogućnost računalu da „vidi“, to jest razumije što se nalazi na fotografiji ili videu. Samo neki od zadataka računalnog vida su prepoznavanje objekata, lica ili znakova u prostoru, analiza pokreta i rekontrukcija događaja. Razvojem računalnog vida dolazi i do razvoja potpuno novih aplikacija za svakodnevnu upotrebu, primjerice aplikacije za praćenje ili prepoznavanje objekata, rekonstrukciju slike i sl. Cilj sustava računalnog vida je oblikovanje modela stvarnog svijeta na temelju slika na način kako je to svojstveno ljudima. Za čovjeka je iščitavanje i interpretacija slike vrlo jednostavna i on može lako prepoznati i prebrojati osobe na slici, pa čak i odrediti njihova emocionalna stanja. Računalni vid, iako još uvijek u izrazito sporije i s puno grešaka, preslikava mogućnosti ljudskog vida pomoću digitalnih slika koristeći tri glavne procesne komponente u sljedećem redoslijedu[2]:

- Pribavljanje slike (eng. *Image acquisition*) ,
- Obrada slike (eng. *Image processing*),
- Analiza i razumijevanje slike (eng. *Image analysis and understanding*).

Cilj obrade slike je stvaranje nove slike na temelju postojeće, dok je cilj računalnog vida razumijevanje sadržaja sa slike. Obrada slike može se koristiti za procesiranje ulaznih podataka koji se upotrebljavaju kod računalnog vida.

Bitno svojstvo koje ljudi posjeduju je sposobnost donošenje odluka i davanje smisla onome što vide u stvarnom svijetu. Omogućavanje računalima i strojevima dijela ovog vizualnog razumijevanja postiže se kroz sljedeće postupke [2]:

• Klasifikacija objekata – uključuje treniranje modela na skupu podataka određenih objekata i zatim model klasificira nove objekte u jednu ili više kategorija određenih treningom.

• Identifikacija objekata – postupak u kojem će model prepoznati specifične instance objekata.

Prvi eksperimenti s računalnim vidom počeli su 1960-tih, a prva komercijalna upotreba je bila 1970. godine sa svrhom da se razlikuje tiskani i ručno pisani tekst. U kasnim 1960-tim

godinama računalni vid nastaje na sveučilištima koji su postali pioniri umjetne inteligencije. Cilj je bio imitirati ljudski vizualni sistem, kao kamen temeljac uspostavljanja inteligentnog ponašanja kod robota.

Gledajući povijest razvoja računalnog vida važno je napomenuti da se zbog širokog spektra potencijalnih aplikacija često spaja s drugim usko povezanim područjima. Tu spadaju obrada slike, fotogrametrija, određivanje poza objekata u 3D-u i računalna grafika. Neki od glavnih algoritama koji su rezultat istraživanja u području računalnog vida danas su dio biblioteka kao što su OpenCV ili Keras [1].

2.3. Programski jezik Python

Python je programski jezik opće namjene, jednostavan, interpretiran i objektno orijentiran jezik kojeg je smislio i stvorio Guido van Rossum 1991. godine. Interpreterski programski jezici su jezici kod kojih se izvorni kod izvršava direktno uz pomoć interpretera, tj. kod ovakvih tipova programskih jezika nema potrebe za kompiliranjem prije izvršavanja, tj. prevođenjem u izvršni oblik. Sadrži module, iznimke, dinamičko povezivanje, dinamičke tipove podataka visoke razine i klase. Ima sučelje za mnoge sistemske pozive i biblioteke, a proširiv je u C ili C++. Također se može koristiti kao produženi jezik za aplikacije kojima je potrebno programibilno sučelje. Python je multiplatformalan jezik tj. izvršava se na Linux-u, macOS-u i Windows-u [3].

Python je odabran za ovaj rad jer je veoma rasprostranjen, besplatan je i jednostavan, te ima opsežnu količinu modula i biblioteka. Štoviše, trenutno je najpopularniji programski jezik za strojno učenje zbog svojih izvrsnih mogućnosti: efikasna i jasna sintaksa, jednostavna integracija s drugim programskim jezicima i najvažnije, opsežna podrška za biblioteke otvorenog koda. Zbog izrazito velike standardne knjižnice za jezik implementacije odabran je Python 3.9. U testiranju i prevođenju programskog koda korišteno je razvojno okruženje JetBrains PyCharm.

2.4. Korištene Python datoteke

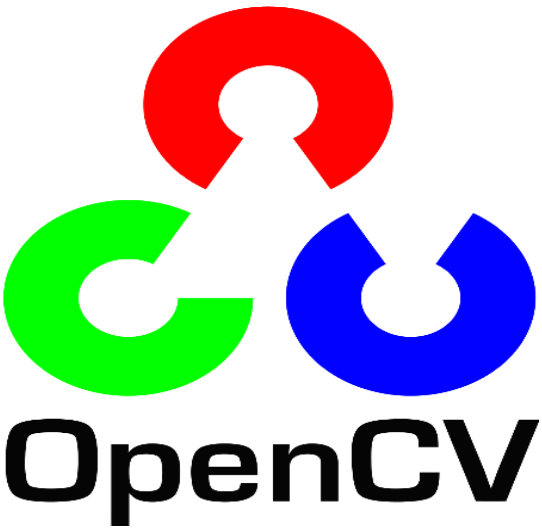
Biblioteka je zbirka prethodno kombiniranih kodova koja se može koristiti iterativno za smanjenje vremena potrebnog za programiranje. Neke od najvažnijih i najčešćih Python datoteka su: Matplotlib, Pandas, Pyglet, SciPy, PyGame, OpenCV, Numpy itd. Posljednje dvije nabrojane će biti korištene u izradi programske aplikacije, stoga je u nastavku dana njihova definicija i opis.

2.4.1. OpenCV

OpenCV (eng. *Open Source Computer Vision*) je najveća i najpopularnija biblioteka računalnog vida i strojnog učenja otvorenog koda u svijetu. Njome se služe na tisuće tvrtki, proizvoda i uređaja, a najviše se koristi za obradu slika i videozapisa radi detekcije oblika,

objekata, teksta, boje itd. OpenCV biblioteka je izdana pod BSD licencom i stoga je besplatna za akademsku i komercijalnu upotrebu [4]. Logotip biblioteke prikazan je na slici 1. Kada je 1999. godine projekt službeno objavljen, glavna inicijativa bila mu je unaprijediti procesorski zahtjevne poslove kao dio serije projekata na području računalne grafike. Prvo alfa izdanje biblioteke bilo je 2000. godine iza koje slijedi pet beta izdanja u razdoblju od 2001. do 2005. godine. Prva 1.0 verzija OpenCV-ja izdana je 2006. godine, a 2008. godine je dobila korporativnu podršku kompanije Willow Garage. Drugo značajno izdanje OpenCV2 objavljeno je 2009. godine i uključuje nove funkcije te bolje implementacije postojećih, u smislu performansi (ponajviše na višejezgrenim sustavima). Službena izdanja danas se objavljuju svakih 6 mjeseci, a u kolovozu 2012. godine podršku nad OpenCV-jem je preuzela neprofitna organizacija OpenCV.org [5].

Napisana je u C i C++ programskim jezicima i podržava korištenje više operacijskih sustava kao što su: MS Windows, Linux, Mac OS i Android. Sadrži sučelja za C++, Python i MATLAB programske jezike. Ističe se s više od 2500 algoritama namjenjenih računalnom vidu, uključujući i algoritme strojnog učenja. Algoritmi se mogu koristiti za otkrivanje i prepoznavanje lica, detekciju objekata, klasificiranje ljudskih kretnji u videozapisima, praćenje objekata u pokretu, izvlačenje 3D modela predmeta, povezivanje niza slika u jednu i sl. OpenCV koristi više od 47 tisuća ljudi, a broj preuzimanja biblioteke prelazi 7 milijuna. Koriste ga mnoge tvrtke kao što su Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota i mnoge istraživačke grupe [5].



Slika 1. OpenCV biblioteka [6]

2.4.2. Numpy

NumPy je Python biblioteka koja se koristi za rad s poljima (eng. *array*) koju ostale biblioteke kao što je Tensorflow koriste interno za provedbu niz operacija na tenzorima. Značajke koje opisuju NumPy su:

- Interaktivnost i jednostavna upotreba ,
- Jednostavna implementacija kompleksnih matematičkih izraza i operacija ,

- Intuitivnost ,
- Otvoreni kod .

Uobičajene upotrebe ove biblioteke su kod izražavanja slika, zvučnih valova i drugih binarnih tokova kao polja realnih brojeva u N-dimenzionalnom prostoru [7].

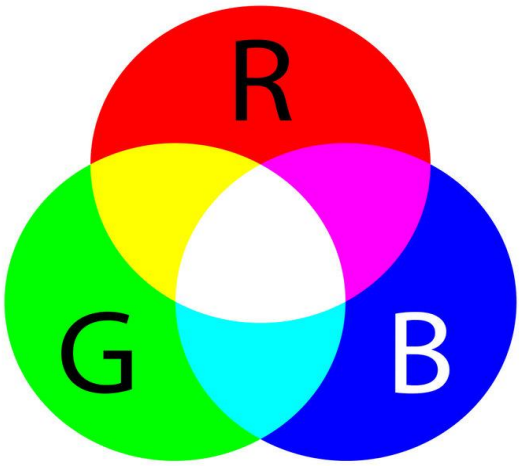
3. DETEKCIJA TEMELJEM BOJE

3.1. Modeli boja

Modeli boja odgovaraju specifikaciji trodimenzionalnog koordinatnog sustava i vidljivog podskupa u tom koordinatnom sustavu u kojem se nalaze sve boje iz određenog područja boja. Svrha modela boje je prikladan način specifikacije boja iz određenog područja boja. Razvijene su dvije skupine modela boja. Prva skupina obuhvaća modele koji su sklopovski orijentirani. Primjeri sklopovski orijentiranih modela boje su RGB (Red, Green, Blue) za CRT monitore, te CMY (Cyan, Magenta, Yellow) i CMYK (Cyan, Magenta, Yellow, Black) za tiskanje u boji. Druga skupina modela obuhvaća korisnički orijentirane modele koji su bliže načinu raspoznavanja svojstava boje od strane korisnika. Primjeri takvih modela su HSV (Hue, Saturation, Value) i HSL (Hue, Saturation, Lightness). Među pojedinim modelima moguća je pretvorba specifikacije boje npr. specifikacija boje u HSV modelu može se pretvoriti u specifikaciju boje u RGB modelu [8].

3.1.1. RGB model

RGB model boja (slika 2) se sastoji od tri primarne boje: crvene, zelene i plave, a miješanjem te tri boje dobivamo sekundarne i tercijalne. Ovaj model je aditivni, svjetlosni model miješanja boja i definiran je u Kartezijevom koordinatnom sustavu. RGB model najčešće se koristi u elektroničkim sustavima kao što su: digitalne i video kamere, pametni telefoni, skeneri, monitori i slično.

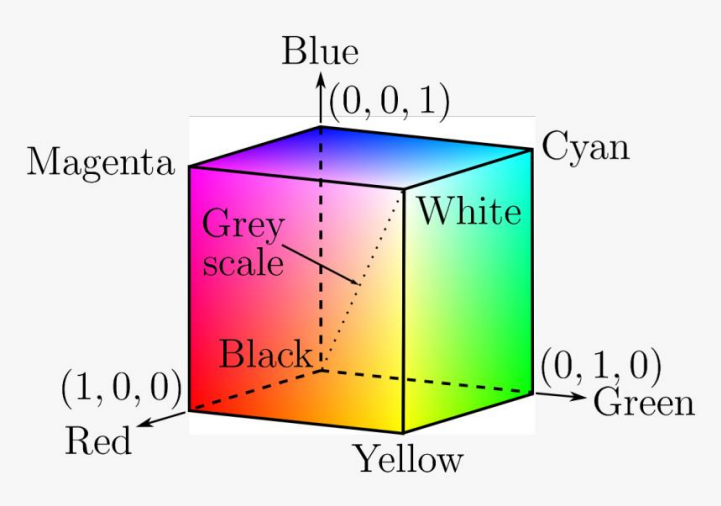


Slika 2. RGB model boja [9]

Svaka boja u RGB modelu se može zapisati u uređenoj trojci (R, G, B) koju redom čine određene vrijednosti crvene, zelene i plave svjetlosti. Vrijednosti te tri komponente imaju raspon od nule do neke maksimalne vrijednosti M koja ovisi o načinu numeričkog označavanja

RGB modela. Uzimajući to u obzir pokazalo se da će se bilo koja odabrana točka morati nalaziti unutar kocke čija je dužina svake stranice jednaka maksimalnoj vrijednosti M. Svaka koordinatna os kocke predstavlja nijanse primarne boje: x os predstavlja zelenu boju, y os plavu, a z os crvenu boju. Maksimalna vrijednost može biti označena postocima, decimalnim brojevima i cijelim brojevima raspona 2^n [9].

Ako se koristi decimalni zapis, maksimalna vrijednost je 1.0, ako se koristi zapis u postocima, maksimalna vrijednost je 100, a ako se koristi 8-bitni zapis maksimalna vrijednost iznosi 255. Kao što je vidljivo na slici 3, koordinata (0,0,0) nalazi se na jednom od vrhova kocke i u toj točki se nalazi crna boja najveće zasićenosti. Suprotno od vrha koordinata (0,0,0) nalazi se vrh čije su koordinate jednake maksimalnoj vrijednosti (M,M,M). U toj točki nalazi se bijela boja, a dijagonala koja povezuje ta dva vrha sadrži različite nijanse sive boje budući da je R=G=B. Ako neku boju želimo prikazati u postocima, 0 bi označavala nedostatak te boje dok bi 100 označavala potpunu zasićenost te boje. Na taj bi način (100, 100, 0) dalo žutu boju, a (100, 0 , 0) crvenu [9].

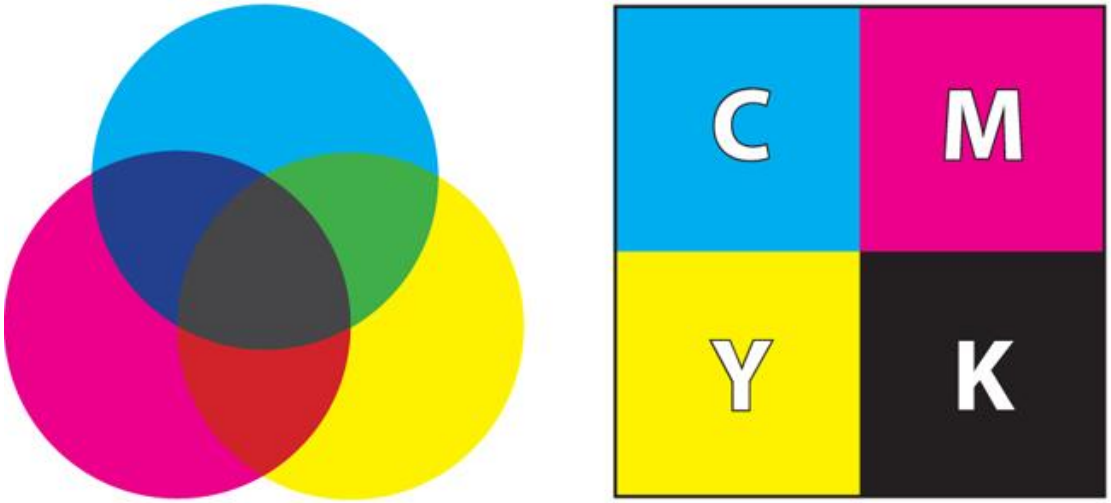


Slika 3. Geometrijska interpretacija RGB modela [10]

3.1.2. CMYK model

Za razliku od RGB modela, gdje se boja na monitoru prikazuje pomoću aditivne kombinacije tri primarne svjetlosti – crvene, zelene i plave boje, CMYK koristi subtraktivnu sintezu. Ovaj model primjenjuje se u uređajima za tiskanje koji nanose pigmente boja na papir. CMYK prostor boja manji je od RGB prostora što znači da tiskarski stroj ne može reproducirati sve boje koje može prikazati monitor Zamišljeno je da je s RGB primarnim bojama osvjetljen bijeli papir za tisak. Kombinacija triju primarnih svjetlosti rezultirat će bijelom bojom. Ako su pak kombinirane dvije primarne svjetlosti, zelena i plava, rezultat će biti cijan boja. Kombinirana crvena i plava svjetlost dat će magentu, a crvena i zelena svjetlost rezultirat će u žutoj boji. Kombinacija dvije primarne svjetlosti RGB modela dat će primarne subtraktivne veličine CMY modela – cijan, magentu i žutu kao što je vidljivo na slici 4. CMY koristi subtraktivnu sintezu

što znači da se boje oduzimaju od bijele svjetlosti umjesto da se dodaju u crnilo kako bi se njihovom kombinacijom stvorila određena boja. Za svaku primarnu boju CMY modela može se reći da je nastala oduzimanjem određene boje iz bijele svjetlosti. Cijan će se dobiti ukoliko od bijele svjetlosti oduzmemo crvenu svjetlost, magentu ako od bijele svjetlosti oduzmemo zelenu svjetlost, te žutu ako od bijele svjetlosti oduzmemo plavu svjetlost. Stoga se može reći da su redom cijan, magenta i žuta komplementarne boje crvenoj, zelenoj te plavoj.

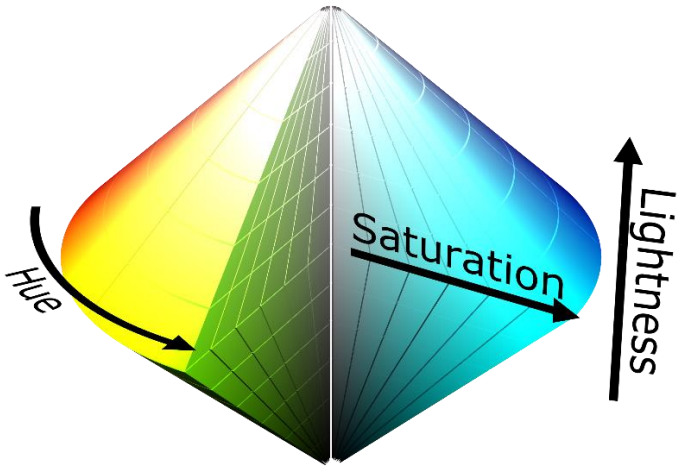


Slika 4. CMYK model boja [11]

Kombinacijom slojeva cijan, magente i žute boje dobije se crna boja jer apsorbiraju svu svjetlost. Međutim, problem CMY modela jest taj da se kombinacijom svih primarnih boja tog modela ne može dobiti vjerodostojan prikaz crne boje. Kako kombinacija tri primarne tinte jako navlažuje papir na kojem se tiska i time se produžuje vrijeme njegova sušenja i konačni prikaz crne boje nije zadovoljavajuć, CMY modelu dodaje se i četvrta komponenta koja predstavlja crnu boju. Kako slovo B već označava plavu boju, za crnu boju je dogovoreno slovo K[9].

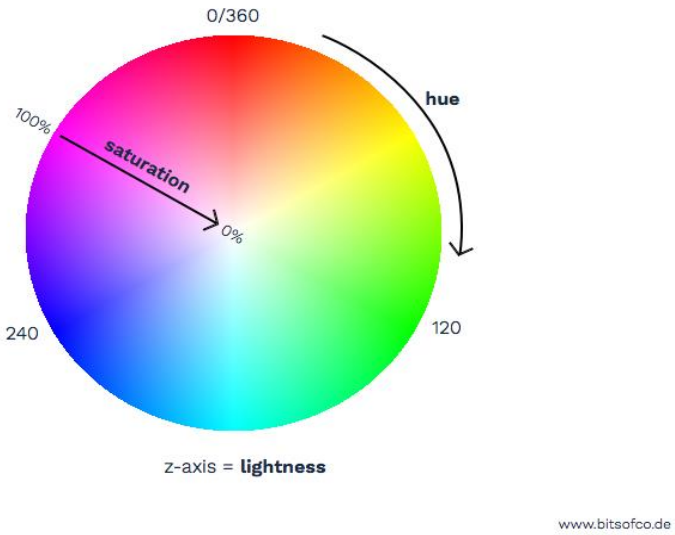
3.1.3. HSL model

HSL model grafički je predstavljen s dva međusobno spojena stošca (slika 5) čiji se vrhovi nalaze sa suprotnih strana. Dva vrha tog trodimenzionalnog tijela definiraju crnu i bijelu boju. Jedna od glavnih prednosti HSL modela u odnosu na RGB model je ta što se komplementarne boje kod HSL modela nalaze jedna nasuprot drugoj što čini cijeli sustav intuitivnijim.



Slika 5. Geometrijska interpretacija HSL modela [11]

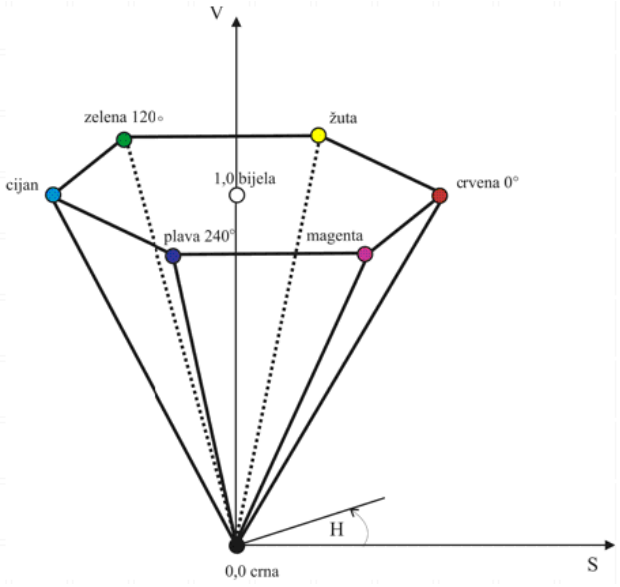
Boja se definira s tri faktora: nijansa, zasićenje i svjetloća. Slovo H predstavlja nijansu boje koja je određena pozicijom na kružnoj paleti boja odnosno kutom između crvene boje koja označava ishodište kuta tj. 0° te zadane boje. Kut se računa u smjeru obrnutom od kazaljke na satu (slika 6). Slovo S predstavlja zasićenost i određena je udaljenošću od centra kružne palete boja. Najveća zasićenost je uz rub kružnice, a najmanja u centru. Zasićenost se iskazuje postotkom, pri čemu je 100% zasićena boja. Naposljetku, slovo L predstavlja intezitet svjetlosti reflektirane od površine i također se iskazuje u postocima. Približavanjem gornjem vrhu stošca svjetlina kružne palete boja će se povećavati, dok će se u suprotnom smjeru svjetlina smanjivati, odnosno povećavat će se tama. Popunjena kružna paleta boja može se zamisliti kao površina koja je nastala nakon što se stožac horizontalno odrezao u određenoj točki [9].



Slika 6. Popunjena kružna paleta boja [9]

3.1.4. HSV / HSB model

HSV model boja, još poznat i kao HSB(Hue, Saturation, Brightness) model, vrlo je sličan HSL modelu. Kako je HSL prostor boje definiran stošcem, tako je raspon vrijednosti zasićenosti boje sve manji prema vrhovima stošca. To predstavlja problem u računalnoj grafici budući da se HSL model koristi za birač boja u programima grafičke obrade, stoga se umjesto stošca koristi valjak kao geometrijska interpretacija HSL modela. Takav pojednostavljeni model osigurava da svaka kružna paleta boja ima jednak promjer, no javljaju se distorzije vezane uz promjenu svjetloće boja. Da bi se to spriječilo razvijen je 1978. godine HSV model koji koristi drukčiju skalu svjetloće na kojoj nema takve distorzije(slika 7). Ovaj model definira iste komponente kao i HSL model, a razlog drukčijeg naziva je identifikacijske prirode [9].



Slika 7. Geometrijska interpretacija HSV modela [10]

HSV model boja definiran je s tri koordinate: tonom boje (engl. hue), zasićenjem boje (engl. saturation) i svjetlinom boje (engl. value, intensity, brightness). Boje u ovom modelu prikazane su u podskupu prostora kojeg omeđuje izvrnuta šesterostrana piramida. Ton boje definira se u bazi piramide kutom od 0° do 360°, gdje je crvena boja ishodište kuta. Kut se mjeri od crvene boje u smjeru suprotnom od kazaljke na satu. Zasićenost boje ima vrijednost od 0% do 100%. Svjetlina boje ima vrijednost od 0% do 100% [9].

Vrlo često se šesterostrana piramida izobličuje u valjak što rezultira u prostoru boja u kojem se najsvjetliji tonovi boja nalaze na vrhu valjka koji je postavljen vertikalno, a na dnu tamniji tonovi. Na samom dnu valjka je crna boja.

U izradi programske aplikacije korišten je HSV model boja čiji je raspon boja dan na slici 8.

Color	Minimum Value			Maximum Value		
	H	S	V	H	S	V
Black	0	0	0	0	0	0
White	0	0	236	0	0	255
Gray	0	0	1	255	50	235
Blue	0	50	0	41	255	255
Green	42	0	0	88	255	255
Yellow	89	50	0	97	255	255
Orange	98	50	0	119	255	210
Beige	99	0	0	120	11	255
Red	120	140	0	128	255	255
Pink	120	0	241	149	255	255
Violet	150	0	0	200	255	255

Slika 8. Raspon boja u HSV modelu [12]

4. DETEKCIJA TEMELJEM OBLIKA

Kod detekcije objekata temeljem oblika jako je važno razgraničiti „objekte od interesa“ od ostatka slike. Tehnika kojom se izdvajaju objekti od interesa se naziva segmentacija. Metode segmentacije koje će se koristiti u ovom radu kod detekcije geometrijskih likova su:

- Tresholding - metoda određivanja praga tj. izdvajanje cijelog objekta od pozadine,
- Edge-based segmentacija - metoda pronalaska rubova. U ovom slučaju, za detekciju rubova koristit će se jedan od najpopularnijih algoritama- Cannyjev algoritam.

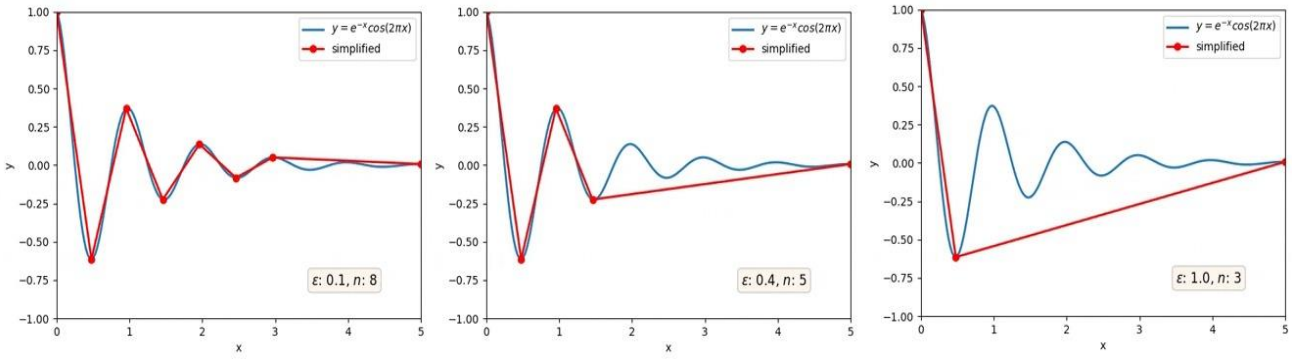
Metoda kojima se razgraničava objekt od interesa od svoje pozadine ima više, ali ovaj završni rad opisat će i analizirati ove dvije navedene.

4.1. Aproksimacija kontura

Aproksimacija kontura je metoda kojom se neka krivulja aproksimira konturom s manjim brojem vrhova, ovisno o preciznosti koju odredimo. OpenCV nudi ugrađenu funkciju koja aproksimira poligonalne krivulje: approxCurve = cv2.approxPolyDP(curve, epsilon, closed), gdje je:

- curve - kontura koju želimo aproksimirati,
- epsilon - najveća udaljenost između krivulje koju želimo aproksimirati i njezine aproksimacije tj. parametar točnosti,
- closed - ako je *True* onda je aproksimirana krivulja zatvorena, u suprotnom je otvorena.

Ova funkcija vraća približnu konturu iste vrste kao i ulazna krivulja, tj. ona koju želimo aproksimirati. Implementacija te funkcije temelji se na Douglas-Peuckerovom algoritmu koji pojednostavljuje krivulju sastavljenu od segmenata linije na sličnu krivulju s manje točaka. Algoritam definira konturu na temelju najveće udaljenosti između izvorne krivulje i pojednostavljene krivulje (Hausdorffova udaljenost između krivulja)[13]. Pojednostavljena krivulja sastoji se od podskupa točaka koje su definirale izvornu krivulju (slika 9). Bio je to jedan od prvih uspješnih algoritama razvijenih za kartografsku generalizaciju.



Slika 9. Prikaz aproksimacije krivulje uz mijenjanje parametra epsilon [14]

4.1.1. Tresholding - određivanje praga

Tresholding je najjednostavniji segmentacijski proces u kojem odvajamo područje slike koje odgovara objektima koje želimo analizirati. Mnogi objekti ili regije slike imaju karakterističnu konstantnu refleksiju ili apsorbciju svjetla na njihovim površinama što nam omogućuje da odredimo neki konstantni iznos tj. prag koji će razdvajati objekte od pozadine. Razdvajanje se temelji na promjeni inteziteta između piksela objekta i piksela pozadine. Da bi se razlikovali pikseli od važnosti, provodi se usporedba svake vrijednosti inteziteta piksela s obzirom na prag. Nakon što se pravilno odvoje važni pikseli, tj. pikseli objekta koji se trebaju detektirati postavlja se određena vrijednost za identifikaciju (pikselima objekta dodijeli se vrijednost crne ili bijele boje, a pikselima pozadine suprotna vrijednost). Određivanje praga je računski nezahtjevna i jednostavna metoda koja ima široku primjenu za različite zadatke. Vrlo se jednostavno izvodi u realnom vremenu uz pomoć adekvatne računarske opreme [15].

Zadatak algoritma je provjeriti sve piksele originalne slike. Algoritam pretvara ulaznu sliku u binarnu (segmentiranu) sliku i to na način:

$$g(i,j) = 1 \;\; za\; f(i,j) \geq T, \qquad (1)$$

$$= 0 \;\; za\; f(i,j) < T, \qquad (2)$$

gdje je T prag, $g\,(i,j) = 1$ su objekti, a $g\,(i,j) = 0$ pozadina ili obrnuto.

Vrlo je bitno odrediti dobar prag jer se samo u rjetkim slučajevima događa da je jedan prag dobar za cijelu sliku (globalni prag). Može se dogoditi da i kod dosta jednostavnih slika dolazi do varijacija koje mogu biti uzrokovane nejednakim osvjetljenjem i nizom drugih faktora. Ovo je prigodna metoda za slučaj detekcije ovog završnog rada jer se objekti međusobno ne dodiruju i jasno se razlikuju od pozadine [15].

4.1.2. Cannyjev algoritam

Canny algoritam za detekciju rubova koristi višekoračni algoritam za otkrivanje širokog raspona rubova na slikama. Razvio ga je John F. Canny 1986. godine i ujedno izradio računsku teoriju otkrivanja rubova [16]. U svom radu Canny je predložio novu metodu za detekciju rubova koja se može podijeliti na ove korake:

- filtriranje pomoću Gaussovog filtra kako bi se uklonio šum sa slike
- pronalazak gradijenata slike
- primjena ne-maksimalnog suzbijanja (eng. *non-maximum suppression*) tj. suzbijanja rubova čija vrijednost nije najveća u usporedbi sa susjednim rubovima kako ne bi bilo lažnih rubova
- praćenje ruba histerezom (eng. *edge tracking*)

Posljednji korak podrazumijeva suzbijanje slabih rubova i onih rubova koji to zapravo nisu pored jakih rubova.

U nastavku će se detaljnije opisati koraci algoritma :

- Smanjenje šuma / buke pomoću Gaussovog filtera - budući da je otkrivanje rubova osjetljivo na šum na slici, prvi korak je ukloniti šumove Gaussovim filtrom 5x5 (3). Time zaglađujemo sliku što nam omogućuje da zanemarimo veći dio detalja i umjesto toga se usredotočimo na stvarnu strukturu.

$$\boldsymbol{K} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \qquad (3)$$

- Pronalaženje gradijenta inteziteta slike - pomoću Sobelovog operatora moguće je odrediti gradijent, odnosno usmjerenu promjenu intenziteta elemenata slike, u horizontalnom i vertikalnom smjeru te ukupne jakosti ruba prema izrazima (4), (5) i (6). **G**x i **G**y predstavljaju gradijente u vodoravnom i vertikalnom smjeru, **A** predstavlja matricu koja sadrži elemente ulazne slike i znak * predstavlja operaciju konvolucije. Prema predznacima se može vidjeti kako pozitivan rezultat ukazuje na prelazak iz tamnijeg u svjetlije područje gledajući sliku prema desno po redovima i prema dolje po stupcima. Sukladno tome, negativan rezultat ukazuje na prelazak iz svjetlijeg u tamnije područje, odnosno prelazak iz područja manjeg u područje većeg intenziteta.

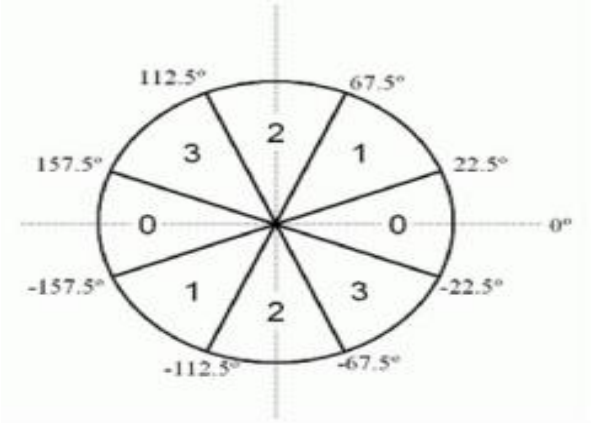
$$\boldsymbol{G}_{x(i,j)} = \boldsymbol{A}_{(i,j)} * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad (4)$$

$$\boldsymbol{G}_{y(i,j)} = \boldsymbol{A}_{(i,j)} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad (5)$$

$$\boldsymbol{G}_{(i,j)} = \sqrt{\boldsymbol{G}_{x(i,j)}^2 + \boldsymbol{G}_{y(i,j)}^2}\ , \qquad (6)$$

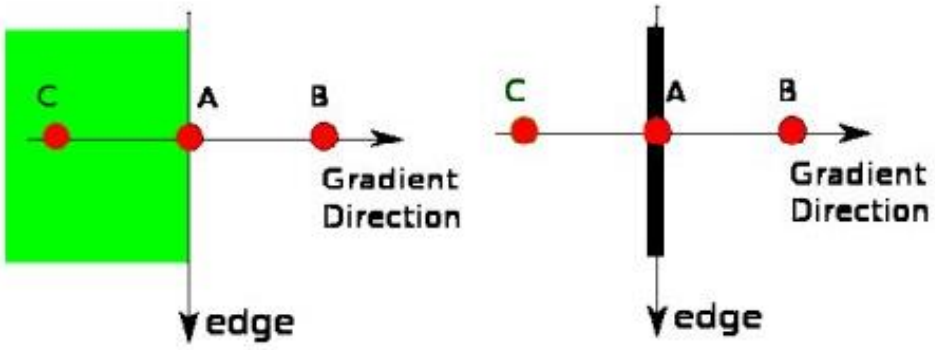
$$\theta = \tan^{-1}\left(\frac{G_{y(i,j)}}{G_{x(i,j)}}\right). \qquad (7)$$

Nakon pronalaska gradijenta, uz pomoć jednadžbe (7) dobiva se kut θ koji je potrebno zaokružiti na jedan od četiri smjera u kojem piksel može imati susjeda. Način na koji se određuju smjerovi prikazan je na slici 10, gdje područja označena brojem 0 odgovaraju kutu od 0°, brojem 1 kutu od 45°, brojem 2 kutu od 90° i brojem 3 kutu od 135° [17].



Slika 10. Određivenje diskretnog smjera ruba [18]

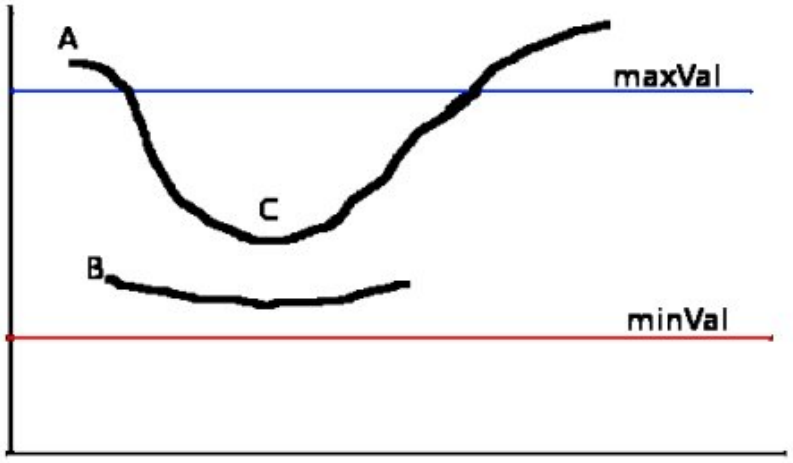
- Ne-maksimalno potiskivanje – vrši se potpuno skeniranje slike kako bi se uklonili svi neželjeni pikseli koji možda ne predstavljaju rub. Ono podrazumijeva usporedbu svakog piksela s njegovim susjedima i zadržavanje samo onih čija je jakost veća od jakosti njegovih susjeda (slika 11). Ostali pikseli se brišu, tj. vrijednost im se postavlja na nulu. Ukratko, rezultat koji se dobije je binarna slika s "tankim rubovima".



Slika 11. Ne-maksimalno potiskivanje [16]

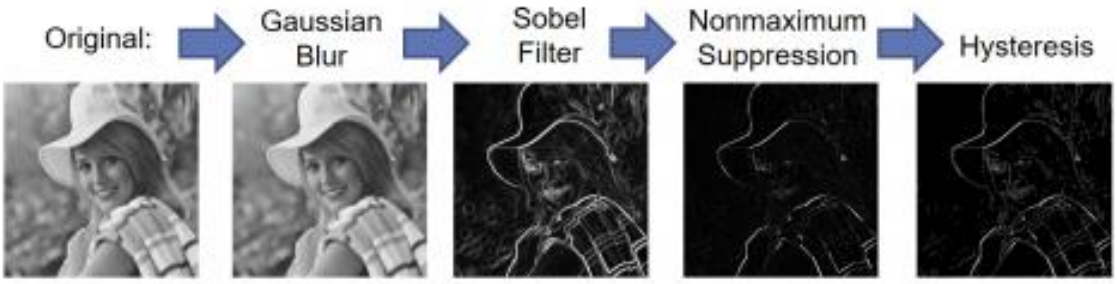
- Praćenje ruba histerezom - ova faza odlučuje koji su od rubova doista pravi rubovi, a koji nisu. Za to su nam potrebne dvije granične vrijednosti, minVal i maxVal. Svi rubovi čiji je gradijent inteziteta iznad gornje granice (maxVal) se zadržavaju i smatraju se jakim rubovima, dok se svi oni čija je vrijednost ispod donje granice (minVal) odbacuju. Rubovi koji se nalaze između dviju granica se smatraju slabim rubovima. Oni se zadržavaju ukoliko im je barem jedan od susjeda jaki rub ili su preko drugih slabih rubova vezani na jaki rub. Grafiči prikaz ovog koraka nalazi se na slici 12. Plava linija predstavlja gornju granicu a crvena donju. Rub A će biti zadržan jer mu je vrijednost iznad gornje granice. Rub B će biti obrisan jer nije

povezan niti s jednim jakim rubom, a rub C će biti zadržan jer je povezan s rubom A [16].



Slika 12. Praćenje ruba histerezom [16]

Ova faza također uklanja šumove malih piksela pod pretpostavkom da su rubovi dugačke crte. Dakle, konačno se dobivaju jaki rubovi na slici (slika13).



Slika 13. Osnovni koraci Cannyjevog algoritma [16]

OpenCV stavlja sve gore navedeno u jednu funkciju, cv.Canny(). Prvi argument te funkcije je naša ulazna slika, a drugi i treći argument su minVal i maxVal. Slika 14 pokazuje rezultat svih gore navedenih koraka.

Slika 14. Rezultat primjene Cannyjevog algoritma [17]

4.2. Houghova transformacija

Houghova transformacija je robusna tehnika procjenjivanja parametara temeljena na načelu glasanja. Ovom transformacijom pronalaze se parametri nesavršenih primjeraka zadane klase oblika. Isprva je korištena za detekciju ravnih linija, a kasnije je proširena i na neke kompleksnije parametarske oblike poput kružnica i elipsa. Primjerci klase oblika definiraju se n-torkom parametara. Svaki element slike (piksel) glasa za parametre svih oblika kojima bi mogao pripadati. Glasovi se zbrajaju u akumulacijskom polju čija dimenzionalnost ovisi o broju parametara koji definiraju oblik. Traženi oblici se pronalaze kao maksimumi tog polja [19].

Houghova transformacija sastoji se od tri osnovna koraka [19]:

- Detekcija primitivnih elemenata slike - svaki element slike ne sadrži jednako vrijedne vrijednosti. Zbog toga se prije postupka glasanja trebaju provesti razna filtriranja kako bi se naglasile one informacije koje su potrebne, odnosno prigušile one informacije koje smetaju. Točnije, izdvajaju se elementi koji su prošli određenu selekciju.
- Preslikavanje slike u parametarski prostor - svaki element slike koji je prošao selekciju iz prvog koraka preslikava se u parametarski prostor definiran akumulacijskim poljem. Ovisno o tome koji se oblici detektiraju određena je n-dimenzionalnost akumulacijskog polja. Ako se radi o pravcu tada je akumulacijsko polje dvodimenzionalno, dok je kod elipse akumulacijsko polje petodimenzionalno. Preslikavanja elemenata u parametarski prostor ostvaruje se glasanjem, odnosno povećanjem vrijednosti u akumulacijskom polju čije su koordinate jednake paru parametara traženog oblika.

- Detekcija lokalnih maksimuma - rezultat dobiven drugim korakom je akumulacijsko polje u kojemu se lokalni maksimumi nalaze na koordinatama koje predstavljaju parametre traženog oblika. Jedan od načina izdvajanja maksimuma je izdvajanje svih vrijednosti akumulacijskog polja koje prelaze neku graničnu vrijednost.

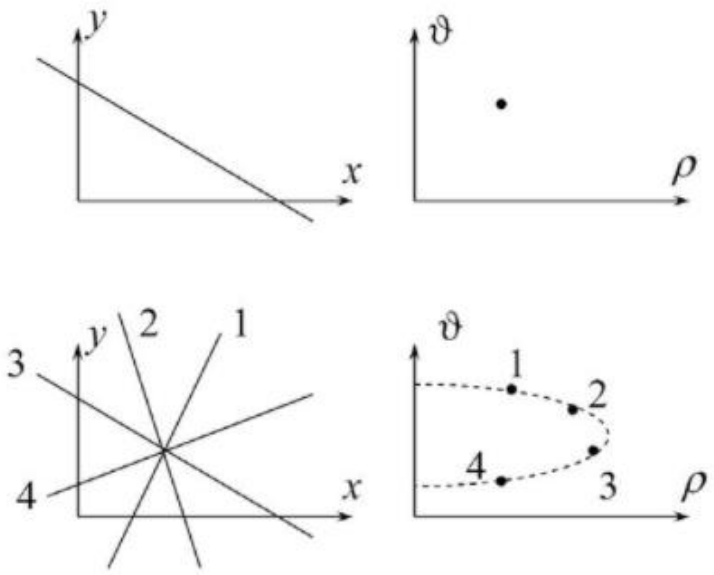
4.2.1. Houghova tranformacija za linije

Linija se predstavlja s dva parametra, nagibom i udaljenosti pravca od ishodišta.
Postavlja se jednadžba pravca u polarnim koordinatama:

$$\rho = x\cos(\alpha) + y\sin(\alpha)\,, \qquad (8)$$

gdje je ρ - udaljenost pravca od ishodišta, a α - nagib normale pravca s obzirom na x os.
Ova jednadžba koristi se umjesto poznate jednadžbe pravca: $y = mx + c$ jer nagib m može imati vrijednosti između - ∞ do +∞, a budući da za transformaciju Houghovi parametri moraju biti ograničeni uzima se jednadžba pravca u polarnim koordinatama [20]. Houghova transformacija pravca je točka u koordinatnom sustavu (ρ, α). Familija pravaca koji prolaze kroz jednu točku se preslikava u skup točaka koje leže na sinusoidi, kao što je prikazano na slici 15.

U OpenCV-ju detekcija linije pomoću Houghove transformacije implementirana je u funkciji cv2.HoughLines() koja uzima sljedeće argumente: rubovi, ρ, α i minimalan broj točaka koje se sjeku za otkrivanje linije. Houghova transformacija linija se primjenjuje u mnogim korisnim aplikacijama u stvarnome svijetu, počevši od skeniranja dokumenata do praćenja traka u prometu kod autonomnih vozila.



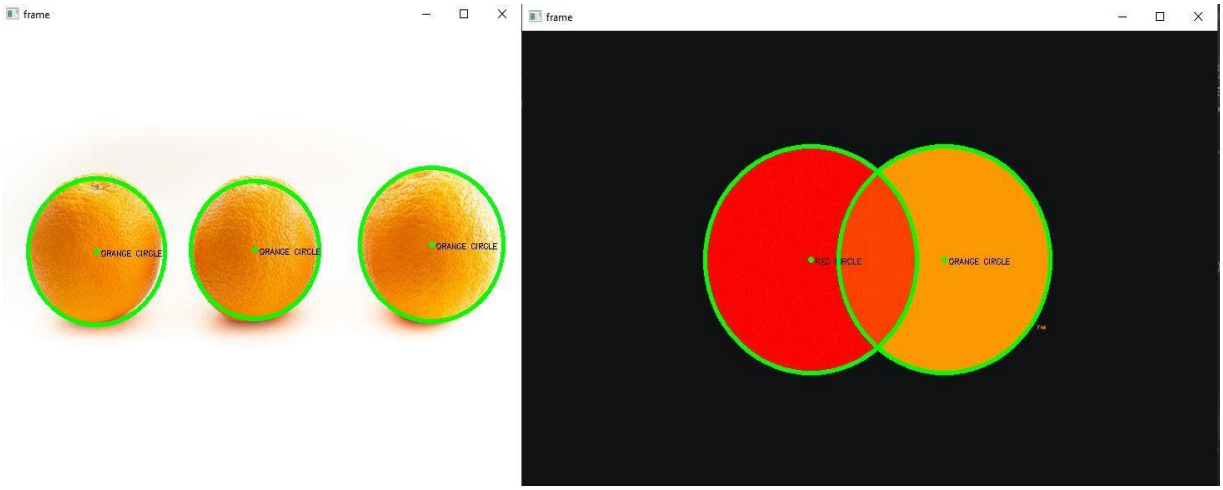
Slika 15. Houghova transformacija [15]

4.2.2. Houghova transformacija za krugove

U OpenCV-u detekcija krugova pomoću Houghove transformacije implementirana je u funkciji cv2.HoughCircles() koja uzima sljedeće argumente [20]:

- Ulazna slika,
- Metoda (određuje način otkrivanja krugova na slikama, najčešće cv2.HOUGH_GRADIENT)
- Dp-obrnuti omjer razlučivosti akumulatora i razlučivosti slike, što je veći dp manje je polje akumulatora
- MinDist - minimalna udaljenost između središta otkrivenih krugova (ako je minDist premalen može se lažno otkriti više krugova, a ako je minDist prevelik neki krugovi se možda neće otkriti)
- Param1 - vrijednost gradijenta koja ze koristi za otkrivanje rubova
- Param2 - vrijednost praga akumulatora
- MinRadius - minimalna veličina radijusa (u pikselima)
- MaxRadius - maksimalna veličina radijusa (u pikselima)

Kako bi se uspješno provela detekcija važno je znati da će trebati prilagođavati parametre da bi postigli što točniju i precizniju detekciju krugova. Primjer detekcije dan je na slici 16.



Slika 16. Primjer Houghove detekcije krugova i detekcije boje

5. IZRADA PROGRAMSKE APLIKACIJE I REZULTATI

5.1. Detekcija oblika

Kao što je već ranije spomenuto, na početku svakog koda moraju se učitati potrebne biblioteke i moduli, u ovom slučaju to su biblioteke OpenCV i Numpy. Pomoću funkcije cv2.imread učitava se određena slika. Nakon što se učita, slika se mora obraditi kako bi se uklonili svi nepotrebni šumovi koji bi mogli smetati kod prepoznavanja objekata. To se radi pomoću funkcije cv2.GaussianBlur. Također, slika u boji se pretvara u sivu sliku pomoću funkcije cv2.cvtColor.

Zadatak ovog završnog rada je izrada programske aplikacije koja će detektirati objekte, u ovom slučaju geometrijske oblike na temelju boje i oblika. Uz rezultate će se potanko opisati način detekcije, a na kraju će se usporediti valjanost odabranih metoda. Također, analizirat će se učinkovitost razvijenog modela u ovisnosti o osvjetljenju prostorije.

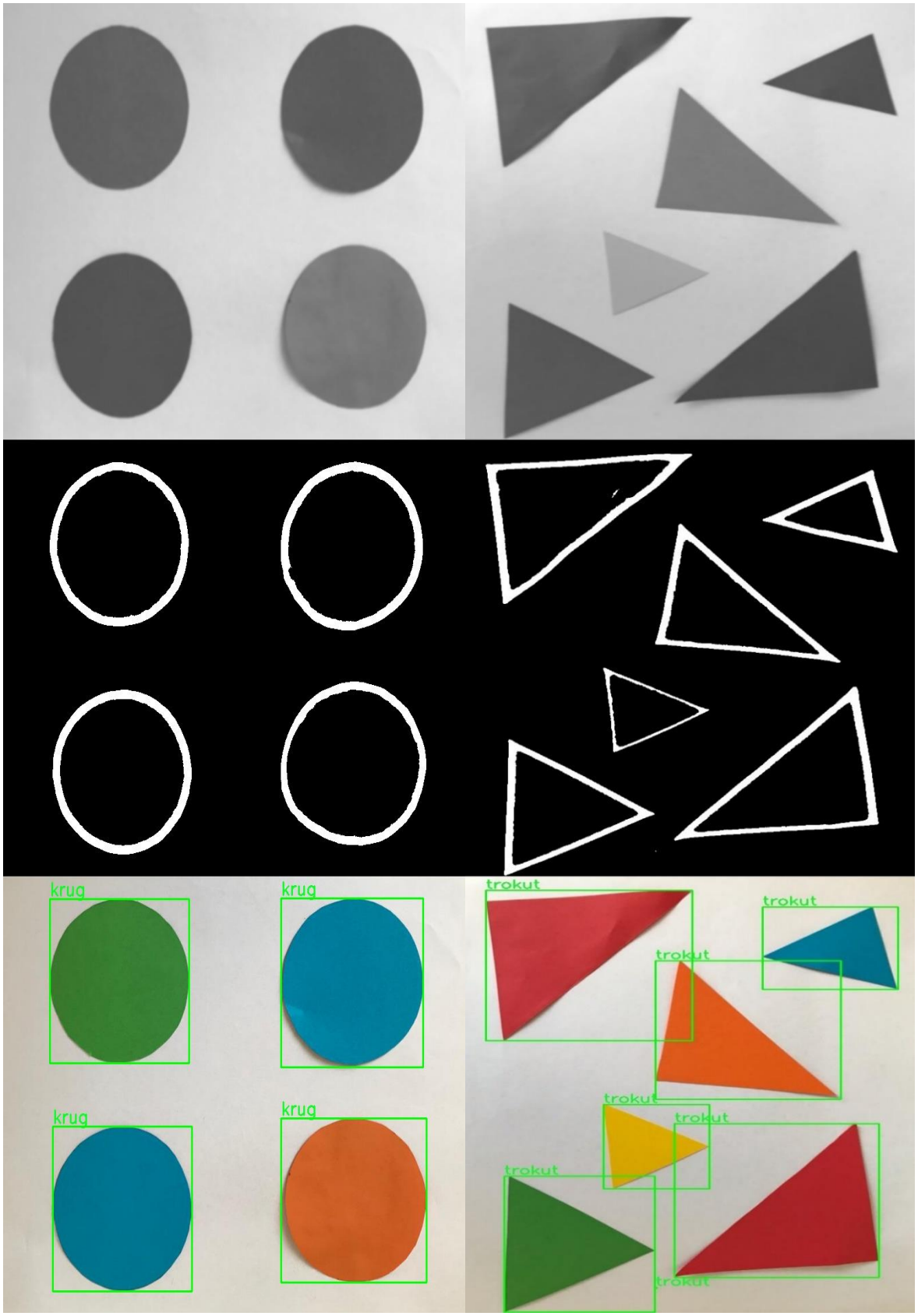
Kod detekcije oblika važno je na neki način razgraničiti objekte koji se žele detektirati od ostatka slike. U ovome modelu koristila su se dva načina pripreme slike za detekciju: Tresholding (određivanje praga) i Cannyjev algoritam. Nakon te dvije tehnike segmentacije slike, oblici se prepoznavaju metodom aproksimacije kontura, odnosno funkcijom cv2.approxPolyDP.

5.1.1. Detekcija geometrijskih oblika segmentacijskim procesom određivanja praga - Tresholding

Na slici 17. prikazani su uspješno prepoznati krugovi i trokuti koristeći Tresholding i aproksimaciju kontura. Prvo se slika u boji pretvara u sivu, a nakon toga se koristi funkcija cv2.adaptiveTreshold kako bi se objekt koji se želi detektirati izdvojio od pozadine.

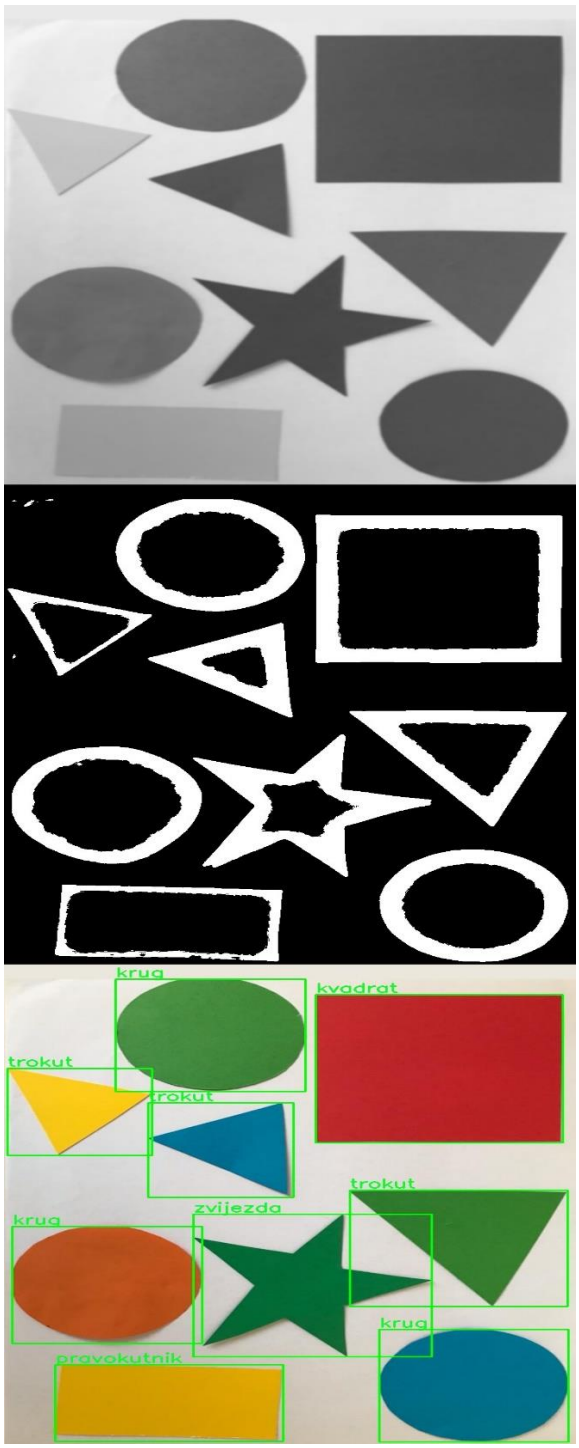
Linija koda kojom se postiže srednji prikaz na slici 17 izgleda ovako:

thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 51, 5)



Slika 17. Programska aplikacija kojom se detektiraju krugovi i trokuti koristeći Tresholding i aproksimaciju kontura

Slika 18. prikazuje navedenu detekciju koja je prepoznala svaki oblik bez greške. U ovom primjeru su uključeni ne samo krugovi i trokuti, nego i kvadrat, pravokutnik i zvijezda pa je zbog toga algoritam najprije imao neke pogrešne detekcije jer nije mogao tično detektirati rub oblika. To je riješeno na način da se u funkciji cv2.adaptiveTreshold umjesto vrijednosti praga 51 stavila vrijednost 71. S tom vrijednosti algoritam je uspješno detektirao objekte na slici.

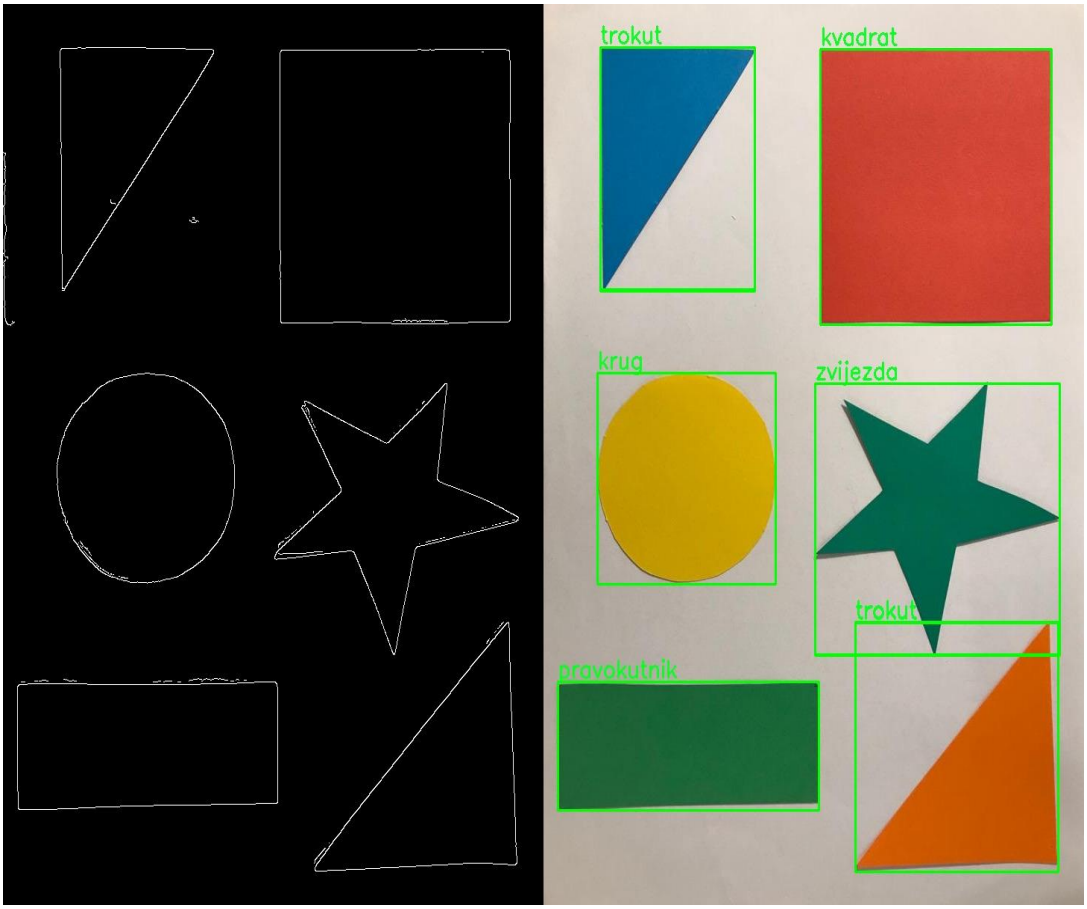


Slika 18. Programska aplikacija koja detektira geometrijske oblike koristeći Tresholding i aproksimaciju kontura

5.1.2. Detekcija geometrijskih oblika Cannyjevim algoritmom

Drugi način kojim su se našli rubovi oblika je Cannyjev algoritam. Slika se obradi i na nju se primjeni Cannyjev algoritam, a onda funkcijom cv2.approxPolyDP aproksimiraju krivulje. Lijeva strana slike 19. prikazuje rubove objekata pronađenih Cannyjevim algoritmom, a desna strana prikazuje detekciju objekata nakon primjenjenje aproksimacije kontura.

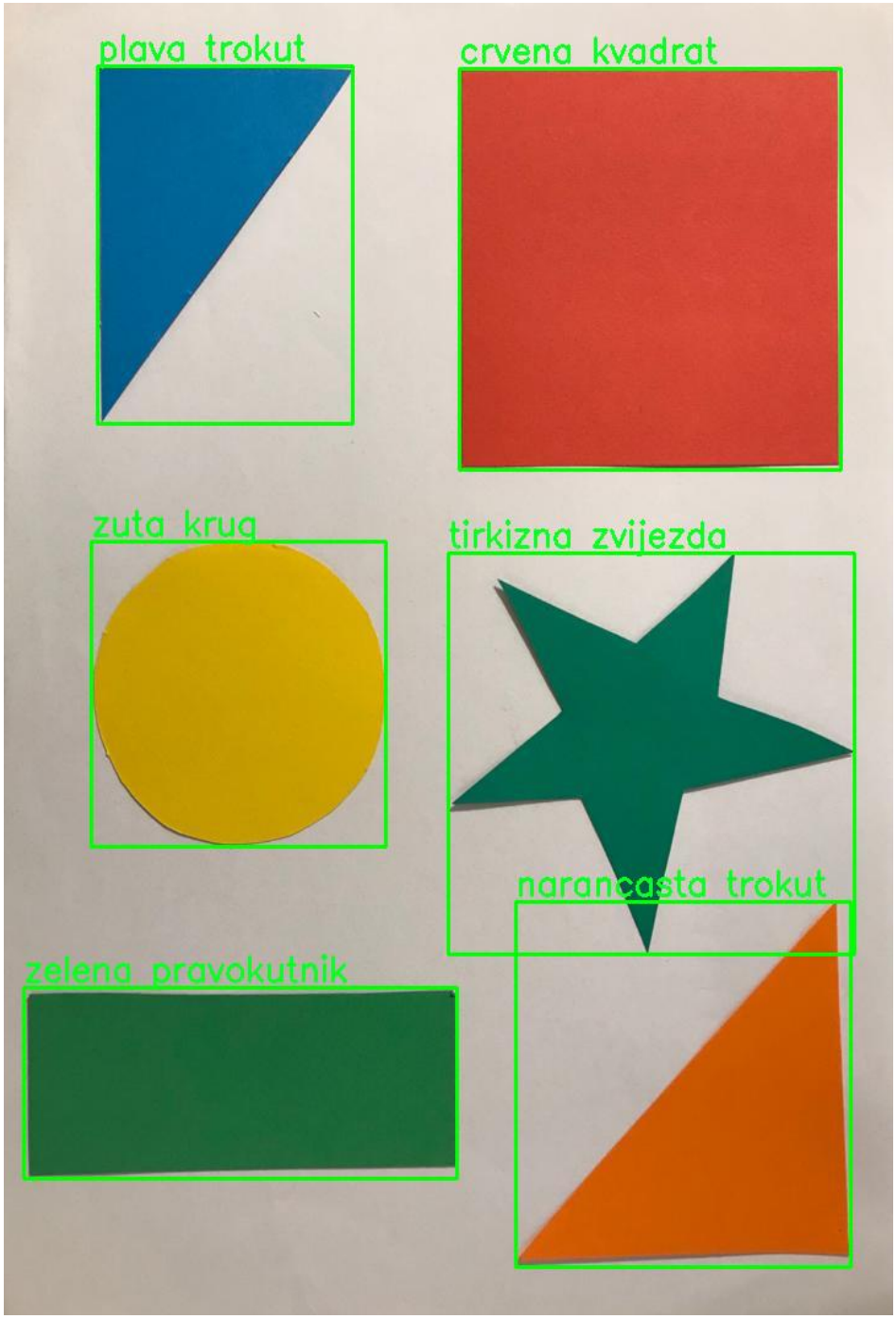
Koristeći Cannyjev algoritam i aproksimaciju kontura uspješno su prepoznati svi objekti na slici.



Slika 19. Programska aplikacija koja detektira geometrijske oblike koristeći Cannyjev algoritam i aproksimaciju konture

5.2. Detekcija oblika i boje

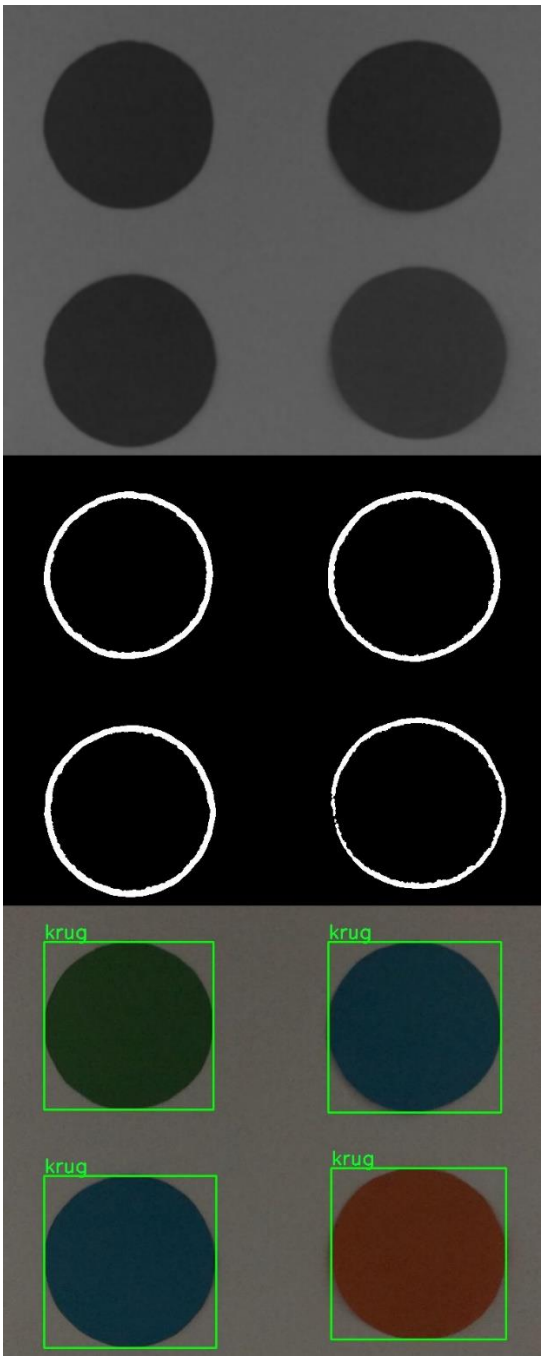
Nakon što su detektirani oblici koristeći dva različita pristupa, detektirana je i boja geometrijskih oblika. Kod detekcije boje bitno je prvo sliku prebaciti iz RGB u HVS model boja. Potom se deriviraju gornje i donje granice pojedine boje i za svaku se konturu traži srednja vrijednost boje u radijusu oko centra konture. Na slici 20. prikazana je uspješna detekcija i geometrijskih oblika i njihovih pripadnih boja. Obje metode su s gotovo istom točnosti detektirale objekte i boje.



Slika 20. Detekcija geometrijskih oblika temeljem boje i oblika

5.3. Analiza učinkovitosti modela u ovisnosti o trenutnim svjetlosnim uvjetima

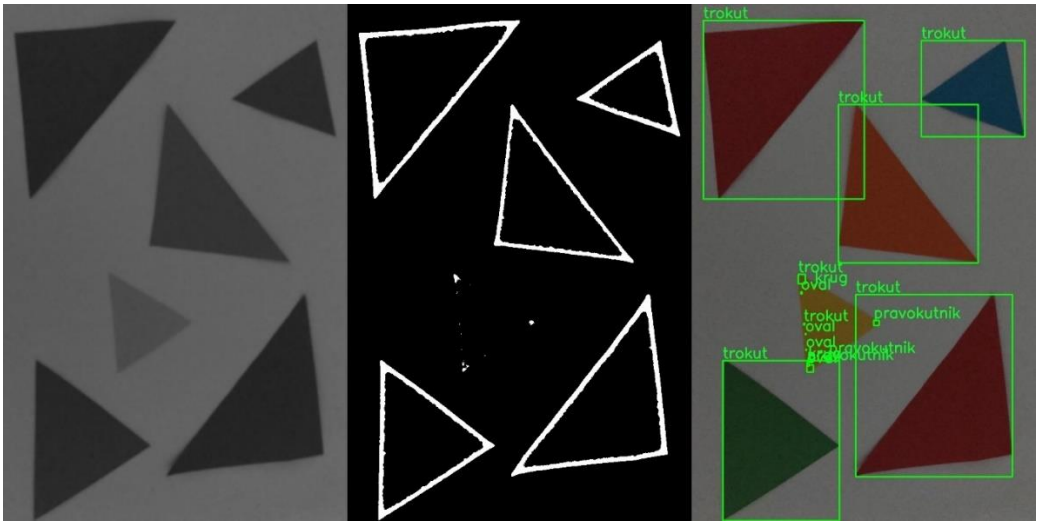
Nakon što smo ustvrdili da obje metode jednako dobro prepoznaju objekte na slici u dobrim svjetlosnim uvjetima, provjerit će se valjanost algoritama u uvjetima prigušene svjetlosti. Slika 21 prikazuje krugove koji su korištenjem Tresholding-a i aproksimacije krivuljama uspješno detektirani. Zbog svoje jednostavnosti, kod krugova nismo zamjetili nedostatke algoritma za drukčije svjetlosne uvjete.



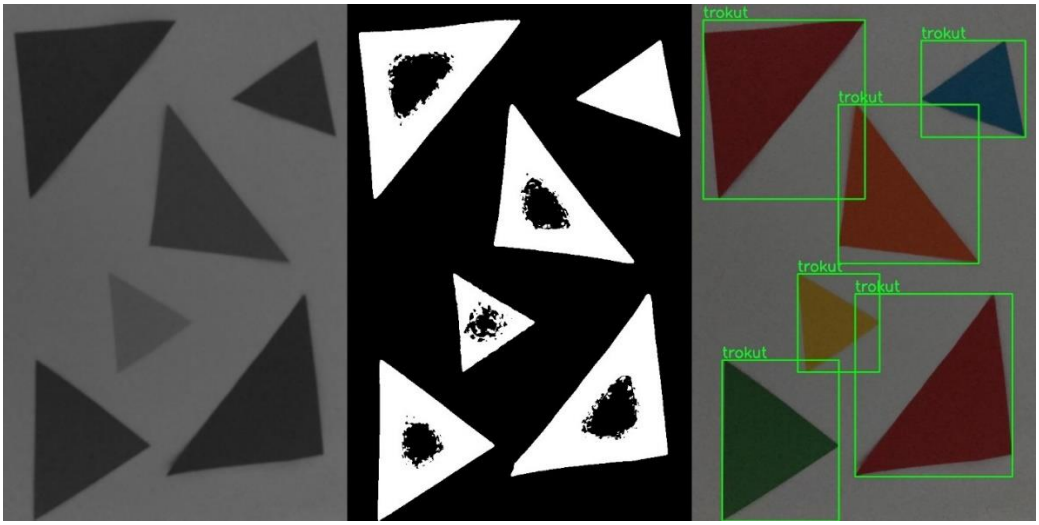
Slika 21. Detekcija krugova koristeći Tresholding i aproksimaciju kontura u tamnijem okruženju

Na slikama 22 i 23 prikazana je detekcija trokuta Tresholding metodom određivanja praga i aproksimacijom kontura u tamnijem okruženju. Koristeći isti algoritam kao i u svjetlosnim

uvjetima nisu se uspjeli uspješno detektirati svi oblici. Problem je bio mali žuti trokut koji nije prepoznat Tresholding metodom otkrivanja rubova. Mijenjanjem parametra T (prag), tj. njegovim povećanjem sa 71 na 251 uspješno je detektiran i mali žuti trokut.

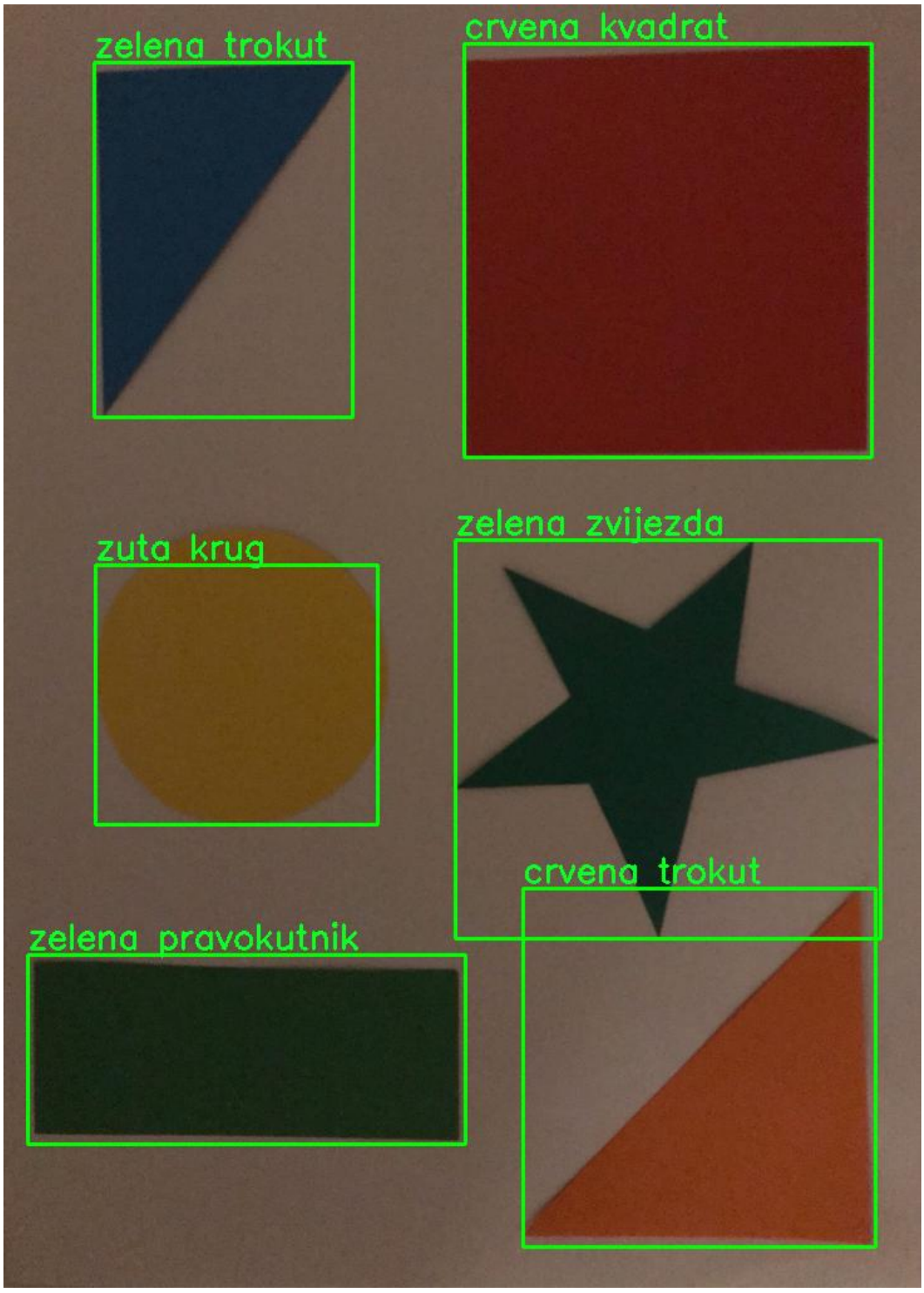


Slika 22. Neuspjela detekcija trokuta koristeći Tresholding i aproksimaciju kontura u tamnijem okruženju



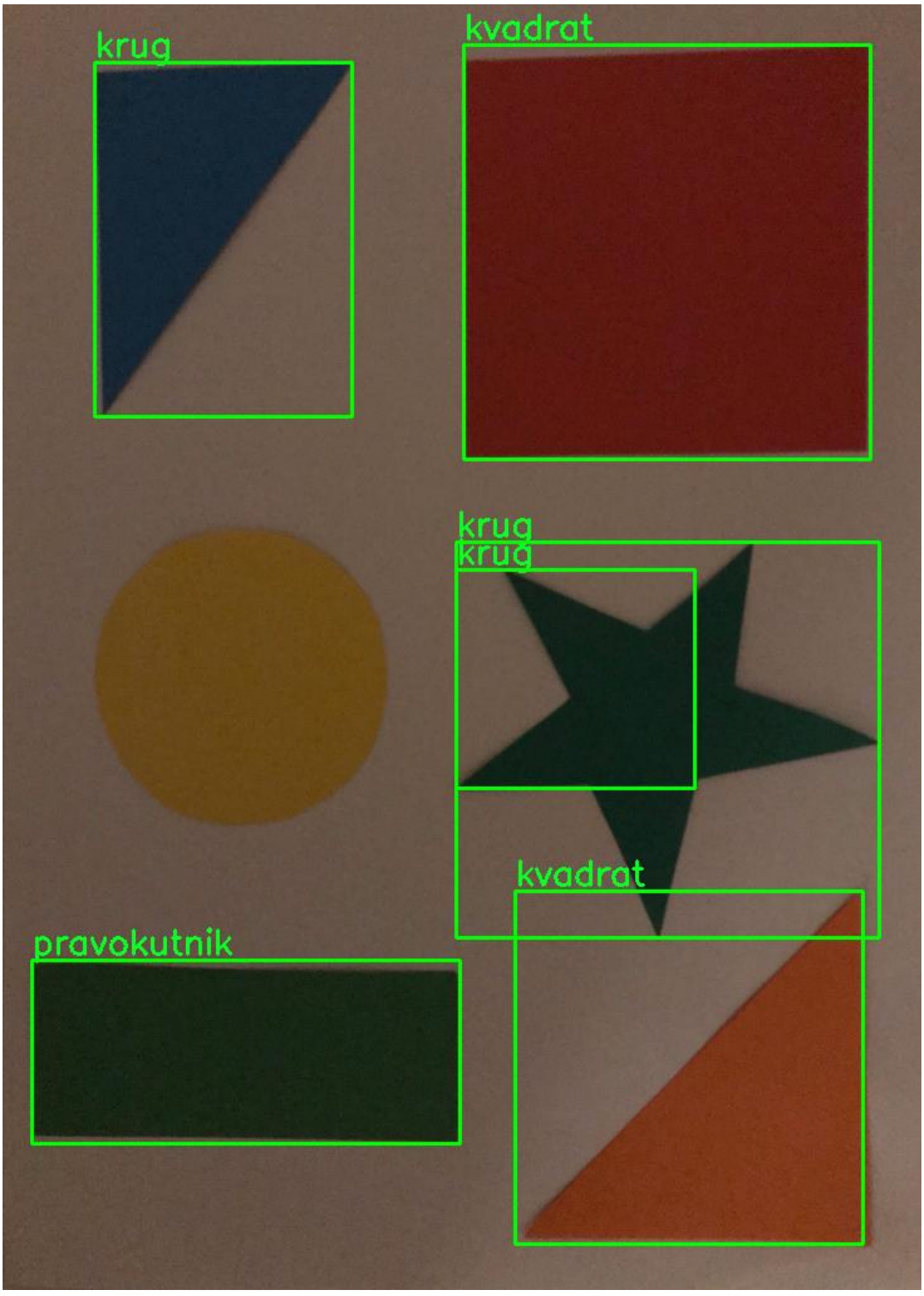
Slika 23. Uspješna detekcija trokuta koristeći Tresholding i aproksimaciju kontura u tamnijem okruženju

Slika 24 prikazuje uspješnu detekciju svih geometrijskih oblika i njihovih pripadnih boja Tresholding metodom detekcije praga i aproksimacijom kontura u uvjetima prigušene svjetlosti.



Slika 24. Detekcija objekata koristeći Tresholding i aproksimaciju kontura u tamnijem okruženju

Za iste postavke i parametre promijenjena je svjetlost i ispitana efikasnost Cannyjevog algoritma za detekciju rubova i aproksimacije kontura. Ovom metodom, uz puno pokušaja i mijenjanja raznih parametara, nije otkriven žuti krug na slici 25,nego bi se uvijek pojavila neka netočna detekcija. Zaključujemo da se prva metoda pokazala malo više adekvatnijom za ovaj slučaj detekcije.

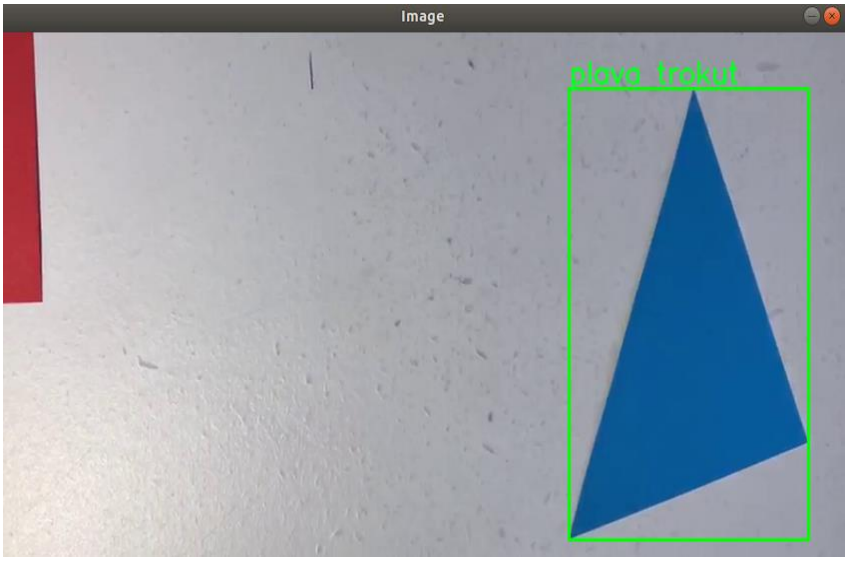


Slika 25. Detekcija objekata koristeći Cannyjev algoritam i aproksimaciju kontura u tamnijem okruženju

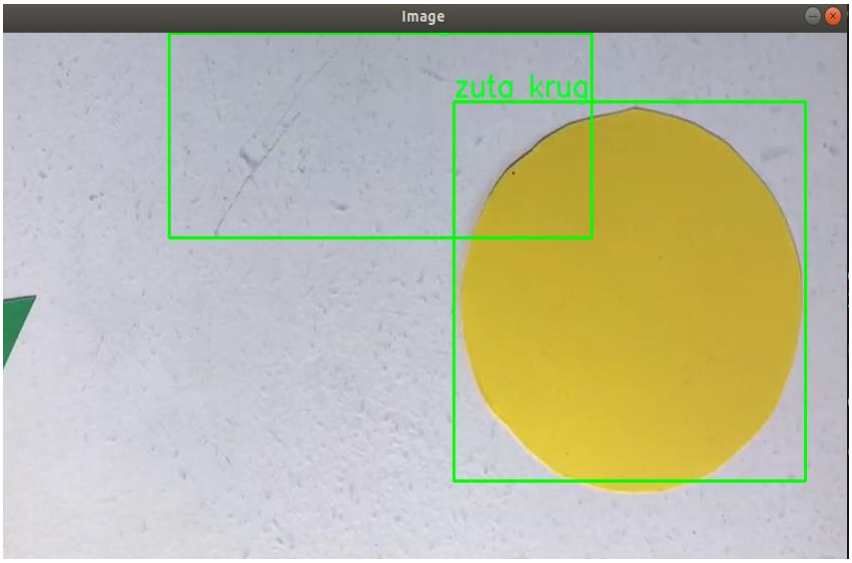
5.5. Detekcija objekata u stvarnom vremenu

Na slikama 26, 27 i 28 prikazana je detekcija objekta u stvarnom vremenu. Dodana je i nesavršena pozadina koja je rezultirala lažnim detekcijama, što je vidljivo na slikama 27 i 28 . Slika 26 uspješno je detektirala trokut plave boje.

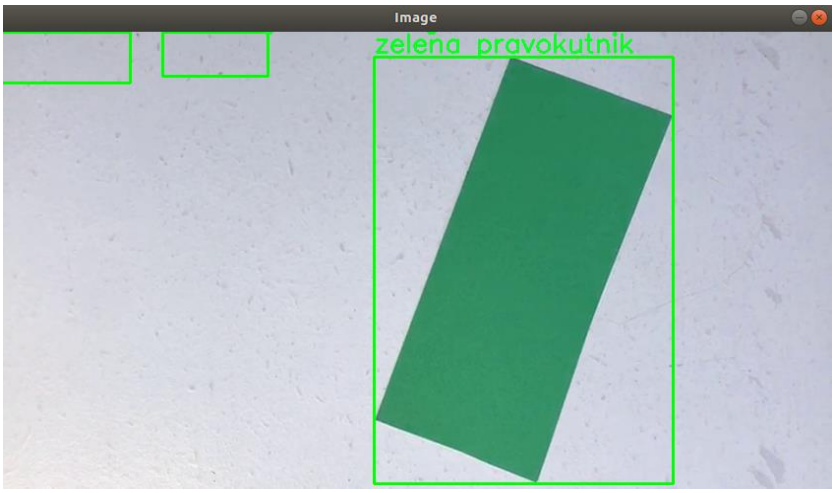
Ovime zaključujemo da složenost algoritama nije predstavljala problem u stvarnom vremenu. Budući rad bio bi pokušaj generaliziranja algoritma kako bi bio prilagođen raznim svjetlosnim uvjetima. Nesavršena pozadina koja se pojavila u videu stvorila je poteškoće, tj. lažne detekcije na kojima bi trebalo raditi. Time se zaključuje da detekcija objekata na slici nije banalan problem i da dosta ovisi o faktorima kao što su pozadina slike, osvjetljenje i sama složenost objekata koji se žele detektirati.



Slika 26. Detekcija plavog trokuta u stvarnom vremenu

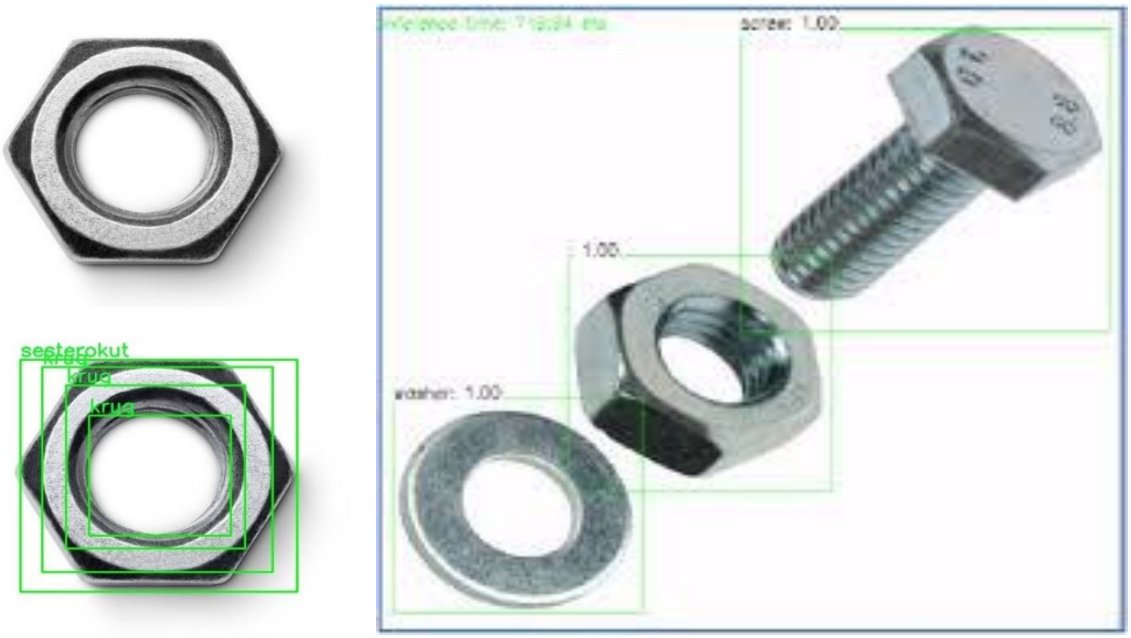


Slika 27. Detekcija žutog kruga u stvarnom vremenu



Slika 28. Detekcija zelenog pravokutnika u stvarnom vremenu

U pokušaju detekcije realnijeg primjera, u ovom slučaju matice, algoritam uspješno detektira složeniji predmet iz tlocrta. Također, može prepoznati oblik predmeta i svih njegovih dijelova, prema slici 29 a). Kako bi se predmet mogao prepoznati iz svih položaja potrebno je algoritam nadograditi npr. pomoću neuronskih mreža. Na slici 29 b) prikazana je detekcija matice, vijka i podložne pločice pomoću neuronskih mreža, što je ujedno i detaljnije i duže, ali u konačnici jednostavnije rješenje za detekciju predmeta u prirodi.



a) **b)**

Slika 29. Detekcija a) matice pomoću Tresholding metode i aproksimacije kontura b) matice, vijka i podložne pločice pomoću neuronskih mreža

6. ZAKLJUČAK

U ovome radu napravljena je programska aplikacija za detekciju objekata temeljem boje i oblika. U sklopu teorijske podloge rada dan je uvid u osnovne informacije o računalnom vidu, programskom jeziku Python i njegovim bibliotekama koje su korištene za izradu aplikacije. Također, detaljno su opisane metode koje su se koristile za segmentaciju slike i sama metoda detekcije objekata. Koristile su se dvije metode za segmentaciju slike, Tresholding metoda određivanja praga i Cannyjev algoritam, a sami oblici detektirani su aproksimacijom krivulja, odnosno funkcijom cv2.approxPolyDP. Metoda Tresholding se pokazala za nijansu boljom u detekciji rubova geometrijskih oblika od Cannyjevog algoritma jer je uspješno detektirala sve geometrijske oblike u normalnim i lošijim svjetlosnim uvjetima. Druga metoda nije uspjela detektirati jedan od šest različitih oblika na slici. Pokazano je da je razvijeni algoritam moguće vrtiti i u stvarnom vremenu na detekciji kroz snimljeni video na kojem su oblici i boje uspješno detektirani. Moguća unaprjeđenja ove programske aplikacije mogu ići u nekoliko smjerova. Prvi bi bio unaprijeđenje korištenih algoritama za detekciju jednostavnih objekata ili korištenje drugih metoda segmentacije i detekcije kako bi se moglo analizirati koja metoda najbolje detektira objekte. Prelazak na učenje pomoću neuronskih mreža za neke složenije probleme od ovoga je također moguć.

LITERATURA

[1] Živkušić D., Primjena računalnog vida u kontroli kvalitete
[2] https://www.cybiant.com/resources/an-introduction-to-computer-vision/
[3] https://hr.wikipedia.org/wiki/Python_(programski_jezik)
[4] https://opencv.org/about/
[5] Žgaljić A., Praćenje pokretnih objekata u video snimkama, 2017.
[6] https://en.wikipedia.org/wiki/OpenCV
[7] https://www.edureka.co/blog/python-libraries/
[8] http://lab425.fesb.hr/igraf/Frames/fP5_1.htm
[9] Habrun M., Usporedba modela boja i primjena u računalnoj grafici, 2018.
[10] Komugović A., Prostor boja, 2015.
[11] https://hudu.hr/modeli-boja-rgb-i-cmyk/1018
[12] A. Elrefaei1 L., Omar Al-musawa M., Abdullah Al-gohany N., Development of an android application for object detection based on color, shape, or local features, The International Journal of Multimedia & Its Applications (IJMA), 2017.
[13] https://theailearner.com/2019/11/22/simple-shape-detection-using-contour-approximation/
[14] https://towardsdatascience.com/simplify-polylines-with-the-douglas-peucker-algorithm-ac8ed487a4a1
[15] https://slideplayer.gr/slide/14907603/
[16] https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
[17] https://www.geeksforgeeks.org/implement-canny-edge-detector-in-python-using-opencv/
[18] Ćorić D., Implementacija algoritma za detekciju rubova u slici na realnu ADAS platformu, 2020.
[19] Šverko M., Houghova transformacija, 2009.
[20] https://learnopencv.com/hough-transform-with-opencv-c-python/

PRILOG

I. Python kod

```
import numpy as np
import cv2
import colorsys

def prepareImageWithThreshold(image):
    blur = cv2.GaussianBlur(image, (1, 1), 0)
    gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)

    # cv2.imshow('gray', gray)
    thresh = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV,51,5)
    # cv2.imshow('thresh', thresh)
    return thresh

def prepareImageWithCanny(image):
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.medianBlur(grayscale, 5)
    canny = cv2.Canny(blur, 5, 40);
    # cv2.imshow("Canny", canny)
    # apply dilation on src image
    kernel = np.ones((3,3),np.uint8)
    dilated_img = cv2.dilate(canny, kernel, iterations = 1)
    # cv2.imshow("Dilates", dilated_img)
    return canny

def prepareImageWithHough(image):
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.medianBlur(grayscale, 5)
    canny = cv2.Canny(blur, 5, 40);
    # cv2.imshow("Canny", canny)
    # apply dilation on src image
    kernel = np.ones((3,3),np.uint8)
    dilated_img = cv2.dilate(canny, kernel, iterations = 2)
    lines = cv2.HoughLinesP(canny, 1, np.pi/180, 5, np.array([]), 10, 10)
    imageCopy =  image.copy()
    imageCopy[:] = (0, 0, 0)
    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(imageCopy, (x1, y1), (x2, y2), (20, 220, 20), 4)

    cv2.imshow("Line Image", imageCopy)
    imageCopy = cv2.cvtColor(imageCopy, cv2.COLOR_BGR2GRAY)

    return imageCopy

def contourDetection(image):
    contours = cv2.findContours(image, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1]

    return contours

def shapeDetection(c):
    shape = ""
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.01 * peri, True)
```

```
    # Remove noise
    if (peri < 600):
        return shape

    # Triangle
    if len(approx) == 3:
        shape = "trokut"

    # Square or rectangle
    elif len(approx) == 4:
        (x, y, w, h) = cv2.boundingRect(approx)
        ar = w / float(h)

        # A square will have an aspect ratio that is approximately
        # equal to one, otherwise, the shape is a rectangle
        shape = "kvadrat" if ar >= 0.95 and ar <= 1.05 else "pravokutnik"

    # Star
    elif len(approx) == 10:
        shape = "zvijezda"

    # Otherwise assume as circle or oval
    else:
        (x, y, w, h) = cv2.boundingRect(approx)
        ar = w / float(h)
        shape = "krug"

    return shape

def saveImage(image, filename):
    # Filename
    filename = '/home/daria/Documents/ZR_detekcija/results/' + filename +
'.png'
    # Saving the image
    cv2.imwrite(filename, image)

def inRange(number, lower, upper):
    return (number > lower and number < upper)

def colorDetection(image, x, y, r):
    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    COLOR_NAMES = ["crvena", "narancasta", "zuta", "zelena", "tirkizna",
"plava", "ljubicasta"]
    COLOR_RANGES_HSV = {
        "crvena": [(0, 50, 10), (10, 255, 255)],
        "narancasta": [(10, 50, 10), (15, 255, 255)],
        "zuta": [(15, 50, 10), (35, 255, 255)],
        "zelena": [(35, 50, 10), (80, 255, 255)],
        "tirkizna": [(80, 50, 10), (100, 255, 255)],
        "plava": [(100, 50, 10), (130, 255, 255)],
        "ljubicasta": [(130, 50, 10), (170, 255, 255)]
    }
    # HSV
    array_hsv = np.array(cv2.mean(hsv_img[y:y+r,x:x+r])).astype(np.uint8)

    for color in COLOR_NAMES:
        colorRange = COLOR_RANGES_HSV[color]
        lower = np.array(colorRange[0])
        upper = np.array(colorRange[1])
```

```
        if (inRange(array_hsv[0], lower[0], upper[0]) and
            inRange(array_hsv[1], lower[1], upper[1]) and
            inRange(array_hsv[2], lower[2], upper[2])):

            return color
    return "unknown_color"

if __name__ == '__main__':
    image =
cv2.imread('/home/daria/Documents/ZR_detakcija/dataset/ime_slike.png')

    imagePrep = prepareImageWithThreshold(image)
    contours = contourDetection(imagePrep)
    contours_poly = [None]*len(contours)

    boundRect = [None]*len(contours)
    centers = [None]*len(contours)
    radius = [None]*len(contours)
    for i, c in enumerate(contours):
        shape = shapeDetection(c)
        x,y,w,h = cv2.boundingRect(c)
        color = ""
        if (shape != ""):
            contours_poly[i] = cv2.approxPolyDP(c, 3, True)
            boundRect[i] = cv2.boundingRect(contours_poly[i])
            centers[i], radius[i] =
cv2.minEnclosingCircle(contours_poly[i])
            cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
            cv2.imshow('cutted contour',image[y:y+h,x:x+w])

            color = colorDetection(image, int(centers[i][0]),
int(centers[i][1]), 6)
            cv2.rectangle(image, (x, y), (x + w, y +h), (0, 255, 0), 2)
            cv2.putText(image, color + " " + shape, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)

    cv2.imshow('image', image)
    cv2.waitKey()
    cv2.destroyAllWindows()
```