Klasifikacija objekata humanoidnim Nao robotom

Vnuk, David Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET PRIMIJENJENE MATEMATIKE I INFORMATIKE

Sveučilišni prijediplomski studij Matematika i računarstvo

KLASIFIKACIJA OBJEKATA HUMANOIDNIM NAO ROBOTOM

ZAVRŠNI RAD

Mentori:

prof. dr. sc. Kristian Sabo

izv. prof. dr. sc. Zoran Tomljanović

Student:

David Matej Vnuk

Osijek, 2024.

SAŽETAK

U ovom završnom radu opisat ćemo način na koji upravljamo humanoidnim NAO robotom te kako pohranjujemo sliku s kamere ugrađene na glavi robota. Prikazat ćemo razvoj ideje, tijek implementacije te proces testiranja. Detaljno ćemo objasniti tri metode koje koristimo kako bismo obradili pohranjenu sliku te ćemo na osnovu prikupljenih informacija, klasificirati objekt u dvije skupine. Negativno klasificiranoj skupini pripadat će objekti koji nisu lopta, dok će pozitivno klasificiranoj skupini pripadati objekti nalik lopti. Spajanje i upravljanje robotom odvijat će se u programskom softveru Choreographe [1], dok će obrada slike biti provedena unutar Python [2] programskog jezika. Nakon implementacije, spojit ćemo se na stvarnog robota te pozvati metode kojima pohranjujemo i obrađujemo sliku. Ovisno o klasifikaciji, robot na glas izgovara kojoj od moguće dvije skupine objekt pripada. Na kraju ćemo rada prikazati rezultate i usporediti rad svih triju metoda na što ćemo se nadovezati u zaključku.

Ključne riječi: humanoidni robot, NAO, klasifikacija, lopta

CLASSIFICATION OF OBJECTS WITH A HUMANOID NAO ROBOT

ABSTRACT

In this final paper, we will describe how we control the humanoid NAO robot and how we store the image from the camera installed on the robot's head. We will show the development of the idea, the implementation process and the testing process. We will explain in detail the three methods we use to process the stored image and, based on the collected information, we will classify the object into two groups. Objects that do not look like a ball will belong to the negatively classified group, while objects that look like balls will belong to the positively classified group. Connecting and controlling the robot will take place in the Choreographe software [1], while the image processing will be done within the Python [2] programming language. After implementation, we will connect to the real robot and call the methods by which we store and process the image. Depending on the classification, the robot says out loud which of the two possible groups the object belongs to. At the end of the paper, we will present the results and compare all three methods, which we will follow up on in the conclusion.

Keywords: humanoid robot, NAO, classification, ball

Sadržaj

Uvod

Ideja ovoga završnog rada bila je objediniti teoriju upravljanja, linearnu algebru, numeričku matematiku te programiranje unutar programskog jezika Python. Željeli smo da obuhvaća upravljanje robotom, analiziranje slike u matričnoj formi te tri različite metode „fitanja“[1] kružnice na koordinatama piksela koji određuju rub objekta sa tzv. „edge-detection“ slike[2].

Bio je cilj naučiti upravljati humanoidnim NAO robotom putem Choreographe programskog softvera, a zatim koristeći kontrole nad pokretima udova usmjeriti robota da gleda u smjeru objekta kojeg klasificiramo. Prijenos slike uživo koju robot vidi također obavljamo putem Choreographe softvera te nam to pomaže kako bismo pravilno mogli usmjeriti robota. Detaljniji opis korištenja softvera dan je u poglavlju 2.1.2 Korištenje. Nakon toga, željeli smo direktno upravljati robotom na način da nam šalje sliku u odgovarajućem formatu, obrađuje ju gore spomenutim metodama te naglas govori rezultat klasifikacije. Za ovakvu vrstu upravljanja koristili smo Python programski jezik. Nešto više o direktnom upravljanju, metodama obrade „edge-detection“ slike i načinu klasifikacije putem Pythona može se pročitati u poglavlju 3 Metode i implementacije. Na kraju, cilj nam je zaključiti kako se naše metode ponašaju, pod kojim uvjetima daju bolji učinak, a koje su im pak mane tako što ćemo komentirati dobivene rezultate.

[1] „fitanje“: engleski naziv za određivanje i prilagođavanje parametara koji će najbolje odgovarati skupu podataka. U ovom slučaju riječ je o pronalasku radijusa i koordinata centra koji će određivati kružnicu koja najbolje odgovara „edge-detection“ slici objekta. Ovaj ćemo izraz koristiti unutar cijelog rada.

[2] „edge-detection“ slika: engleski naziv za sliku koja je dobivena jednom od metoda koje iz početne slike objekta stvaraju novu sliku na kojoj je prikazan samo rub uslikanog objekta

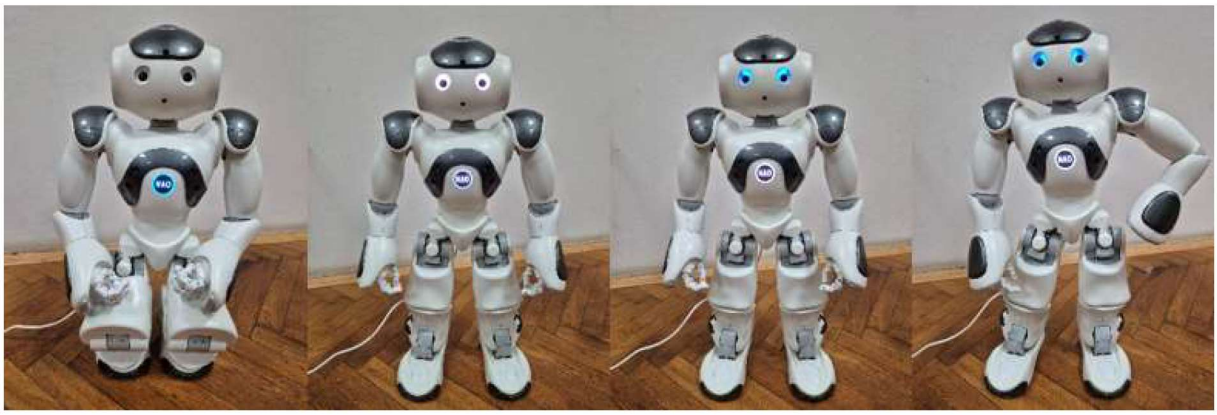
Poglavlje 1

Pokretanje i spajanje robota

Humanoidni NAO robot je iznimno lak za spajanje i rukovanje. Detaljnu dokumentaciju za rukovanje robotom možemo pronaći na [3]. Nakon pažljivog vađenja iz zaštitnog kofera, postavljamo ga u čučeći položaj te ga spajamo na punjač (vidi slike 1 i 2).

Slika 1 i 2: Spajanje robota na punjač

Nakon što se robot ponešto napuni, palimo ga tako što tri sekunde držimo NAO tipku. Robot se budi i ustaje.

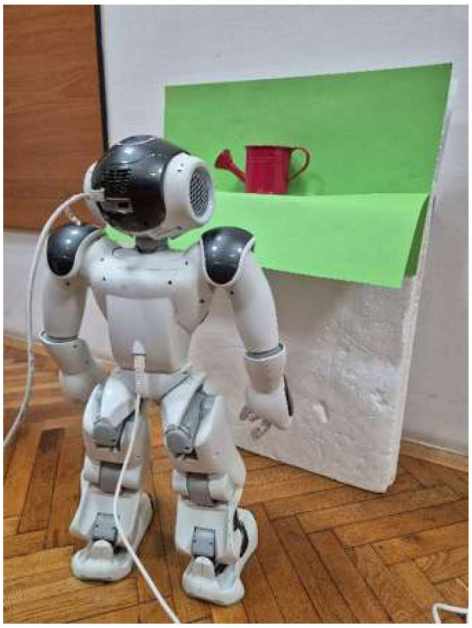


Slika 3: Proces buđenja

Kako bismo robota spojili na računalo, spajamo mrežni kabel u glavu robota te u računalo (vidi slike 4 i 5).

Slike 4 i 5: Spajanje na računalo

Na posljetku usmjeravamo cijelog robota da gleda prema stalku za objekte koje ćemo klasificirati.



Slike 6 i 7: Usmjeravanje

Konačno usmjeravanje kamere na glavi robota napravit ćemo unutar Choreographe programskog softvera, a proces ćemo opisati u narednom poglavlju.

Poglavlje 2

Korištenje softvera

U ovom poglavlju objasnit ćemo kako smo se služili Choreographe programskim softverom, koje smo opcije koristili te odakle smo preuzeli isti. Također, objasnit ćemo što je sve potrebno kako bismo spojili programske biblioteke robota unutar programskog jezika Python kojeg smo prethodno instalirali na naše računalo. Osim toga, navest ćemo potrebne verzije samog programskog jezika, ali i biblioteka koje su kompatibilne za upravljanje humanoidnim NAO robotom.

2.1 Choreographe

Choreographe je programski softver u vlasništvu tvrtke SoftBank Robotics [4]. Koristi se za kreiranje animacija te za kontroliranje i dijalog s robotom. Navedene usluge koje softver pruža omogućene su za korisnike koji posjeduju robota, ali i za one kojima je vizualna simulacija dovoljna kako bi isprobali upravljanje robotom. Program sadrži korisničko sučelje u kojemu možemo pomoću blokovskih funkcija animirati i upravljati robotom. Također, sadrži razne informacije o robotu kao što su pozicija svih udova, napunjenost baterije, glasnoća govora i sl. Većinu ćemo navedenih funkcija opisati u poglavlju 2.1.2 Korištenje.

2.1.1 Instalacija

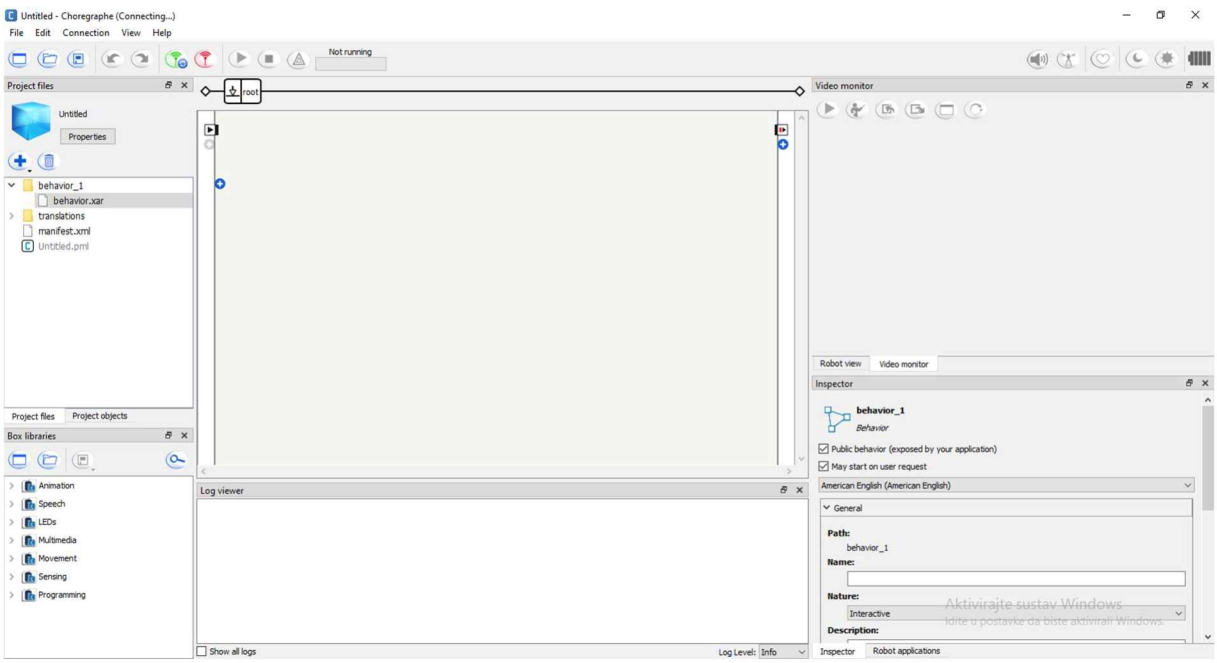
Preuzimanje softvera vršimo na poveznici [5]. Odabiremo operativni sustav kojega posjedujemo na računalu te instaliramo 2.8.7 – Setup verziju. Instalaciju provodimo na uobičajen način te nakon iste možemo pokrenuti instalirani softver.

2.1.2 Korištenje

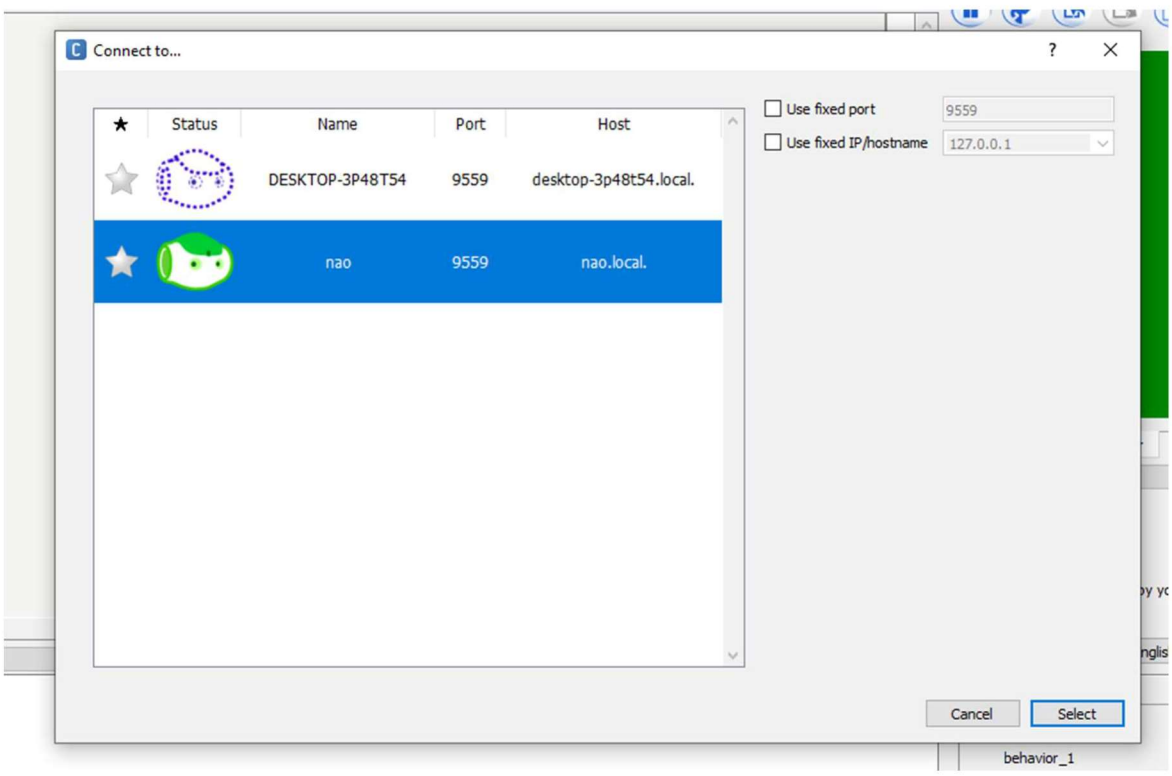
Prije samog opisa načina na koji se koristi Choreographe programski softver, željeli bismo naglasiti da ostale funkcionalnosti koje nisu korištene unutar ovog rada nismo dodatno opisivali jer se detaljna dokumentacija za Choreographe softver može naći na internetskoj stranici [6]. Istu smo dokumentaciju koristili kako bismo se upoznali s funkcionalnostima koje su nam trebale, a njihovo korištenje nadalje opisujemo.

Unutar Choreographe softvera pritiskom na ikonu bežičnog spajanja (vidi sliku 8) prikazuje nam se izbornik u kojemu bismo trebali moći izabrati plavog, „DESKTOP“ robota ili zelenog „NAO“ robota (slika 9). Plavi robot označava simulacijskog robota, a zeleni fizičkog.

U poglavlju 1 Pokretanje i spajanje robota, detaljno je opisan proces spajanja fizičkog NAO robota s računalom. Ako su svi koraci ispravno učinjeni, zeleni robot bi trebao biti vidljiv.

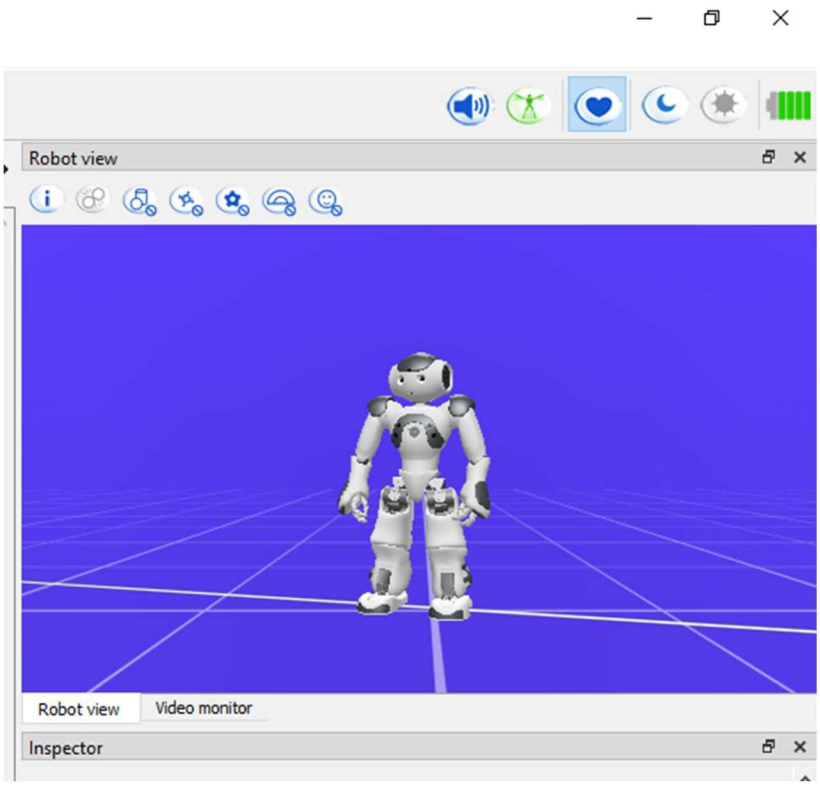


Slika 8: Choreographe programski softver



Slika 9: Ponuđene opcije spajanja

Ukoliko je sve ispravno povezano, u desnom bismo dijelu softvera trebali moći vidjeti robota koji ima jednake pozicije udova kao fizički robot (vidi sliku 10).

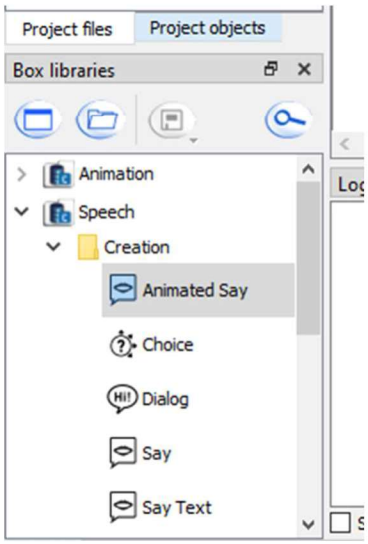


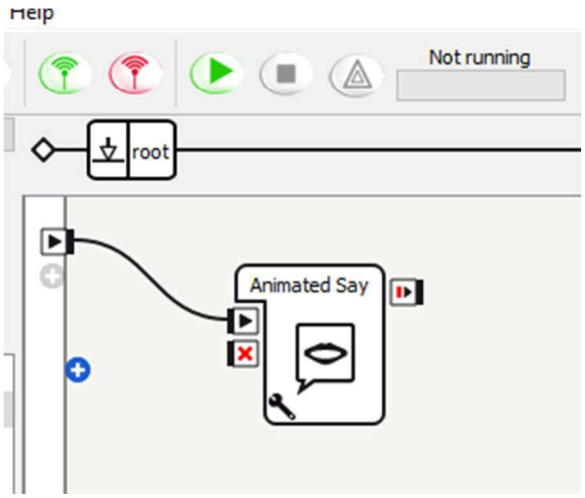
Slika 10: Simulacija robota u prostoru

Kao što možete primijetiti imamo prikazane razne informacije o robotu kao što su razina napunjenosti baterije, položaj udova, glasnoća izgovora i slično.

Jedinu funkcionalnost koju nismo uspjeli iskoristiti, ali smatramo da je bitna te ćemo ju kratko spomenuti jesu blokovske funkcije. Choreographe softver ima već pozamašan broj ugrađenih funkcija koje se slažu i pozivaju u blokovima. Tako dakle, možemo pokretati različite dijelove tijela, kontrolirati animacije, senzore ili izreći rečenicu (vidi sliku 11).

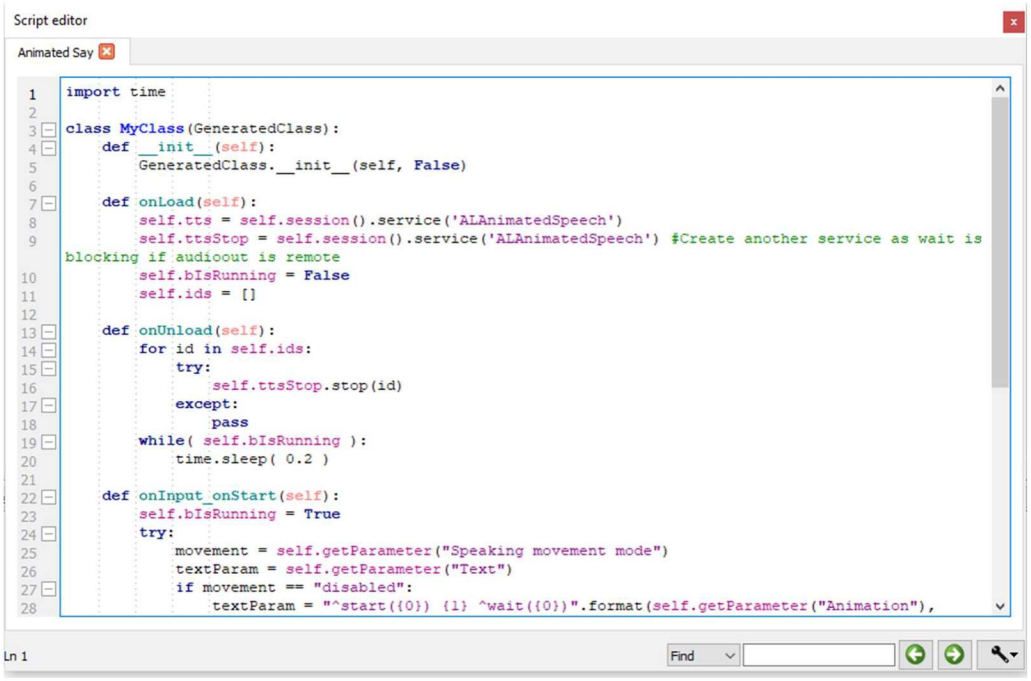
Na slici 12 vidimo primjer kako iz izbornika u lijevom donjem kutu softvera (slika 11) možemo odabrati blokovsku funkciju te ju pokazivačem prevući u središnji dio ekrana. Takve funkcije pozivamo pritiskom na „play“ dugme ili F5 na tipkovnici.





Slike 11 i 12: Odabir blokovskih funkcija

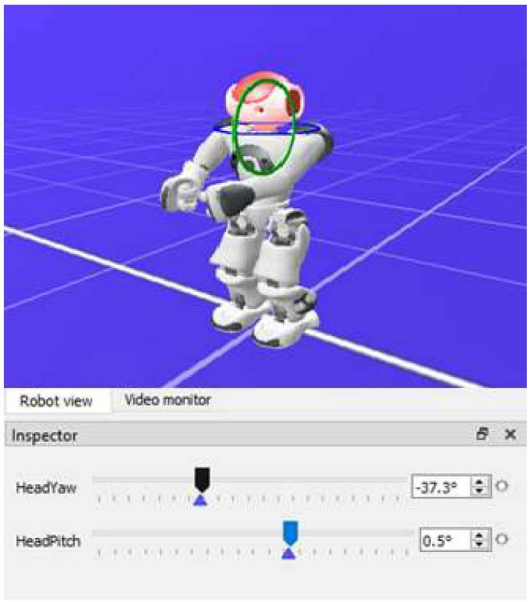
Iako se dvostrukim pritiskom na blokovsku funkciju otvara skripta koju možemo mijenjati (vidi sliku 13), za upravljanje i programiranje robotom na nižoj razini koristit ćemo direktno spajanje biblioteka NAO robota unutar instaliranog Python programskog jezika na računalu. Razlog tomu jest što unutar Python okruženja Choreographe softvera nedostaju biblioteke koje su nam potrebne (više u poglavlju 2.2 Python).



Slika 13: Skripta blokovske funkcije

Glavni razlog zbog kojeg koristimo Choreographe jest opcija „Video monitoring“ koja nam omogućava da u stvarnom vremenu gledamo sliku koju robot vidi (slika 15). Na taj način možemo vidjeti je li objekt, kojeg ćemo klasificirati, centriran.

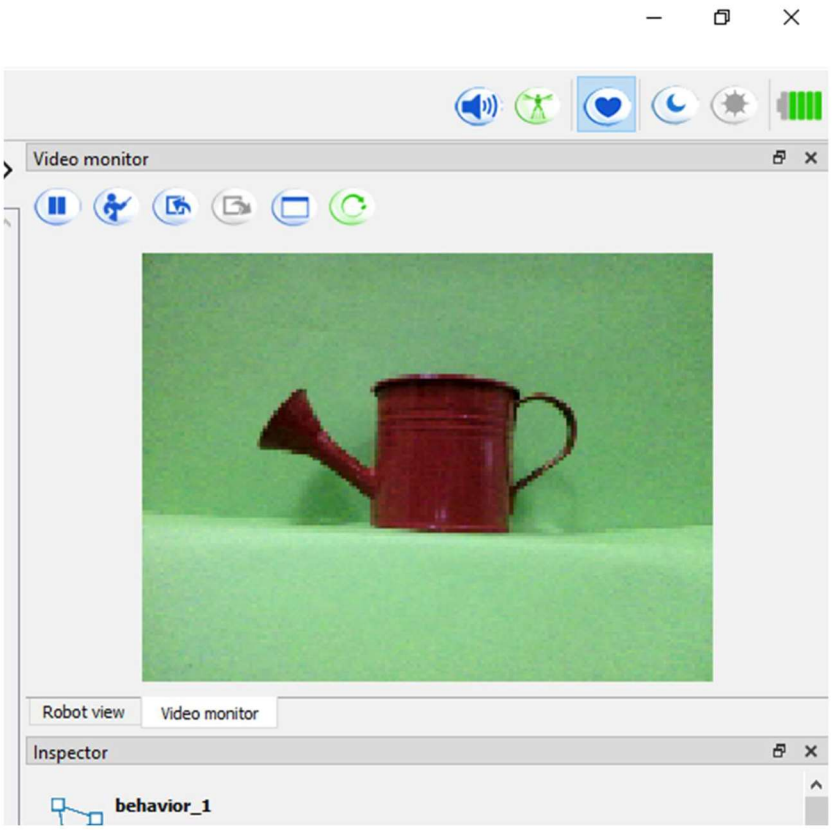
Ukoliko objekt nije centriran vraćamo se na opciju „Robot view“ koju vidimo na slici 10 te klikom pokazivača na glavu robota dobivamo opciju pomicanja glave prikazanu na slici 14.



Slika 14: Upravljanje pozicijom glave robota

Funkcijom „HeadYaw“, prikazanoj na slici 14, glavu robota pomičemo lijevo, odnosno desno, a funkcijom „HeadPitch“ pomičemo glavu gore, odnosno dolje.

Želimo da naš objekt bude centriran kao na slici 15.



Slika 15: Centrirani objekt

Kada smo centrirali objekt, spremni smo pokrenuti metode obrade slike i „fitanja“ kružnice koje implementiramo u Python programskom jeziku. Detaljan opis istoga možete pronaći u sljedećem poglavlju.

2.2 Python

Upravljanje i programiranje humanoidnog NAO robota u programskom jeziku Python nosi sa sobom određene poteškoće. Naime, robot je bio javno dostupan od 2011. godine te je stoga kompatibilan samo sa starijim verzijama Pythona i njegovim bibliotekama.

2.2.1 Instalacija

Odlaskom na internetsku stranicu [7] možemo preuzeti 2.7.18 verziju Pythona koja je jedina, po našim testiranjima, funkcionirala na robotu te nije dovodila u sukob verzije različitih biblioteka i paketa koje smo instalirali.

Nakon klasične instalacije, potrebno je instalirati SDK – „Software Development Kit“ [5] kojeg pruža kompanija SoftBank Robotics. To je skup alata koji nam omogućavaju uspostavljanje veze s NAO robotom te izgradnju naših funkcija kojima upravljamo i programiramo modulima istoga. Nešto više o korištenju i programiranju modula reći ćemo u poglavlju 3.2 Implementacija.

Nadalje, opisat ćemo povezivanje SDK-a unutar Python okruženja.

Ukoliko je Python27 instaliran na C disku, pristupamo datoteci na putanji „C:\Python27\Lib\site-packages“ te ondje pohranjujemo SDK datoteku koju preimenujemo u „pynaoqi“.

Zatim je potrebno u PATH varijablu okruženja dodati nekoliko putanja:

C:\Python27\Lib

C:\Python27\Lib\Scripts

Nakon toga trebamo dodati novu varijablu okruženja:

Ime varijable: **PYTHONPATH**

Vrijednost varijable: **C:\Python27\Lib\site-packages\pynaoqi\lib**

Sada bi „import naoqi“ trebao raditi unutar Pythona.

Potrebne Python biblioteke i njihove verzije jesu:

pip install matplotlib==2.2.5 (bitno je instalirati ovu verziju)

pip install scikit-image (verzija u radu 0.14.5)

pip install numpy (verzija u radu 1.16.6)

pip install pillow (verzija u radu 6.2.2)

pip install opencv-python

Izrazito je bitno pratiti korake instalacije i koristiti verzije koje smo naveli. Tijekom izrade završnog rada, mnogo puta nam se događalo da verzije nisu kompatibilne te smo na neki način velikim brojem pokušaja morali „pogoditi“ ili se snaći u pronalasku verzije koja će raditi.

U sljedećem ćemo poglavlju reći nešto više o metodama koje smo koristili i kako smo ih implementirali pomoću Python programskog jezika. Također, navest ćemo gdje se originalni kod može pronaći i isprobati.

Poglavlje 3

Metode, klasifikacija i implementacije

Metode koje ćemo razmotriti jesu redom „naivna“ metoda, metoda minimiziranja algebarske udaljenosti te metoda minimiziranja geometrijske udaljenosti.

Sve tri metode imaju jednak cilj. Pronaći parametre „fitane“ kružnice koja će najbolje odgovarati detektiranom rubu objekta. Međutim, način na koji dolazimo do istih je drugačiji te ćemo u narednom tekstu objasniti svaku od metoda zasebno.

U naivnoj metodi određujemo dva parametra „fitane“ kružnice – radijus i centar kojima ćemo iscrtavati standardnu jednadžbu kružnice. Više u poglavlju 3.1 Naivna metoda.

Drugom metodom određujemo parametre algebarski zadane jednadžbe kružnice (jednadžba (3.3)) u ravnini kojima ponovno računamo radijus i centar izvodom formula. U tom ćemo slučaju dobiti sustav linearnih jednadžbi koji ćemo rješavati uz pomoć SVD (singularne) dekompozicije[3]. Više o ovoj metodi pisat ćemo u poglavlju 3.2 Metoda minimizacije algebarske udaljenosti.

Trećom metodom rješavamo sustav nelinearnih jednadžbi te ćemo stoga koristiti Gauss-Newtonovu metodu[4] iteracija. Za početni korak uzet ćemo upravo rješenje iz druge metode. Na taj bismo način trebali dobiti preciznije izračune radijusa i koordinata centra.

Za sve tri metode želimo da naša „fitana“ kružnica što bolje odgovara koordinatama piksela koji određuju rub objekta tj. „edge-detection“ slici. To bi značilo da ćemo, nakon preuzimanja slike s robota, prvo načiniti „edge-detection“ sliku koja sadrži samo piksele ruba objekta, a zatim pokušati odrediti kružnicu koja najbolje pokriva koordinate piksela koji određuju rub.

Ideja je da na osnovu usporedbe razlike oblika „edge-detection“ objekta i „fitane“ kružnice zaključimo radi li se o lopti ili o nekom drugom obliku.

[3] Više o SVD dekompoziciji može se pročitati na [11]
[4] Više o Gauss-Newtonovoj metodi može se pročitati na [12]

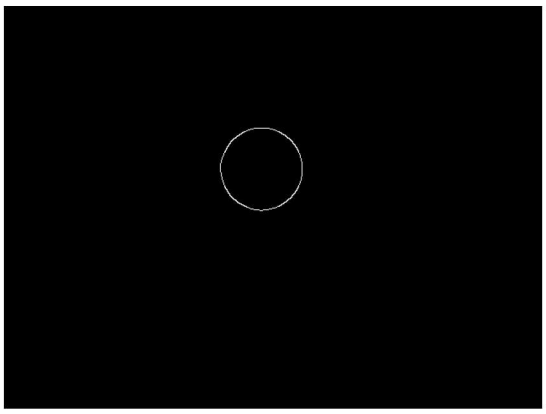
3.1 Naivna metoda

Razlog zbog kojeg smo ovu metodu nazvali naivnom je način na koji radi. Vrlo je intuitivan i zapravo je najlakši način na koji bi riješili ovaj problem uz korištenje jednostavne matematike.

Kao što smo u uvodu ovog poglavlja rekli, naivnom metodom želimo odrediti parametre kružnice (radijus - R te (X,Y) točku centra) koja najbolje pokriva rub objekta.

Kako bismo to mogli, trebamo preuzeti sliku s robota (npr. slika 16) te ju obraditi putem gotove „Canny“ funkcije unutar biblioteke „OpenCV“. Ta će nam funkcija vratiti sliku koja na koordinatama piksela koji određuju rub objekta s preuzete slike ima bijelu boju, dok je cijela pozadina i unutrašnjost objekta crne boje (slika 17). O oba pojma kao i o preuzimanju slike s robota pisat ćemo nešto više u narednom poglavlju Implementacija, jer se u ovome poglavlju želimo nešto više posvetiti metodi određivanja parametara „fitane“ kružnice naivnom metodom.

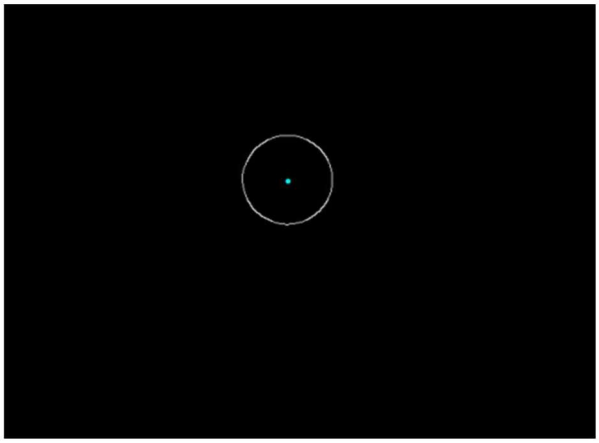
Slika 16: Sačuvana slika s robota



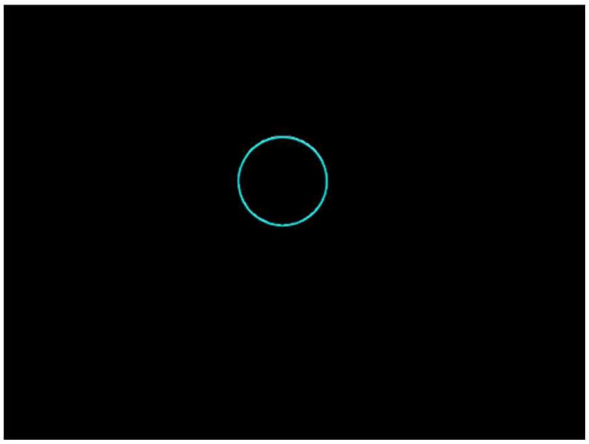
Slika 17: „Edge-detection“ slika

Nakon što imamo „edge-detection“ sliku ruba, možemo pokrenuti našu metodu. Metoda prvo određuje centar „fitane kružnice“ na način da uzme *medijan* svih x-koordinata te *medijan* svih y-koordinata piksela koji određuju rub. Na taj bismo način doista dobili centar svih bijelih piksela (slika 18). Zatim bismo prolazili po svim bijelim pikselima i određivali euklidsku udaljenost od dobivenog centra. Uzeli bismo sve izračunate vrijednosti i odredili *prosjek*[5]. Tu bismo prosječnu udaljenost centra od svih piksela uzeli kao radijus naše „fitane“ kružnice (slika 19).

[5] U poglavlju rezultati objasnit ćemo kako smo za ovaj dio metode isprobali i medijan, a za centar isprobali prosjek te ćemo ih direktno usporediti



Slika 18: Centar „fitane“ kružnice

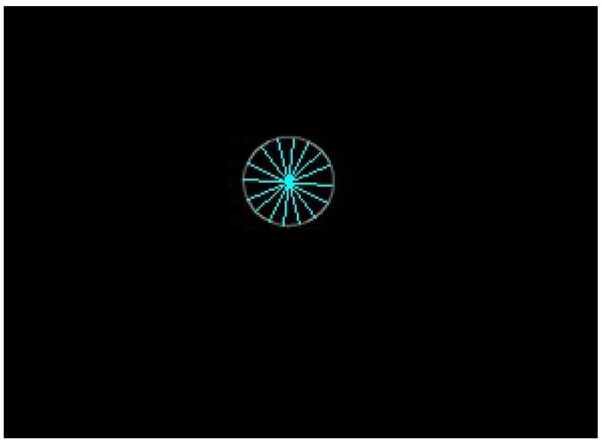


Slika 19: Radijus „fitane“ kružnice

Važno je za naglasiti da je ovo dakako idealan slučaj i idealan rezultat „edge-detection“ funkcije te smo upravo zato mogli imati veliku preciznost u iscrtavanju „fitane“ kružnice kao i u procesu klasifikacije. U poglavlju 4 Rezultati, dotaknut ćemo se mnogih primjera koji imaju različite oblike te ćemo objasniti kako smetnje i greške prilikom određivanja ruba mogu naštetiti „fitanju“ kružnice, a posljedično i klasifikaciji. Također, raspravit ćemo o razlikama u rezultatima pri korištenju prosjeka te medijana za određivanje centra i radijusa „fitane kružnice“. Konkretno u ovom poglavlju, kao što smo gore napisali, opisana je metoda u kojoj smo koristili medijan za određivanje centra, a prosjek za određivanje radijusa.

3.1.1 Klasifikacija

Nakon izračunatih parametara „fitane kružnice“ računamo našu „loss funkciju“[6] MSE – „Mean Square Error“ na način da izračunamo euklidsku udaljenost centra od svakog bijelog piksela (slika 20), a onda za svaki od njih računamo odstupanje od idealnog radijusa naše „fitane“ kružnice (formula (3.1)). Od svih izračunatih odstupanja, kako i samo ime (MSE) kaže, uzimamo prosjek nakon što ih kvadriramo (formula (3.2)). Na taj način dobivamo prosječno kvadratno odstupanje našeg objekta, tj. ruba objekta, od idealne kružnice.



Slika 20: Vizualni prikaz računanja udaljenosti za MSE „loss funkciju“

Detaljno pokažimo način na koji smo izračunavali našu MSE „loss funkciju“.

Ukoliko centar označimo s koordinatama (p, q), pozicije bijelih piksela označimo kao $p_1, \dots, p_n$, gdje je $p_i = (x_i, y_i), i = 1 \dots n$, a razlike i-tog piksela s $d_i, i = 1 \dots n$, te radijus s r, razlike možemo zapisati kao:

(3.1) $$d_i = \sqrt{(x_i - p)^2 + (y_i - q)^2} - r, \quad i = 1 \dots n$$

[6] „Loss funkcija“ - poznata i kao funkcija gubitka ili funkcija troška. Ona kvantificira koliko dobro model predviđa stvarne podatke, odnosno koliko su predikcije modela različite od stvarnih vrijednosti. U kontekstu klasifikacije, označava koliko je objekt sličan, odnosno različit lopti.

Konačno, MSE računamo prema sljedećoj formuli:

(3.2) $$\frac{1}{n}\sum_{i=1}^{n}(d_i)^2$$

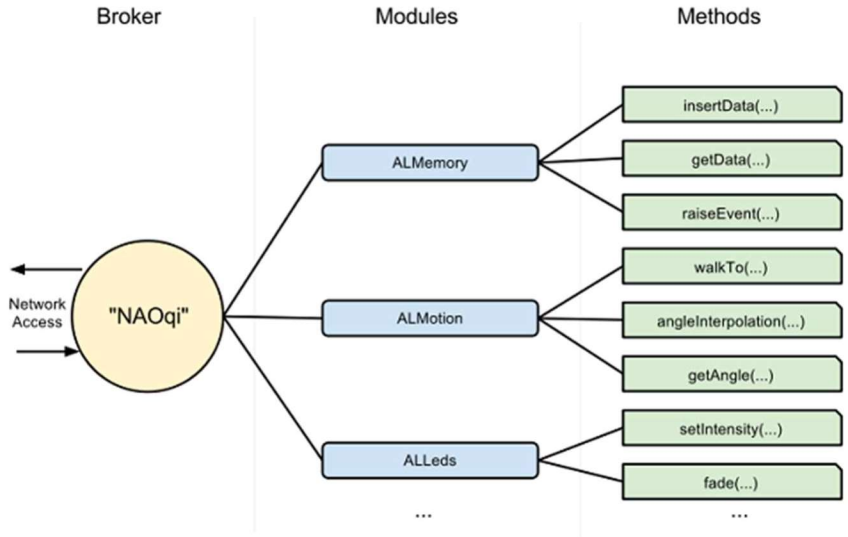
Na posljetku, potrebno je odrediti granicu za MSE koja će odrediti klasifikaciju. Ukoliko je MSE veći od postavljene granice, tzv. „thresholda“, onda će objekt biti negativno klasificiran, tj. objekt nije lopta, a ako je MSE manji ili jednak postavljenoj granici, bit će pozitivno klasificiran, tj. objekt je lopta. Unutar sljedećeg poglavlja objasnit ćemo kako smo s obzirom na klasifikaciju upravljali robotom da na glas kaže radi li se o lopti ili ne.

3.1.2 Implementacija

Implementaciju nećemo obraditi na način da objašnjavamo programski kod, nego ćemo objasniti što smo sve koristili i ukratko pojasniti kako neke ugrađene funkcije iz biblioteka rade. Programski kod direktno prati opis metode iz poglavlja 3.1 Naivna metoda te je doista čitljiv. Naravno, dajemo pristup programskom kodu na GitHub linku [8] za sve one koji žele detaljnije proučiti implementaciju, kao i pristup dokumentaciji robota koju smo i sami koristili [3].

Najbitnije je spomenuti da koristimo ALProxy klasu iz naoqi biblioteke koju smo dodali unutar poglavlja 2.2.1 Instalacija. Iz iste ćemo pozivati razne ugrađene „Proxy“ elemente.

„Proxy“ je ime za objekt koji predstavlja modul robota. Tako na primjer ako kreiramo „Proxy“ na različiti modul, dobit ćemo objekt koji ima sve metode koji taj „Proxy“ sadrži. Na slici 21 vidimo kako NAOqi biblioteka radi poput brokera, tj. posrednika koji s obzirom na odabrani „Proxy“ pruža različite metode za korištenje.



Slika 21: Odnos između poslužitelja, njegovih „Proxya“ i metoda unutar istih

U našem smo kodu koristili „ALVideoDevice“ koji označava „Proxy“ za kameru robota. Označili smo taj proxy kao camProxy te koristili njegove dvije funkcije (metode). Funkciju „subscribeCamera“ koristili smo da bismo se priključili na kameru, a funkciju „getImageRemote“ kako bismo sačuvali sliku koju robot vidi. Naravno, na posljetku morali smo se odjaviti s korištenja kamere metodom „unsubscribe“.

Jer metoda „getImageRemote“ vraća „RAW“ format slike gdje za svaki piksel imamo R, G i B vrijednosti koje opisuju njegovu boju, koristimo klasu Image iz PIL biblioteke („Pillow“ paket koji je instaliran u poglavlju 2.2.1 Instalacija).

PIL, tzv. „Python Imaging Library“ pruža podršku za razne formate slike i prilično velike mogućnosti obrade istih. Funkcija „frombytes“ služi upravo za slučaj kada želimo kreirati sliku direktno iz gore spomenutog RAW formata. Ona stvara RGB sliku dane širine i visine koja se zatim može pohraniti pomoću funkcije „save“ u željenom formatu. Mi smo konkretno odabrali „PNG“ format slike.

Nakon pohrane slike radimo „edge-detection“ koristeći „Canny edge-detector“ iz OPENCV („cv2“) biblioteke. Prije nego što se primijeni Canny algoritam[7] za detekciju rubova, potrebno je poduzeti nekoliko pripremnih koraka kako bi se osigurala učinkovitost i točnost detekcije. Ovi koraci uključuju konverziju u odgovarajući prostor boja koje promatramo, izdvajanje kanala svjetline i zamućivanje kako bi se objekt lakše mogao raspoznati od pozadine.

Zatim se primjenjuje „Canny“ algoritam koji daje sliku ruba na kojoj dalje primjenjujemo metode (naivnu ili naredne koje ćemo opisati).

U main dijelu spajamo se na robota te pozivamo sve do sad opisane funkcije prateći redoslijed. Važno je za naglasiti da je za spajanje s robotom, direktno putem Pythona, potreban IP[8] robota. IP robota dobijemo na način da pritisnemo „NAO“ tipku na robotu. On će na glas izreći svoj IP (vidi sliku 22).

[7] „Canny algoritam“ – ugrađeni algoritam za detekciju ruba objekta unutar OpenCV biblioteke. Nešto više o istome može se pročitati na [9].
[8] IP adresa – označava adresu internetskog protokola koju svaki uređaj povezan s mrežom posjeduje. U našem smo slučaju mrežnim kablom povezali robota i računalo.

Slika 22: Robot izgovara IP

Kao što smo objasnili u poglavlju 3.1 Naivna metoda, s obzirom na klasifikaciju, robot će izreći „It is a ball“ ili „It is not a ball“. Kako bismo to mogli, u krajnjem ćemo koraku koristiti „Proxy“ zvan „ALTextToSpeech“ i njegovu metodu „say“.

3.2 Metoda minimizacije algebarske udaljenosti

Metoda minimiziranja algebarske udaljenosti [9] računa koeficijente a, b i c iz algebarske reprezentacije kružnice dane izrazom (3.3).

(3.3)
$$F(x) = ax^T x + b^T x + c = 0,$$
$$a \neq 0,$$
$$x, b \in \mathbb{R}^2$$

Naime, ako raspišemo izraze za dane x i b vektore dobijemo:

$$x = (x_1, x_2)^T, \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

(3.4)
$$x^T x = {x_1}^2 + {x_2}^2$$

(3.5)
$$b^T x = b_1 x_1 + b_2 x_2$$

Ubacimo li dobivene jednadžbe (3.4) i (3.5) u početnu dobijemo:

(3.6)
$$a({x_1}^2 + {x_2}^2) + b_1 x_1 + b_2 x_2 + c = 0$$

što odgovara jednadžbi kružnice

$$(x - p)^2 + (y - q)^2 = r^2,$$

[9] Ideju za korištenje ove metode dobili smo čitajući rad Least-Squares Fitting of Circles and Ellipses [10]

jer se može zapisati upravo u formi kao (3.6):

$$x^2 - 2px + p^2 + y^2 - 2qy + q^2 = r^2$$
$$x^2 + y^2 - 2px - 2qy + (p^2 + q^2 - r^2) = 0.$$

Dakle, s obzirom na našu algebarsku reprezentaciju kružnice danu s (3.3) te izvedenu formulu (3.6) imamo vektor nepoznanica koje želimo izračunati kako bismo dobili nepoznate parametre „fitane“ kružnice. Označimo ih kao elemente vektora *u*:

(3.7) $$u = (a, b_1, b_2, c)^T$$

Pretpostavimo da imamo m podataka $x_i = (x_{i1}, x_{i2})^T$, gdje je $i = 1,2, \dots, m$.

Uvrštavanjem elemenata vektora x_i u jednadžbu (3.6), gdje je $x_1 = x_{i1}$, a $x_2 = x_{i2}$ dobijemo:

$$a(x_{i1}^2 + x_{i2}^2) + b_1 x_{i1} + b_2 x_{i2} + c = 0$$

Zapišimo to kao linearni sustav $Bu = 0$ gdje je u dan izrazom (3.7):

Za svaki će podatak $x_i = (x_{i1}, x_{i2})$, i-ti redak matrice B biti jednak:

$$(x_{i1}^2 + x_{i2}^2, x_{i1}, x_{i2}, 1)$$

Konstruiranjem svih redova za sve podatke $i = 1, 2, \dots, m$ dobijemo matricu B:

(3.8) $$B = \begin{pmatrix} x_{11}^2 + x_{12}^2 & x_{11} & x_{12} & 1 \\ x_{21}^2 + x_{22}^2 & x_{21} & x_{22} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1}^2 + x_{m2}^2 & x_{m1} & x_{m2} & 1 \end{pmatrix}$$

Dakle, linearni sustav $Bu = 0$ sadrži m jednadžbi.

Da bismo dobili netrivijalno rješenje, uzimamo da je $u_1 = 1$ ili $\|u\|_2 = 1$.

Jasno je da za broj podataka (točaka koje čine „edge-detection“ sliku) veći od 3, tj. za $m > 3$, općenito sustav neće uvijek imati rješenje jer bi u suprotnom sve točke morale zadovoljavati jednadžbu kružnice.

Zato rješavamo sustav $Bu = r$, gdje za u uzimamo onog koji minimizira $\|r\|_2$

Ovime imamo problem minimizacije uz uvjet: $\|Bu\|_2 \rightarrow min$ uz uvjet $\|u\|_2 = 1$

S obzirom da imamo sustav linearnih jednadžbi, rješenje ćemo pronaći koristeći singularnu (SVD) dekompoziciju [11].

Rješenje je ekvivalentno pronalasku desnog singularnog vektora pridruženog najmanjoj singularnoj vrijednosti matrice B kojeg ćemo pronaći računajući U, S i V matrice singularnom dekompozicijom [10].

U sljedećem ćemo odlomku opisati teorijski način izračunavanja desnog singularnog vektora pridruženog najmanjoj singularnoj vrijednosti, dok u praksi postoje efikasne metode za izračunavanje SVD dekompozicije.

Kako bismo pronašli desni singularni vektor pridružen najmanjoj singularnoj vrijednosti, potrebno je pronaći matricu $M = B^T B$ te odrediti svojstvene vrijednosti iz $\det(M - \lambda I) = 0$ gdje je I jedinična matrica, a λ svojstvena vrijednost. Zatim, za svaku svojstvenu vrijednost λ_i izračunavamo svojstveni vektor k_i rješavajući $Mk_i = \lambda_i k_i$. Na kraju, pomoću svojstvenih vektora konstruiramo matricu V^T čiji su retci izračunati svojstveni vektori k_i. S obzirom na dijagonalno poredane singularne vrijednosti od najveće prema najmanjoj, desni singularni vektor pridružen najmanjoj singularnoj vrijednosti je upravo zadnji redak matrice V^T (slika 24).

Time ćemo dobiti vektor u (3.7) koji sadrži koeficijente a, b_1, b_2 i c.

Zato nam je potrebno izraziti formule za koordinatu centra i radijus kako bismo uz dobivene koeficijente a, b_1, b_2 i c mogli izračunati iste.

Iz (3.6) možemo pronaći koordinate centra te izraziti radijus za $x = (x_1, x_2)^T$:

$$a(x_1^2 + x_2^2) + b_1 x_1 + b_2 x_2 + c = 0$$

Grupiranjem imamo:

$$a\left(x_1^2 + \frac{b_1}{a} x_1\right) + a\left(x_2^2 + \frac{b_2}{a} x_2\right) + c = 0$$

Nadopunimo zagrade do punih kvadrata:

$$a\left(\left(x_1+\frac{b_1}{2a}\right)^2-\left(\frac{b_1}{2a}\right)^2\right)+a\left(\left(x_2+\frac{b_2}{2a}\right)^2-\left(\frac{b_2}{2a}\right)^2\right)+c=0$$

Pojednostavimo izraz:

$$a\left(x_1+\frac{b_1}{2a}\right)^2+a\left(x_2+\frac{b_2}{2a}\right)^2=\frac{b_1^2}{4a}+\frac{b_2^2}{4a}-c$$

$$\left(x_1+\frac{b_1}{2a}\right)^2+\left(x_2+\frac{b_2}{2a}\right)^2=\frac{b_1^2+b_2^2-4ac}{4a^2}$$

(3.9) $$\left(x_1+\frac{b_1}{2a}\right)^2+\left(x_2+\frac{b_2}{2a}\right)^2=\frac{(\|b\|_2)^2}{4a^2}-\frac{c}{a},\ a\neq 0$$

(3.9) je jednadžba kružnice iz koje pročitamo centar i radijus ukoliko je desna strana pozitivna.

(3.10) Centar $z=(p,q)=\left(-\frac{b_1}{2a},-\frac{b_2}{2a}\right)$

(3.11) Radijus $r=\sqrt{\frac{(\|b\|_2)^2}{4a^2}-\frac{c}{a}}$

Nakon izračunatih parametara (3.10) i (3.11) iscrtavamo „fitanu“ kružnicu te ju uspoređujemo s „edge-detection“ slikom pomoću MSE „loss funkcije“. Na isti način kao i za naivnu metodu što je opisano u poglavlju 3.1.1 Klasifikacija računamo vrijednost MSE te vršimo klasifikaciju pomoću „threshold“ vrijednosti.

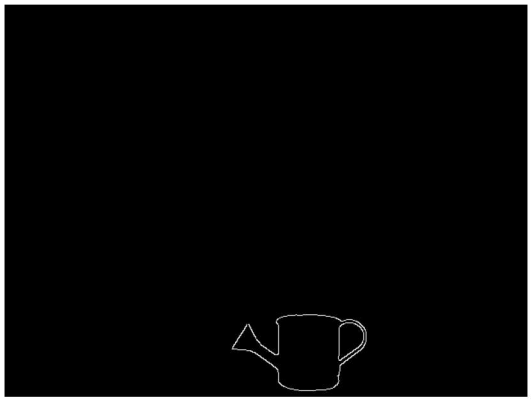
Ova metoda je izrazito jednostavna, ali ima svoje nedostatke. Nerijetko nismo sigurni što točno minimiziramo u geometrijskom smislu te nam daje rezultate koje ne bismo očekivali.

Njezino rješenje često se koristi kao početna aproksimacija u metodi koju ćemo opisati u poglavlju 3.3 Metoda minimizacije geometrijske udaljenosti.

3.2.1 Implementacija

Jednako kao i u 3.1.2 Implementacija naivne metode, nećemo direktno prolaziti kroz kod već ćemo ukratko opisati što isti radi. Na GitHub linku [8] može se direktno pristupiti kodu kojim smo implementirali metodu minimizacije algebarske udaljenosti.

Naime, u našem kodu prvo detektiramo bijele piksele jer oni označavaju rub našeg objekta (vidi sliku 23).

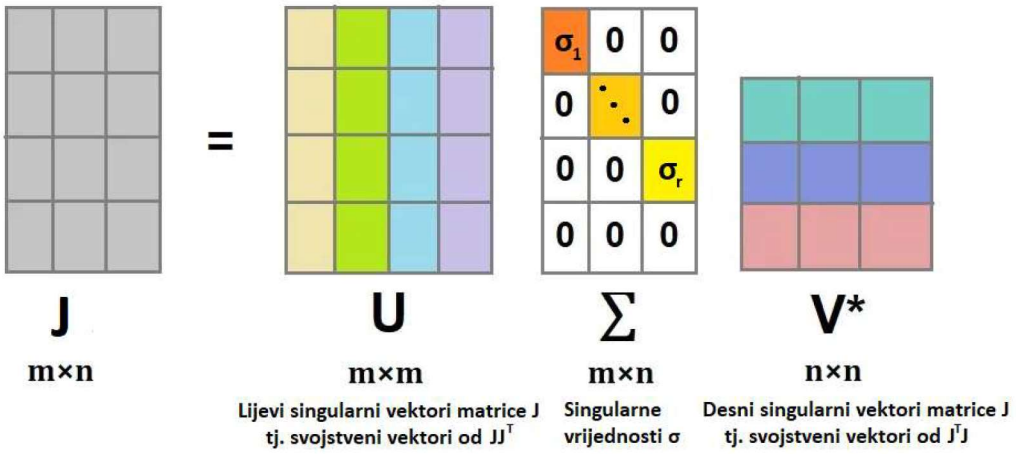


Slika 23: „Edge-detection“ slika

Ti bijeli pikseli su upravo naši podaci x_i od kojih konstruiramo matricu B po uzoru na (3.8).

Ugrađenom SVD metodom unutar numpy biblioteke dobijemo U, S i V matrice.

Kao što smo u prethodnom poglavlju napomenuli, rješenje, odnosno vektor nepoznanica $\boldsymbol{u}$, je desni singularni vektor koji je pridružen najmanjoj singularnoj vrijednosti matrice B (prema [10]). Stoga uzimamo zadnji singularni vektor iz matrice V^T znajući da su naše singularne vrijednosti dijagonalno poredane u matrici S od najveće prema najmanjoj (vidi sliku 24).



Slika 24: U, S i V matrice singularne dekompozicije

Jer je to rješenje zapravo vektor $\boldsymbol{u}$, njegovi elementi odgovaraju nepoznanicama a, b_1, b_2 i c.

Nakon što imamo sve nepoznanice, preostalo je uvrstiti ih u formule za određivanje centra i radijusa koje smo izveli u prošlom poglavlju (formule 3.10 i 3.11).

Na kraju iscrtavamo „fitanu“ kružnicu jer imamo sve potrebne parametre te na jednak način vršimo klasifikaciju izračunavanjem odstupanja od detektiranog ruba objekta pomoću MSE koji je opisan u poglavlju 3.1.1 Klasifikacija.

3.3 Metoda minimizacije geometrijske udaljenosti

U ovoj metodi[10] minimiziramo sumu kvadrata udaljenosti $d_i^2 = \left(||z - x_i|| - r\right)^2$ gdje su koordinate centra i podataka dane sa $z = (z_1, z_2), x_i = (x_{i1}, x_{i2})$ tj. minimiziramo izraz:

$$\sum_{i=1}^{m} d_i^2 = \sum_{i=1}^{m} \left(||z - x_i|| - r\right)^2$$

$$d_i(v) = \sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2} - r \tag{3.12}$$

Želimo naći $\boldsymbol{v}$ takav da je $\sum_{i=1}^{m} d_i(v)^2$ minimalna, a $v = (z_1, z_2, r)^T$

Jer se ovdje očigledno (prema 3.12) radi o sustavu nelinearnih jednadžbi koristit ćemo Gauss-Newtonovu metodu [12] za rješavanje. Njome ćemo prvo inicijalizirati nepoznate parametre s nekom početnom aproksimacijom, a zatim ćemo računati funkciju reziduala $d_i(v)$ koja u našem slučaju predstavlja razliku udaljenosti točke x_i od našeg centra z. Nakon toga, računamo Jacobian uz pomoć kojega se u svakoj od iteracija formira sustav normalnih jednadžbi zapisan preko Jacobiana i vektora reziduala. Rješavanjem sustava dobijemo korekcijski vektor kojime ažuriramo parametre. Postupak nastavljamo dok se ne dostigne postavljeni broj iteracija ili promjena u korekcijskim vrijednostima bude manja od tolerancije.

Kao što smo napisali, za Gauss-Newtonovu metodu potreban nam je Jacobian koji sadrži sve parcijalne derivacije funkcije $d_i(v)$ po svim nepoznanicama od $\boldsymbol{v}$, tj. po z_1, z_2 i r:

$$\frac{\partial d_i}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2} - r\right) = \frac{z_1 - x_{i1}}{\sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2}}$$

$$\frac{\partial d_i}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2} - r\right) = \frac{z_2 - x_{i2}}{\sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2}}$$

$$\frac{\partial d_i}{\partial r} = \frac{\partial}{\partial r}\left(\sqrt{(z_1 - x_{i1})^2 + (z_2 - x_{i2})^2} - r\right) = -1$$

[10] Ideju za korištenje ove metode dobili smo čitajući rad Least-Squares Fitting of Circles and Ellipses [10]

Konstruirajmo Jacobian matricu $J(v)$:

$$J(v) = \begin{pmatrix} \frac{z_1 - x_{11}}{\sqrt{(z_1 - x_{11})^2 + (z_2 - x_{12})^2}} & \frac{z_2 - x_{12}}{\sqrt{(z_1 - x_{11})^2 + (z_2 - x_{12})^2}} & -1 \\ \vdots & \vdots & \vdots \\ \frac{z_1 - x_{m1}}{\sqrt{(z_1 - x_{m1})^2 + (z_2 - x_{m2})^2}} & \frac{z_2 - x_{m2}}{\sqrt{(z_1 - x_{m1})^2 + (z_2 - x_{m2})^2}} & -1 \end{pmatrix}$$

Za provođenje Gauss-Newtonove metode potrebna nam je još samo početna aproksimacija koju ćemo dobiti tako što izračunamo sustav linearnih jednadžbi iz 3.2 Metoda minimizacije algebarske udaljenosti te za početne aproksimacije centra i radijusa uzmemo rješenja formula (3.10) i (3.11).

Zatim na objašnjen način provodimo Gauss-Newtonovu metodu sa zadanim brojem iteracija i tolerancijom.

Nakon što se metoda zaustavi, iscrtavamo kružnicu s dobivenim parametrima radijusa i centra. Klasifikaciju ponovno provodimo korištenjem „loss funkcije“ MSE i „thresholdom“ što smo detaljno opisali u poglavlju 3.1.1 Klasifikacija.

U sljedećem ćemo poglavlju opisati kako smo ju implementirali, a u rezultatima ćemo prikazati jesmo li vidjeli značajnu razliku između sve tri metode.

U teoriji bi treća metoda trebala biti najtočnija te očekujemo da će dati najpreciznija „fitanu“ kružnicu s obzirom na „edge-detection“ sliku.

3.3.1 Implementacija

I za ovu ćemo metodu implementaciju ukratko opisati riječima te nećemo prolaziti detaljno kroz kod. Također, kod kojim je metoda implementirana bit će dostupan na GitHub linku [8].

Unutar implementacije imamo funkciju kojom konstruiramo Jacobian matricu te funkciju kojom računamo reziduale d_i.

Jednako kao i u prošloj metodi, kreiramo matricu B te koristeći ugrađenu funkciju SVD iz numpy biblioteke dobivamo U, S i V matrice.

Uzimamo desni singularni vektor pridružen najmanjoj singularnoj vrijednosti te iz njega očitavamo vrijednosti nepoznanica a, b_1, b_2 i c s kojima izračunavamo koordinate centra (z_1 i z_2) i radijus r prema (3.10) i (3.11).

Te su nam vrijednosti početna aproksimacija za Gauss-Newtonovu metodu koju provodimo jednom for petljom u kojoj provjeravamo jesmo li dostigli zadani broj iteracija (100) te jesmo li u zadanoj toleranciji (10^{-6}). Koristimo numpy ugrađene funkcije kako bi sve radilo učinkovito i brzo.

Nakon završetka metode, iscrtavamo „fitanu“ kružnicu te provodimo klasifikaciju izračunavanjem odstupanja od detektiranog ruba objekta pomoću MSE koji je opisan u poglavlju 3.1.1 Klasifikacija.

Poglavlje 4

Rezultati

4.1 Rezultati naivne metode

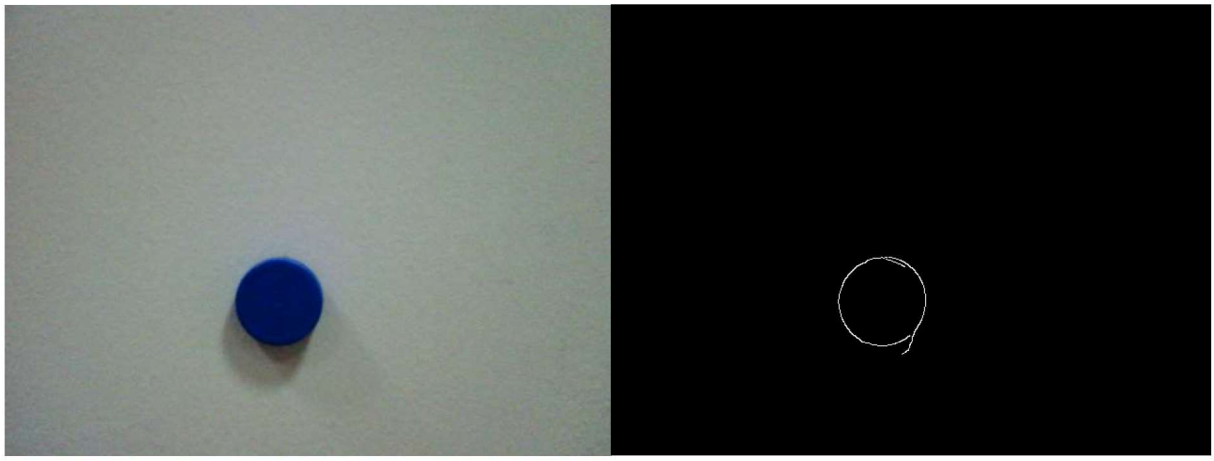
U ovom poglavlju razmotrit ćemo razne načine na koje smo pokušali dobiti što bolje rezultate klasifikacije uz naivnu metodu. Također, provjerit ćemo utječe li različit pristup određivanju centra i radijusa „fitane“ kružnice na rezultate klasifikacije. Odnosno, utječe li uzimanje medijana ili prosjeka, kojima izračunavamo parametre „fitane“ kružnice, na klasifikaciju objekta.

Za testiranje koristili smo tridesetak objekata, ali zbog količine nećemo prikazati rezultate za svaki pojedini objekt, već će se istima moći pristupiti na GitHub linku [8].

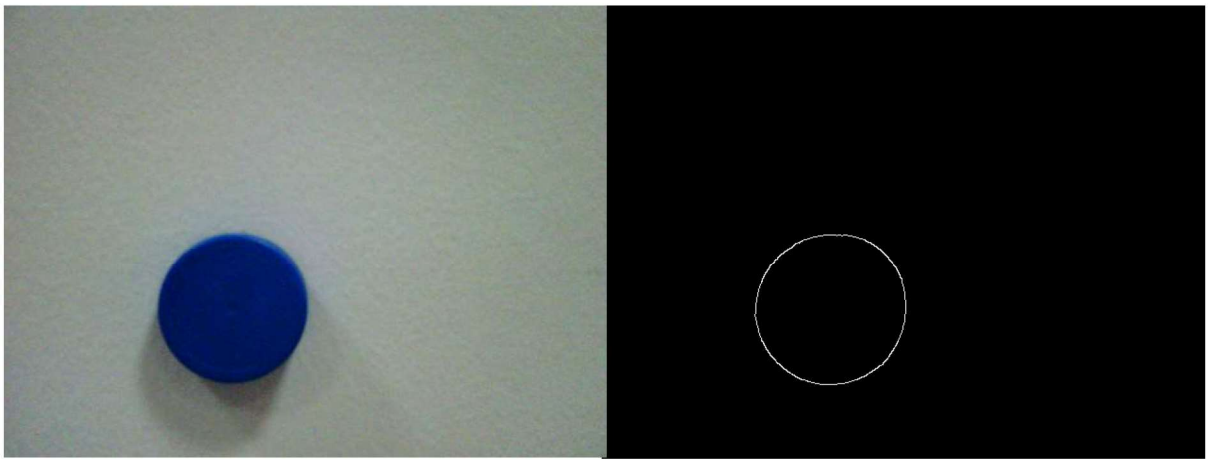
Na samom početku krenuli smo s klasifikacijom slika na nijansi bijele pozadine tj. zidu. Kao što se može primijetiti na slikama ispod, rezultati „edge-detectiona“ nisu sjajni jer imamo očigledne smetnje – „noise“, a ne jasan rub objekta kojeg bi ljudskim okom znali odrediti.

Slika 25: Slika na bijeloj pozadini i primjer „edge-detection“ slike sa smetnjama

Također, shvatili smo da je nerijetko bitna udaljenost od samog predmeta te da približavanjem robota većinom dobijemo bolju „edge-detection“ sliku (usporedi slike 26 i 27) što je bilo i očekivano.



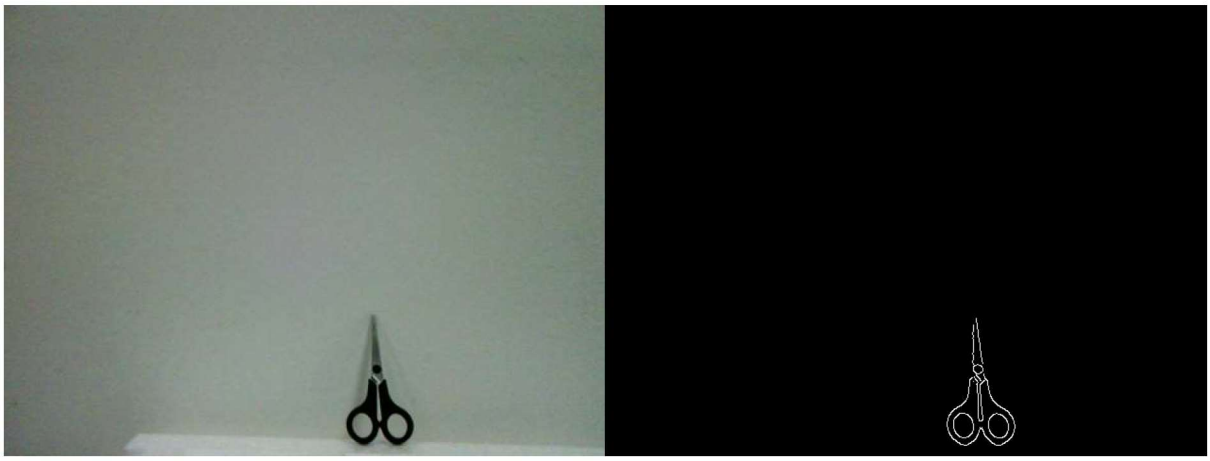
Slika 26: Primjer detekcije kad je objekt dalje



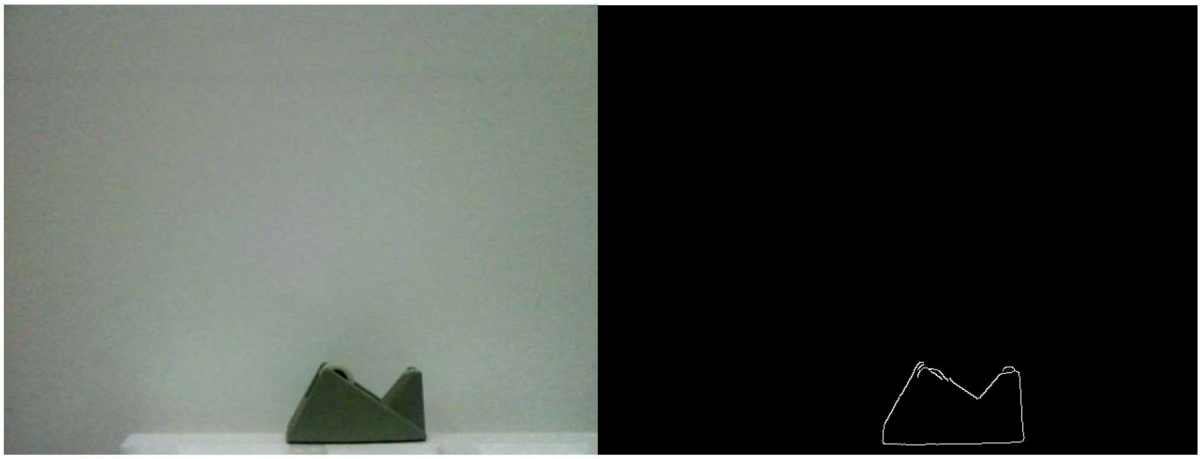
Slika 27: Primjer detekcije kad je objekt bliže

Naravno, bilo je i vrlo dobrih rezultata koje možemo vidjeti na slikama 28, 29 i 30.

Slika 28: Primjer dobre detekcije ruba



Slika 29: Primjer dobre detekcije ruba

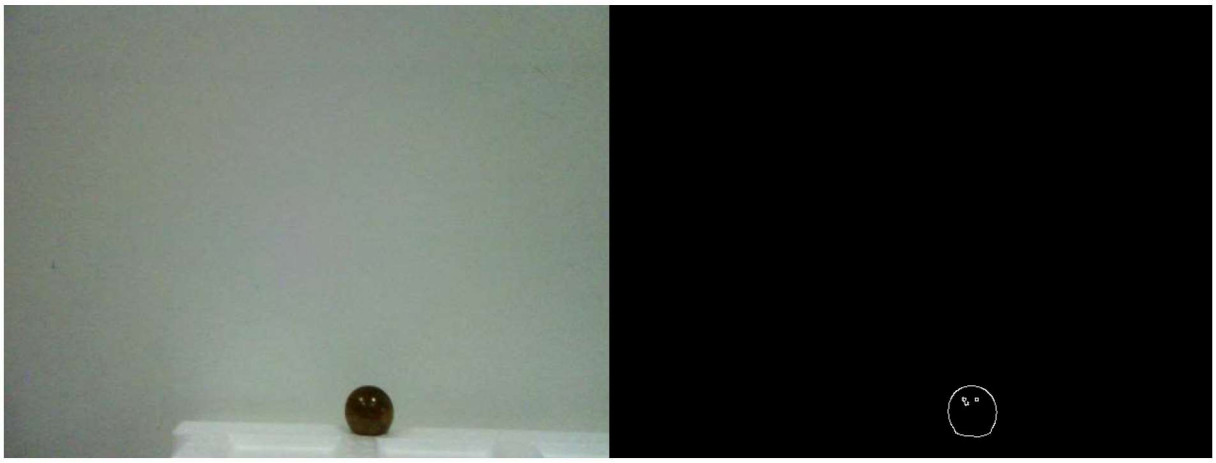


Slika 30: Primjer dobre detekcije ruba

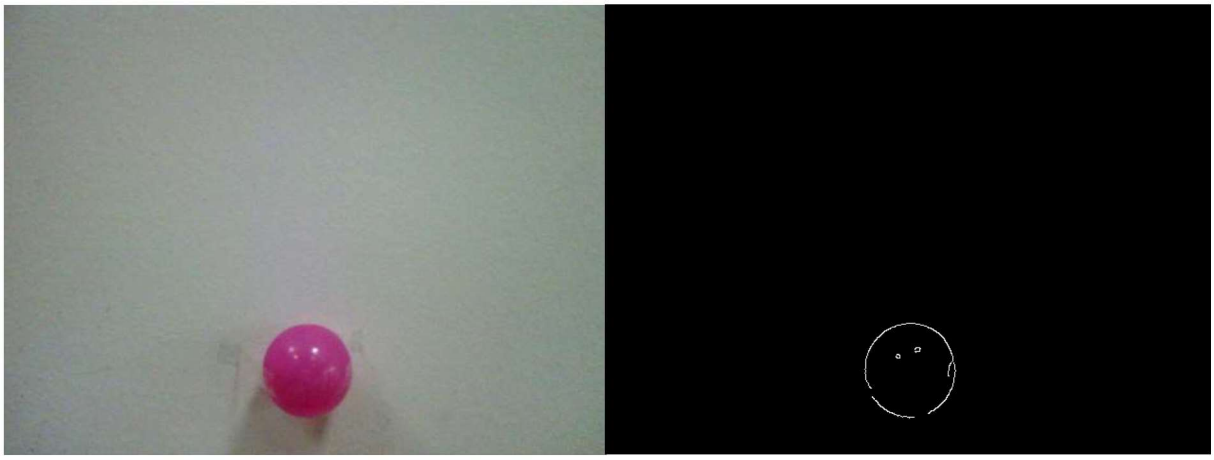
Iako bi se ljudima činilo da su čak i malo lošiji primjeri, koje dajemo u nastavku, zapravo dobri, vidjet ćemo da će našoj metodi i manje smetnje, koje nisu dio vanjskog ruba objekta, a detektirane su, smetati prilikom računanja parametara „fitane kružnice“. Samim time će i „MSE“ tj. prosječno kvadratno odstupanje od „fitane kružnice“ biti veliko te će objekt koji je lopta ipak klasificirati negativno.

Takve pogreške su posljedica činjenice da smetnje prilikom izračuna „pomaknu“ parametre „fitane“ kružnice u svoju stranu te je direktno „MSE“ veći jer se „fitana“ kružnica i „edge-detection“ slika ne poklapaju.

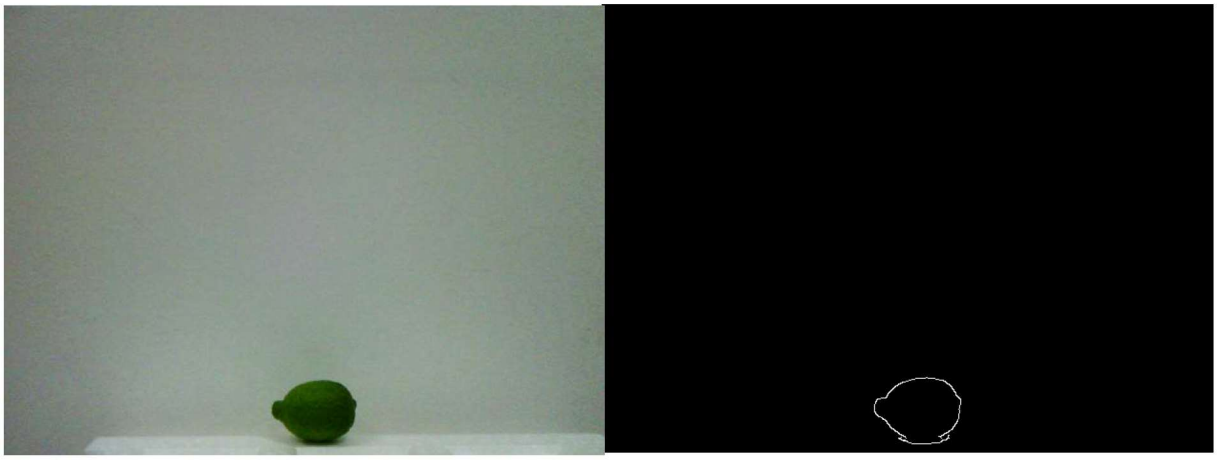
Dajemo primjere koji čovjeku izgledaju relativno dobro, ali zbog sitnih smetnji od refleksije i sjene dolazi do pogrešaka u računanju parametara „fitane kružnice“:



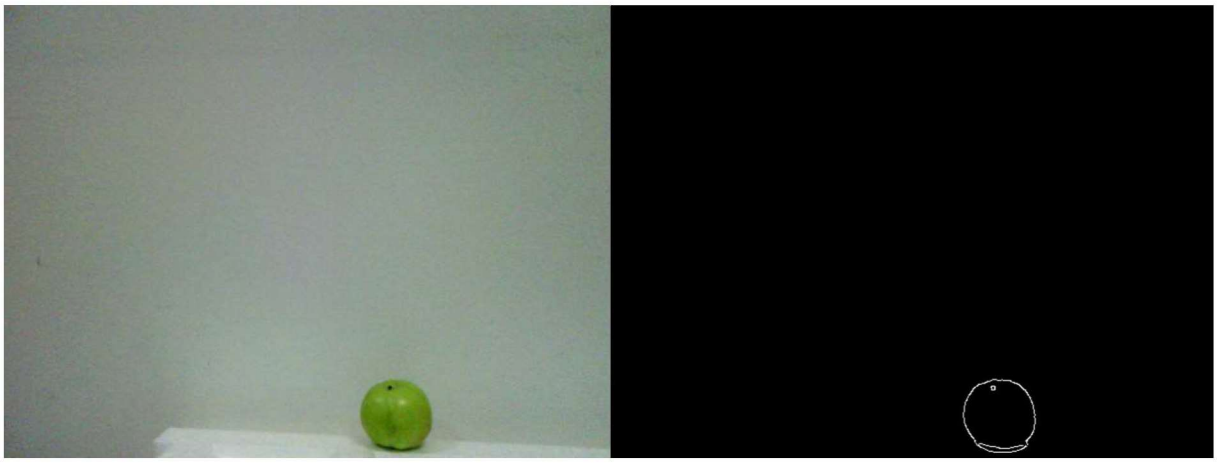
Slika 31: Primjer sa smetnjama od refleksije



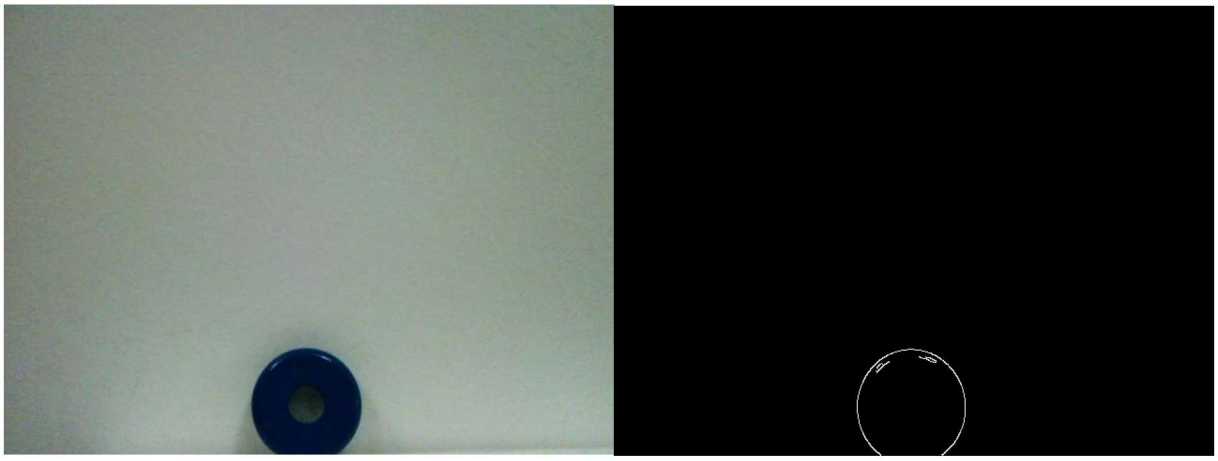
Slika 32: Primjer sa smetnjama od refleksije



Slika 33: Primjer sa smetnjama od sjene



Slika 34: Primjer sa smetnjama od sjene



Slike 35: Primjer sa smetnjama od refleksije

Pogledajmo na primjerima kako će odsjaj na objektu, koji je doista lopta, pomaknuti „fitanu“ kružnicu u smjeru odsjaja te će objekt biti pogrešno klasificiran.

Na primjerima također pokazujemo kako se naša metoda ponaša uz određivanje centra „fitane“ kružnice pomoću (redom) prosjeka i medijana.

Slika 36: MSE (redom) = 81.36 , 133.46 (FALSE, FALSE za threshold = 50)

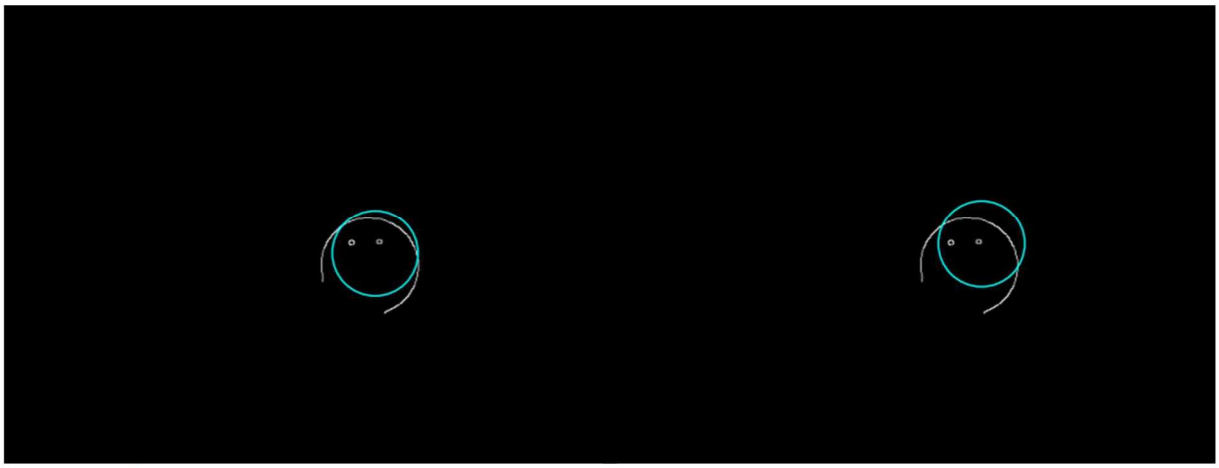
Slika 37: MSE (redom) = 152.95 , 509.99 (FALSE, FALSE za threshold = 50)

Slika 38: MSE (redom) = 40.34 , 100.09 (TRUE,FALSE za threshold = 50)

Iako je test napravljen na malo primjera, s obzirom na razliku u „MSE“ vrijednosti mogli bismo nagađati da je metoda s korištenjem prosjeka za izračun centra nešto bolja.

Također, testirali smo upotrebu medijana i prosjeka za određivanje radijusa „fitane kružnice“ te razlike gotovo i nema. Tj. razlike u MSE vrijednostima su zanemarive pa ih nismo dodatno uspoređivali kako ne bismo bespotrebno širili opseg rada. Znatiželjni mogu pristupiti kodu na GitHub linku [8] te sami promjenom medijana i prosjeka vidjeti o kojoj se razlici radi.

Međutim, iako bi se onda odsjaj možda i mogao riješiti s korištenjem prosjeka za određivanje centra te podizanjem granice tj. „thresholda“, u slučaju lošije „edge-detection“ slike, gdje fali dio ruba objekta, nismo imali rješenje (slika 39). Stoga smo odustali od korištenja bijele pozadine.



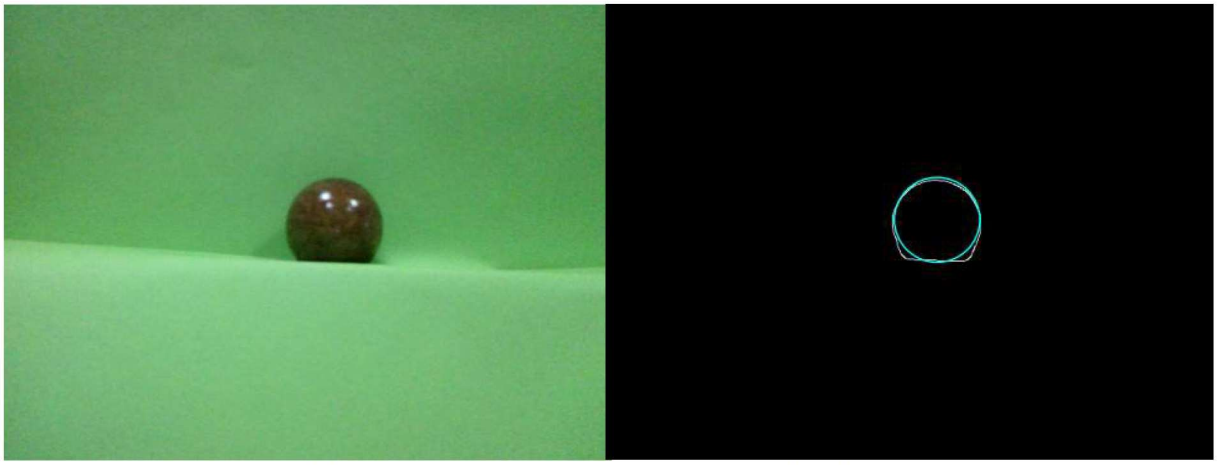
Slika 39: MSE (redom) = 148.83 , 334.07 (FALSE, FALSE za threshold = 50)

Željeli smo poboljšati rezultate metode te smo umjesto zida počeli koristiti zeleni hamer papir kao „green-screen“. Razlog zbog kojeg smo odabrali zelenu pozadinu je taj što su naši odsjaji od svjetla bili bijele boje, kao i zid, te smo zbog toga ranije imali detektirane smetnje. Naime, algoritam je detektirao odsjaj kao dio pozadine te ga izrezivao iz objekta.

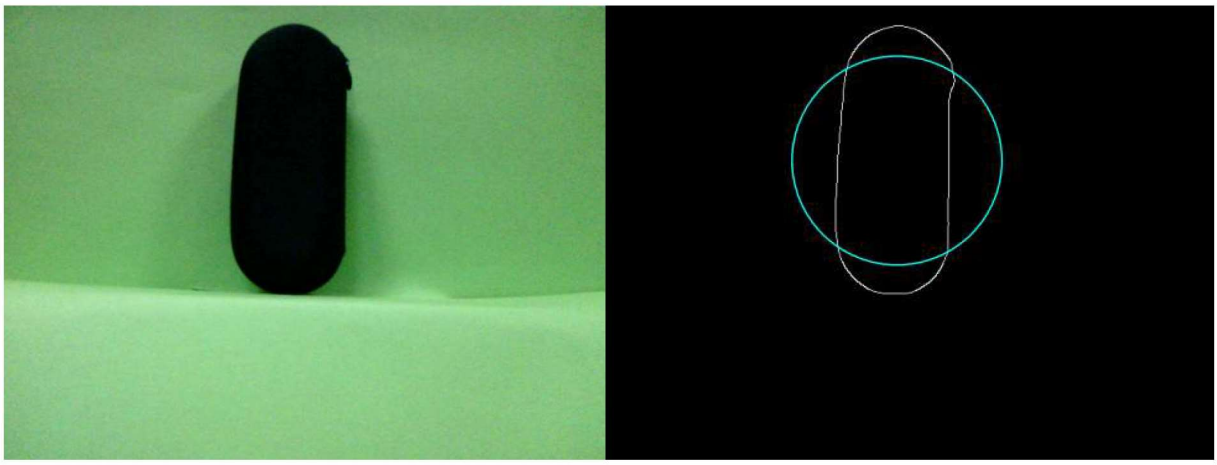
Odsjaj na zelenoj pozadini, koji je i dalje na slici, nije detektiran kao pozadina već kao dio objekta te ga zbog toga algoritam ne odstranjuje. Također, spomenuto zamućivanje iz objašnjenja metode u poglavlju 3.1 Naivna metoda dodatno je pomoglo da se i manje smetnje na samom pozivanju „Canny“ algoritma ne detektiraju.

Nadalje prilažemo rezultate koji su iznenađujuće dobri:

Slika 40: MSE = 2078.33 (FALSE za threshold = 50)

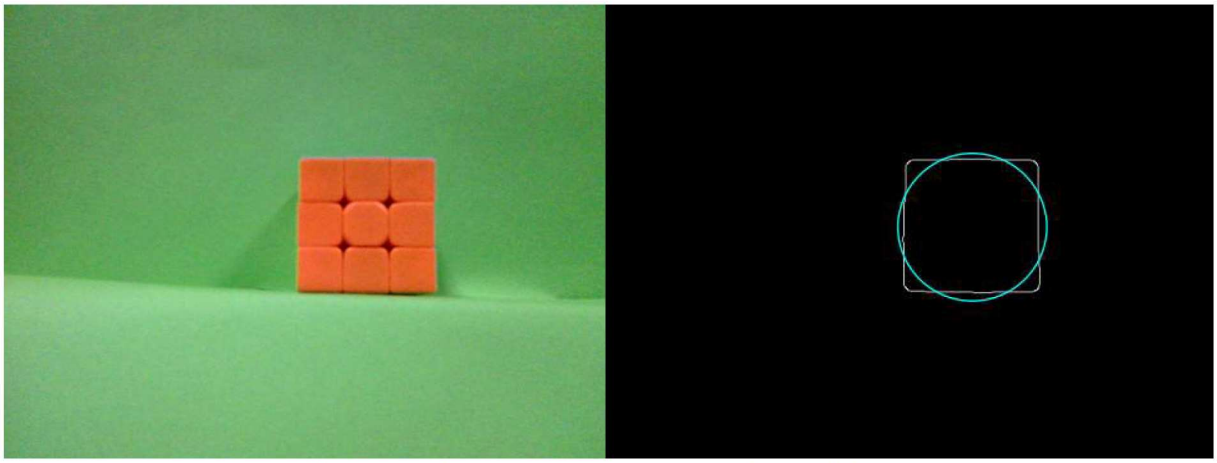


Slika 41: MSE = 11.39 (TRUE za threshold = 50)

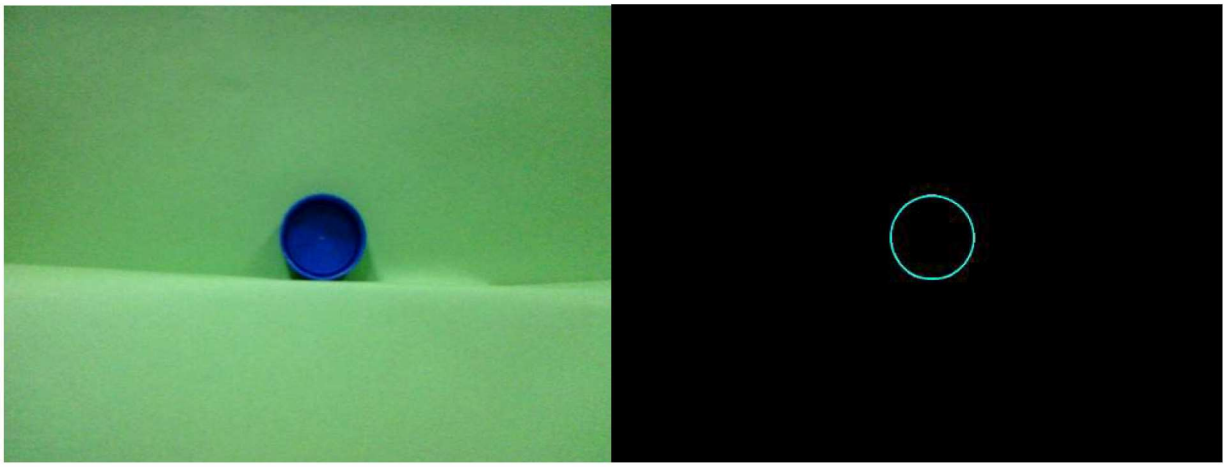


Slika 42: MSE = 862.37 (FALSE za threshold = 50)

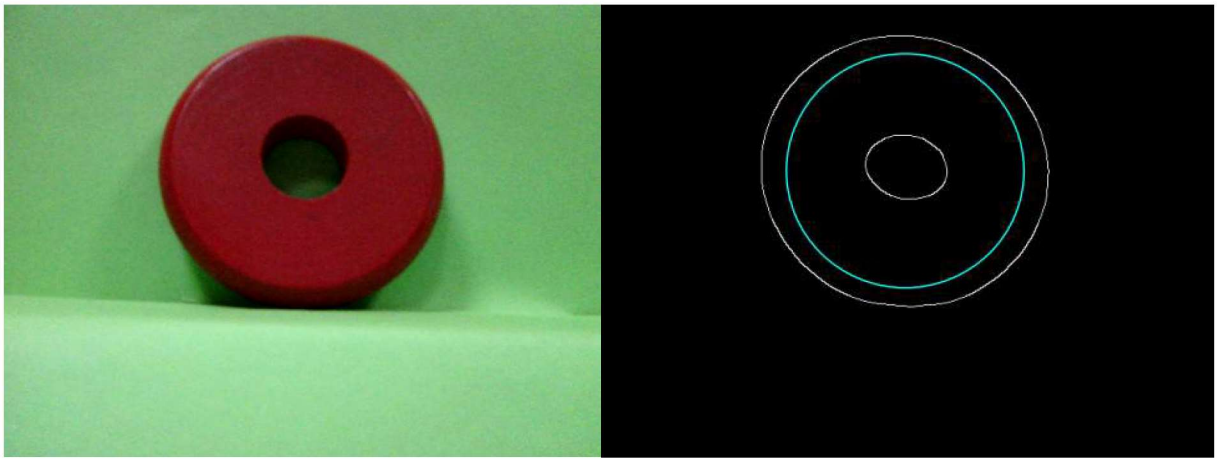
Slika 43: MSE = 8.67 (TRUE za threshold = 50)



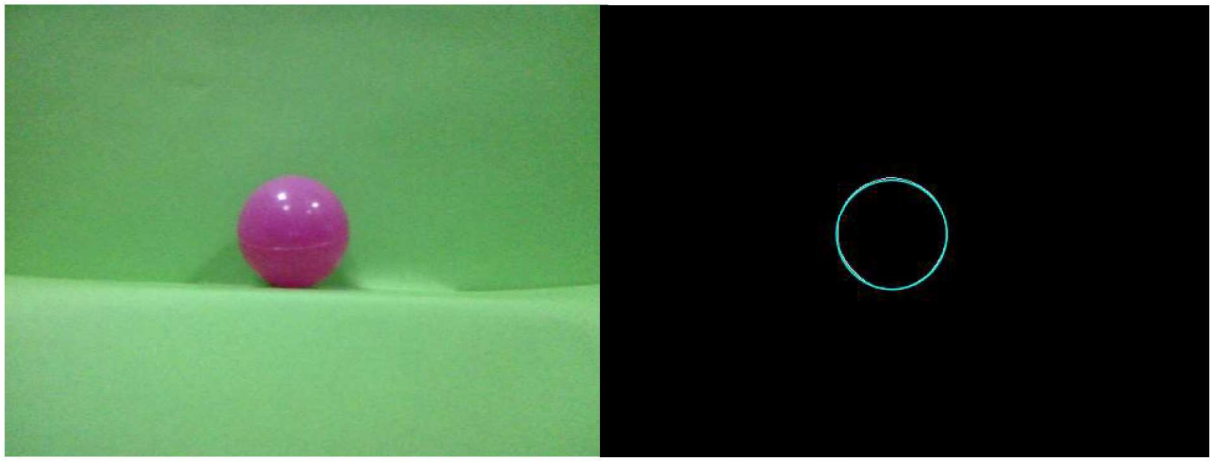
Slika 44: MSE = 68.61 (FALSE za threshold = 50)



Slika 45: MSE = 0.28 (TRUE za threshold = 50)



Slika 46: MSE = 1938.94 (FALSE za threshold = 50)



Slika 47: MSE = 1.23 (TRUE za threshold = 50)

Napominjemo da smo za određivanje centra i radijusa „fitane“ kružnice koristili prosjek jer nam se on, kako smo gore i objasnili, bio pokazao boljim izborom za klasifikaciju na bijeloj pozadini. U ovom slučaju, gdje su „edge detection“ slike izvrsne jer koristimo zelenu pozadinu, niti ne bi bilo značajne razlike u klasifikaciji korištenjem medijana ili prosjeka.

Međutim, „MSE“ vrijednosti su, kao i parametri „fitane“ kružnice, i dalje ponešto bolji korištenjem prosjeka za izračun centra i radijusa. Slično kao u slučaju klasifikacije na bijeloj pozadini.

Iako ne možemo prikazati rezultate za svaki pojedini objekt, ostvarili smo točnu klasifikaciju za svih tridesetak objekata koji su korišteni prilikom testiranja. Takvi su rezultati izvrsni, ali ne možemo tvrditi da metoda uz zelenu pozadinu neće griješiti.

Rezultate za svaki od objekata možete pogledati na GitHub linku [8].

Važno je za naglasiti da, iako klasificiramo loptu, objekti koji su okrugli (čep boce i sl.) bit će pozitivno klasificirani. Razlog tomu je nedostatak treće dimenzije koju ne provjeravamo.

Takve klasifikacije direktno ne smatramo niti kao točne niti kao pogrešne jer je to nedostatak metode koji za sad ne rješavamo.

Završno, kao digresiju, dajemo primjer pokušaja korištenja maksimuma i minimuma udaljenosti za određivanje radijusa „fitane“ kružnice.

Naime, osim spomenutog prosjeka i medijana za određivanje istoga, pokušali smo koristiti maksimum i minimum. Takvi nam se pokušaji, u slučaju „edge-detection“ slike koja sadrži smetnje od refleksije, nisu pokazali uspješnima (vidi sliku 48).

Slika 48: Korištenje minimuma i maksimuma u određivanju radijusa „fitane“ kružnice

Rezultati su očekivani jer se smetnje od refleksije nalaze unutar objekta te će samim time, korištenjem minimuma, radijus biti poprilično malen u ovisnosti o poziciji centra. (vidi lijevi dio slike 48).

U slučaju korištenja maksimuma, za slike koje sadrže smetnje izvan ruba samog objekta, radijus će biti veći nego što bi trebao jer će uzeti u obzir najdalji piksel koji je „edge-detection“ algoritam detektirao kao dio ruba.

Na kraju ovoga poglavlja, objedinimo sve rezultate unutar tablice radi lakše preglednosti. Tablica sadrži objekte koje smo detaljno opisali unutar rada, a rezultati za ostale objekte, čije slike nismo prikazali u radu, mogu se generirati pokretanjem algoritma koji se nalazi na GitHub linku [8].

Iako smo gore detaljnije objasnili zašto smo se odlučili za korištenje zelene pozadine, dodatno ćemo u tablici prikazati rezultate za obje pozadine. Naglašavamo da u tablici prilažemo rezultate na bijeloj pozadini koje smo dobili korištenjem prosjeka za računanje centra. Takav nam je pristup, kao što smo prije objasnili, dao bolje rezultate.

Stoga, stupce „MSE vrijednost“ i „Rezultat klasifikacije“ dijelimo na rezultate na bijeloj, odnosno zelenoj pozadini kako bismo pokazali razliku u rezultatima koja je direktno utjecala na odabir zelene pozadine.

Objekt	**MSE vrijednost** **Bijela / zelena pozadina**	**Rezultat klasifikacije** **Bijela / zelena pozadina**
Kantica za vodu	181.39 / 2078.33	Nije lopta / Nije lopta
Smeđa loptica	40.34 / 11.39	Lopta je / Lopta je
Futrola za naočale	362.79 / 862.37	Nije lopta / Nije lopta
Limun	18.43 / 8.67	Lopta je / Lopta je
Rubikova kocka	96.37 / 68.61	Nije lopta / Nije lopta
Čep boce	7.87 / 0.28	Lopta je / Lopta je
Disk s rupom u sredini	83.41 / 1938.94	Nije lopta / Nije lopta
Ružičasta loptica	148.83 / 1.23	Nije lopta / Lopta je

Tabica 1: Prikaz rezultata naivne metode na bijeloj i zelenoj pozadini

Iako bismo iz tablice mogli zaključiti da algoritam poprilično dobro radi i na bijeloj pozadini, vidimo da su MSE vrijednosti puno manje kada bi očekivano trebale biti veće, jer objekt nije

lopta, dok su vrijednosti puno veće kada bi trebale biti manje, tj. kada je objekt lopta (pogledaj zadnji red tablice 1).

Tako dakle, metoda na zelenoj pozadini doista ispravlja potencijalne greške koje se događaju prilikom detekcije ruba na bijeloj pozadini kada sjena ili odsjaj svjetla zasmetaju algoritmu.

„Threshold“, tj. granična vrijednost MSE vrijednosti za koju ćemo reći da je objekt lopta, ili da nije lopta u slučaju da je MSE vrijednost veća od thresholda, postavljena je na 50. Stečenim iskustvom na rezultatima velikog broja ponavljanja, zaključili smo kako je ta vrijednost dovoljna da imamo točnu klasifikaciju za sve objekte iz našeg odabranog skupa.

Tako dakle, i kada algoritam dobije nešto lošiji rezultat zbog različitih čimbenika (lošija osvjetljenost prostorije, sjena i sl.) s takvom smo se granicom osigurali da objekti koji su lopta i dalje budu klasificirani pozitivno.

4.2 Rezultati metode minimizacije algebarske udaljenosti

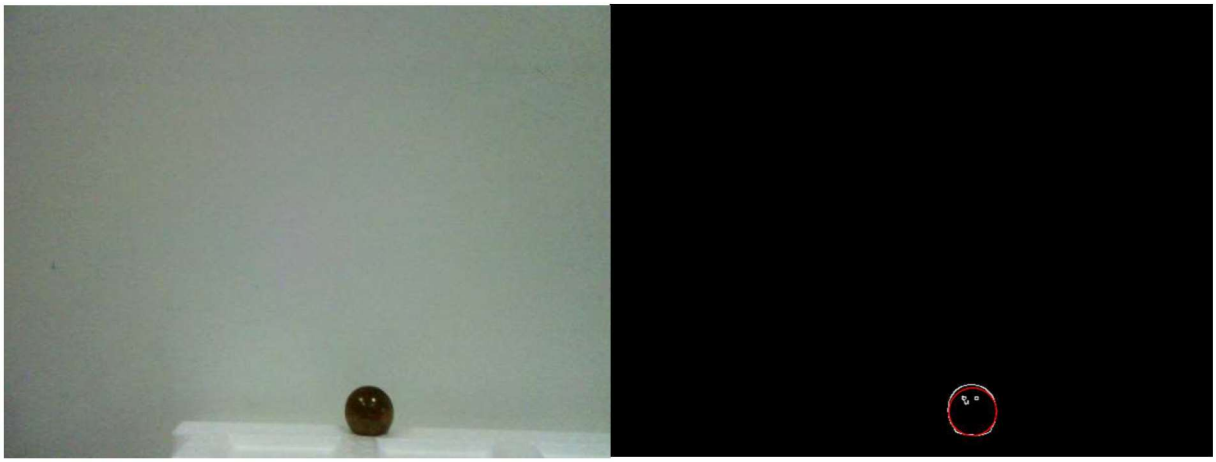
Kao što smo u poglavlju 4.1 Rezultati naivne metode napisali, imali smo probleme prilikom „fitanja“ kružnice, kod lošijih „edge-detection“ slika, kada smo slikali objekte na bijeloj pozadini tj. zidu. Takve su slike sadržavale odsjaje svjetla, sjenu ili su jednostavno dijelovi ruba bili nepotpuni.

U ovom ćemo poglavlju prikazati rezultate metode minimizacije algebarske udaljenosti koju smo opisali u poglavlju 3.2, upravo na slikama sa smetnjama.

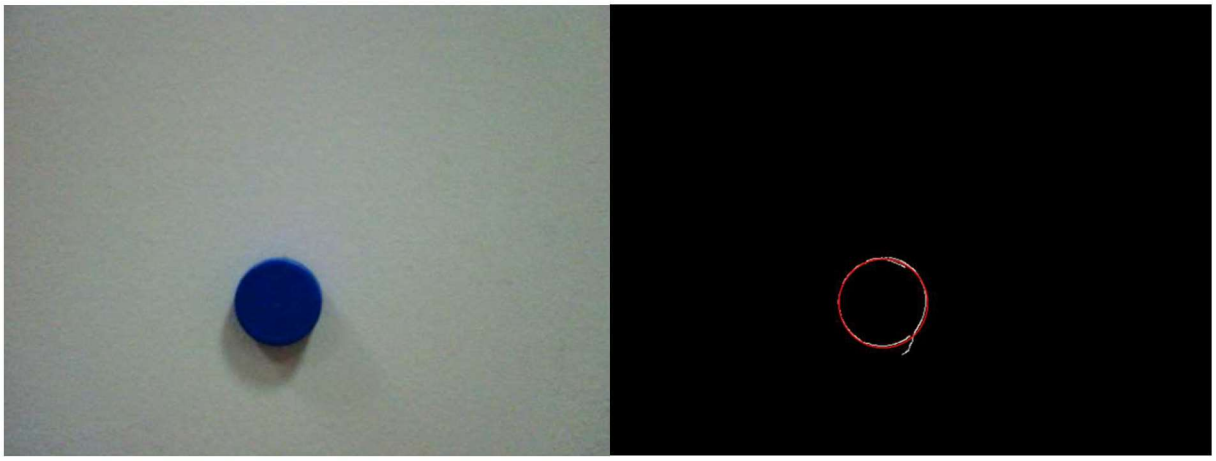
Prvo ćemo krenuti sa spornim slikama, a na kraju ćemo prikazati i klasifikaciju na boljim „edge-detection“ slikama na kojima niti naivna metoda nije griješila.

Napominjemo da nećemo prikazati klasifikaciju za svih tridesetak objekata. Klasifikacija za ostale objekte može se pronaći na GitHub linku [8].

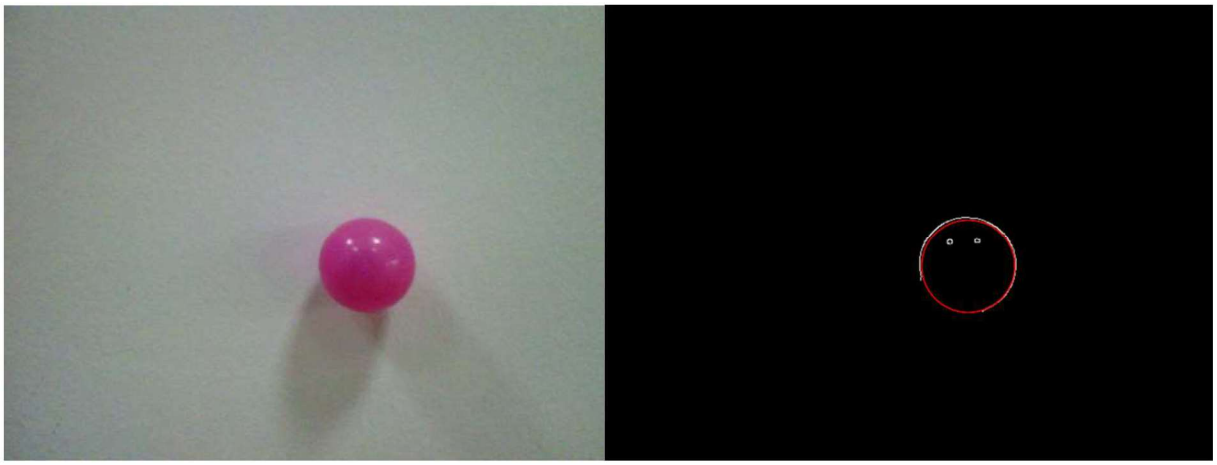
Pokažimo primjere klasifikacije sa smetnjama od odsjaja, sjene ili s nepotpunim rubom:



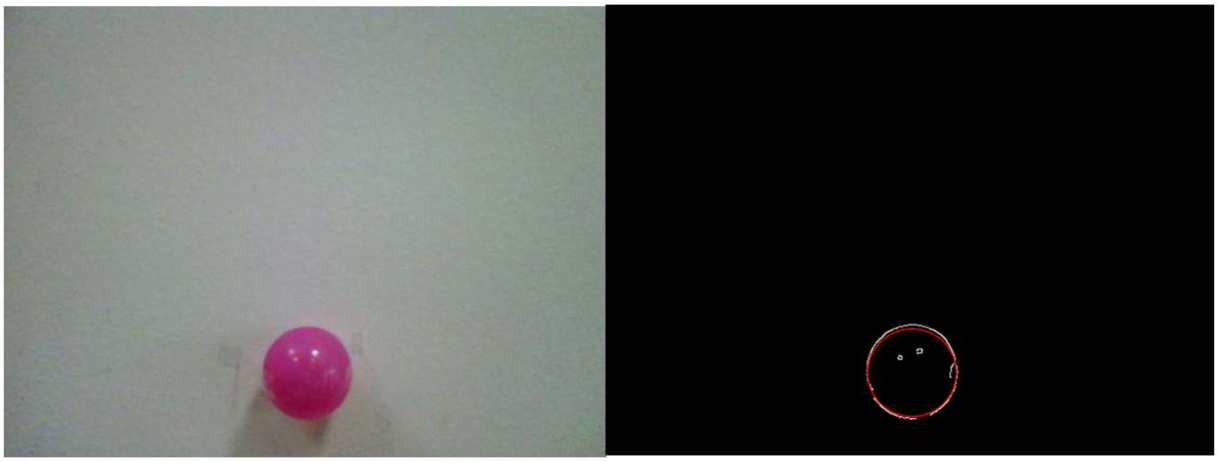
Slika 49: MSE = 23.56 (TRUE za threshold = 50)



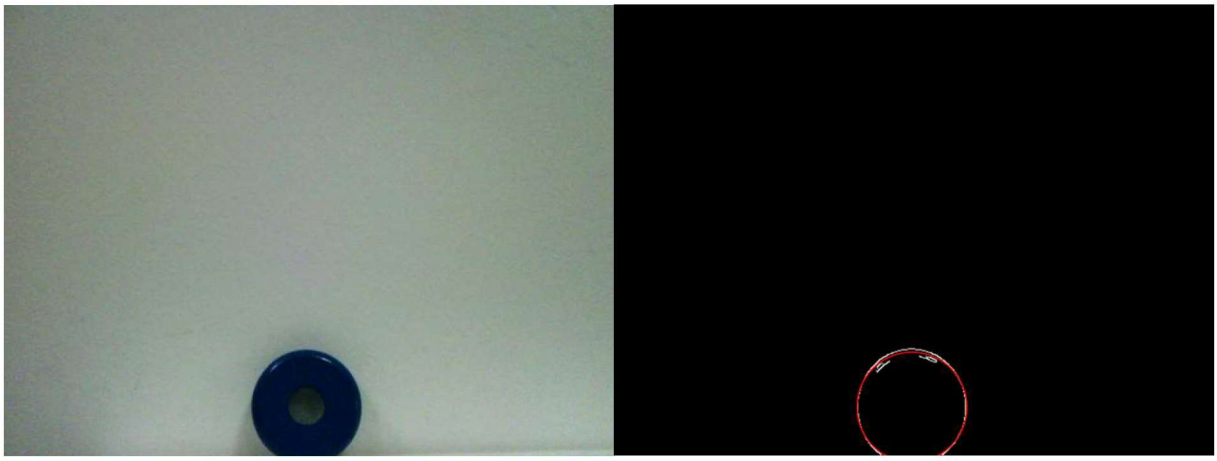
Slika 50: MSE = 5.95 (TRUE za threshold = 50)



Slika 51: MSE = 43.14 (TRUE za threshold = 50)

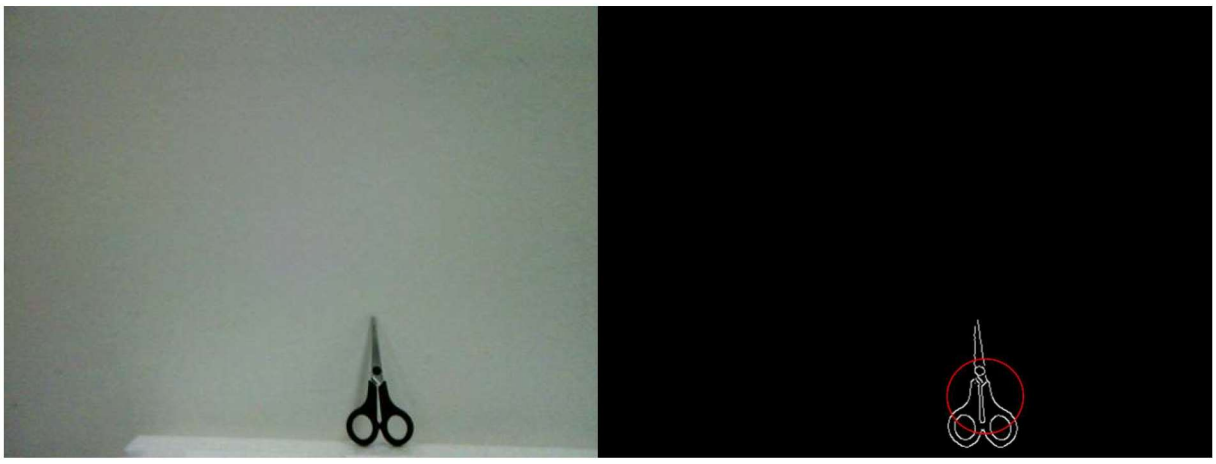


Slika 52: MSE = 57.27 (TRUE za threshold povećan na 60)



Slika 53: MSE = 5.03 (TRUE za threshold = 50)

Slika 54: MSE = 180.06 (FALSE za threshold = 50)



Slika 55: MSE = 302.62 (FALSE za threshold = 50)

Iz priloženih rezultata jasno vidimo da ova metoda puno bolje, za razliku od naivne metode, pronalazi parametre „fitane“ kružnice jer naočigled znatno točnije pokriva rub na „edge-detection“ slici.

Bitno je za napomenuti da bi u ovom slučaju „threshold“ bilo dobro još povećati jer smo na slici 52 imali nešto veću vrijednost „loss funkcije“. Naime, ako je odsjaj svjetla u sredini objekta te ako je objekt istodobno nešto veći, može se dogoditi da i MSE vrijednost bude veća jer će se razlika koju smo objasnili u poglavlju 3.1.1 Klasifikacija računati i za bijele piksele koji su smetnja. Oni će tako, iako je „fitana“ kružnica gotovo savršena, i dalje ponešto povećati MSE vrijednost.

Metoda u potpunosti uklanja nedostatke naivne metode te ima potpunu točnost u klasifikaciji i na zelenoj i na bijeloj pozadini. Rezultate na zelenoj pozadini nismo prikazivali jer bismo time povećavali opseg rada, ali jasno je da će metoda na njoj raditi još i bolje jer su i same „edge-detection“ slike potpunije. Rezultatima na objektima slikanim na zelenoj pozadini može se pristupiti na GitHub linku [8].

4.3 Rezultati metode minimizacije geometrijske udaljenosti

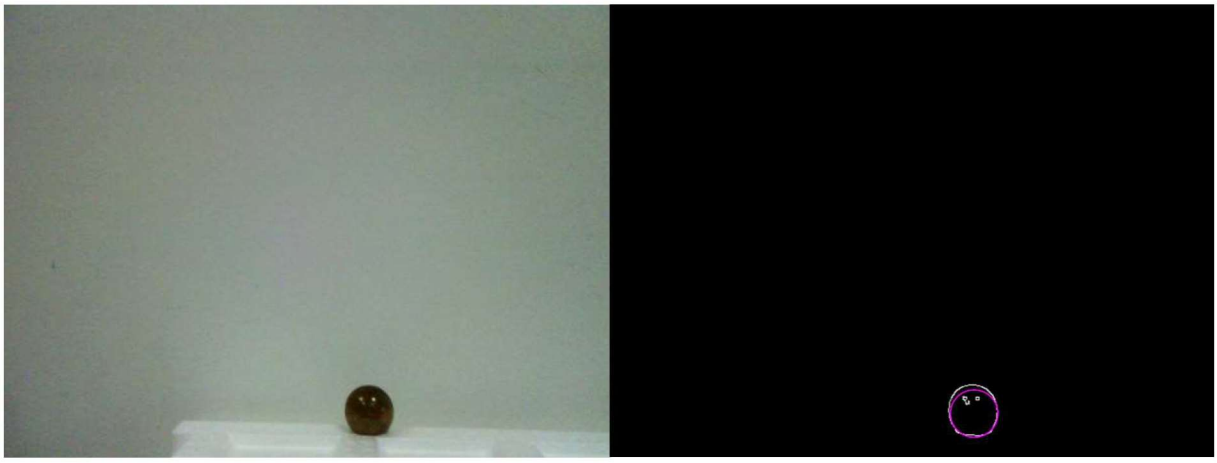
I za ovu ćemo metodu prikazati rezultate za „edge-detection“ slike s kojima je naivna metoda imala poteškoća. Takve su slike sadržavale odsjaje svjetla, sjenu ili su jednostavno dijelovi ruba bili nepotpuni.

Prikazat ćemo rezultate metode minimizacije geometrijske udaljenosti koju smo opisali u poglavlju 3.3.

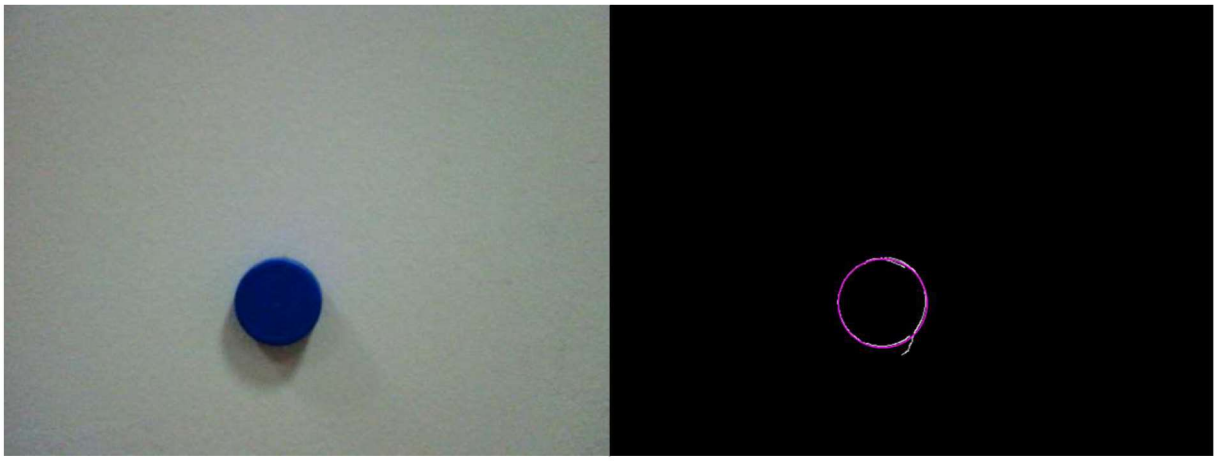
Opet ćemo krenuti sa spornim slikama, a nakon toga ćemo prikazati i klasifikaciju na boljim „edge-detection“ slikama na kojoj niti naivna metoda nije griješila. Na samom kraju usporedit ćemo rezultate minimizacije algebarske i geometrijske udaljenosti.

Napominjemo da nećemo prikazati klasifikaciju za svih tridesetak objekata. Klasifikacija za ostale objekte može se pronaći na GitHub linku [8].

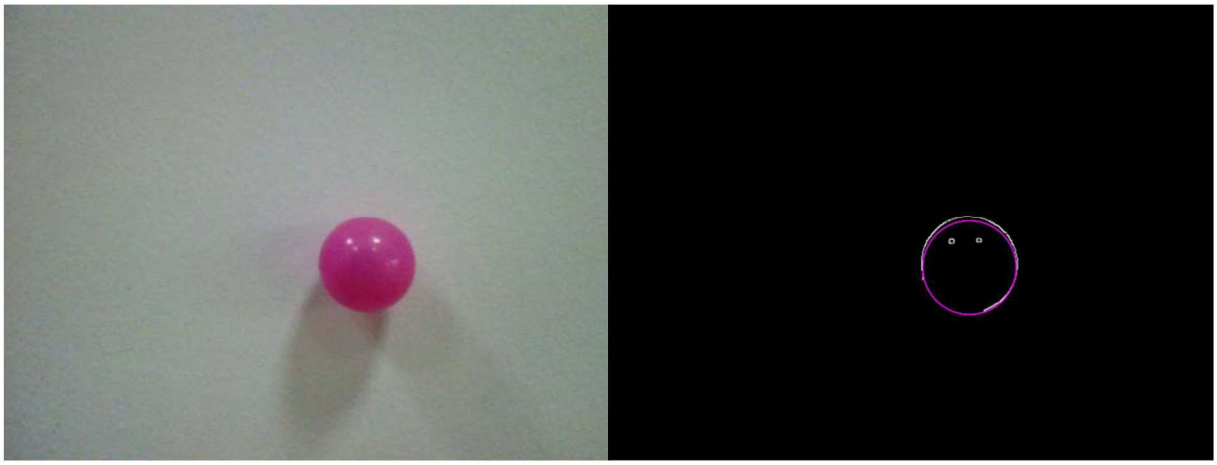
Pokažimo primjere klasifikacije sa smetnjama od odsjaja, sjene ili s nepotpunim rubom:



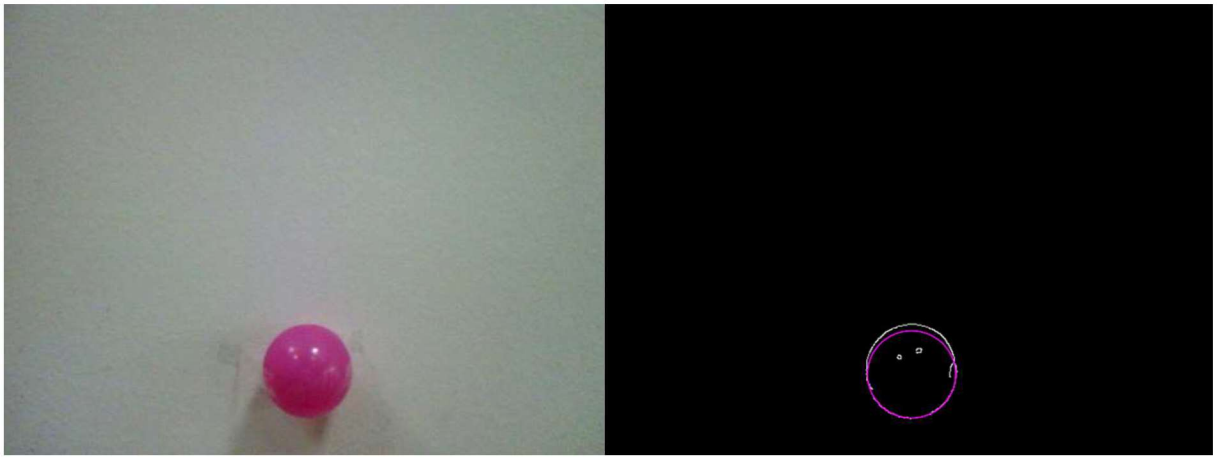
Slika 56: MSE = 21.75 (TRUE za threshold = 50)



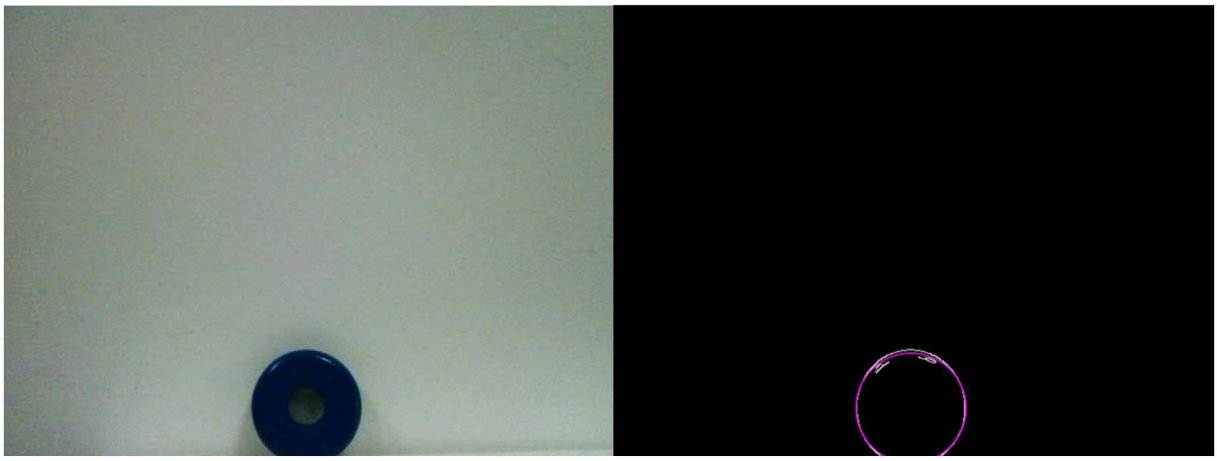
Slika 57: MSE = 5.88 (TRUE za threshold = 50)



Slika 58: MSE = 41.27 (TRUE za threshold = 50)

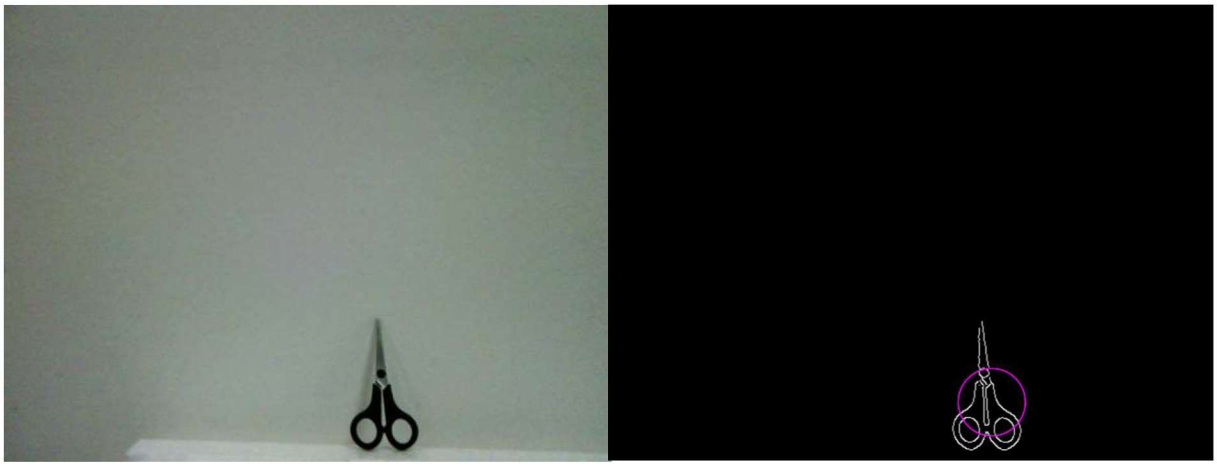


Slika 59: MSE = 54.44 (TRUE za threshold povećan na 60)



Slika 60: MSE = 5.02 (TRUE za threshold = 50)

Slika 61: MSE = 175.93 (FALSE za threshold = 50)



Slika 62: MSE = 280.31 (FALSE za threshold = 50)

Na prikazanim rezultatima vidimo da su parametri i MSE vrijednosti gotovo jednake za metodu minimizacije algebarske i metodu minimizacije geometrijske udaljenosti.

Uspoređivanjem, vidimo naznake da bi, s obzirom da je MSE ponešto ipak manji za ovu metodu, metoda minimizacije geometrijske udaljenosti mogla davati bolje rezultate od metode minimizacije algebarske udaljenosti.

Ova metoda također u potpunosti uklanja nedostatke naivne metode te ima potpunu točnost u klasifikaciji i na zelenoj i na bijeloj pozadini. Rezultate na zelenoj pozadini nismo prikazivali jer bismo time povećavali opseg rada, a jasno je da će metoda na njoj raditi još i bolje jer su i same „edge-detection“ slike potpunije. Rezultatima na objektima slikanim na zelenoj pozadini može se pristupiti na GitHub linku [8].

4.4 Usporedba rezultata svih triju metoda

U ovom ćemo poglavlju, unutar tablice, usporediti MSE vrijednosti i rezultate klasifikacije na bijeloj pozadini za sve tri metode.

Tablica sadrži objekte koje smo detaljno opisali unutar rada, a rezultati za ostale objekte, čije slike nismo prikazali u radu, mogu se generirati pokretanjem algoritma koji se nalazi na GitHub linku [8].

	Naivna metoda		Min. alg. udaljenosti		Min. geo. udaljenosti	
Objekt	**MSE vrijednost**	**Rezultat klasifikacije**	**MSE vrijednost**	**Rezultat klasifikacije**	**MSE vrijednost**	**Rezultat klasifikacije**
Kantica za vodu	181.39	Nije lopta	180.06	Nije lopta	175.93	Nije lopta
Smeđa loptica	40.34	Lopta je	23.56	Lopta je	21.75	Lopta je
Futrola naočale	362.79	Nije lopta	357.66	Nije lopta	340.34	Nije lopta
Limun	18.43	Lopta je	13.26	Lopta je	13.14	Lopta je
Rubikova kocka	96.37	Nije lopta	98.3	Nije lopta	95.76	Nije lopta
Čep boce	7.87	Lopta je	5.95	Lopta je	5.88	Lopta je
Crveni disk	83.41	Nije lopta	49.49	Lopta je	45.93	Lopta je
Ružičasta loptica	148.83	Nije lopta	43.14	Lopta je	41.27	Lopta je
Plavi disk	152.95	Nije lopta	5.03	Lopta je	5.02	Lopta je
Škare	297.97	Nije lopta	302.62	Nije lopta	280.31	Nije lopta
Stalak za selotejp	323.1	Nije lopta	267.25	Nije lopta	258.42	Nije lopta
Jabuka	10.72	Lopta je	5.86	Lopta je	5.85	Lopta je

Tablica 2: Usporedba rezultata sve tri metode

Vidimo kako metoda minimizacije algebarske udaljenosti i metoda minimizacije geometrijske udaljenosti upotpunjuju naivnu metodu te doista ostvaruju bolje rezultate na slikama sa smetnjama od sjene ili odsjaja.

„Threshold“, tj. granična vrijednost MSE vrijednosti za koju ćemo reći da je objekt lopta, ili da nije lopta u slučaju da je MSE vrijednost veća od thresholda, postavljena je na 50 za sve tri metode. Stečenim iskustvom na rezultatima velikog broja ponavljanja, zaključili smo kako je ta vrijednost dovoljna da imamo točnu klasifikaciju za sve objekte iz našeg odabranog skupa za metode minimizacije algebarske i geometrijske udaljenosti.

Iako, za razliku od naivne metode, druge dvije metode u pravilu nisu podložne greškama koje se događaju uslijed smetnja na slici, ipak smo se postavljanjem takve granice osigurali da objekti koji su lopta i dalje budu klasificirani pozitivno ukoliko se dogodi nekakva iznimka.

S obzirom da u metodi minimizacije geometrijske udaljenosti koristimo Gauss-Newtonovu metodu, morali smo postaviti zaustavni kriterij u obliku broja maksimalnih iteracija i tolerancije na promjenu vrijednosti korekcijskog vektora. U našem slučaju, koristili smo 100 za broj maksimalnih iteracija, a toleranciju 10^{-6}. Takvim smo vrijednostima dobili zadovoljavajuće rezultate.

Poglavlje 5

Zaključak

Uspjeli smo povezati se na robota, na nižoj razini programirati njegove funkcije direktnim spajanjem u Python programskom jeziku te sačuvati sliku na kojoj smo uspješno izvršili klasifikaciju.

Zadovoljni smo rezultatima svih triju metoda te smo zapravo nadmašili očekivanja s obzirom da na našim objektima robot i metode ne griješe. Također, trajanje cjelokupnog procesa pohrane i obrade slike te klasifikacije je iznimno brzo jer robot momentalno izgovara odluku o klasifikaciji. Naša naivna metoda radi bez greške na slikama objekata uslikanim na zelenoj pozadini, dok ostale dvije metode upotpunjavaju njezine nedostatke te bez greške klasificiraju slike objekata na bijeloj pozadini koje sadrže smetnje od odsjaja svjetla, sjene ili nedostatka detektiranog ruba. Time su naši ciljevi u potpunosti ostvareni.

Važno je za naglasiti da ne tvrdimo da metode neće griješiti te da zasigurno postoje objekti koji bi na neki način naštetili određivanju ruba, a time i konačnoj klasifikaciji.

Iako se u ovome radu nismo zanimali sa „edge-detection“ slikama nastalim od objekata s više nijansi boja i tekstura te također nismo radili klasifikaciju objekata u prostoru gdje bi ostali objekti mogli smetati, smatramo da će nam takvi problemi klasifikacije otvoriti mogućnosti za dodatno proučavanje metoda koje bismo detaljno analizirali i opisali u budućim radovima.

Literatura

[1] http://doc.aldebaran.com/1-14/software/choregraphe/choregraphe_overview.html, 1.8.2024.

[2] https://www.python.org/about/, 1.8.2024.

[3] http://doc.aldebaran.com/2-1/home_nao.html, 1.8.2024.

[4] https://www.softbankrobotics.com/, 1.8.2024.

[5] https://www.aldebaran.com/en/support/nao-6/downloads-softwares, 1.8.2024.

[6] http://doc.aldebaran.com/1-14/software/choregraphe/index.html, 1.8.2024.

[7] https://www.python.org/downloads/, 1.8.2024.

[8] https://github.com/DMV1204/ZavrsniRad

[9] https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123, 1.8.2024.

[10] Walter Gander, Gene H. Golub, Rolf Strebel, *Least-Squares Fitting of Circles and Ellipses*, Odjel za informatiku, Sveučilište Stanford, Kalifornija, 1994.

[11] Ninoslav Truhar, *Numerička linearna algebra*, Odjel za matematiku, Sveučilište J. J. Strossmayera u Osijeku, 2010.

[12] Rudolf Scitovski, *Numerička matematika*, Odjel za matematiku, Sveučilište J. J. Strossmayera u Osijeku, 2004.