

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

# **Implementacija Harrisovog detektora kuteva**

## **Tehnička dokumentacija**

### **Verzija 1.0**

**Studentski tim:** Benjamin Horvat  
Dino Kovač  
Mak Krnic  
Nikola Munđer  
Dino Pačandi  
Ivan Weber

**Nastavnik:** Prof. dr.sc. Siniša Šegvić

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## Sadržaj

1. Uvod.....	3
2. Prototipiranje u Pythonu.....	4
2.1 Implementacija Harrisovog detektora kuteva u Pythonu.....	4
2.2 Grafički prikaz gradijenata.....	10
2.3 Primjeri.....	14
2.4 Zaključak.....	19
3. Zaglađivanje slike.....	20
3.1 Zaglađivanje Gaussovim algoritmom.....	20
3.2 Implementacija Gaussovog algoritma.....	20
4. Harrisov odziv.....	22
4.1 Pronalaženje važnih točaka računanjem Harrisovog odziva.....	22
4.2 Računanje Harrisovog odziva.....	22
4.3 Programska implementacija računanja Harrisovog odziva.....	23
5. Odbacivanje nepotrebnih točaka.....	24
6. Rezultati.....	26
7. Zaključak.....	30
8. Upute za korištenje.....	31
8.1 Prototip u Pythonu.....	31
8.2 Implementacija u C++-u.....	32
9. Literatura.....	33

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 1. Uvod

Računalni vid je područje umjetne inteligencije koje se bavi prepoznavanjem dvodimenzionalnih i trodimenzionalnih predmeta. Dobivanjem, procesuiranjem i analiziranjem slika iz stvarnog svijeta dobivamo numeričke vrijednosti pomoću kojih računamo bitne značajke koje su nam potrebne iz neke scene. Koncept značajki se koristi kako bi se označio dio informacije koji je važan za rješavanje računalnog zadatka određene aplikacije.

Izričito, značajke se odnose na:

- rezultat opće operacije susjedstva točke (detektor značajki) primijenjen na sliku
- određene strukture u slici, npr. jednostavne strukture poput točke ili ruba, pa sve do kompleksnijih struktura poput predmeta

Ostali oblici značajki se odnose na pokret u nizu slika, struktura definiranih krivuljama ili granicama između različitih područja u slici, te njihovih svojstava.

Koncept značajki je općenit i izbor značajki u određenom sustavu računalnog vida ovisi o specifičnom problemu.

U našem izlaganju i problemu bavimo se značajkama koje mogu biti:

- rub
- kut

Nas konkretno zanima kut koji ćemo detektirati Harrisovim algoritmom za detekciju kuteva. Harris i Stephens su usavršili Moravecov detektor kuteva uzevši u obzir diferencijal rezultata kuta s obzirom na smjer, umjesto da su koristili pomicanje zakrpe. Taj rezultat kuta zapravo je razlika kvadrata. Ideja Harrisovog algoritma je lociranje točaka interesa u čijem susjedstvu postoje rubovi koji imaju gradijent u barem dva različita smjera, čime je kut detektiran.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 2. Prototipiranje u Pythonu

Python je skriptni jezik, pogodan za prototipiranje i brz razvoj modela zahvaljujući svojoj jednostavnoj sintaksi.

Kako pripada interpretiranim jezicima, brzina mu nije jedna od glavnih odlika, ali zbog izuzetno jednostavne i lako čitljive sintakse, sam programski kod izgleda kao pseudokod te ga to čini izuzetno povoljnim za prototipiranje.

Pythonova standardna biblioteka sadrži bogati skup funkcija te uz dodatak biblioteka otvorenog koda *numpy*, *scipy*, *PIL* i *Matplotlib* omogućava široki skup operacija na slikama odnosno njihovu obradu.

### 2.1 Implementacija Harrisovog detektora kuteva u Pythonu

Prije samog izračuna Harrisovog odziva potrebno je sliku pretvoriti u *grayscale* format kako bismo olakšali samo računanje. Nakon toga potrebno je izračunati odziv Harrisove funkcije za svaki piksel (točku) na slici.

Prvo je potrebno izračunati elemente matrice. To radimo u sljedećih par koraka:

```
filters.gaussian_filter(Slika, (sigma, sigma), (0, 1), Ix)
filters.gaussian_filter(Slika, (sigma, sigma), (1, 0), Iy)
```

$I_x$  i  $I_y$  su derivati slike u smjeru  $x$ -osi, odnosno  $y$ -osi. Odmah se primjenjuje i Gaussov filter kako bi se postiglo zaglađivanje slike i eliminirali nepotrebni rubovi.

Nakon toga računaju se sami elementi matrice:

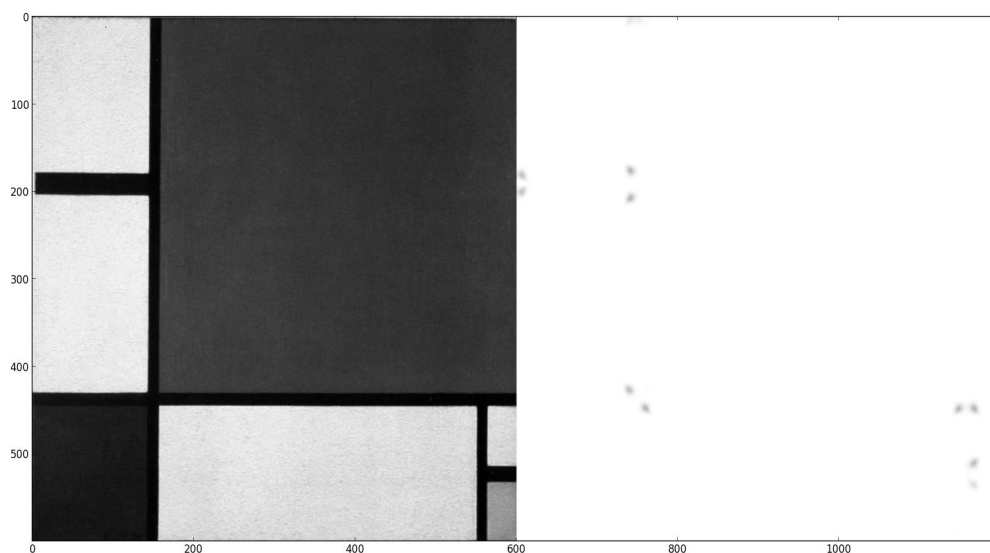
```
Ixx = filters.gaussian_filter(Ix*Ix, sigma)
Iyy = filters.gaussian_filter(Iy*Iy, sigma)
Ixy = filters.gaussian_filter(Ix*Iy, sigma)
```

Ponovno se koristi Gaussov kernel, ali ovaj puta kao težinska funkcija (prozor) unutar kojeg se uspoređuju pikseli. Nakon toga računa se sam odziv Harrisove funkcije. Kako bi izbjegli računanje svojstvenih

vrijednosti za svaki piksel na slici koristimo aproksimaciju  $\frac{\text{Determinanta}}{\text{Trag}}$  ( time također i izbjegavamo korištenje konstante  $k$  u izračunu koja se određuje empirijski ).

Nakon tih izračuna dobivamo *sliku* koja sadrži vrijednost Harrisove funkcije za svaki piksel ulazne slike za koju smo i računali odziv.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

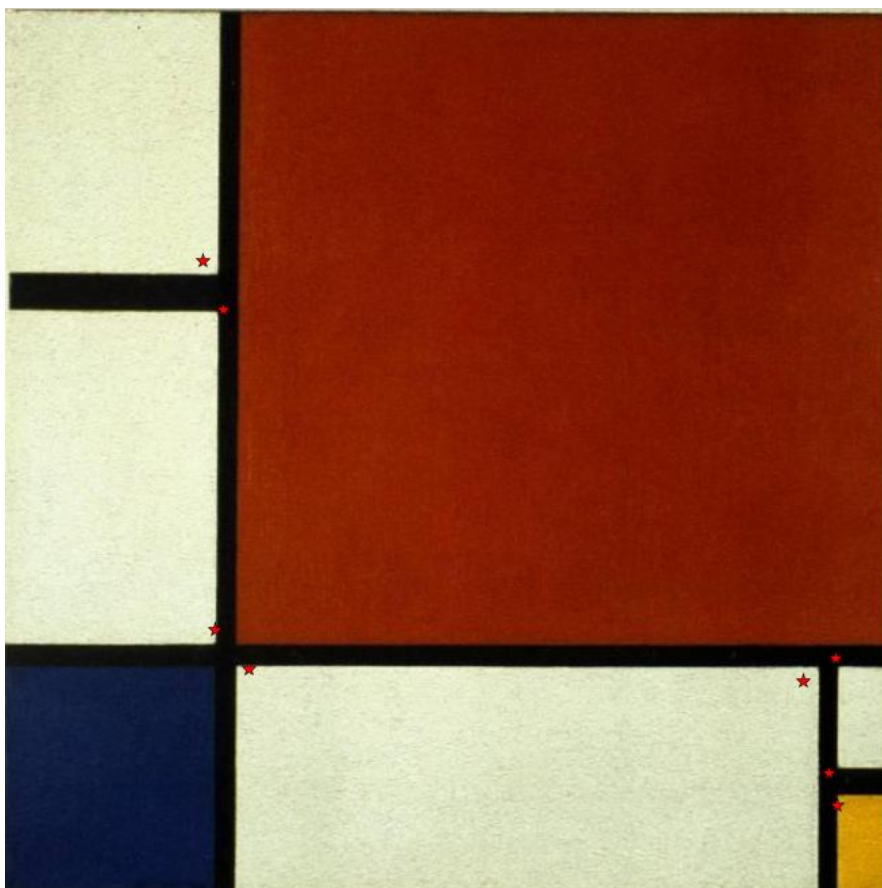


*Slika 1. Odziv Harrisove funkcije*

Slika 1 prikazuje odziv Harrisove funkcije. Na lijevoj strani nalazi se ulazna slika, a desno odziv. Primjećuje se zatamnjenja koje se nalaze na mjestima kuteva kod originalne slike.

Sljedeći korak je odrediti kuteve na temelju dobivenog odziva. Sve vrijednosti odziva koje su iznad neke granice proglasimo mogućim kutevima, a ostale odbacimo. Nakon toga gledamo vrijednosti odziva svih mogućih kuteva, počevši od najveće. Ukoliko u nekom prozoru oko tog kuta ne postoji niti jedan drugi kut, on se proglašava novim kutem te se zabranjuje pojava drugih kuteva unutar njegovog prozora. Na taj način sprječava se višestruka detekcija istog kuta.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 2. Detektirani kutevi*

Slika 2 prikazuje sliku s označenim detektiranim kutevima. Dolazi do višestruke detekcije jer je korištena premala dozvoljena udaljenost dvaju kuteva (pre mali prozor).

## UPARIVANJE SLIKA U PYTHONU

Ukoliko želimo pronaći odgovarajuće točke na dvije slike koristeći Harrisove kuteve, oni sami po sebi neće biti dovoljni. Potrebno je dodati neki opisnik koji će sadržavati informacije o okolnim pikselima tog kuta.

Za svaki izračunati kut uzimaju se svi pikseli unutar prozora neke određene veličine. Veličine prozora moraju biti jednake za obje slike. Zatim se opisnik svakog pojedinog kuta jedne slike uspoređuje s opisnikom svakog pojedinog kuta druge slike. Za uspoređivanje se koristi normalizirana kros-korelacija, metoda koja je robusna na promjene razine osvjetljenja.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

$$ncc(Opisnik1, Opisnik2) = \sum_x \frac{Opisnik1(x) - \mu_1}{\sigma_1} * \frac{Opisnik2(x) - \mu_2}{\sigma_2} \quad (1)$$

$n$  označava broj piksela u prozoru,  $\mu$  označava srednju vrijednost piksela unutar prozora, a  $\sigma$  je standardna devijacija unutar tog prozora. Stupanj podudaranja skalira se izlaznom vrijednošću funkcije. Čim je vrijednost veća i sama razina podudaranja je veća. Uzimamo da se kutevi podudaraju ako je vrijednost funkcije(1) veća od neke određene vrijednosti.

Kako je moguće da jedan kut prve slike u usporedbi s više kuteva druge slike poprimi vrijednost funkcije(1) veću od granične, uzima se samo najveća vrijednost. To ilustrira sljedeći odsječak koda:

```
d = -ones((len(desc1), len(desc2)))
for i in range(len(desc1)):
    for j in range(len(desc2)):
        d1 = (desc1[i] - mean(desc1[i])) / std(desc1[i])
        d2 = (desc2[j] - mean(desc2[j])) / std(desc2[j])
        ncc_value = sum(d1 * d2) / (n-1)
        if ncc_value > threshold:
            d[i,j] = ncc_value

ndx = argsort(-d)
matchscores = ndx[:,0]
```

Pomoćna matrica čiji su elementi isprva svi jednaki -1 popunjava se vrijednostima funkcije(1), ukoliko je ona veća od neke granične vrijednosti. Nakon iteracije po svim mogućim kombinacijama opisnika kuteva prve i druge slike, matrica se množi s -1 te se svaki redak uzlazno sortira tako da indeks stupca najmanjeg elementa dođe na prvo mjesto. Pošto smo matricu pomnožili s -1 najveća vrijednost postala je najmanja i nakon sortiranja nalazi se na prvom mjestu.

Kao dodatna provjera može se provesti provjera simetričnosti uparivanja tako da se prvo provjeri podudaranje kuteva prve slike s kutevima druge slike, a zatim podudaranje kuteva druge slike s kutevima prve slike. Ukoliko je u oba slučaja rezultat jednak, veća je vjerojatnost stvarnog podudaranja.

```
matches_12 = match(desc1, desc2, threshold)
matches_21 = match(desc2, desc1, threshold)
```

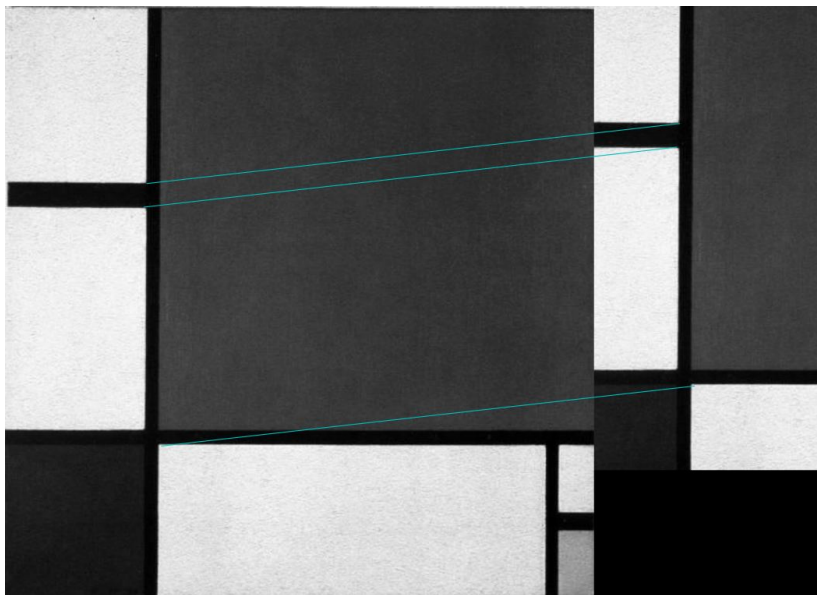
Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

```

pom_12 = where(matches_12 >= 0) [0]
for n in pom_12:
    if matches_21[matches_12[n]] != n:
        matches_12[n] = -1

```

Ukoliko se indeksi ne poklapaju, taj pridruženi par kuteva se odbacuje.



*Slika 3. Uparivanje slika*

Iz originalne slike izrezan je dio te je provedeno uparivanje izrezanog djela s originalom [Slika 1]. Sva tri kuta uspješno su uparena. U ovom slučaju nije došlo do promjene rezolucije slike ili rotacije što je i razlog uspješnom uparivanju.



Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 4. Pokušaj uparivanja dvije slike*

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

Na slici 4 prikazan je pokušaj uparivanja dvije slike. Iz originalne slike izrezan je neboder te je izvršeno uparivanje korištenjem Harrisovih kuteva i normalizirane kros-kovarijacije. Dobiveno je više pogrešnih rezultata nego ispravnih. Razlog tome je sama normalizirana kros-kovarijacija koja ne daje najbolju moguću informaciju o podudaranju te velika osjetljivost same metode na rotaciju slike i promjenu u rezoluciji. Također, za postizanje boljih rezultata potrebno je koristiti mali  $\sigma$  ( $\sim 1$ ) što ovdje nije bio slučaj.

## 2.2 Grafički prikaz gradijenata

Jedan od zadataka ovog projekta bio je grafički prikaz gradijenata slike, odnosno kreiranje funkcije koja bi omogućila prikaz promjene intenziteta slike na točno određenom području, tj. 'prozoru', čije se dimenzije i položaj mogu specificirati po volji. Uz to, dobiveni gradijenti bi se grafički aproksimirali elipsom, dajući općeniti osjećaj 'pružanja' promjene intenziteta na određenom prozoru. Određivanje potrebnih podataka za konstruiranje takve elipse se uglavnom baziralo na svojstvenim vrijednostima i svojstvenim vektorima, kao što ćemo pokazati u nastavku.

Za početak, uzimamo sliku (koja je prethodna bila pretvorena u grayscale format) te nad njom primjenjujemo Gaussove filtre, čime se slika zamućuje, smanjujući broj informacija koje treba obraditi. Iz takve slike se vade koordinate točaka po X i Y osi.

```

Ix = zeros(im.shape)
filters.gaussian_filter(im, (sigma,sigma), (0,1), Ix)
Iy = zeros(im.shape)
filters.gaussian_filter(im, (sigma,sigma), (1,0), Iy)

```

Program prikazuje sliku, nakon čega se klikom miša specificira točka koja će služiti kao središte prozora iz kojeg se uzima uzorak gradijenta.

Sada kada imamo potrebne gradijente, prelazimo na drugi dio naše funkcije, crtanje elipse koja bi aproksimirala naš skup gradijenata. Za crtanje jedne elipse trebaju nam sljedeći podaci: koordinate centra, duljina velike poluosi, duljina male poluosi te smjer pružanja elipse.

Centar takve elipse će imati X i Y koordinate koje odgovaraju prosjeku svih X i Y koordinata dobivenih gradijenata. Dobivamo ih pomoću sljedeće dvije naredbe:

```

xS=sum(IxF)/len(IxF)
yS=sum(IyF)/len(IyF)

```

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

Prelazimo na računanje duljina poluosi, za što će nam biti potrebne svojstvene vrijednosti, za računanje kojih će nam pak trebati kovarijacijska matrica. Za početak, centriramo X i Y koordinate gradijenata oko ishodišta te ih spajamo u jedno n\*2 polje, gdje je n broj gradijenata.

```
for i in range(0,len(IxF)):
    IxF1.append(IxF[i]-xS)
    IyF1.append(IyF[i]-yS)
XY = [ ]
XY.append(IxF1)
XY.append(IyF1)
```

Kovarijacijska matrica prikazuje ovisnost jedne varijable o drugoj, u našem slučaju, ovisnost X o X, Y o Y te X o Y. Za dvije varijable ta matrica je oblika:

$$S = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} \quad (2)$$

Gdje  $\sigma_{11}$  predstavlja korelaciju X sa X,  $\sigma_{22}$  Y sa Y, a  $\sigma_{12}$  i  $\sigma_{21}$  su jednakog iznosa i oba predstavljaju ovisnost X o Y i obrnuto. Da bi dobili matricu koristimo sljedeću formulu:

$$S = \frac{1}{n-1} * XY * XY^T \quad (3)$$

Gdje je n broj gradijenata, XY dvodimenzionalna matrica koja sadrži sve X i Y koordinate gradijenata, a  $XY^T$  njena transponirana verzija. Prikazana formula za S je implementirana sljedećom linijom koda:

```
S=(1./(len(IxF)-1))*dot(XY,XYT)
```

Nakon dobivanja kovarijacijske matrice S, možemo početi računati svojstvene vrijednosti. Za dvodimenzionalnu matricu, odnosno matricu za slučaj s dvije varijable, imamo dvije svojstvene vrijednosti dobivene sljedećom formulom:

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

$$\lambda_{1,2} = \frac{O_{11} + O_{22} \pm \sqrt{(O_{11} - O_{22})^2 + 4 O_{12}^2}}{2} \quad (4)$$

Što u Pythonu ostvarujemo s ove dvije linije koda:

```
eva1=0.5*(S[0,0]+S[1,1]+sqrt(pow((S[0,0]-S[1,1]),2)+4*pow(S[0,1],2)))
eva2=0.5*(S[0,0]+S[1,1]-sqrt(pow((S[0,0]-S[1,1]),2)+4*pow(S[0,1],2)))
```

Dobivene svojstvene vrijednosti određuju dimenzije elipse koju moramo konstruirati, duljinu njezine malene i velike poluosi.

Nakon toga, računamo elemente svojstvenih vektora.

$$v_1 = \frac{O_{12}}{\sqrt{(\lambda_1 - O_{11})^2 - O_{12}^2}} \quad (5)$$

$$v_2 = \frac{\lambda_1 - O_{11}}{\sqrt{(\lambda_1 - O_{11})^2 - O_{12}^2}} \quad (6)$$

Za koje odgovarajući kod glasi:

```
eve1=S[0,1]/sqrt(pow((eva1-S[0,0]),2)+pow(S[0,1],2))
eve2=(eva1-S[0,0])/sqrt(pow((eva1-S[0,0]),2)+pow(S[0,1],2))
```

Vektori određuju smjer promjena, smjer pružanja gradijenata, a iz njih se može dobiti kut koji elipsa zatvara s X osi, što nam je potrebna za crtanje iste. Iako bi iz dobivenih vrijednosti mogli dobiti svojstvene vektore  $V_1$  i  $V_2$ , gdje je:

$$V_1 = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad V_2 = \begin{bmatrix} v_2 \\ -v_1 \end{bmatrix} \quad (7)$$

Isti nam neće trebati, jer je kut koji dobijemo provođenjem sljedećih naredbi dovoljan za određivanja smjera elipse:

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

```
tang=eve2/eve1
kut=arctan(tang)*180./pi
```

Zapravo, on je skalarna reprezentacija prvog vektora izražena u radijanima, a budući da naredba koju koristimo za crtanje elipse podrazumijeva dani kut kao smjer pružanja velike poluosi, dok prvi svojstveni vektor određuje pružanje prve svojstvene vrijednosti, koja određuje veliku poluos, to je sve što trebamo (za manju poluos se podrazumijeva da je okomita na veliku).

Valja napomenuti da se u nekim vrlo rijetkim (ali i dalje mogućim) slučajevima može dogoditi da nazivnik iznosi 0, što bi uzrokovalo prekid rada programa. Takve greške smo dobili isključivo u slučajevima u kojima je jedna varijabla konstantna, odnosno ima za sve gradijente istu vrijednost. Ukoliko se radi o X varijabli, postavljamo  $v_1$  na 0,  $v_2$  na 1, kut na 90 stupnjeva, ukoliko se radi o Y varijabli,  $v_1$  na 1,  $v_2$  na 0, a kut na 0 stupnjeva.

Uz koordinate središta te kuta koji velika poluos zatvara s X osi, jedino nam ostaje odrediti duljine poluosi. U pravilu, formule za duljinu poluosi su:

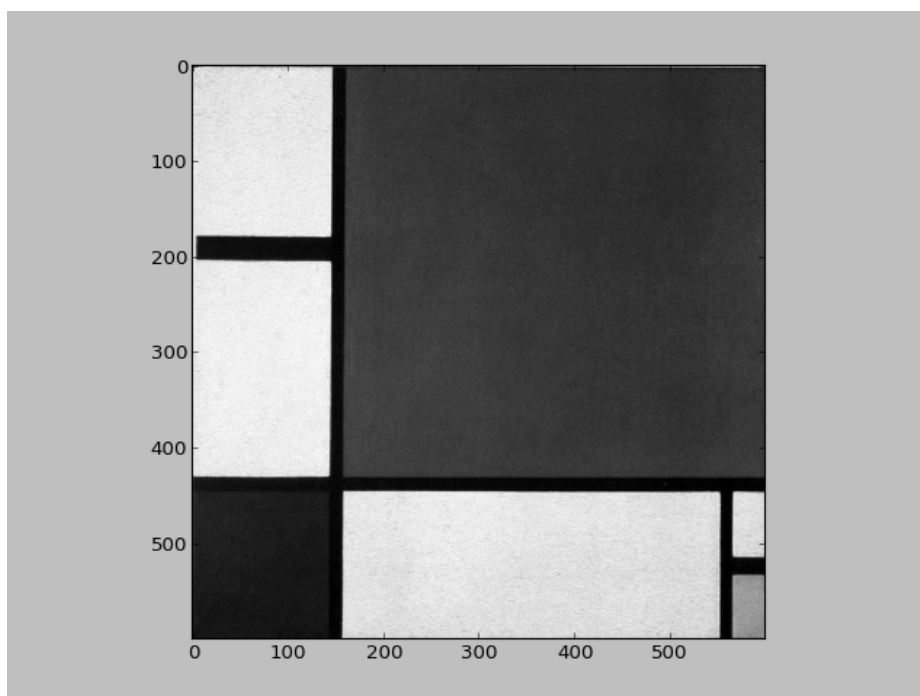
$$R_{1,2} = 2\sqrt{\lambda_{1,2}} \quad (8)$$

Međutim, ova formula je prikladna za aproksimiranje skupova s uniformnom raspodjelom. To kod nas nije slučaj te je korištena formula malo preinačena:

$$R_{1,2} = 4\sqrt{\lambda_{1,2}} \quad (9)$$

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 2.3 Primjeri

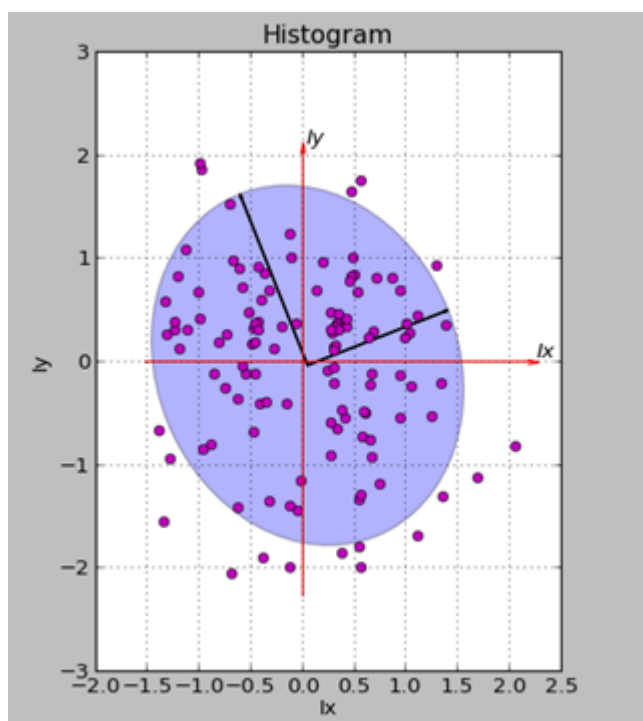


*Slika 5. Testna slika*

Slika 5 prikazuje sliku koju ćemo koristiti za testiranje crtanja histograma gradijenata i pripadne elipse. Možemo kliknuti po želji na bilo koji dio slike kako bi pozicionirali prozor.

Za točku s koordinatama (400,200), dobivamo sljedeći histogram:

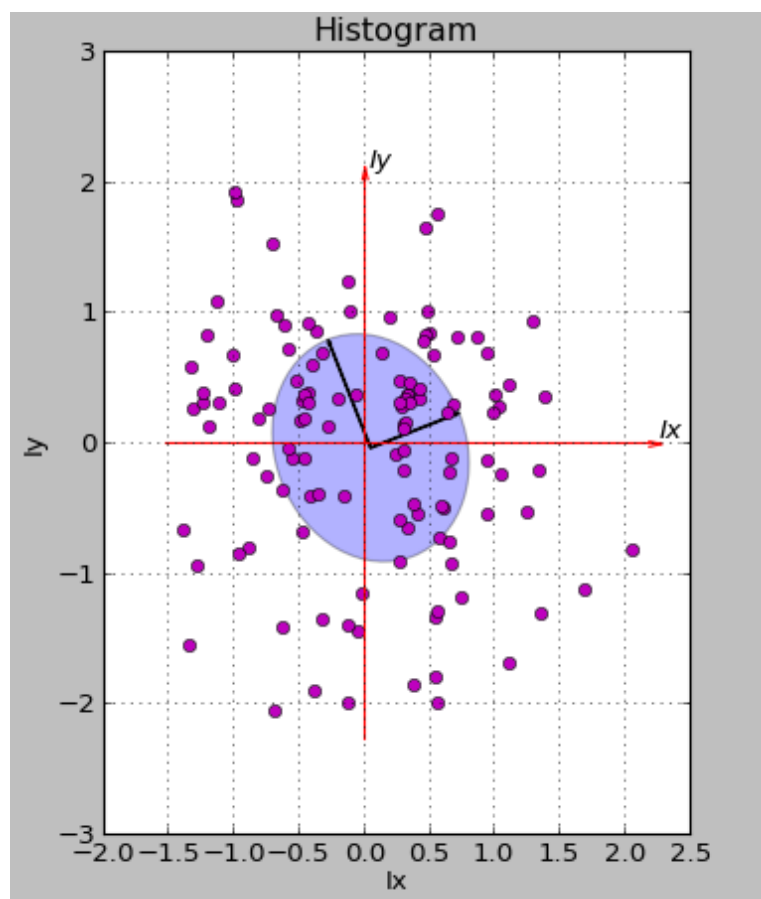
Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 6. Histogram svih gradijenata*

Slika 6 prikazuje dobiveni histogram na kojem su prikazani svi gradijenti, elipsa koja aproksimira to polje gradijenata te svojstveni vektori, odnosno poluosi elipse.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



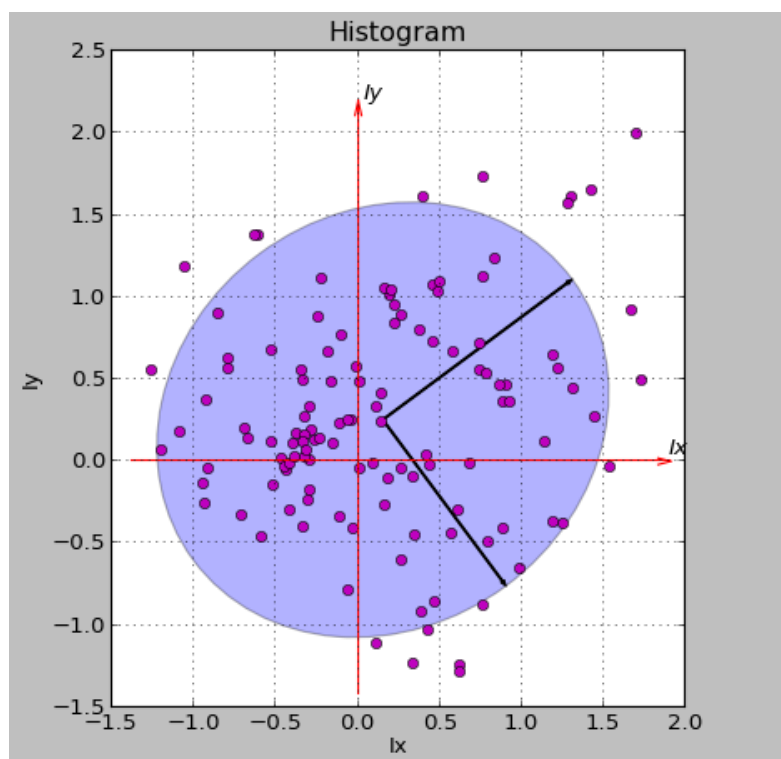
Slika 7. Histogram aproksimacije gradijenata

Slika 7 prikazuje aproksimaciju istog prozora, ali uz korištenje formule nepromijenjene formule za poluosi. Očito pokriva manji broj točaka.

Na kraju, naš kod treba ispitati za tri glavna slučaja, slučaj u kojem obrađujemo rub, kut i ravninu.



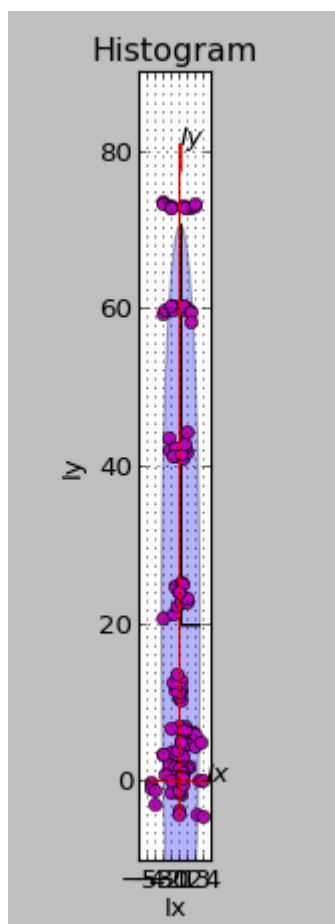
Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 8. Histogram ravnine*

Slika 8 prikazuje histogram dobiven za prozor pozicioniran na ravninu. Rezultat je okrugla elipsa malenih dimenzija.

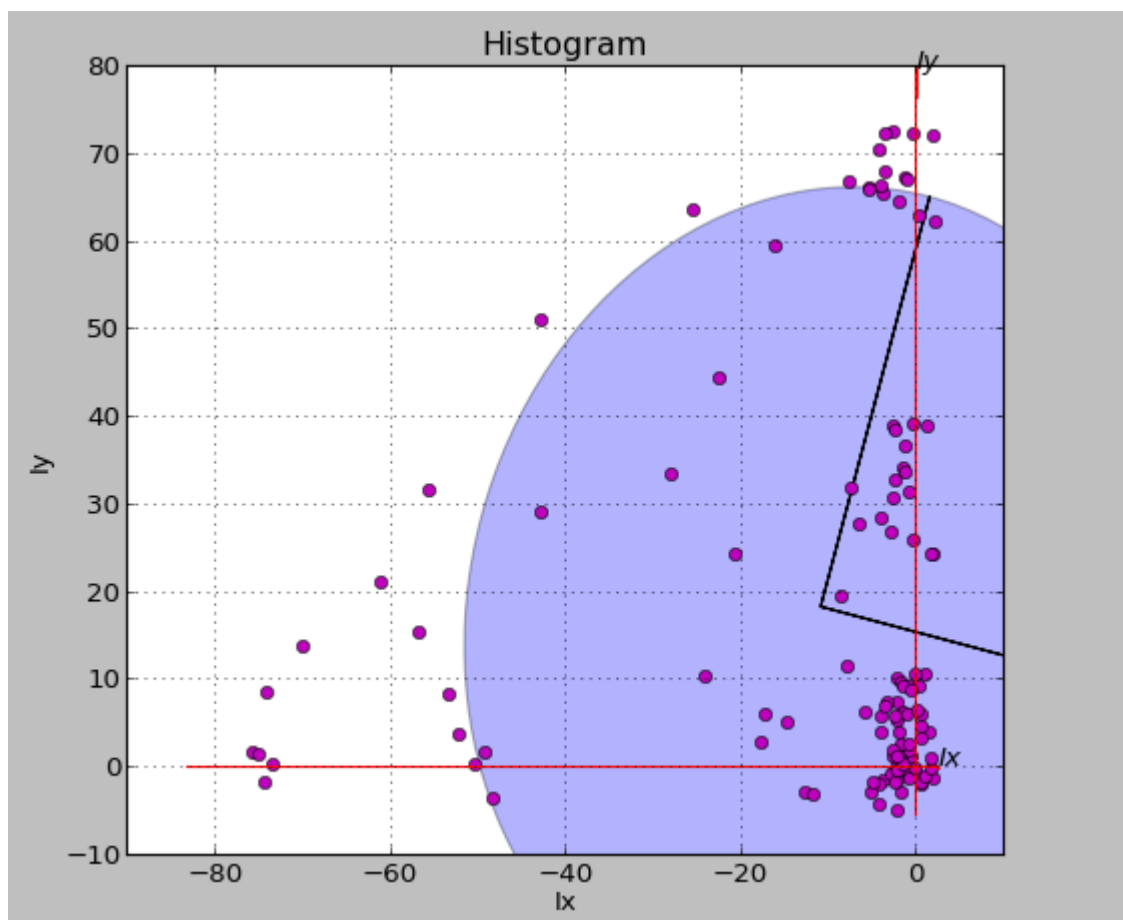
Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 9. Histogram ruba*

Slika 9 prikazuje histogram dobiven za prozor pozicioniran na horizontalni rub. Rezultat je izduljena elipsa, s velikom poluosi puno duljom od male, a budući da se u ovom slučaju radi o horizontalnom rubu, njeno pružanje je zamalo paralelno s Y osi.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



Slika 10. Histogram kuta

Slika 10 prikazuje histogram dobiven za prozor pozicioniran na kut. Baš kao i kod ravnine, rezultat je okrugla elipsa, samo ovaj put s puno duljim poluosima. Nažalost, nismo uspjeli riješiti problem centriranja grafa za ovaj slučaj.

## 2.4 Zaključak

Dobivene elipse dobro aproksimiraju iscertane gradijente pa bi mogli zaključiti da naš kod radi dobro. Iako ima nekih nedostataka, kao što je problem centriranja slike prikazan u zadnjem primjeru, kod dobro radi za veliku većinu primjera te koristi formule umjesto pozivanja predefiniranih funkcija, čineći ga donekle pogodnijim kao izvor za učenje.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

### 3. Zaglađivanje slike

#### 3.1 Zaglađivanje Gaussovim algoritmom

Budući da fotografije snimljene fotoaparatom ili kamerama često imaju smetnje, kako bi se minimizirao učinak tih smetnji na konačni rezultat analize slike i detekcije kutova potrebno ih je ukloniti ili barem umanjiti. Kao jedan od jednostavnih, a učinkovitih algoritama za zaglađivanje slike, u ovoj se implementaciji koristi Gaussov filtar. To znači da se slika zaglađuje prema funkciji (10).

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{(x^2+y^2)}{2\sigma^2}}, \quad (10)$$

gdje su  $x$  i  $y$  koordinate točke, a  $\sigma$  standardna devijacija.

#### 3.2 Implementacija Gaussovog algoritma

Gaussov filtar je, matematički, konvolucija slike i Gaussove funkcije. Obzirom na to da radimo s određenim i relativno malim brojem točaka, sve vrijednosti matrice (*kernela*) mogu se unaprijed izračunati te se tako ostvaruje bitna ušteda procesorskog vremena. Primjer takve matrice, veličine 5x5 dan je u nastavku:

$$G = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 4 & 8 & 4 & 1 \\ 2 & 8 & 16 & 8 & 2 \\ 1 & 4 & 8 & 4 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix} \quad (11)$$

Programski, ta je konvolucija implementirana u dva prolaza (svaki za jednu dimenziju) kako bi se dobio što brži rezultat. Za svaki se stupac svakog retka (odnosno, za svaku točku u matrici) prvo izračuna suma umnožaka horizontalnog okruženja točke s *kernelom* za horizontalno zaglađivanje, a zatim se isti postupak ponavlja za vertikalno zaglađivanje. Pri tome je bitno napomenuti da se radi na kopijama iste matrice (slike), kako ne bi nastale smetnje radi toga što su prethodne točke već "zamagljene", a naredne nisu.

Za lakšu ilustraciju slijedi pseudokod:

```

za svaki redak matrice originalna_matrica {
    za svaki stupac matrice originalna_matrica {
        vrijednost_točke = 0
        za svaki element el 1D matrice kernel_horiz {
            vrijednost_točke = el * originalna_matrica[trenutni_redak,
trenutni_stupac + pozicija_u_kernelu - radijus_zaglađenja]
        }
    }
}

```

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

```

        privremena_matrica[trenutni_redak, trenutni_stupac] = vrijednost_točke
    }
}

za svaki redak matrice originalna_matrica {
    za svaki stupac matrice originalna_matrica {
        vrijednost_točke = 0
        za svaki element el 1D matrice kernel_vert {
            vrijednost_točke = el * originalna_matrica[trenutni_redak +
pozicija_u_kernelu - radijus_zaglađenja, trenutni_stupac]
        }
        ciljna_matrica[trenutni_redak, trenutni_stupac] = vrijednost_točke
    }
}

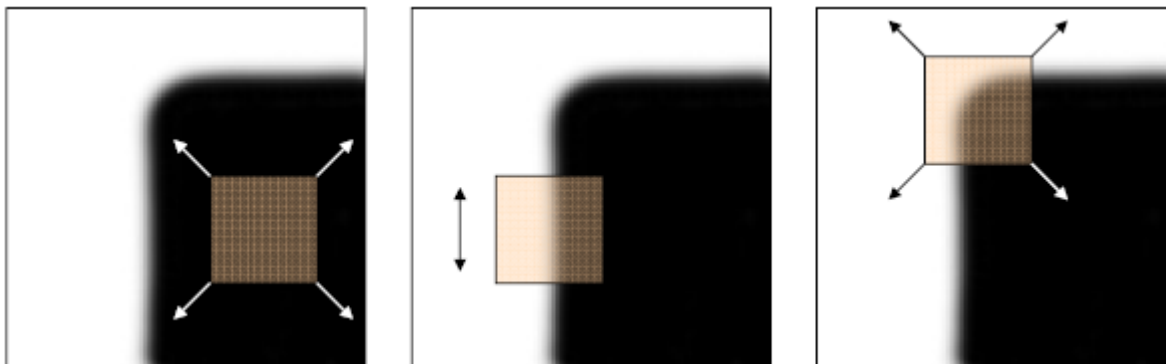
```

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 4. Harrisov odziv

### 4.1 Pronalaženje važnih točaka računanjem Harrisovog odziva

Prilikom obrade slika najvažnije nam je pronaći karakteristične točke koje će nam pomoći u analizi. Za nas su najznačajnije točke na slici kutevi, one jednoznačno određuju poziciju objekata na fotografijama. To nam je važno kako bismo mogli raspoznavati objekte, promatrati njihovu sličnost i pomake pri usporedbi više fotografija. Harrisov odziv je funkcija kojom pomicanjem okvira matematičkim postupkom izražavamo promjene osvjjetljenja za okruženje svake točke. Ukoliko ne dolazi do značajne promjene osvjjetljenja kad taj okvir pomičemo oko neke pozicije, ta točka nam nije zanimljiva. Ako do veće promjene dolazi samo kad okvir pomičemo u jednom smjeru, pronašli smo rub. Najbitnije je pronaći one pozicije kod kojih prilikom micanja okvira uočavamo velike promjene u svim smjerovima.



Slika 11. Micanje okvira prilikom računanja Harrisovog odziva

### 4.2 Računanje Harrisovog odziva

Matematička formula za izračun Harrisovog odziva je sljedeća:

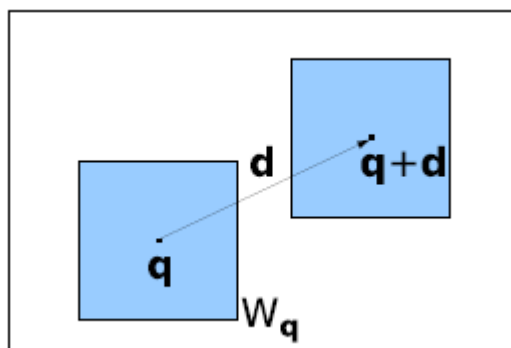
$$E_q(d) = \sum_{x, y \in W(q)} [I(x+d_x, y+d_y) - I(x, y)]^2 \quad (12)$$

Pritom je  $E_q(d)$  različitost susjedstva točke  $q$ , u ovisnosti o pomaku  $d$ .

$W(q)$  je okruženje oko točke  $q$ , odnosno okvir.

$I(x+d_x, y+d_y)$  je intenzitet na poziciji  $(x, y)$  s pomakom  $d$ .

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



Slika 12. Prikaz formule za izračun Harrisovog odziva

### 4.3 Programska implementacija računanja Harrisovog odziva

Pseudokod programske implementacije je sljedeći:

```

dx = apsolutna vrijednost prve Gausove derivacije po x smjeru;
dy = apsolutna vrijednost prve Gausove derivacije po y smjeru;

Ixx = dx*dx;
Iyy = dy*dy;
Ixy = dx*dy;
Det = Ixx*Iyy - Ixy*Ixy;
Trace = Ixx+Iyy;

Vrati (Det/Trace)

```

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 5. Odbacivanje nepotrebnih točaka

Prilikom pronalaženja kutova, moguće je da se pronade više točaka na istom mjestu nego što je to potrebno. To se događa ako je više nezavisnih kuteva u malom okruženju i zbog toga što za više točaka oko jednog kuta vrijede uvjeti koje smo postavili za promjenu intenziteta. Pri tome se događa da točka koja je na samom kutu ima pridruženu najveću vrijednost. Zato nam je potreban postupak kojim ćemo odbaciti sav višak i zadržati samo one točke koje imaju najveću važnost kako bismo odredili samo kuteve.

Za ovaj postupak potrebni su nam podaci koje sami oblikujemo po našoj potrebi. To su granica kojom uvjetujemo minimalnu udaljenost točaka i prag kojim određujemo minimalne vrijednosti koje točkama moraju biti pridružene kako bi se očitale kao kutevi.

Postupak je sljedeći:

1. određivanje praga u odnosu na maksimalnu očitane vrijednost
2. raspoznavanje koje točke imaju veće vrijednosti od praga
3. dobivanje koordinata tih točaka
4. označavanje točaka koje zbog minimalne udaljenosti dolaze u obzir za razmatranje
5. provjeravanje zadovoljavaju li točke oba uvjeta (udaljenosti i praga)
6. vraćanje filtriranih točaka

Ovako izgleda ideja programskog rješenja:

```
funkcija get_harris_points (ulazna_matrica, minimalna_udaljenost, prag)
{
    prag_kuta = prag * maksimum (ulazna_matrica);
    nova_matrica = (ulazna_matrica > prag_kuta);
    koordinate = odredi_nenule (nova_matrica);
    vrijednosti = pridruži_vrijednosti (koordinate);

    dopuštene_lokacije = popuni_nulama;
    dopuštene_lokacije = popuni_jedinicama (udaljenost > minimalna_udaljenost);

    filtrirani_kutevi = provjera(dopuštene_lokacije) i koordinate

    vrati filtrirani_kutevi;
}
```

Konačni prag kojim se uvjetuje koje točke dolaze u obzir ovisan je o maksimalnoj vrijednosti matrice i praga kojim od te maksimalne vrijednosti uzimamo neki postotak kao minimalnu vrijednost koju treba zadovoljiti.



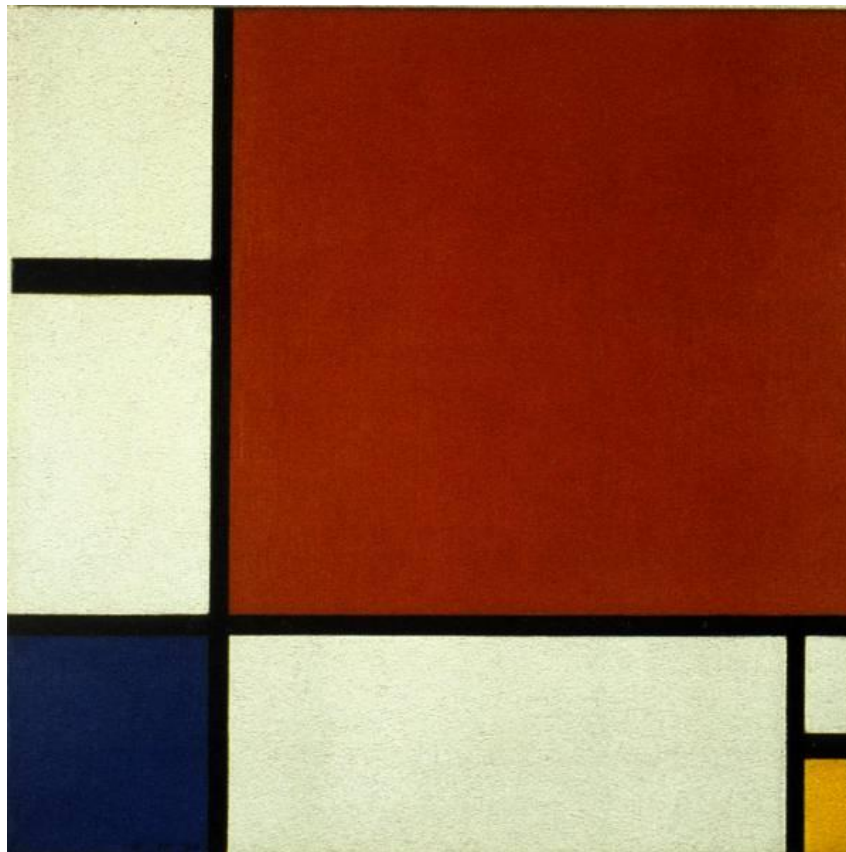
Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

Koordinate i vrijednosti su matrice u koje pohranjujemo točne pozicije tih točaka koje su zadovoljile prethodni uvjet. Zatim u dopuštene vrijednosti pohranjujemo zastavice kojima provjeravamo koje od točaka možemo razmatrati. Vraćamo vrijednost onih točaka koje su udaljenije od minimalne zadane udaljenosti i veće od nekog postotka maksimalne vrijednosti početne matrice.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 6. Rezultati

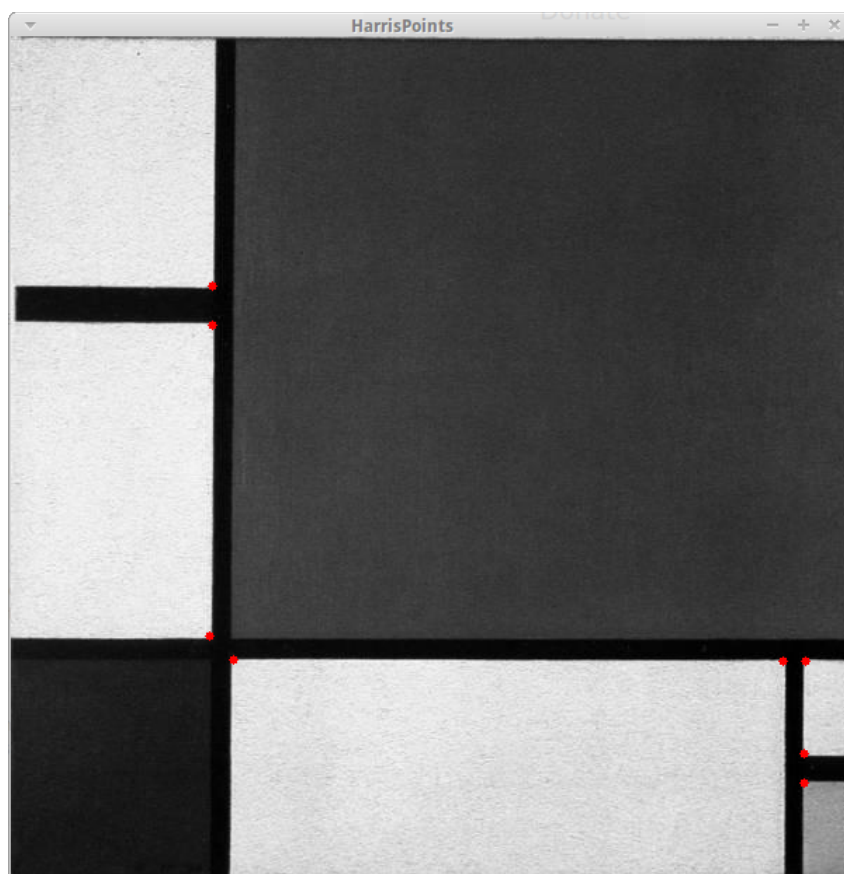
Za pregled rezultata koristi se Slika 13.



*Slika 13. Testna slika*

Razlog tome je što slika ima lako prepoznatljive značajke, odnosno kuteve. Nakon što implementaciji Harrisovog algoritma kao ulaz damo navedenu sliku, kao izlaz dobivamo sliku 14.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

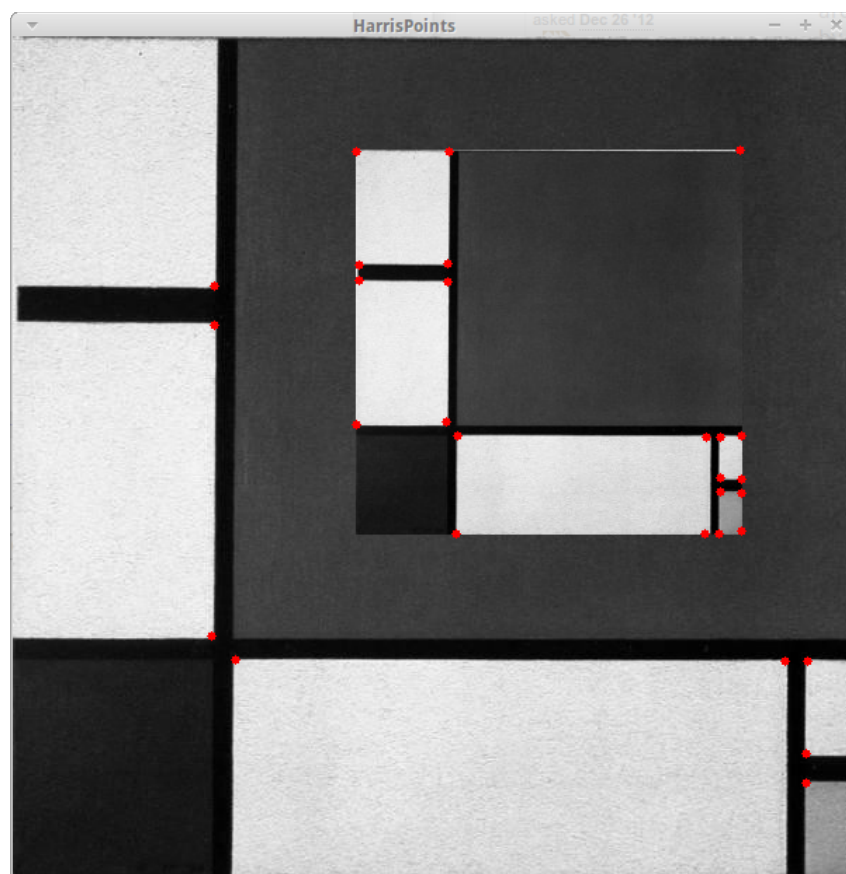


*Slika 14. Rezultat za osnovnu testnu sliku*

Na rezultatnoj slici se jasno vide kutevi koje bismo odabrali da ih određujemo bez pomoći računala. Zaključujemo da implementacija radi dobro za osnovni slučaj. Sljedeći slučaj koji se promatra jest ista slika u koju je u najveće polje ubačena umanjena te rotirana verzija iste slike. Na taj način se može procijeniti koliko je implementacija algoritma neovisna o rotaciji slike. Pritom treba imati na umu da nije riječ samo o rotaciji, već da je u pitanju i skaliranje slike koje bi također moglo utjecati na rezultat. Kako bi doskočili tom problemu, uvodimo još jednu sliku u kojoj eliminiramo rotaciju. Dakle, tu promatramo utjecaj samo skaliranja na testnoj slici.

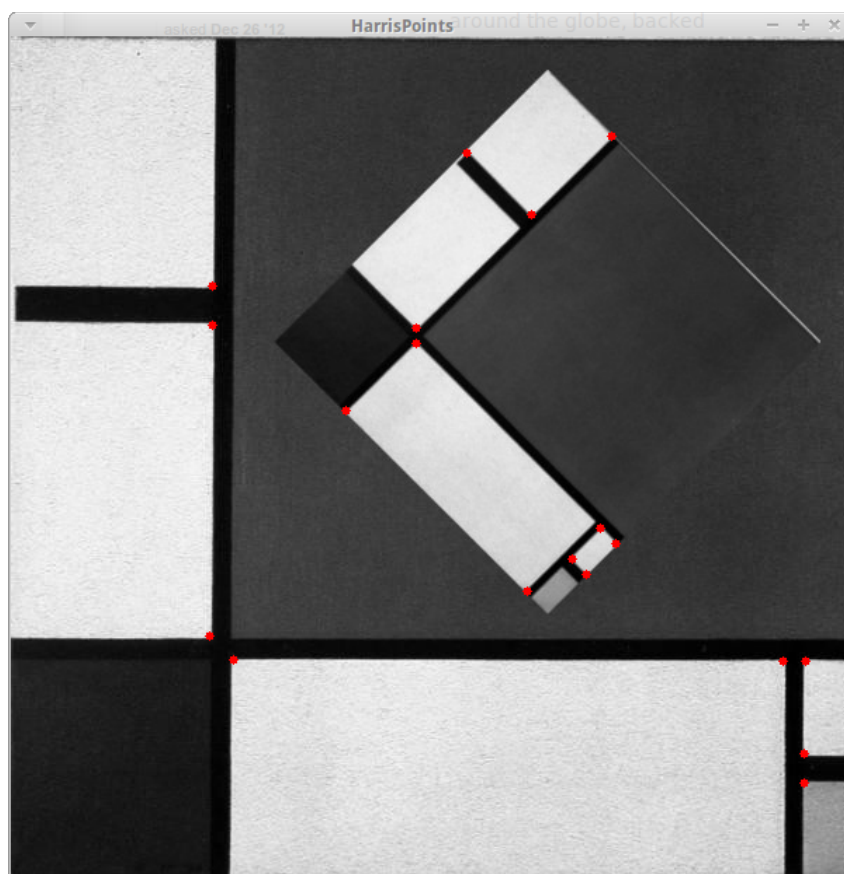
Dobiveni izlaz programa prikazan je na slikama 15 i 16.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 15. Rezultat za testnu sliku s ubačenom nerotiranom slikom*

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12



*Slika 16. Rezultat za testnu sliku s ubačenom rotiranom slikom*

Iz rezultata se može opaziti da program detektira manje kuteva na rotiranoj slici, no još uvijek detektira većinu istih kuteva kao na slici bez rotacije. Takav rezultat nije idealan, ali je dovoljno dobar za korištenje. Također se vidi da skaliranje slike na više od trećine prvotne veličine ne mijenja detekciju kuteva bitno.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 7. Zaključak

Programski jezik Python pogodan je za brzo prototipiranje programske podrške zbog jednostavne sintakse te interpretirane prirode. Pokazuje se da veliki broj biblioteka otvorenog koda također pridonosi lakom prototipiranju.

Nadalje, Harrisov algoritam za detekciju kuteva je dobar alat za pronalazak računalu "pamtljivih" točaka na slici. Dovoljno je računalno efikasan te učinkovit za primjene u području računalnog vida kao što su praćenje objekta na snimci, prepoznavanje objekata, itd. Iz navedenih rezultata možemo zaključiti da je pogodan i za situacije kada postoji mogućnost rotacije objekta koji nas zanima, odnosno njegove slike.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 8. Upute za korištenje

Sljedeće upute za korištenje programskog koda.

### 8.1 Prototip u Pythonu

Potrebno je imati instalirano sljedeće:

- Python 2.7.3
- Python Imaging Library(PIL)
- Numpy
- Scipy
- Matplotlib

#### 8.1.1 Harrisov detektor kuteva

Nakon pokretanja Python IDLE( Python GUI ) i uvođenja potrebnih modula potrebno je pokrenuti detektor kuteva unošenjem sljedećeg koda:

```
im = array('put_do_neke_jpgSlike').convert('L')
harrisim = compute_harris_response(im)
filtered_coords = get_harris_points(harrisim,10, 0.01)
plot_harris_points(im, filtered_coords)
```

#### 8.1.2 Grafički prikaz gradijenata

Potrebno je unijeti sljedeći kod:

```
im = array(Image.open('imeslike.jpg').convert('L'))
grad(im, sigma, prozor)
```

Gdje su sigma i prozor neobavezni argumenti s pretpostavljenim vrijednostima 1 i 5. Sigma je korišten za Gaussovo zamućenje, a prozor određuje veličinu prozora kod računanja gradijenta.

#### 8.1.3 Uparivanje slika

Nakon pokretanja Python IDLE( Python GUI ) i uvođenja potrebnih modula, potrebno je učitati dvije slike koje želimo upariti. Prije uparivanja slike je potrebno pretvoriti u *grayscale* format te ih nakon toga prebaciti u *array* tip.

Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

```
slika = array(Image.open('put_do_neke_jpgSlike').convert('L'))
```

Nakon što smo učitali slike koje želimo usporediti, jednostavno pozovemo funkciju `upari()` iz *Harris* modula.

```
harris.upari(slika1,slika2)
```

Dodatno je moguće podešavati vrijednost  $\sigma$ , širinu prozora iz kojeg se uzima uzorak opisnika kuta koja uvećana za jedan ujedno služi i kao minimalna udaljenost dvaju kuteva na slikama.

## 8.2 Implementacija u C++-u

Za kompajliranje je potrebno imati instaliran OpenCV 2.4.3, najnoviju verziju u vrijeme pisanja ovog teksta.

Nakon što se kompajlira pomoću priložene *Makefile* datoteke, program se poziva na sljedeći način iz komandne linije:

```
./harrisDetector [-d] [-s] [-k kernel_size] [-t threshold] -i ime_slike
```

Pritom su parametri unutar uglatih zagrada neobavezni. Objašnjenje korištenja prikazano je u tablici 1.

Tablica 1: Detaljno objašnjenje parametara

Parametar	Objašnjenje
-d	Uključuje crtanje dijagnostičkih slika za međukorake algoritma.
-s	Onemogućuje početno zaglađivanje slike.
-k	Postavlja radijus za zaglađivanje, pretpostavljena vrijednost je 3.
-t	Postavlja graničnu vrijednost za odbacivanje nepotrebnih točaka u intervalu $<0,1>$ , pretpostavljena vrijednost je 0.1.
-i	Postavlja putanju do ulazne slike.

Program se gasi pritiskom na tipku Esc.



Implementacija Harrisovog detektora kuteva	Verzija: 1.0
Tehnička dokumentacija	Datum: 07/01/12

## 9. Literatura

1. Solem, J.E., Programming Computer Vision with Python, Sebastopol: O'Reilly, 2012
2. Sobel derivatives,  
[http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel\\_derivatives/sobel\\_derivatives.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html),  
15.12.2012.
3. OpenCV reference manual, release 2.4.2, July 2012.
4. Gaussian blur, [http://en.wikipedia.org/wiki/Gaussian\\_blur](http://en.wikipedia.org/wiki/Gaussian_blur), 9.12.2012.
5. Vito Macchia, OpenCV's C++ interface, 30.6.2010., <http://www.aishack.in/2010/07/opencvs-c-interface/>,  
8.12.2012.
6. Matplotlib reference guide, <http://matplotlib.org/>, 10.12.2012.
7. Numpy reference guide, <http://docs.scipy.org/doc/numpy/reference/>, 10.12.2012.
8. Wijewickrema, S.N.R., Paplinski, A.P., Principal Component Analysis for the Approximation of a Fruit as an Ellipse, Australia, Monash University, 2004.
9. Intuitive explanation to Harris corner detector, Andrey,  
<http://matlabcorner.wordpress.com/2012/11/17/does-harris-corner-detector-finds-corners-intuitive-explanation-to-harris-corner-detector/>, 10.12.2012.
10. Principal Component Analysis, Jolliffe, I.T., Second edition, Springer, Aberdeen, 2002.