

1. Detekcija objekata i njihove orijentacije na osnovu histograma gradijenata

U ovom dokumentu opisana je ideja i algoritam detekcije objekata (konkretno automobila) i njihove orijentacije na osnovu orijentacije histograma gradijenata. Algoritam je implementiran u *Python* programskom jeziku u *Spyder* okruženju.

1.1. Ulazne slike

Kao ulazni tip slika korišćene su slike automobila *Passat*, ukupno 101 slika. Format korišćenih slika je *JPEG*. Na slici 1. biće prikazana jedna od ulaznih slika.



Slika 1. Ulazna slika

Učitavanje slika izvršeno je u nekoliko liniji koda uz pomoć *OpenCV* biblioteke(1.1).

```
inimg_slika_sa_kojom_se_poredi = cv2.imread("passat_main.jpg")  
  
height = np.size(inimg_slika_sa_kojom_se_poredi, 0)  
  
width = np.size(inimg_slika_sa_kojom_se_poredi, 1)
```

(1.1)

1.2. Vektori karakteristika

Osnovni zapis prikaza boja na slikama jeste *RGB* model boje: (engl. *red, green, blue*). Svaki element (engl. *pixel*) predstavljen je s tri komponente: crvenom, zelenom i plavom, od kojih svaka komponenta ima vrijednost između 0 i 255. Pošto algoritam koristi histograme gradjenata potrebno je učitane slike konvertovati u *grayscale* slike. Konverzija *RGB* slike u *grayscale* sliku izvršena je u liniji koda, takođe uz pomoć *OpenCV* biblioteke(1.2).

```
inimg1_gray = cv2.cvtColor(inimg_slika_sa_kojom_se_poredi, cv2.COLOR_RGB2GRAY) (1.2)
```

1.3. Algoritam

U ovom radu predloženo je otkrivanje područja od interesa (*Region of Interes - ROI*) objekata. Za testiranje, odnosno, detekciju korišćene su fotografije automobila. Pomoću histograma gradjenata (*Histogram of Oriented Gradient - HOG*) deskriptora i podudaranja elemenata samo u *ROI*. Ovim pristupom povećava se tačnost algoritma i štedi se na vremenu računanja.

1.3.1. Histogram orijentacije gradjenata (HOG)

Osnovna ideja histograma orijentacije gradijenta je podjela slike na više dijelova, tzv. blokova te izrada histograma koji nam govore o učestalosti smjera gradjenata elemenata slike unutar svakog bloka. Veličina blokova u ovom radu jeste 128x64 piksela (1.3).

```
blockSize = (16, 16) (1.3)

blockStride = (8, 8)

cellSize = (8, 8)

nbins = 9
```

Ti blokovi često mogu biti okarakterisani izgledom i oblikom objekta sa prilično dobro raspoređenim gradijentima lokalnog intenziteta ili rubova samog objekta. Pri određivanju većine deskriptora slika, prvi korak čini normalizacija boje i osvjetljenja slike što je već spomenuto, a nakon toga se vrši izračunavanje gradjenata slika. Proces izračunavanja gradjenata slike naziva se konvolucija. Konvolucijom izvorne slike *I* sa filterom Sobel, gdje su

$$Sx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ i } Sy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{ aproksimacije horizontalnih i vertikalnih derivata,}$$

dobijamo dvije nove slike $I_x = I * S_x$ i $I_y = I * S_y$. Primjer konvolucije slike sa filterom dimenzija 3x2 dat je na slici 2.

I₁₁	I₁₂	I₁₃	I₁₄	I₁₅	I₁₆	I₁₇	I₁₈	I₁₉
I₂₁	I₂₂	I₂₃	I₂₄	I₂₅	I₂₆	I₂₇	I₂₈	I₂₉
I₃₁	I₃₂	I₃₃	I₃₄	I₃₅	I₃₆	I₃₇	I₃₈	I₃₉
I₄₁	I₄₂	I₄₃	I₄₄	I₄₅	I₄₆	I₄₇	I₄₈	I₄₉
I₅₁	I₅₂	I₅₃	I₅₄	I₅₅	I₅₆	I₅₇	I₅₈	I₅₉
I₆₁	I₆₂	I₆₃	I₆₄	I₆₅	I₆₆	I₆₇	I₆₈	I₆₉

K₁₁	K₁₂	K₁₃
K₂₁	K₂₂	K₂₃

Slika 2. Primjer slike i filtera za konvoluciju

Konvolucija se izvodi pomjeranjem filtera preko svih dijelova slike koje filter može u potpunosti prekriti, počevši od gornjeg lijevog ugla. Svaka pozicija filtera određuje jedan izlazni element slike, čija vrijednost se računa množenjem vrijednosti filtera i odgovarajućeg elementa slike za svaku ćeliju filtera te sabiranjem svih tih vrijednosti. Primjer računanja izlazne vrijednosti jednog elementa slike, npr. O_{57} prikazan je jednačinom (1.4).

$$O_{57} = I_{57} * K_{11} + I_{58} * K_{12} + I_{59} * K_{13} + I_{67} * K_{21} + I_{68} * K_{22} + I_{69} * K_{23} \quad (1.4)$$

Opšti matematički izraz za računanje konvolucije dat je jednačinom (1.5), gdje su $M \times N$ dimenzije slike i $m \times n$ dimenzije filtera, varijabla i ima vrijednosti od 1 do $M - m + 1$, a varijabla j ima vrijednosti 1 do $N - n + 1$.

$$O_{i,j} = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1) \cdot K(k,l) \quad (1.5)$$

Nakon završenog filtriranja slike, vrši se računanje iznosa i orijentacije gradijenata za pojedinačne elemente slike. Ukupni gradijenat elemenata slike računa se jednačinom (1.6):

$$|G| = \sqrt{S_x^2 + S_y^2} \quad (1.6)$$

Za ovaj korak iskorišćena je biblioteka *OpenCV*.

Nakon izvršene konvolucije, u linijama koda (1.7) algoritam prolazi kroz sve slike i njihove regione od interesa i vrši poređenja jedne slike sa ostalima kako bi pronašao najsličniju sliku, odnosno detektovao objekat sa istom orijentacijom:

```

sum_distance
= []

sum_all_distance = []
num_of_img = 16
Prefix = "trainingImgs/passat ("
for n in range(1, num_of_img, 1):
    s=str(n)
    inimg1 = cv2.imread( "trainingImgs/passat (" + s + ").jpg")
    inimg1_b_gray = cv2.cvtColor(inimg1, cv2.COLOR_RGB2GRAY)
    for i in range(0, width - 64, 1):
        for j in range(0, height - 128, 24):
            roi1 = inimg1_gray[j: j + 128, i: i + 64]
            roi2 = inimg1_b_gray[j: j + 128, i: i + 64]
            cv2.imwrite("roi1.jpg", roi1)
            roi1a = roi1.size
            roi2a = roi2.size
            d = cv2.HOGDescriptor((64, 128), blockSize, blockStride,
cellSize, nbins)
            d2 = cv2.HOGDescriptor((64, 128), blockSize, blockStride,
cellSize, nbins)
            winStride = (8, 8)
            padding = (8, 8)
            locations = ((10, 20),)
            descriptorValues1 = d.compute(roi1)
            descriptorValues2 = d2.compute(roi2)
            A = descriptorValues1
            B = descriptorValues2

```

(1.7)

Zahvaljujući lokalnim varijacijama u osvjetljenju i kontrastu boja, iznosi gradijenata takođe variraju. Dodatno poboljšanje performansi algoritma postiže se preklapanjem blokova, tako da svaka ćelija doprinosi konačnom rezultatu više puta. Konačni rezultat slike je tada vektor normalizovanih histograma blokova (1.6).

Sljedeći korak jeste određivanje magnituda/opsega (1.8) i uslova za histogram gradijenata.

```

average_dist
= 0

percent = (n / num_of_img) * 100.0
ps = str(percent)

s = "Processing" + ps + " % "
print(s)
for k in range(0, sum_distance.__len__(), 1):
    average_dist += sum_distance[k]
average_dist /= sum_distance.__sizeof__()
sum_all_distance.append(average_dist)
del sum_distance[:]

```

(1.8)

Poređenjem tih magnituda dobijaju se uslovi za detekciju objekta i njegove orijentacije. Najmanja razlika između magnituda ustvari ukazuje na sliku na kojoj je objekat detektovan (1.9) i ta slika prikazana je u prozoru.

```

min =
10000

rememberL = 0
for l in range(0, num_of_img - 1, 1):
    if(sum_all_distance[l] < min):
        min = sum_all_distance[l]
        rememberL = l + 1

f=str(rememberL)
cv2.imwrite("main.jpg", inimg_slika_sa_kojom_se_poredi)
cv2.imshow("1", inimg_slika_sa_kojom_se_poredi)
ucitaj = cv2.imread(Prefix + f + ".jpg")
cv2.imshow("2", ucitaj)
cv2.imwrite("najslicnija.jpg", ucitaj)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

(1.9)

