DETECTIA BENZILOR DE CIRCULATIE

Student: Luchian Bogdan-Ionut

Grupa: CI-1B

1. Enuntul problemei

O problema de baza in dezvoltarea vehiculelor autonome consta in detectia benzilor de circulatie. Solutia acestui proiect va va fi construita folosind un detector de muchii (ex. Canny) si transformata Hough pentru detectia liniilor drepte.

2. Descrierea solutiei

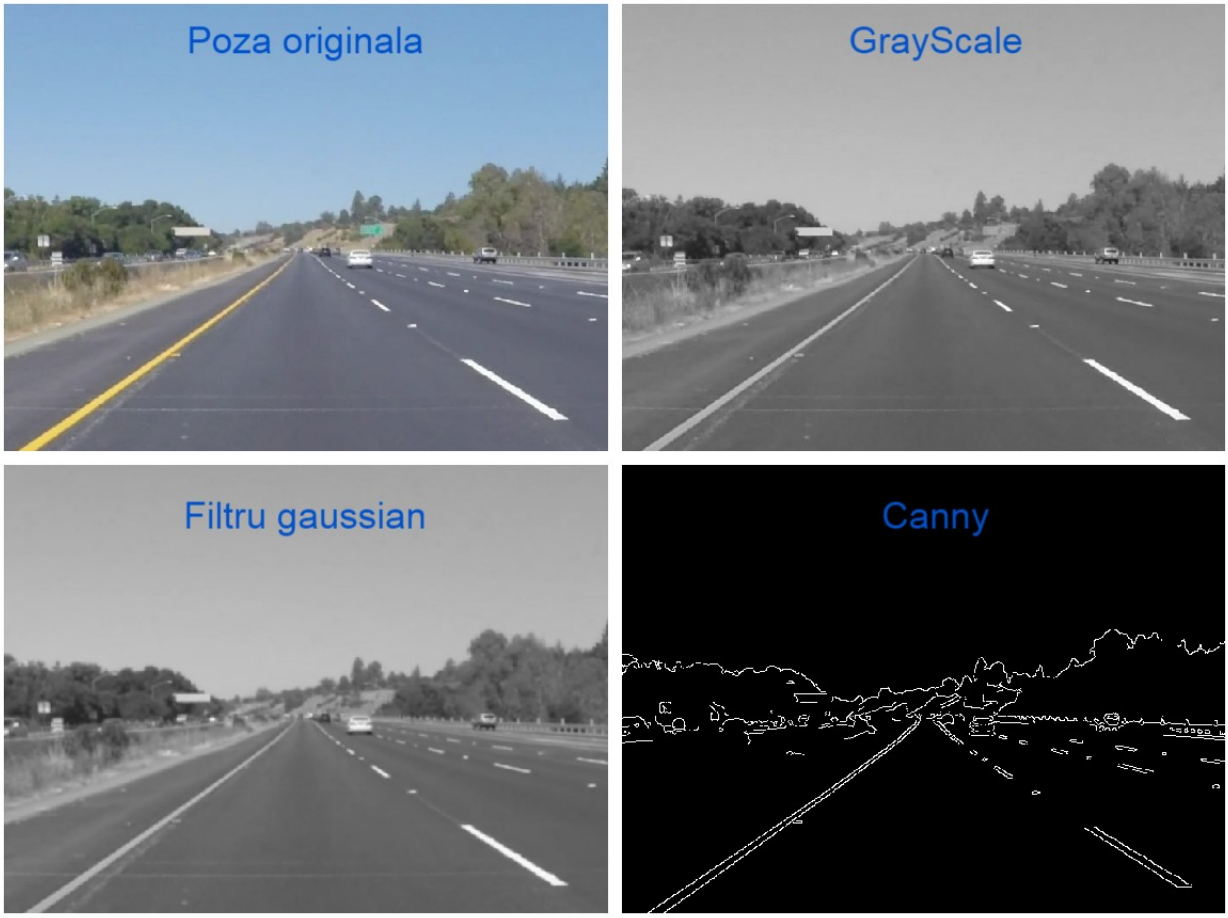
Pentru inceput vom incarca o imagine din colecţia noastră pentru a ne testa algoritmul.

```
image_path = r"C:\Users\Lenovo\PycharmProjects\pythonProject\test_images\
solidWhiteCurve.jpg"
image1 = cv2.imread(image_path)
plt.imshow(image1)
```

In continuare vom define 3 functii pentru manipularea imaginii.

```
def grey(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
def gauss(image):
    return cv2.GaussianBlur(image, (5, 5), 0)
def canny(image):
    edges = cv2.Canny(image,50,150)
    return edges
```

- *GreyScale*: aceasta ajută la creşterea contrastului culorilor, facilitând identificarea modificărilor intensităţii pixelilor.
- *Filtrul gaussian*: Scopul filtrului este de a reduce zgomotul din imagine.
- *Canny*: Cu acesta detectăm marginile din imagine.

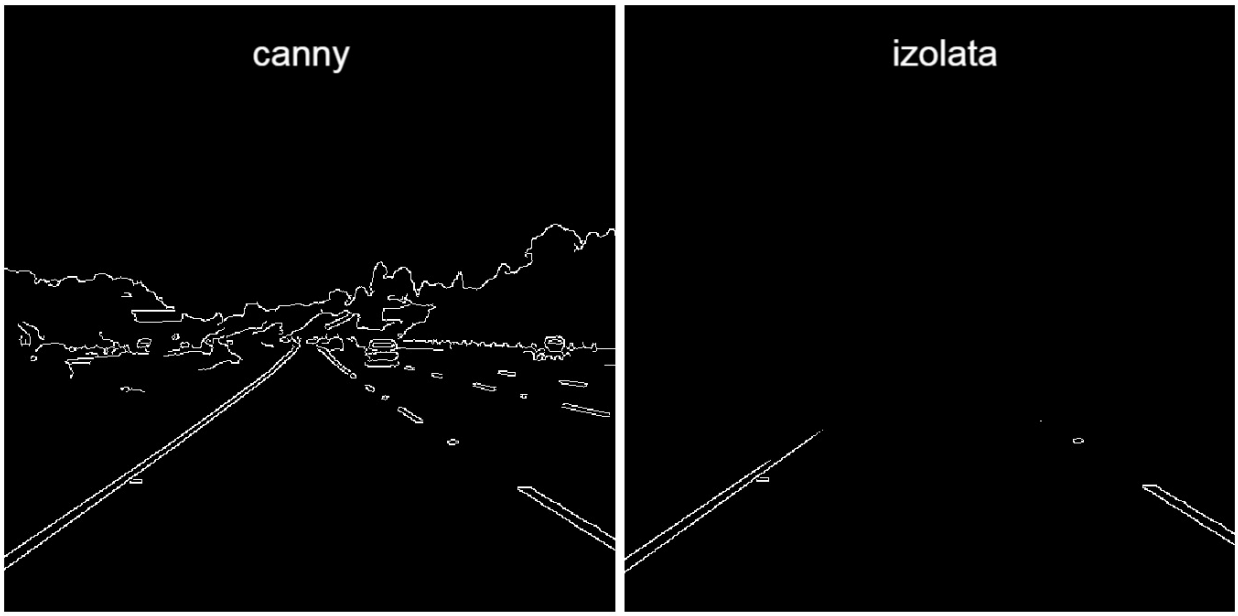


Acum că am definit toate marginile din imagine, trebuie să izolăm marginile care corespund liniilor de bandă. Vom folosi urmatoarea functie:

```
def region(image):
    height, width = image.shape
    triangle = np.array([[(100, height), (475, 325), (width, height)]])
    mask = np.zeros_like(image)
    mask = cv2.fillPoly(mask, triangle, 255)
    mask = cv2.bitwise_and(image, mask)
    return mask
```

Această funcţie va izola o anumită regiune codificată în imaginea în care sunt liniile de bandă. Este nevoie de un parametru, imaginea Canny şi transmite regiunea izolată.

În imaginea noastră, există intensităţi de doi pixeli: alb-negru. Pixelii negri au valoarea 0, iar pixelii albi au valoarea 255. În binar de 8 biţi, 0 se transformă în 00000000 şi 255 se traduce în 11111111. Pentru bit şi operaţie, vom folosi valorile binare ale pixelului. Am repeta această operaţiune pentru fiecare pixel de pe imagine, rezultând doar marginile din ieşirea măştii.



Urmatorul pas este să definim funcția care transformă aceste margini în linii. Acest lucru se va face cu ajutorul Transformatei Hough, care transformă acele grupuri de pixeli albi din regiunea noastră izolată în linii reale.

```
lines = cv2.HoughLinesP(isolated, 2, np.pi/180, 100, np.array([]),
minLineLength=40, maxLineGap=5)
```

Dacă nu facem o medie a liniilor, acestea par foarte agitate, deoarece *cv2.HoughLinesP* produce o grămadă de segmente de linii în loc de o linie mare. De aceea pentru a trasa liniile vom define o functie numita “average”:

```
def average(image, lines):
    left = []
    right = []
    for line in lines:
        print(line)
        x1, y1, x2, y2 = line.reshape(4)
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        y_int = parameters[1]
        if slope < 0:
            left.append((slope, y_int))
        else:
            right.append((slope, y_int))
    right_avg = np.average(right, axis=0)
    left_avg = np.average(left, axis=0)
    left_line = make_points(image, left_avg)
    right_line = make_points(image, right_avg)
    return np.array([left_line, right_line])
```

Acesta va găsi panta medie și interceptarea y a segmentelor de linie din stânga și din dreapta și va afișa două linii solide (una în stânga și alta în dreapta). Fiecare segment de linie are

2 coordonate: una denotă începutul liniei, iar cealaltă marchează sfârşitul liniei. Folosind aceste coordonate, vom calcula pante şi interceptările y ale fiecărui segment de linie.

Apoi, vom colecta toate pantele segmentelor de linie şi vom clasifica fiecare segment de linie fie în lista corespunzătoare cu linia stângă sau dreapta (panta negativă = linia stângă, panta pozitivă = linia dreaptă).

Iar la final trebuie să luăm media pârtiilor şi a interceptărilor Y din ambele liste.

```
right_avg = np.average(right, axis=0)
left_avg = np.average(left, axis=0)
left_line = make_points(image, left_avg)
right_line = make_points(image, right_avg)
```

Acum, că avem panta medie şi interceptarea y pentru ambele liste, vom defini punctele de început şi final pentru ambele liste.

```
def make_points(image, average):
    slope, y_int = average
    y1 = image.shape[0]
    y2 = int(y1 * (3/5))
    x1 = int((y1 - y_int) // slope)
    x2 = int((y2 - y_int) // slope)
    return np.array([x1, y1, x2, y2])
```

Această funcţie preia doi parametri, imaginea cu liniile de bandă şi lista cu panta medie şi y_int ale liniei, şi afişează punctele de început şi de sfârşit pentru fiecare linie. Calculăm coordonatele *x* rearanjând ecuaţia unei linii, de la $y = mx + b$ la $x = (yb) / m$.

Cu toate acestea, această funcţie nu afişează liniile, ci calculează doar punctele necesare pentru afişarea acestor linii. Urmeaza să creăm o funcţie care ia aceste puncte şi face linii din ele.

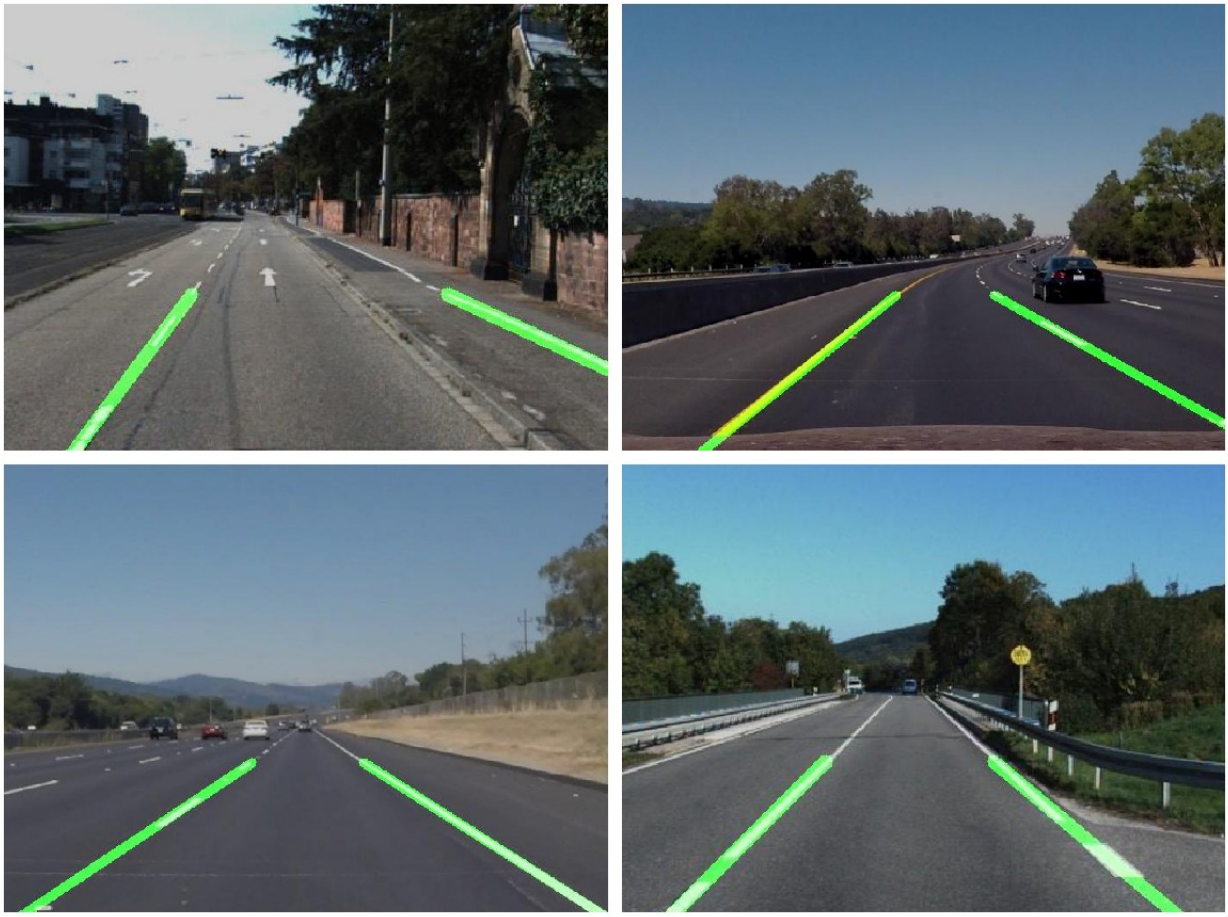
```
def display_lines(image, lines):
    lines_image = np.zeros_like(image)
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line
            cv2.line(lines_image, (x1, y1), (x2, y2), (0, 255, 0), 10)
    return lines_image
```

Aceasta functie presupunea crearea unei imagini cu aceleaşi dimensiuni ca şi imaginea noastră original, ne asigurăm că listele cu punctele de linie nu sunt goale, parcurgem listele şi extragem cele două perechi de coordinate, creăm linia şi o lipim pe imaginea întunecată.

Prin apelarea funcţiei *cv2.addWeighted* dăm o greutate 0,8 fiecărui pixel din imaginea reală, făcându-i puţin mai întunecaţi. Poza finală va arăta aşa cum se poate observa în figura de mai jos.

3) Rezultate favorabile

Pasii descrisi mai sus functioneaza in mare parte pe imaginile ce prezinta un drum drept, atat cu linii continue cat si cu linii discontinue pe partea stanga si dreapta a vehiculului.



4) Rezultate nefavorabile

Acest algoritm nu functioneaza intotdeauna. Una dintre aceste cauze este faptul ca pozele ce intalnesc un viraj/curba nu fac bine triunghiurile din functia isolated, ajungand ca linii sa inceapa intotdeauna din partea dreapta jos. O alta problema ar fi ca unele poze nu prezinta linia continua/prima linie din dreapta fapt pentru care se va trasa o singura linie cu verde si anume cea din stanga.

5) Video cu rezultatul segmentarii

Pentru a putea genera benzile pentru un videoclip a trebuit sa aplicam toate functiile mentionate anterior. Putem spune ca un videoclip este doar o grămadă de imagini care apar una după alta foarte repede.

```
#Video
video = r"C:\Users\Lenovo\PycharmProjects\pythonProject\test_videos\1.mp4"
cap = cv2.VideoCapture(video)
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        gaus = gauss(frame)
```

```
        edges = cv2.Canny(gaus,50,150)
        isolated = region(edges)
        lines = cv2.HoughLinesP(isolated, 2, np.pi/180, 50, np.array([]),
minLineLength=40, maxLineGap=5)
        averaged_lines = average(frame, lines)
        black_lines = display_lines(frame, averaged_lines)
        lanes = cv2.ad1dWeighted(frame, 0.8, black_lines, 1, 1)
        cv2.imshow("frame", lanes)
cap.release()
cv2.destroyAllWindows()
```