Sprawozdanie 2. AiPO Mirosław Kołodziej

April 30, 2023

1 Laboratorium 5

1.1 a) Proszę wczytać przykładowy obraz lab5_1.jpg i skonwertować go do obrazu w skali szarości.

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_1.jpg")
cv2_imshow(image)
```

[ ]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

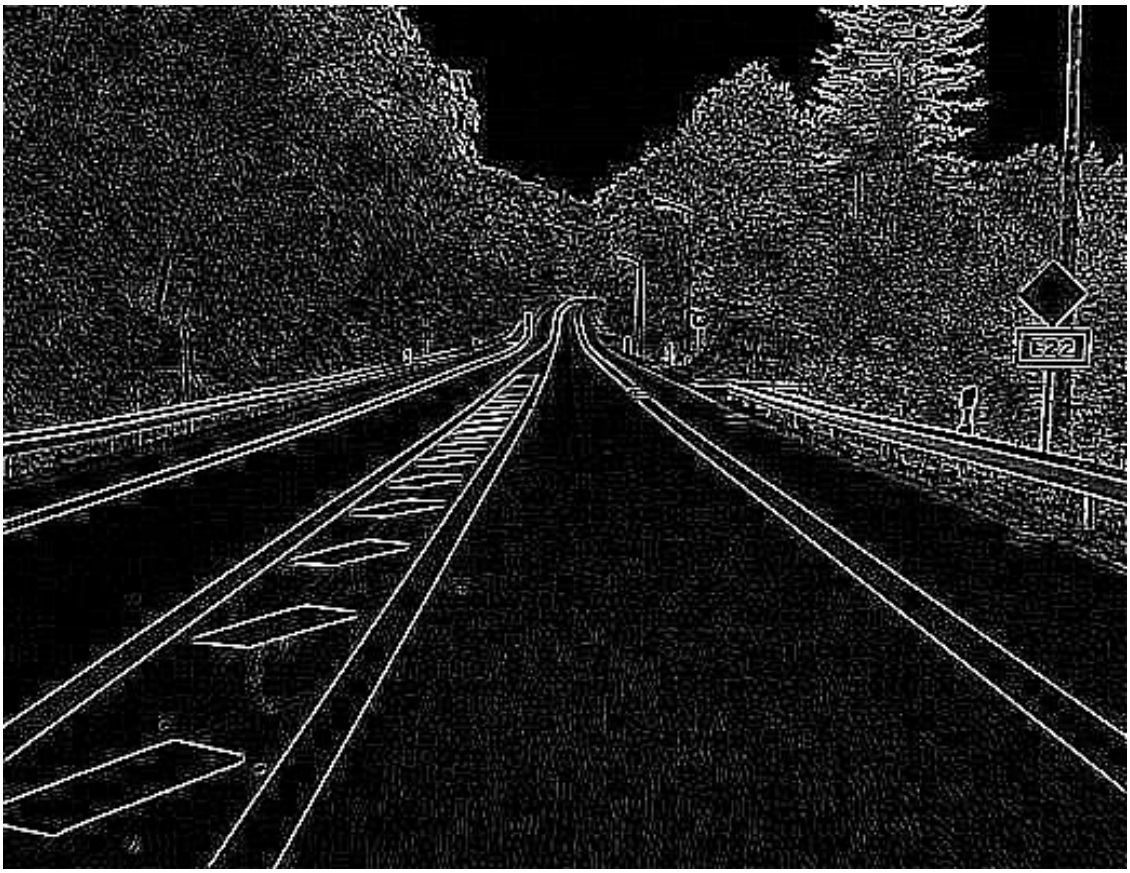
1.2 b) Proszę sprawdzić i porównać wynik wykrywania krawędzi metodami

1.3 a. Laplace

1.4 b. Canny

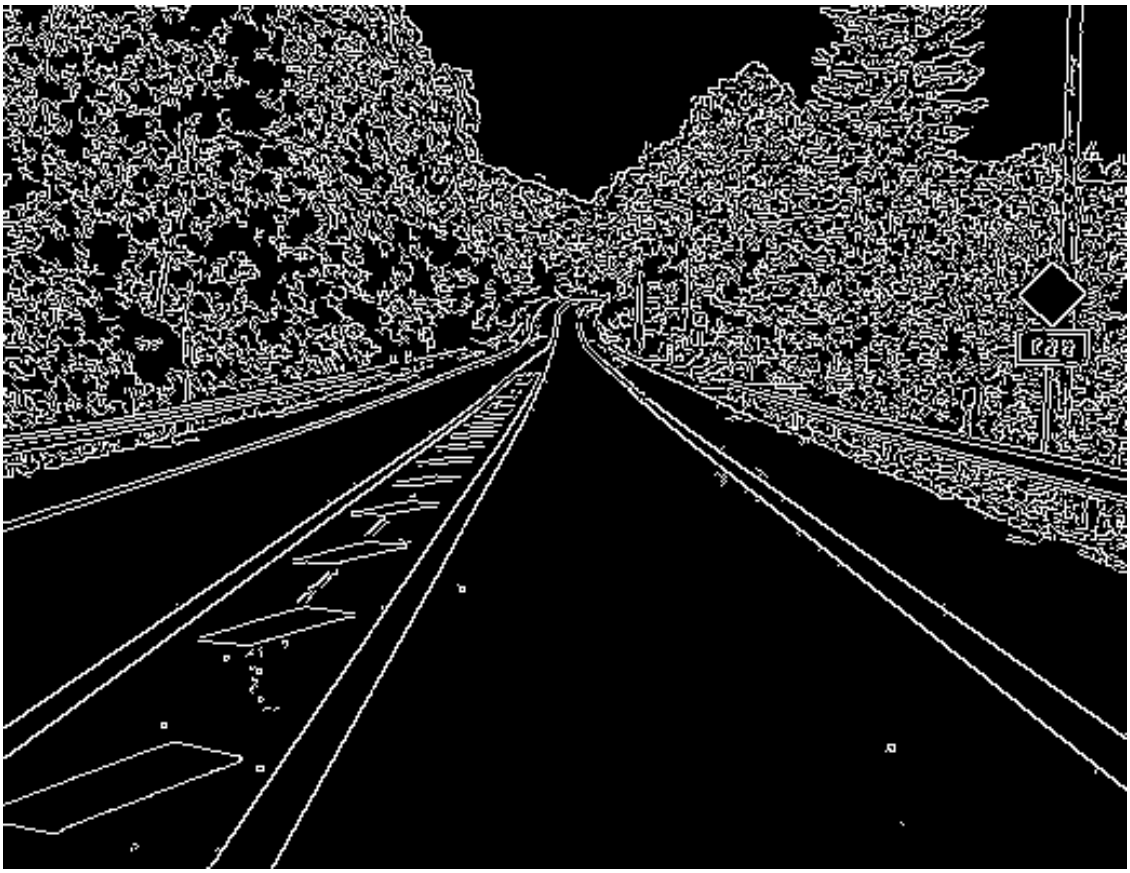
1.5 dla obrazu oryginalnego i obrazu z nałożonym filtrem gaussowskim (5x5). Wynikowy obraz można poddać progowaniu.

[ ]:
```
lap = cv2.Laplacian(gray, cv2.CV_16S, ksize=3)
cv2_imshow(lap)
```



[ ]:
```
gauss = cv2.GaussianBlur(gray,(5,5),0)
lap2 = cv2.Laplacian(gauss, cv2.CV_16S, ksize=3)
cv2_imshow(lap2)
```

[ ]:
```
can = cv2.Canny(gray,50,150)
cv2_imshow(can)
```



[ ]:
```
can2 = cv2.Canny(gauss,50,150)
cv2_imshow(can2)
```

Metoda Canny jest bardziej odporna na szumy oraz działa lepiej w przypadku krawędzi o niskim kontraście. Dodatkowo daje lepsze wyniki krawędziowe i umożliwia identyfikowanie krawędzi o różnej grubości. Nałożenie filtra na obraz sprawiło, że wykrywane są tylko wyraźne krawędzie i poprawiło wyniki w przypadku obu metod. Mimo wszystko, wciąż lepsze wyniki daje metoda Canny.

1.6 c) Proszę zbadać wpływ parametrów minVal, maxVal i kSize (nie ma w Canny) na wynik działania algorytmu Canny.

[ ]:
```
for i, val1 in enumerate([3,5,7,9]):
  for j, val2 in enumerate([50,100,150]):
    for k, val3 in enumerate([150,200,250]):
      print(val1, val2, val3)
      gauss = cv2.GaussianBlur(gray,(val1,val1),0)
      can = cv2.Canny(gauss,val2,val3)
      cv2_imshow(can)
```

```
3 50 150
```

3 50 200

3 50 250

```
3 100 150
```

```
3 100 200
```

```
3 100 250
```

```
3 150 150
```

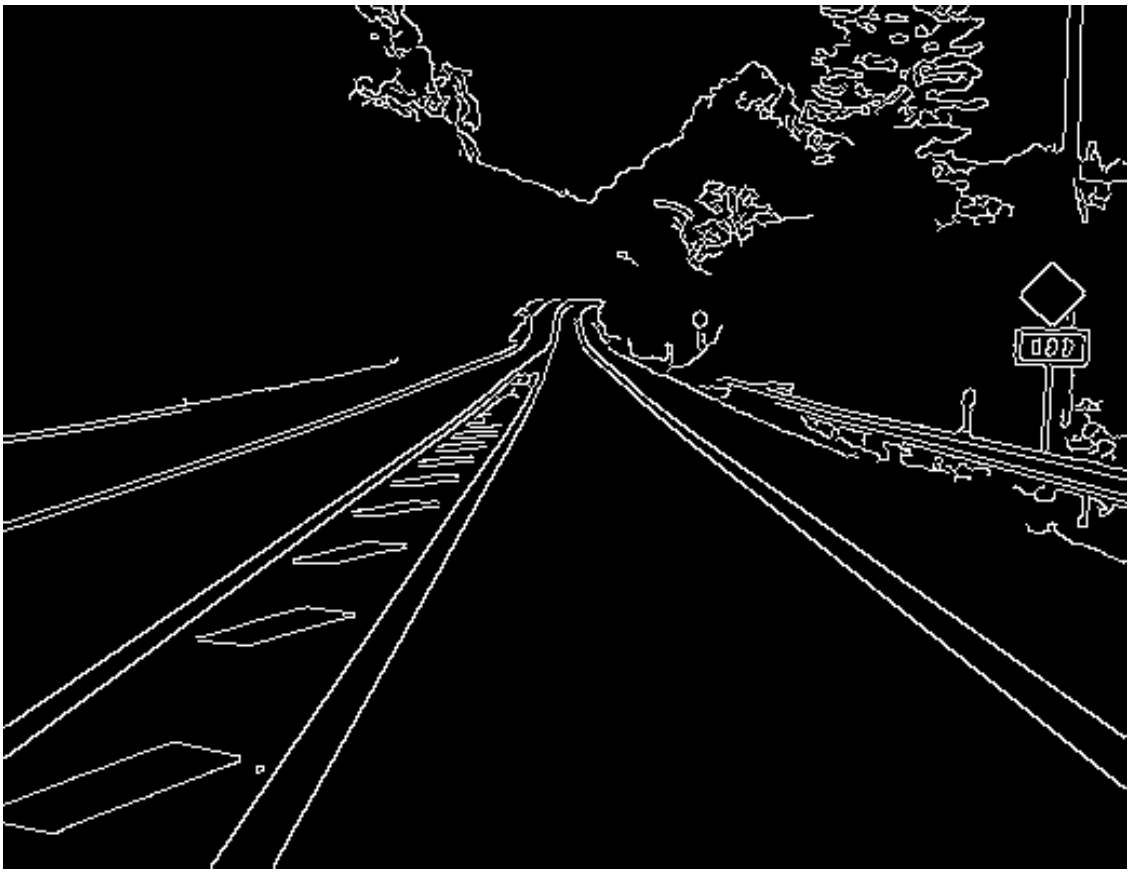
```
3 150 200
```

```
3 150 250
```

5 50 150

5 50 200

5 50 250



```
5 100 150
```

5 100 200

```
5 100 250
```

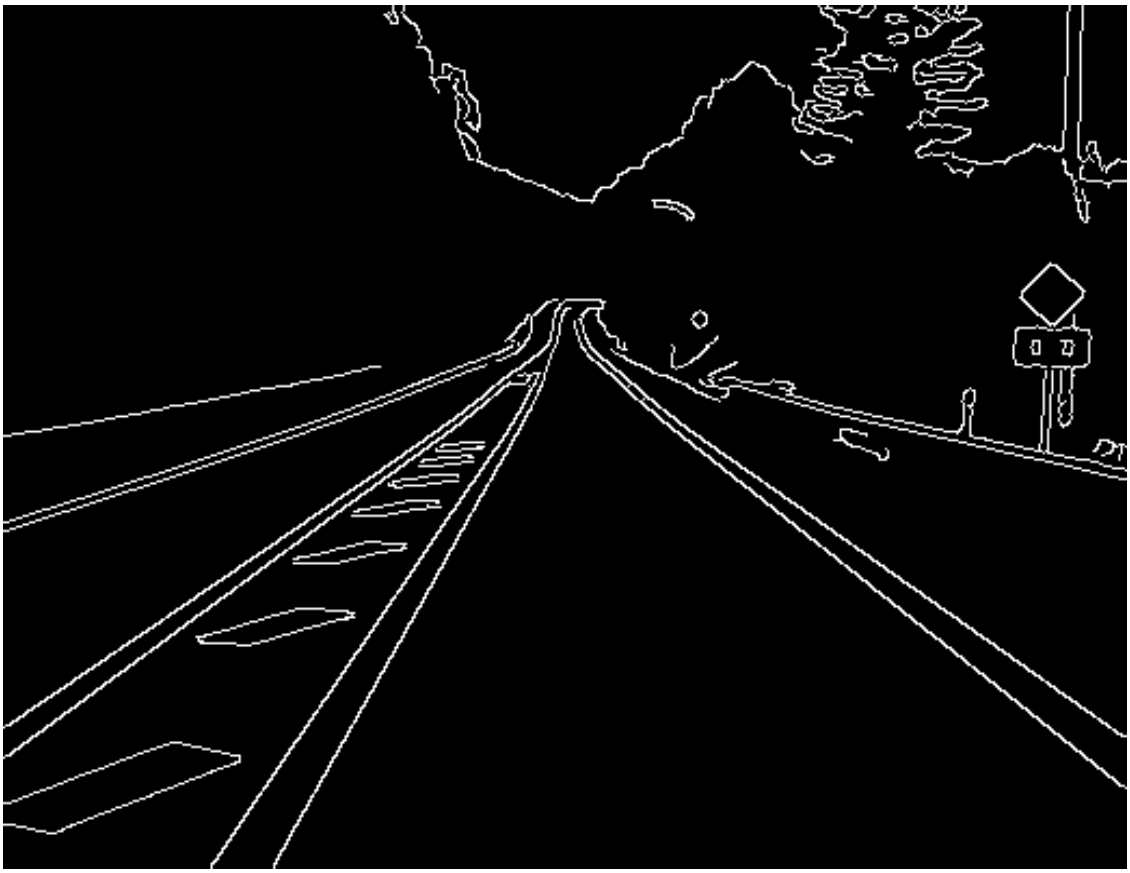
```
5 150 150
```

5 150 200

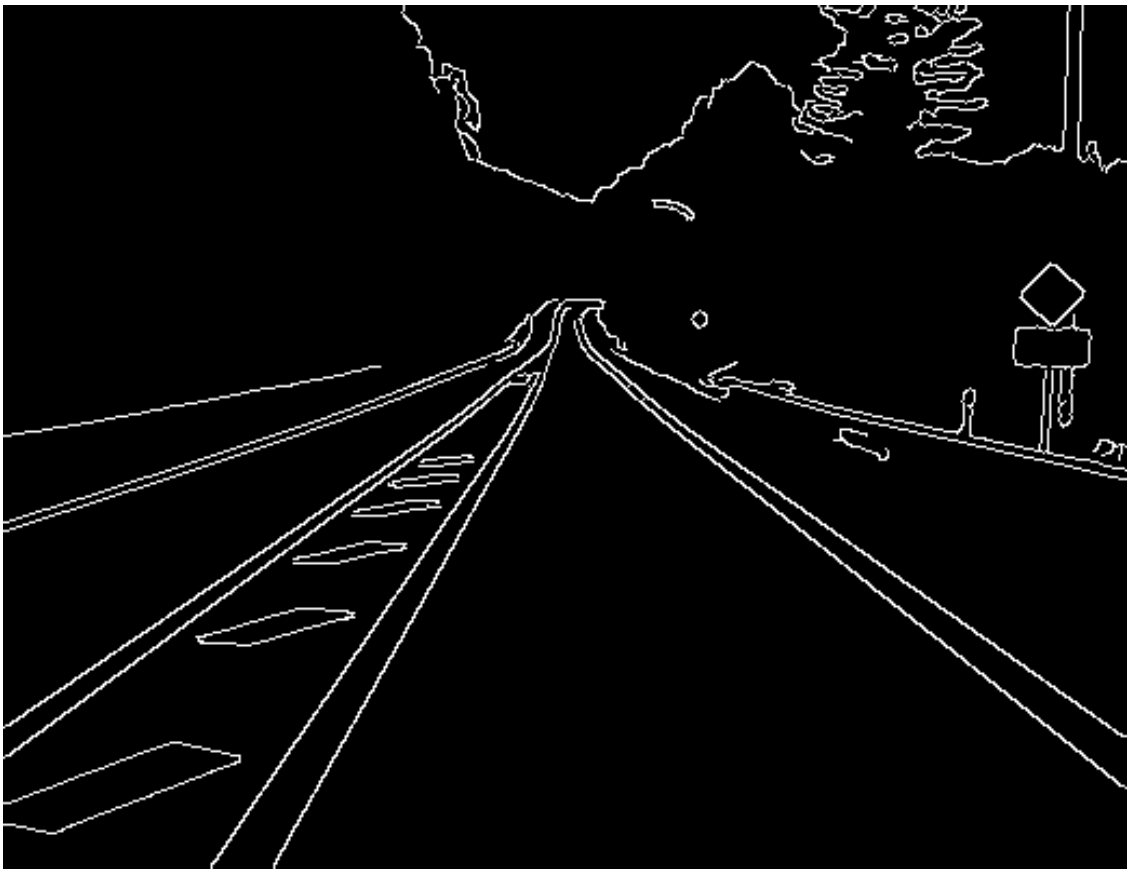
5 150 250

7 50 150

7 50 200

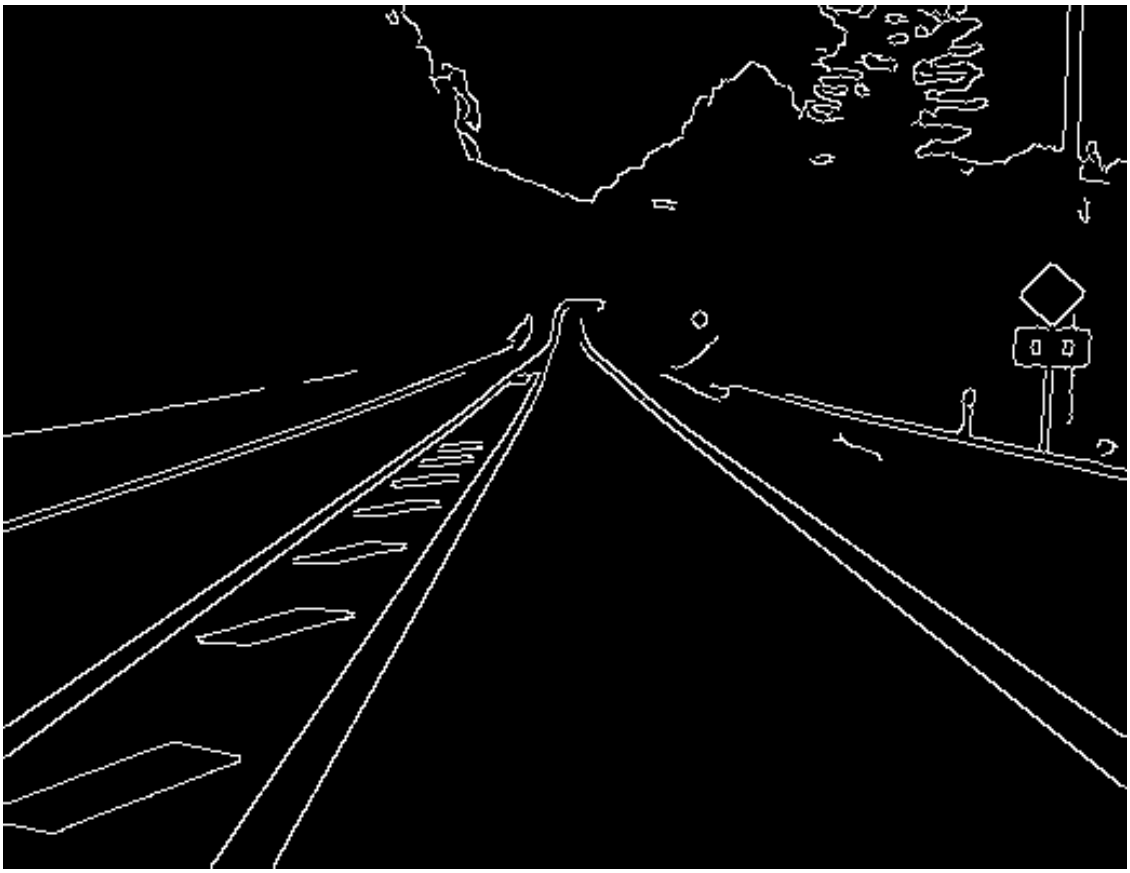


7 50 250

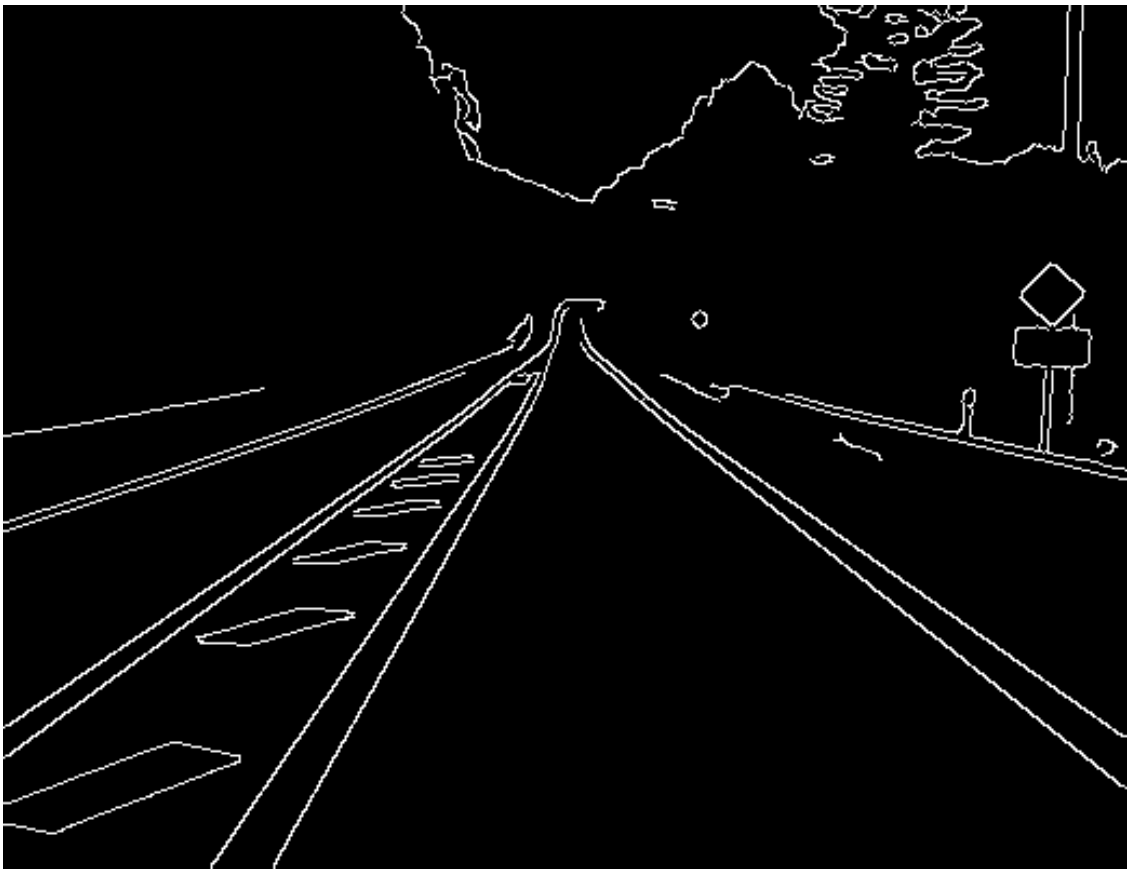


```
7 100 150
```

```
7  100  200
```



```
7 100 250
```

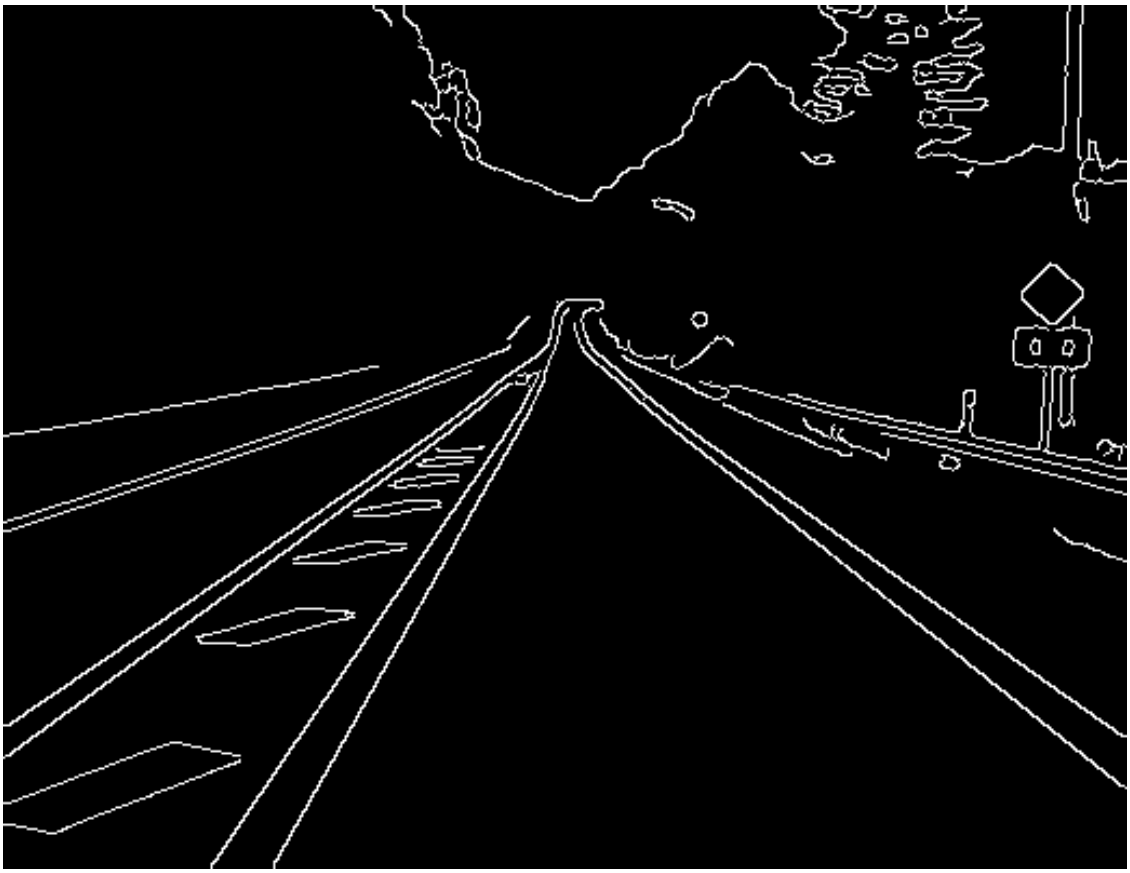


```
7 150 150
```

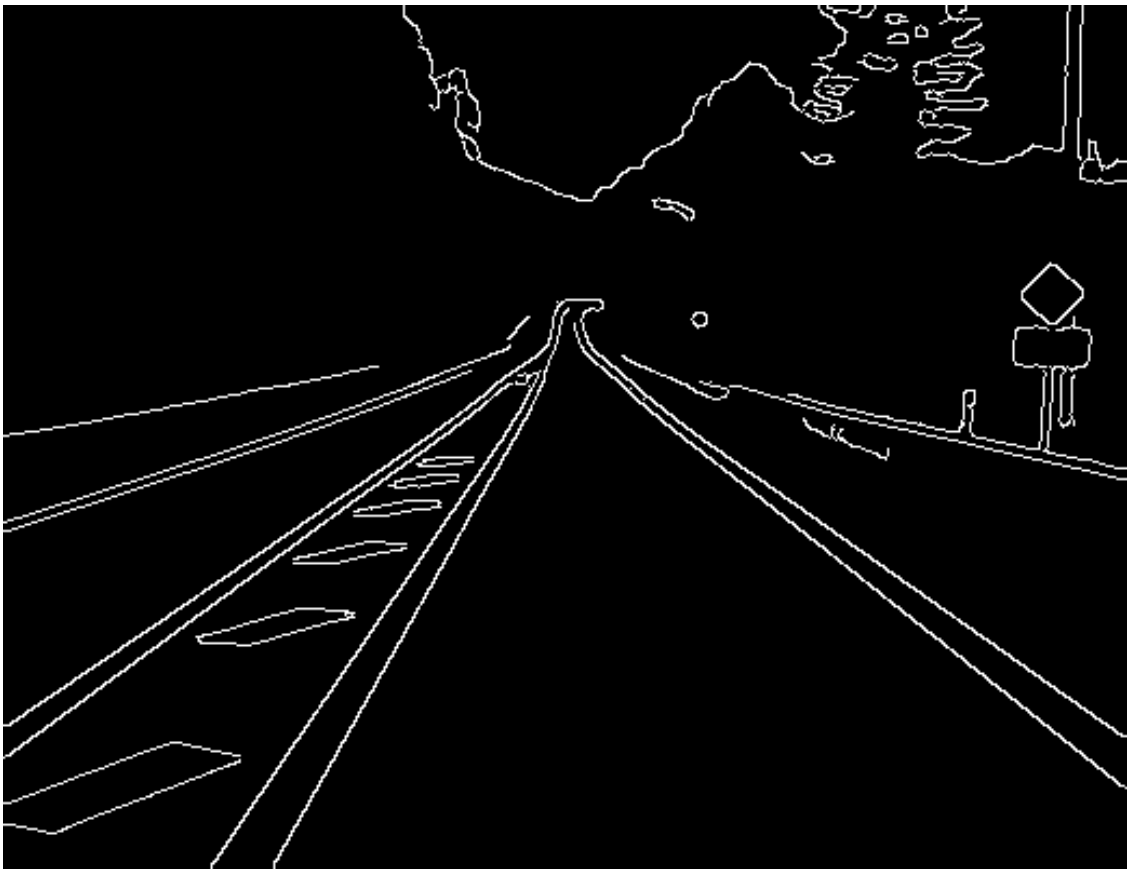
7 150 200

```
7 150 250
```

9 50 150

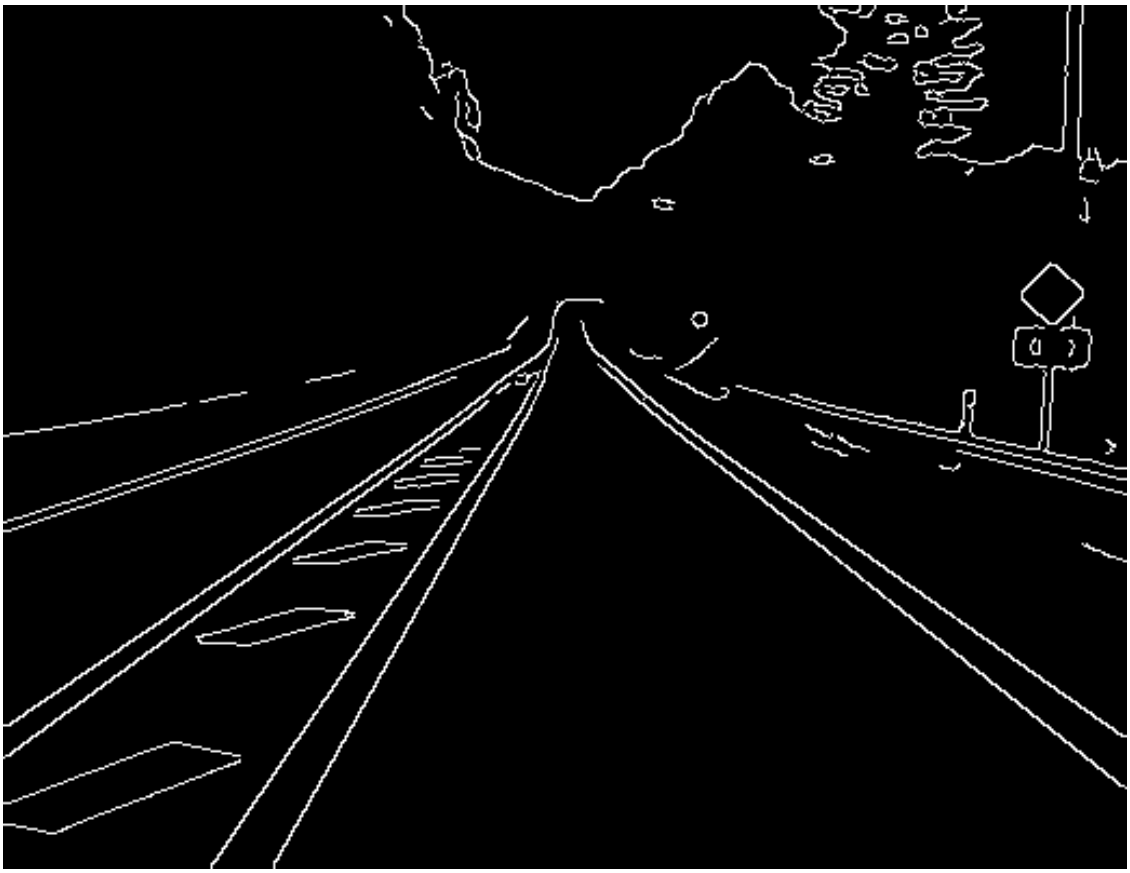


9 50 200

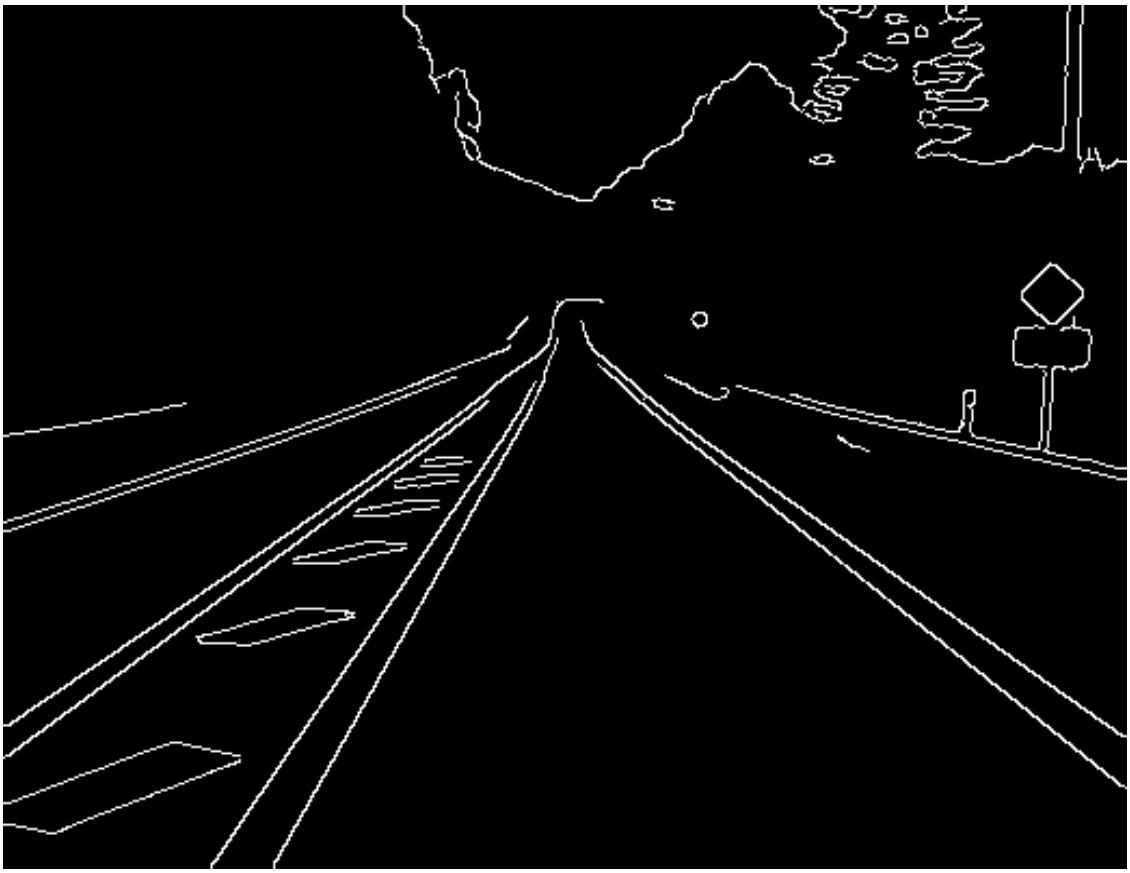


9 50 250

```
9 100 150
```



9 100 200



```
9 100 250
```

```
9 150 150
```

```
9 150 200
```

```
9 150 250
```

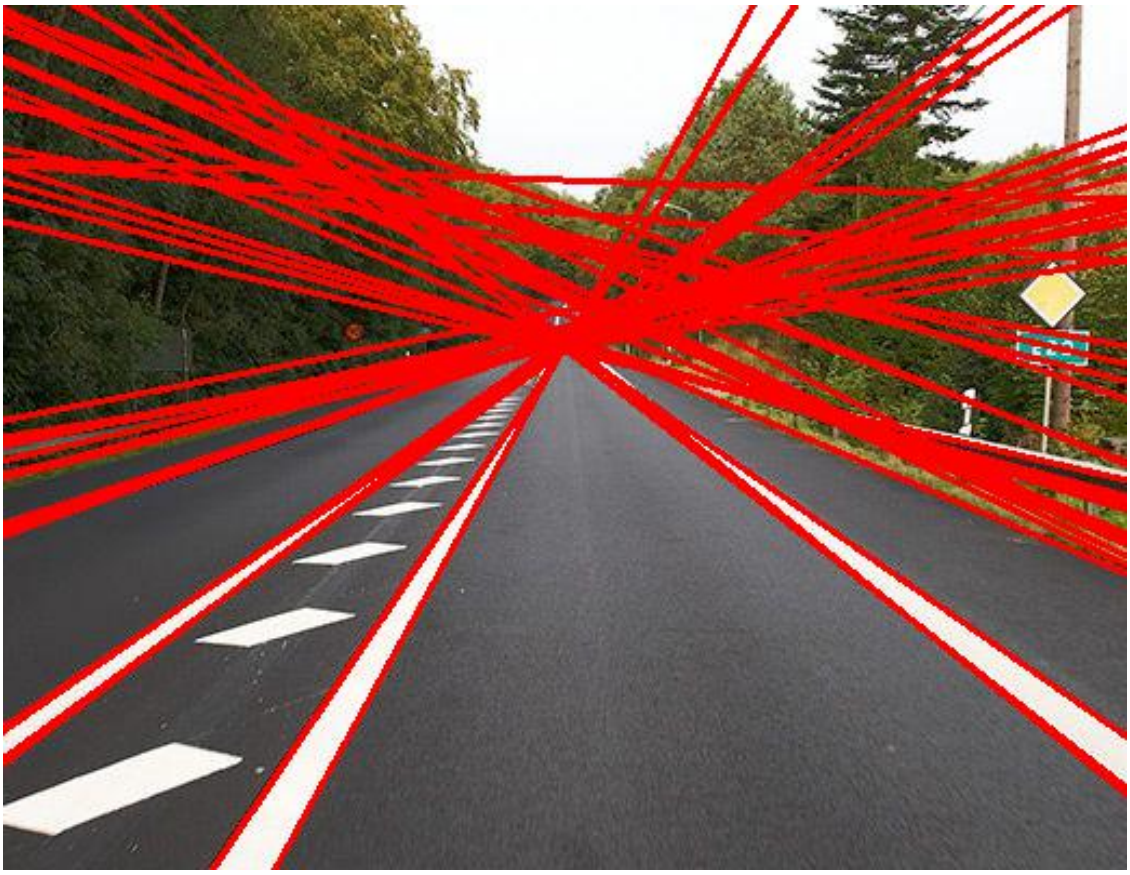
Parametry minVal i maxVal odpowiadają za dolny i górny próg przy wykonywaniu progowania krawędzi. Piksele o intensywności poniżej dolnego progu są odrzucane jako niebędące częścią krawędzi, podczas gdy piksele o intensywności powyżej górnego progu są uznawane za część krawędzi. Piksele o intensywności pomiędzy progami są uznawane za część krawędzi, gdy są połączone z pikselami które zostały już oznaczone jako krawędź.

Zbyt niskie wartości progów mogą prowadzić do wykrywania szumu jako krawędzi, natomiast zbyt wysokie wartości mogą prowadzić do pomijania niektórych krawędzi.

1.7 d) Proszę dokonać wykrywania linii metodą transformacji Hough

[ ]:
```
import numpy as np
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_1.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
for r_theta in lines:
    arr = np.array(r_theta[0], dtype=np.float64)
    r, theta = arr
    a = np.cos(theta)
    b = np.sin(theta)
```

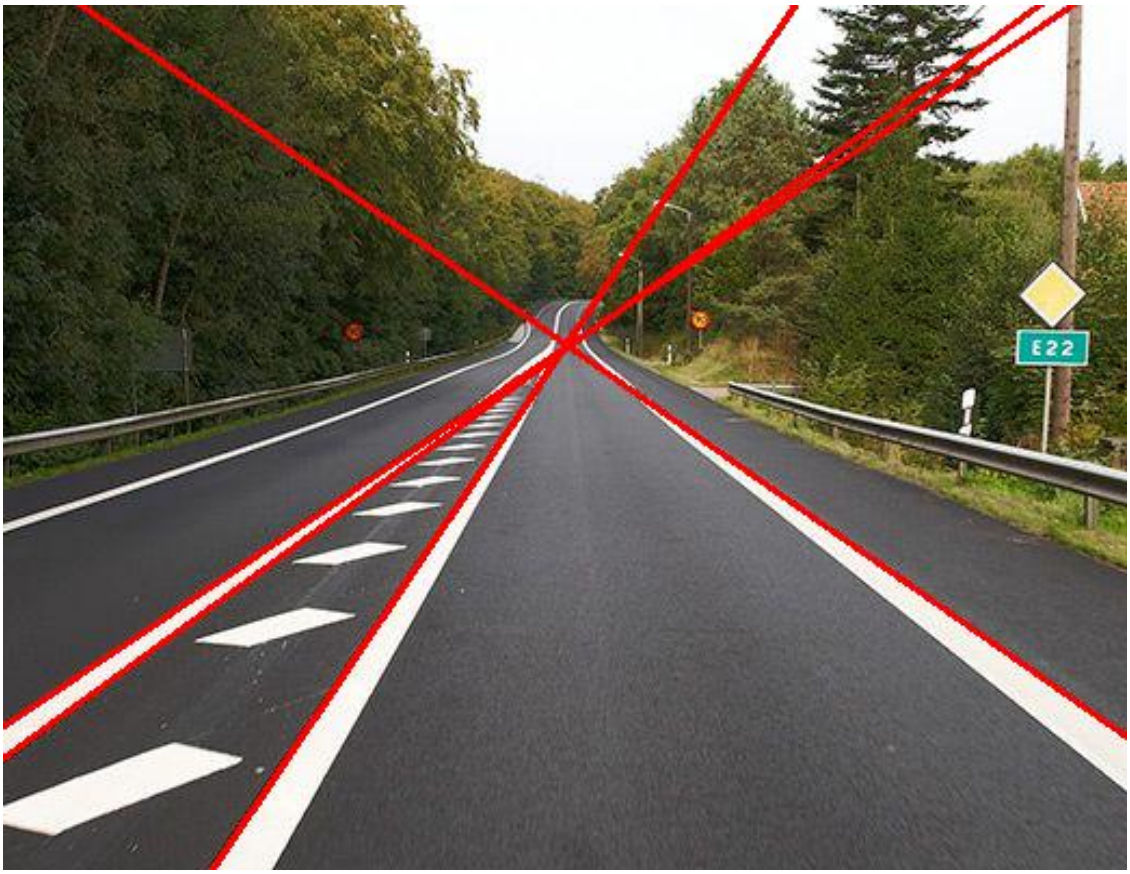
```
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2_imshow(image)
```



1.8 e) Proszę zbadać wpływ progu na wynik działania transformacji Hough

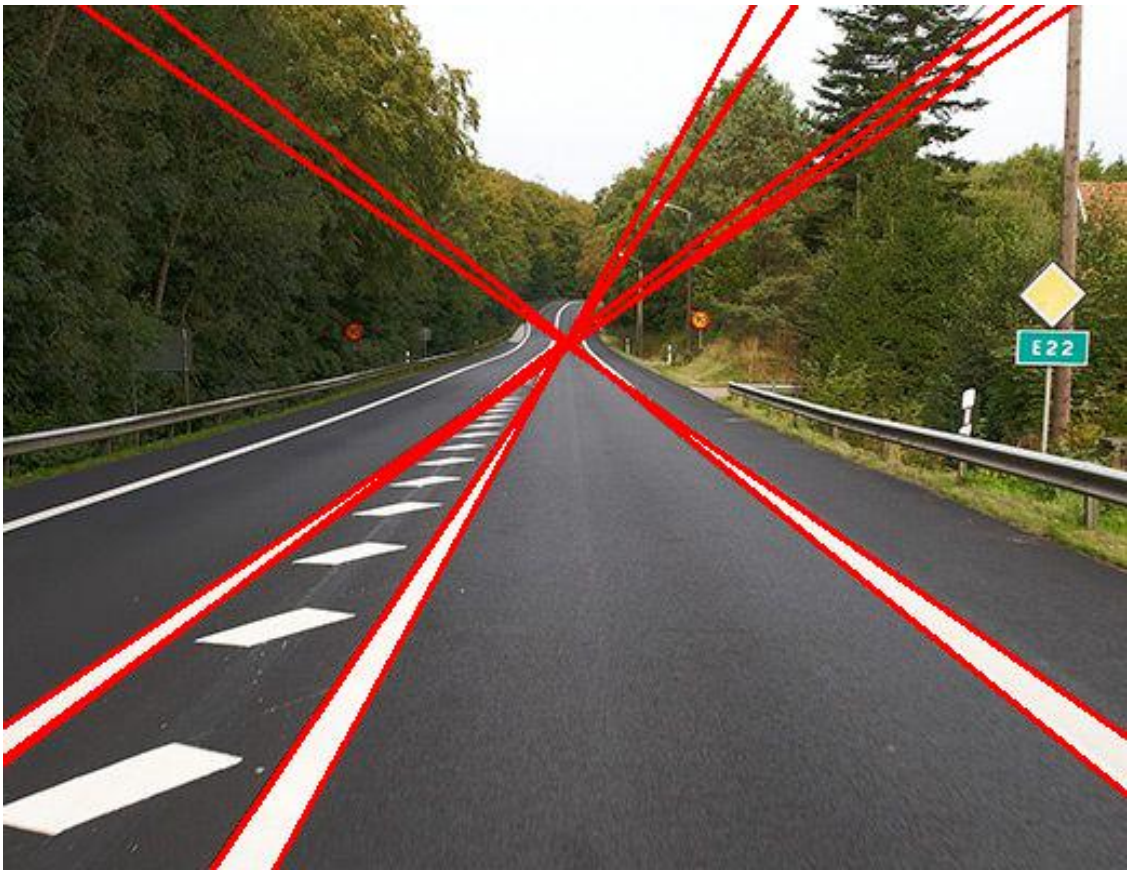
[ ]:
```
import numpy as np
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_1.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 250)
for r_theta in lines:
    arr = np.array(r_theta[0], dtype=np.float64)
    r, theta = arr
```

```
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2_imshow(image)
```



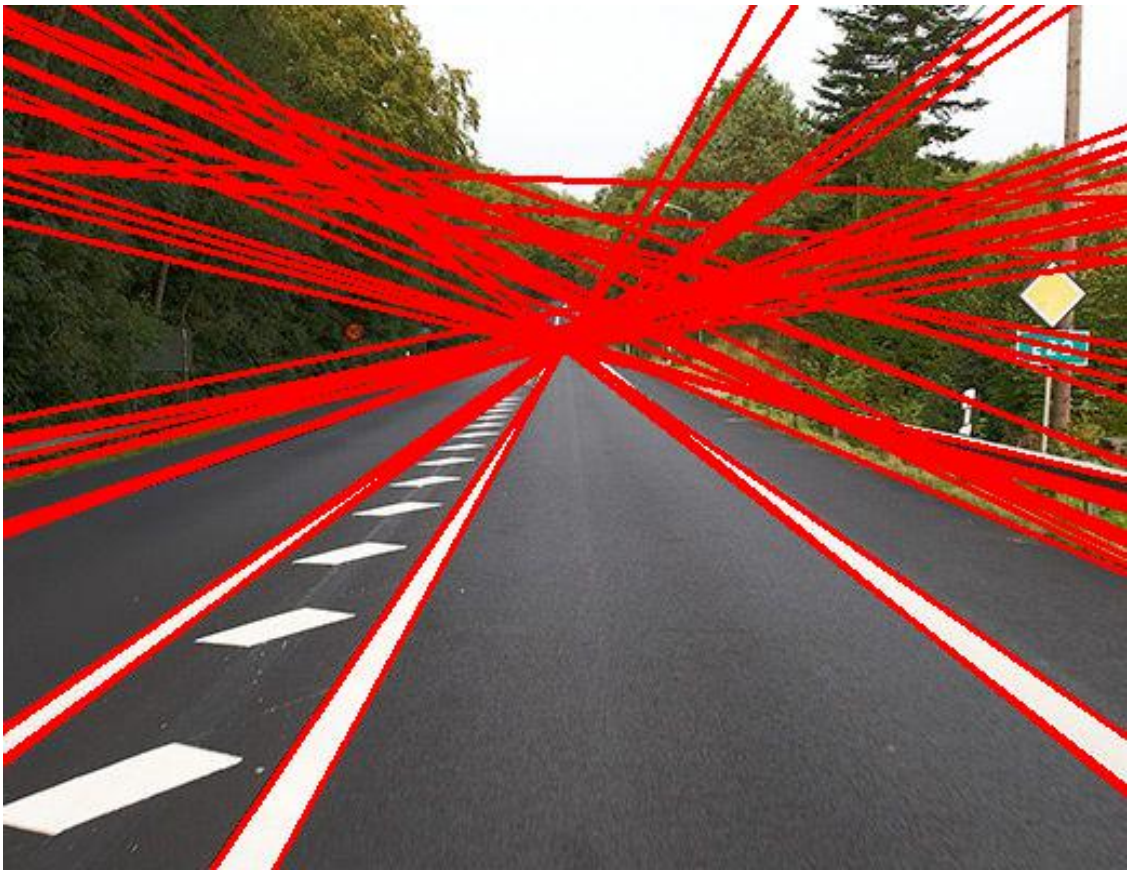
[ ]:
```
import numpy as np
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_1.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 235)
for r_theta in lines:
    arr = np.array(r_theta[0], dtype=np.float64)
    r, theta = arr
```

```
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2_imshow(image)
```



[ ]:
```
import numpy as np
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_1.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
for r_theta in lines:
    arr = np.array(r_theta[0], dtype=np.float64)
    r, theta = arr
```

```
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*r
    y0 = b*r
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2_imshow(image)
```



Zmniejszenie wartości progu powoduje, że algorytm wykrywa coraz większą ilość krawędzi, przy mniejszych wartościach często wykrywa je błędnie. Gdy próg jest zaś zbyt wysoki, większość linii zostaje pominiętych.

1.9 f) Proszę wczytać drugi z przykładowych obrazów lab5_2.png, podobnie go skonwertować i dokonać wykrywania okręgów

[ ]:
```
import numpy as np
image = cv2.imread("/content/drive/MyDrive/AiPO/lab5_2.png")
image = cv2.resize(image, dsize = None, fx = 0.5, fy = 0.5, interpolation = cv2.
 ↪INTER_NEAREST)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_copy = image.copy()

gauss = cv2.GaussianBlur(gray, (5, 5), 0, )

circles = cv2.HoughCircles(gauss, method = cv2.HOUGH_GRADIENT, dp = 1,
                           minDist = 80, param1 = 60, param2 = 21, minRadius =␣
 ↪85, maxRadius = 90)

circles = np.uint16(np.around(circles))

for circle in circles[0,:]:
    cv2.circle(image_copy, center = (circle[0], circle[1]), radius = circle[2],
                             color = (0, 255, 0),thickness = 4)
    cv2.circle(image_copy, center = (circle[0], circle[1]), radius = 2,
                             color = (0, 0, 255), thickness = 3)
cv2_imshow(image_copy)
```

2 Laboratorium 6

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np
```

2.1 Proszę wczytać przykładowy krótki film przedstawiający obraz z kamery samochodu poruszającego się po drodze i zrealizować na nim metodę wykrywania linii poprzez transformację Hough. Wynik należy zwizualizować na sekwencji wideo.

[ ]:
```
cam  = cv2.VideoCapture('/content/drive/MyDrive/AiPO/vid1.mov')
# pobranie informacji o rozmiarze klatki i ilości klatek na sekundę
width = int(cam.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cam.get(cv2.CAP_PROP_FPS))
```

[ ]:
```
# ustawienie parametrów wynikowego wideo
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
wyjscie = cv2.VideoWriter('/content/drive/MyDrive/AiPO/plik.mp4', fourcc, fps,␣
 ↪(width, height))

while(cam.isOpened()):
    # wczytanie kolejnej klatki
    ret, frame = cam.read()
    #przerwanie pętli w przypadku zakończenia filmu
    if not ret:
      cam.release()
      break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # wykrycie krawędzi w klatce
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)
    lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

    # rysowanie wykrytych linii na klatce
    if lines is not None:
        for line in lines:
            rho, theta = line[0]
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            x1 = int(x0 + 1000*(-b))
            y1 = int(y0 + 1000*(a))
            x2 = int(x0 - 1000*(-b))
            y2 = int(y0 - 1000*(a))
            cv2.line(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

    #Zapis klatki do pliku wyjściowego
    wyjscie.write(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

```
cam.release()
cv2.destroyAllWindows()
wyjscie.release()
```

2.2 Wczytać drugi film, przedstawiający nagranie samochodów na autostradzie. Przenieść do odcieni szarości. Stworzyć sekwencję różnic pomiędzy kolejnymi klatkami. Utworzone różnice wykorzystać jako maski na oryginalnych klatkach.

[ ]:
```
cap  = cv2.VideoCapture('/content/drive/MyDrive/AiPO/vid2.mov')
# pobranie informacji o rozmiarze klatki i ilości klatek na sekundę
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
```

[ ]:
```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
frame_size = (width, height)

out = cv2.VideoWriter('/content/drive/MyDrive/AiPO/plik2.avi', fourcc, fps,␣
 ↪frame_size)

while(cap.isOpened()):
    # wczytanie kolejnej klatki
    ret, frame = cap.read()

    #przerwanie pętli w przypadku zakończenia filmu
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    gray = cv2.resize(gray, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

    # wygenerowanie maski na podstawie różnicy między kolejnymi klatkami
    if 'prev_frame' in locals():
        diff = cv2.absdiff(gray, prev_frame)
        _, mask = cv2.threshold(diff, 100, 255, cv2.THRESH_BINARY+ cv2.
 ↪THRESH_OTSU)
        mask = cv2.dilate(mask, None, iterations=2)
        mask = cv2.resize(mask, frame_size[::-1])
    else:
        mask = np.zeros_like(gray)

    #nałożenie maski na oryginalną klatkę
```

```
    masked_frame = cv2.bitwise_and(frame, frame, mask=cv2.resize(mask, (frame.
 ↪shape[1], frame.shape[0])))

    out.write(masked_frame)

    # aktualizacja poprzedniej klatki
    prev_frame = gray

cap.release()
out.release()
cv2.destroyAllWindows()
```

3 Laboratorium 7

3.1 A. Proszę porównać metody Harrisa, SIFT, SURF, FAST i ORB w zadaniu rozpoznawania punktów charakterystycznych dla obrazu lab5_2.png. Dla której metody wyniki wydają się bliższe intuicji? Która wydaje się być bardziej „aplikowalna" (np. ze względu na czas obliczeń)?

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/drive/MyDrive/AiPO/lab5_2.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

[ ]:
```
import numpy as np
# Metoda Harrisa
harris = cv2.cornerHarris(gray, 7, 9, 0.07)
keypoints_harris = np.argwhere(harris > 0.01 * harris.max())
keypoints_harris = [cv2.KeyPoint(float(x[1]), float(x[0]), 2) for x in␣
 ↪keypoints_harris]


img_harris = cv2.drawKeypoints(img.copy(), keypoints_harris, None, color=(0,␣
 ↪255, 0), flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
for point in keypoints_harris:
    cv2.circle(img_harris, (int(point.pt[0]), int(point.pt[1])), 3, (0, 0,␣
 ↪255), thickness=-1)
```

```
cv2_imshow(img_harris)
```

[ ]:
```
# Metoda SIFT
sift = cv2.xfeatures2d.SIFT_create()
keypoints_sift = sift.detect(gray, None)

img_sift = img.copy()

cv2.drawKeypoints(img_sift, keypoints_sift, img_sift, color=(0,255,0))

cv2_imshow(img_sift)
```

[ ]:
```
# Metoda FAST
fast = cv2.FastFeatureDetector_create()
keypoints_fast = fast.detect(gray, None)


img_fast = img.copy()
cv2.drawKeypoints(img_fast, keypoints_fast, img_fast, color=(0,255,0))

cv2_imshow(img_fast)
```

[ ]:
```
# Metoda ORB
orb = cv2.ORB_create()
keypoints_orb, descriptors = orb.detectAndCompute(gray, None)
img_orb = img.copy()

cv2.drawKeypoints(img_orb, keypoints_orb, img_orb, color=(0,255,0))
cv2_imshow(img_orb)
```

1. Metoda Harrisa opiera się na wykrywaniu krawędzi w obrazie i analizie zmiany intensywności wokół krawędzi. Metoda ta jest szybka, ale może mieć trudności z wykrywaniem punktów charakterystycznych na obrazach o dużym szumie.
2. SIFT to metoda wykrywania punktów charakterystycznych, która działa na wielu skalach. Ta metoda jest odporna na zmiany skali i rotacje, ale może być kosztowna obliczeniowo.
3. FAST to szybka metoda wykrywania punktów charakterystycznych polegająca na wykrywaniu pikseli, które mają wartości intensywności znacznie różniące się od ich sąsiadów. Metoda powinna być skuteczna na obrazach o niskim szumie.
4. ORB to połączenie algorytmów FAST i BRIEF. Umożliwia szybkie i skuteczne wykrywanie punktów charakterystycznych. ORB może działać na wielu skalach i jest odporny na zmiany rotacji.

Patrząc na uzyskane wyniki, najlepiej sprawdziła się metoda Harrisa (pod względem wizualnym). Zaznaczyła ona najlepiej charakterystyczne punkty. Miała jednak problem z łączeniami obrazków. Metody FAST i SIFT zaznaczyły zdecydowanie zbyt dużo punktów. Może to być spowodowane zaszumieniem obrazka w kilku miejscach. ORB dla kilku części obrazka poradziło sobie dobrze, jednak dla większości obrazka nie potrafiło znaleźć punktów szczególnych.

3.2 B. Proszę spróbować wykryć twarz w sekwencji wideo – nagranej na kamerze lub ściągniętej z serwisu youtube. W tym celu korzystamy z kaskady Haara:

[ ]:
```
!pip install pytube
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting pytube
  Downloading pytube-12.1.3-py3-none-any.whl (57 kB)
                             57.2/57.2 kB
3.6 MB/s eta 0:00:00
Installing collected packages: pytube
Successfully installed pytube-12.1.3
```

[ ]:
```
from pytube import YouTube

yt = YouTube('https://www.youtube.com/watch?v=ZOdT17E8PZI')
stream = yt.streams.filter(file_extension='mp4', res='720p').first()
stream.download()

# Wczytanie wideo i kaskady Haara
cap = cv2.VideoCapture('/content/Ronaldo Siuuu.mp4')
face_cascade = cv2.CascadeClassifier('/content/drive/MyDrive/AiPO/
 ↪haarcascade_frontalface_default.xml')

# Odczytanie parametrów wideo
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Utworzenie obiektu do zapisu wideo
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output.mp4', fourcc, fps, (width, height))


# Wykrywanie twarzy w sekwencji wideo
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    # Zakreślenie twarzy ramką
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    out.write(frame)


cap.release()
out.release()
cv2.destroyAllWindows()
```