High-level feature detectie (practicum 1)

In deze opdracht gaan we een plaatje van de A2 inladen en bekijken of we automatisch de weg en verkeersborden kunnen detecteren. Hiervoor maken we gebruik van het basisscript zoals we dat in week 1 ook gebruikt hebben, alleen laden we nu **`snelwegA2.png`** in. Ook hebben we alvast de **`numpy`** module geïmporteerd voor onze wiskundige berekeningen:

```
import cv2 as cv
import numpy as np

#Leest het plaatje van file
#LET OP: geef het juiste pad op naar het plaatje
file ='snelwegA2.png';

#laten we vast wat kleuren definiëren om
#straks de featues te tekenen.
kleur_rood = [0,0,255]
kleur_groen = [0,255,0]
kleur_blauw = [255,0,0]

img = cv.imread(file)

#test of het plaatje goed is ingelezen
assert img is not None

cv.imshow("origineel", img)

#HIER KOMT ONZE FEATURE DETECTIE CODE


#het programma stopt als je de 'q' intyped
while True:
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```

Als we het script runnen, zien we het plaatje hier rechts. Op het plaatje zien we dat de wegen links en rechts van de auto beginnen en in een rechte lijn voor ons uit strekken. Uit ervaring weten we dat de weg altijd zo eruitziet. De verkeersborden voor de snelheid, zijn cirkels met cijfers erin en deze zullen altijd hetzelfde formaat hebben.

Gelukkig voor ons is er een zeer goede en gemakkelijke manier om lijnen en cirkels in een plaatje te vinden, de Hough transform[1] en wordt al sinds 1972(!) gebruikt.

Laten we beginnen met het zoeken van de **rechte lijnen**.

1. De eerste stappen zijn verbeteringen van het beeld:
    a. Converteren naar grijswaarden
    b. Detecteren van alle randen door middel van Canny Edge Detectie:

    We gebruiken bewust een nieuw plaatje **`gray`** en een plaatje **`edges`**, omdat we:
    - het originele plaatje later willen gebruiken om de gevonden features te tekenen
    - het gray plaatje ook gaan gebruiken voor de cirkel detectie.

[1] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, Vol. 15, pp. 11–15 (January, 1972)

De drempelwaardes zijn gezet zodat er niet teveel kleine, zwakke edges achterblijven. Je kan met deze 100 en 400 spelen om een beetje een gevoel te krijgen, wat de invloed op deze waardes zijn op de lijn detectie.

```
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray, 100, 400, apertureSize = 3)
cv.imshow("edges", edges)
```

2. Bekijk het volgende youtube filmpje: https://www.youtube.com/watch?v=4zHbI-fFIlI . Thales Sehn Körting legt daar in 6 minuten heel goed uit hoe een Hough Transform rechte lijnen kan vinden. De wiskunde hoef je niet te begrijpen, maar de namen van de variabelen zijn van belang om de lijnen straks goed te kunnen vinden

3. Laten we dit algoritme dan maar implementeren:

```
lijnen = cv.HoughLines(edges, 1, np.pi/180, 150)
```

Het **cv.Houghlines** commando, neemt als input de gevonden **edges** van de Canny Edge detectie. 1 is de initiële waarde voor ρ (rho) en np.pi/180 is de initiële waarde voor θ (theta). Deze zijn niet echt belangrijker, de 150 zijn veel belangrijker, deze geven aan hoeveel punten op dezelfde lijn moeten liggen om als een lijn te worden gedetecteerd.

4. De lijnen die we terugkrijgen bestaat uit een 3D array, waarbij voor elke gedetecteerde lijn de ρ en θ zijn opgeslagen. Deze kunnen we terugrekenen naar de coördinaten in het x,y coördinaten stelsel. Wil je de wiskunde begrijpen, kijk dan naar deze video's.

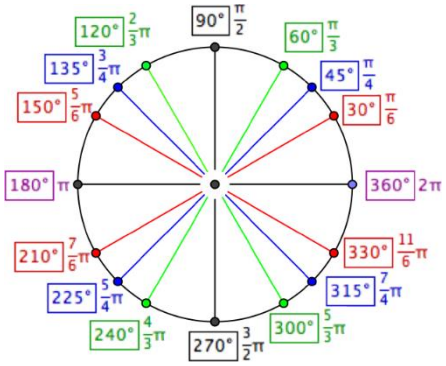
```
nlijnen, punt, nDimenties = lijnen.shape
for i in range(nlijnen):
    rho, theta = lijnen[i][0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(img, (x1,y1), (x2,y2) ,kleur_rood , 2)

cv.imshow('resultaat', img)
```

Uiteindelijk krijgen we voor elke ρ en θ een begin en eind coördinaat (welke ver buiten ons plaatje liggen, het gaat ons om de lijn).
cv.line tekent vervolgens een rode lijn op het **img** met lijndikte 2 .

Laten we ons script runnen en kijken wat er gebeurd.

5. Oei, dat waren wel veel lijnen. Speel nu eens met de drempelwaarde 150 van het **cv.Houghlines** commando. Kun je enkele niet interessante lijnen wegtoveren?

6. Er zijn ook nog andere manieren om dit te doen. We hebben namelijk nog voorkennis over de lijnen. We weten dat ze altijd in de zelfde richting lopen. We hoeven dus alleen maar aan het **`cv.Houghlines`** commando te vertellen dat we geïnteresseerd zijn in bepaalde hoeken van θ!



Hieronder zie je een cirkel waarin lijnen in een bepaalde richting lopen. De bijbehorende pi (π) is aangegeven en kunnen we gebruiken om de waarde van θ te bepalen.
Omdat in ons plaatje een lijn van linksonder naar rechtsboven hetzelfde is als een lijn van rechtsboven naar linksonder, gebruiken wij alleen de rechterkant van de cirkel (de waardes van 0 tot π en niet van π tot 2π)

Het lijkt erop dat we voor de lijnen aan de rechterkant van de weg onze θ kunnen limiteren van 2/3 π tot 5/6 π. Aan de linkerkant lijkt 1/3 π tot 2/5 π een goede gok.
Zet je eerste lijn detectie in het commentaar en vervang deze door:

```
#lijnen = cv.HoughLines(edges, 1, np.pi/180,150)
lijnenRechts = cv.HoughLines(edges, 1, np.pi/180, 150,
                        min_theta=2/3*np.pi,max_theta=5/6*np.pi)
lijnenLinks = cv.HoughLines(edges, 1, np.pi/180, 300,
                        min_theta=1/3*np.pi,max_theta=2/5*np.pi)
lijnen = np.concatenate((lijnenRechts, lijnenLinks), axis = 0)
```

De np.concatenate plakt de gevonden matrices aan elkaar zodat we de code bij punt 4 niet hoeven te veranderen.

Laten we beginnen met het zoeken van de **cirkels**.

1. De eerste stappen zijn verbeteringen van het beeld zijn hetzelfde als de rechte lijnen, we pakken nu dus meteen het beeld **`edges`** om onze cirkels op te detecteren.
2. Bekijk het volgende youtube filmpje: https://www.youtube.com/watch?v=Ltqt24SQQoI Thales Sehn Körting legt daar in 5 minuten heel goed uit hoe een Hough Transform cirkels kan vinden.
3. Dan gaan we nu openCv aan het werk zetten:

```
circles = cv.HoughCircles(edges, cv.HOUGH_GRADIENT, 1, minDist=20,
          param1=50,param2=30,minRadius=5,maxRadius=20)
```

De enige parameters die interessant zijn hier is de
- **`minDist`**, die zegt dat we een volgende circel pas op 20 pixels van de huidige cirkel mogen tellen.
- **`minRadius`** en **`maxRadius`**. Daarin gegeven we aan hoe groot de cirkel is die we zoeken (in pixels).

4. Dat gaan we ook weer op het scherm tekenen. Daarvoor moeten we eerst heel vervelend, de gevonden cirkels uit een lijst van 1 element halen en de getallen omzetten naar integers:

```
# circles is een lijst met 1 element:
# Dit ene lijst element is ook weer lijst met punten.
# Elk punt bestaat uit center_x, center_y, radius - in floats.
# haal deze lijst uit de enkele lijst en maak er mooie integers van:
circles = np.uint16(np.around(circles))[0,:]
```

```
for centerx, centery, radius in circles:
    # teken de buitenste rand van de circel, met lijndikte 2
    cv.circle(img,(centerx,centery),radius,kleur_groen,2)

    # teken nu de binnenkant als punt
    #(radius=2pixels groot, lijndikte 3)
    cv.circle(img,(centerx,centery),2,kleur_blauw,3)
```

5. Nu kunnen we eindelijk gaan tekenen en ons eindresultaat bewonderen:

```
cv.imshow('resultaat', img)

cv.imwrite("snelweg_output.jpg", img);
```