
# Computer Vision
# Assignment 1

**Students:**

Mennatullah Ibrahim Mahmoud - 6221
Nour El-din Hazem Abdelsalam - 6261
Anas Ahmed Bekheit - 6659

## Part 1: Cartoonifying

First step is applying the median filter to reduce noise. Then, we apply the laplacian filter and use thresholding to detect edges. We then pass a bilateral filter to flatten the flat spaces without blurring the edges.

**Code:**

```
[67] def cartoonify(img):
       gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
       smooth_image = cv2.medianBlur(gray_image,7)
       edge_image = cv2.Laplacian(smooth_image,-1,ksize=5)
       _, thresh1 = cv2.threshold(edge_image, 120, 255, cv2.THRESH_BINARY_INV)
       bilateral = cv2.bilateralFilter(img, d= 9, sigmaColor= 75, sigmaSpace= 42)
       bilateral[thresh1==0] = 0
       return bilateral
```

**Results:**

**Part 2: Lane Detection**

First step is edge detection using Canny edge detector. The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986.

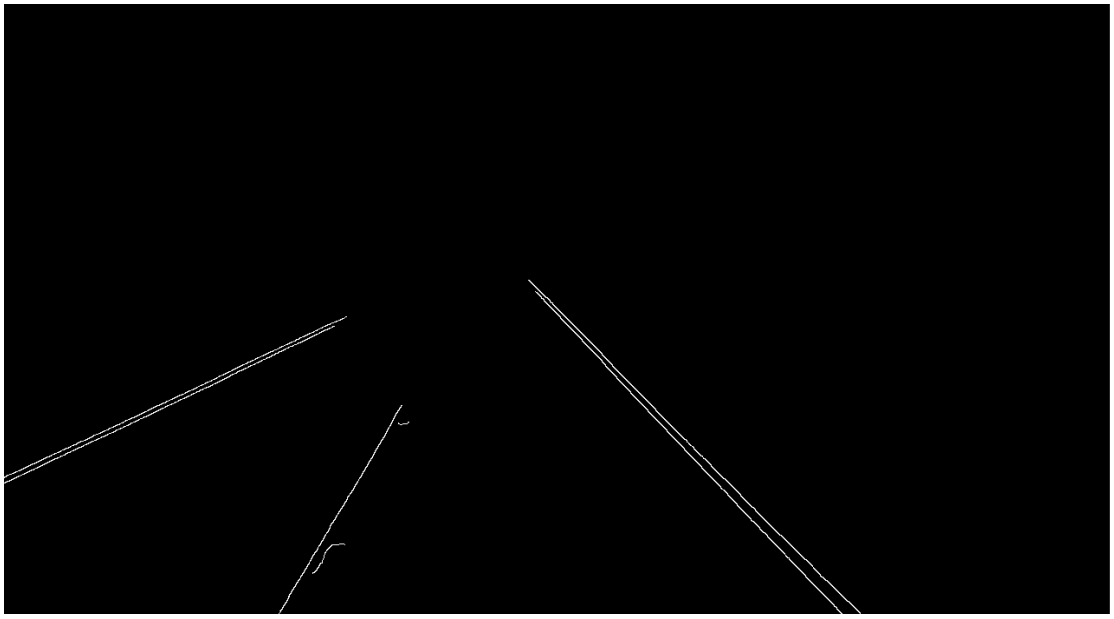
Next step is to focus on the region of interest in the photo which is the lane this is done by masking the parts outside of the road.

**Code:**

```
[333] def edge_detection(img):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        smooth_image = cv2.medianBlur(img,7)
        edge_img = cv2.Canny(smooth_image,100,200)
        return edge_img

[334] def roi(img):
        img[0:320,:]=0
        img[:,1000:]=0
        return img
```

**Results:**



Next step is performing hough transform. In each a non-zero point in the edge image we calculate the voting of its rows with theta from 0 to 180 and we fill the accumulator array.

**Code:**

```
def hough_transform(edge_img):
  non_zeros = np.nonzero(edge_img)
  dict_hough = {}
  diagonal = math.ceil(math.sqrt(edge_img.shape[0]**2 + edge_img.shape[1]**2))
  for x,y in zip(non_zeros[0],non_zeros[1]):
    for theta in range(0,180):
      row = round(x*math.cos(theta*math.pi/180) + y*math.sin(theta*math.pi/180))+diagonal
      if(row,theta) in dict_hough.keys():
        dict_hough[(row,theta)]=dict_hough[(row,theta)]+1
      else:
        dict_hough[(row,theta)]=1
  return dict_hough
```

```
[488] diagonal = math.ceil(math.sqrt(edge_img.shape[0]**2 + edge_img.shape[1]**2))
      accumulated_array = np.zeros(shape=(diagonal*2,180))
```

```
[489] for element in dict_hough.keys():
        accumulated_array[element[0]][element[1]] = dict_hough[element]
```

Using a threshold = 50, we ignore the lines with less than 50 votes

**Code:**

```
[490] print(len(dict_hough.keys()))
      print(len(np.nonzero(accumulated_array)[0]))

      111468
      111468

[491] accumulated_array[accumulated_array<50]=0
      print(len(np.nonzero(accumulated_array)[0]))

      148
```

Next step is the Non-Maxima Suppression to suppress the duplicate lines due to the noise in the edge image

**Code:**

**Non Maxima Suppression**

```
[546] #range = 100 x 10
      #non-maximum supression
      i = 0
      j = 0
      shift_in_i = 100
      shift_in_j = 10
      m1 = accumulated_array.shape[0]
      m2 = accumulated_array.shape[1]
      counter=0
      while(i<m1):
        j=0
        while(j<m2):
          max = np.max(accumulated_array[i:min(i+shift_in_i,m1),j:min(j+shift_in_j,m2)])
          max_coordinates = np.where(accumulated_array[i:min(i+shift_in_i,m1),j:min(j+shift_in_j,m2)]==max)
          accumulated_array[i:min(i+shift_in_i,m1),j:min(j+shift_in_j,m2)] = 0
          accumulated_array[i:min(i+shift_in_i,m1),j:min(j+shift_in_j,m2)][max_coordinates[0][0],max_coordinates[1][0]] = max
          j = j+shift_in_j
        i += shift_in_i

[542] len(np.nonzero(accumulated_array)[0])

      6
```

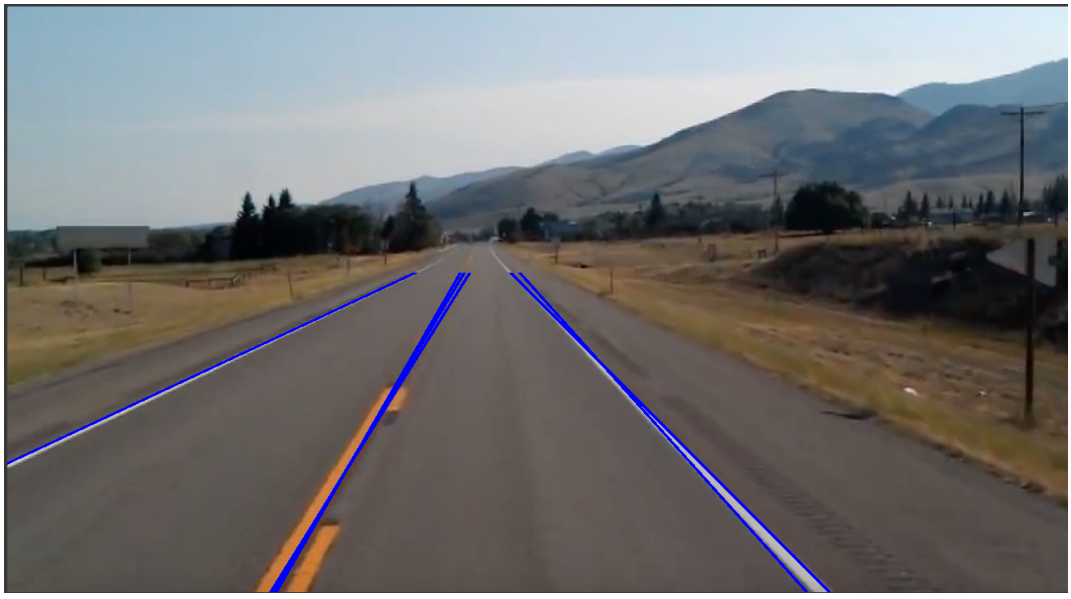
Finally we use the ro and theta to draw the final lines on the original image. Using the Region of Interest mask, we also mask the extended lines outside the lane.

## Code:

```
[543] def draw_lines(img):
        nonzeros_indices = np.nonzero(accumulated_array)
        for row,theta in zip(nonzeros_indices[0],nonzeros_indices[1]):
          row = row - diagonal
          theta = theta * math.pi / 180
          sine = math.sin(theta)
          cosine = math.cos(theta)
          xo = row * cosine
          yo = row * sine
          x1 = round(xo - 2000 * sine)
          y1 = round(yo + 2000 * cosine)
          x2 = round(xo + 2000 * sine)
          y2 = round(yo - 2000 * cosine)
          img = cv2.line(img, (y1,x1), (y2,x2), (255, 0, 0) , 2)
        return img
```

```
[547] road_image = cv2.imread("road.jpg")
      lined_image = draw_lines(road_image)
      final_image = cv2.imread("road.jpg")
      lined_image[0:320,:] = final_image[0:320,:]
```

## Results:



Link:
https://colab.research.google.com/drive/17VFQ9ZJdkAvzbNFjPyCPUsytn19VPBq2#scrollTo=li9v2B1MQF_G