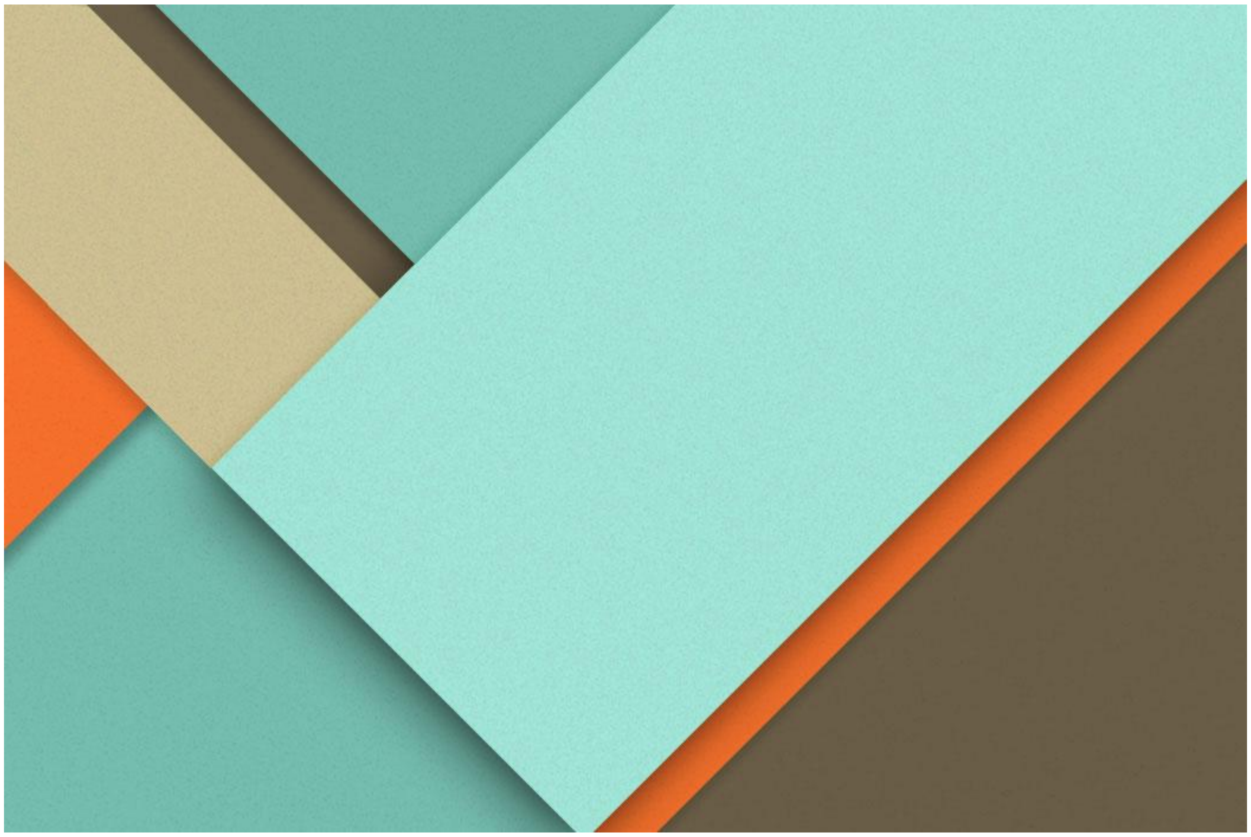


Computer Vision

Assignment 1 – Cartoonify And Road Lane Detection

—

Abdallah Yasser Ibrahim – 18015026

Ahmed Bahgat Elsherif – 18010078

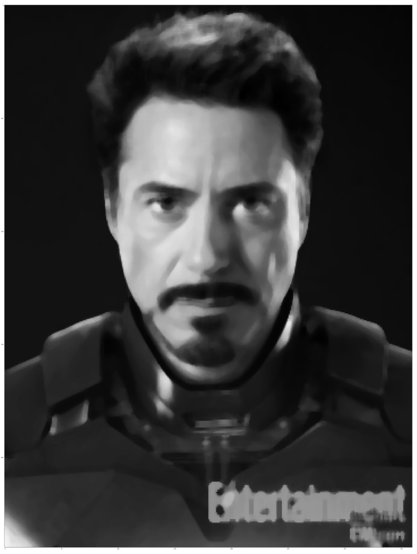
Cartoonify

We used this image to apply the cartoonify effect:

We want to get two images:

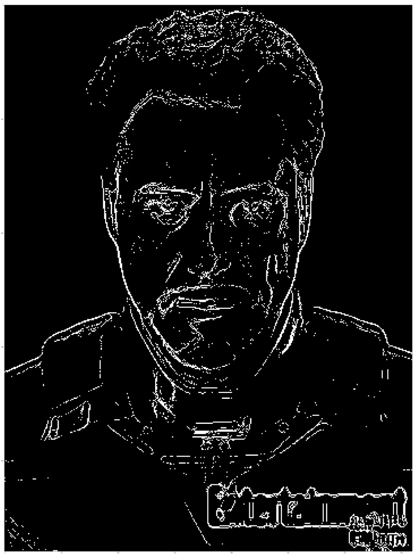
- The black outline. We get this by using laplacian edge detection.
- The colored painting. We get this by applying a bilateral filter that does smoothing and also preserves the edges.

Convert to Grayscale and Remove Noise



The first step towards edge detection is to convert to grayscale and remove the noise using the median blur (which also helps remove weak edges):

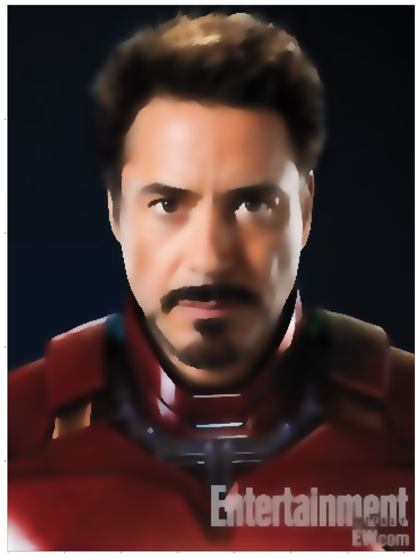
Edge Detection using Laplacian Filter



Now that we have an image with all pixel values between 0-255, we can use the laplacian filter to find the edges. It creates zero crossing at the edge locations. We then apply a threshold of value 10 so that all values are either 0 or 255.

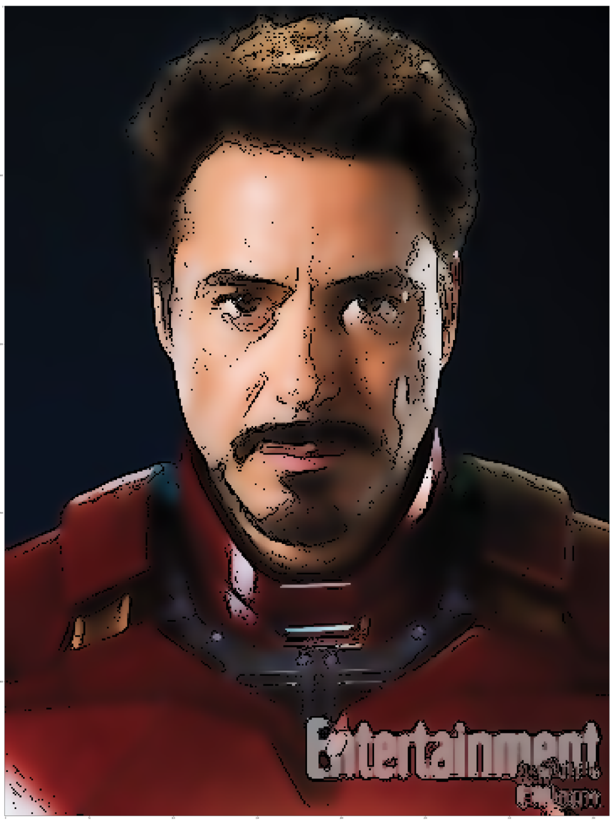
Generating a Color Painting

We want to smooth the colors to make it look like a painting, which has more solid colors and less details. We use the bilateral filter as it preserves the edges. Instead of using a large bilateral filter (which would be slow), we use a smaller filter (9*9) three times to generate a similar effect. We chose the sigma parameter equal 100, which is high and creates a better cartoon effect.



Overlaying the Edges on the Color Painting

The final step is to combine the two images. We want to add the black edges on the color image. We do this by starting with a black background and copying the values from the color painting if that position doesn't have an edge.



Road Lane Detection

The goal is to detect the lanes on roads. We represent the lane as a straight line and use the **Hough transform** to find these lines.

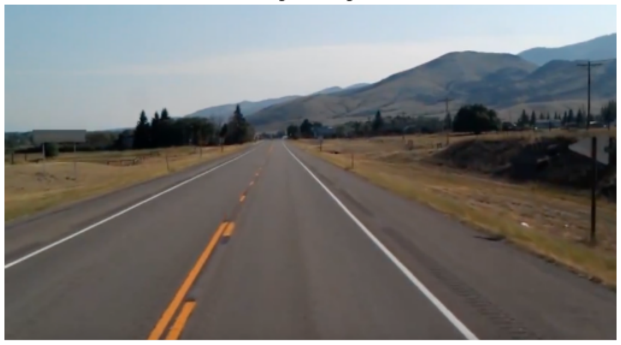
Smoothing Grayscale Image

The first step is to convert to grayscale and use the median smoothing filter to remove noise while preserving edge structure.

Edge Detection

To find the lines, we need to know where the edge pixels are. We used **Canny edge detection algorithm** with a **low threshold of 100** and a **high threshold of 150** which is relatively high. The thresholds were chosen empirically.

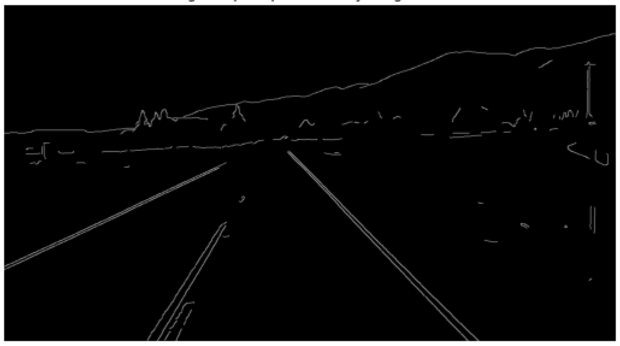
Original Image



Grayscale image

Median-blurred image

Edge map output of Canny's algorithm



Region of Interest

We define the region of interest as a polygon with manually selected vertices. Only points inside this region of the image are considered for the hough transform algorithm.

```
# define ROI
coords = np.array([(0, 700), (0, 510), (350, 360), (670, 340), (1100, 700)])
```

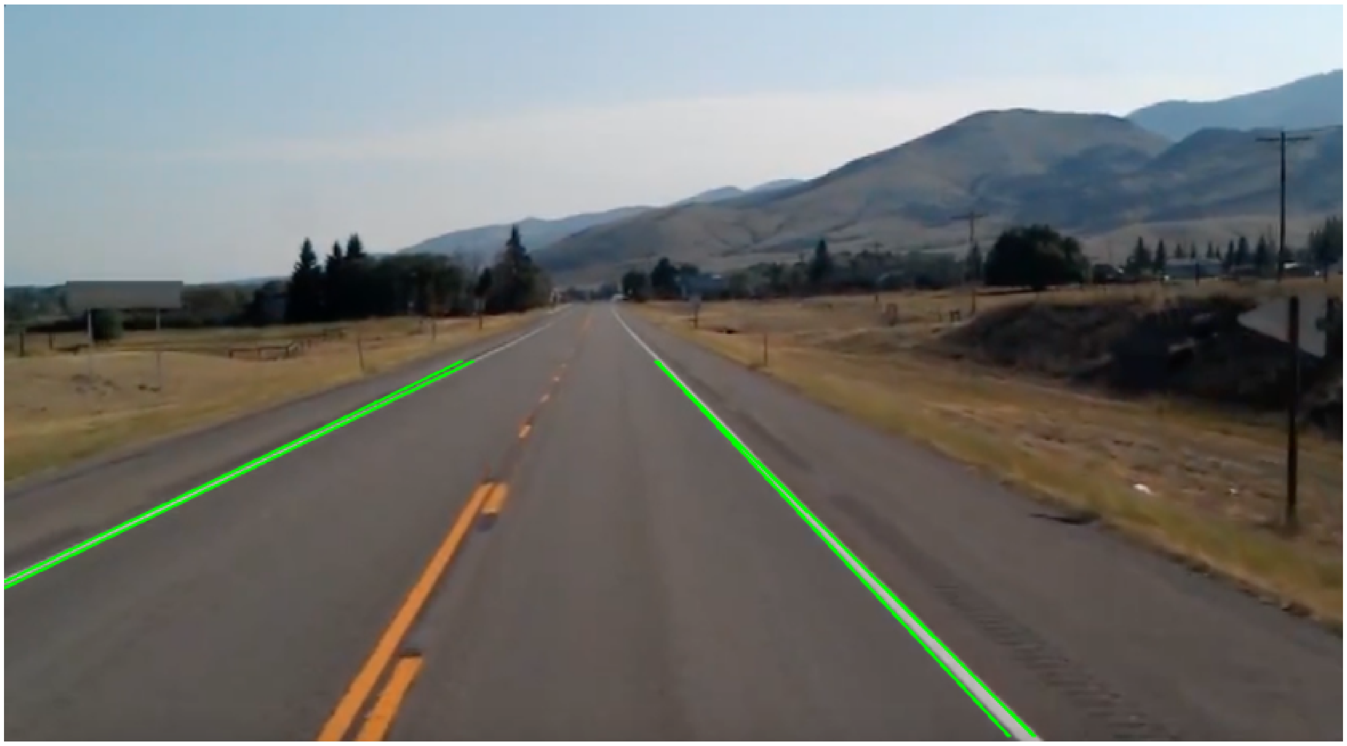
Hough Transform

We used binning when iterating over the angles θ. By default, bin width is set to 1°.

We also applied non-maximum suppression + a threshold of 50 on the values of the accumulator array to select only the strongest lines.

Results

Lines detected by Hough transform



Lines detected by Hough transform