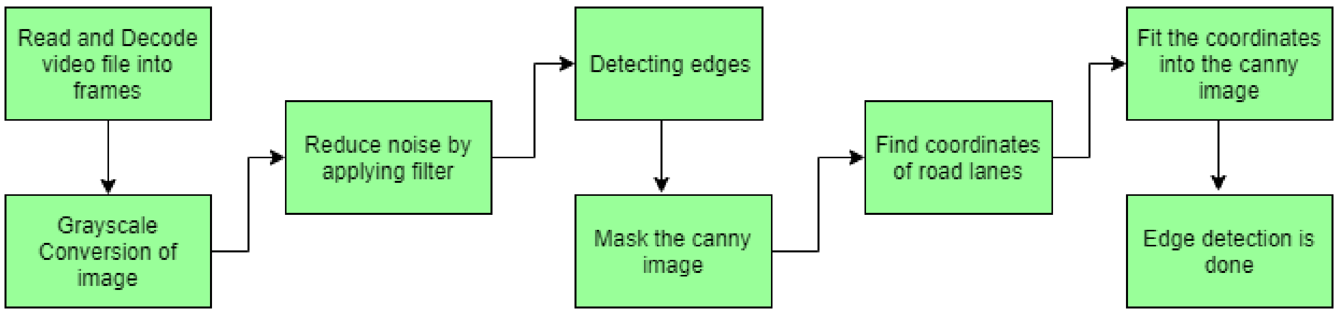
OpenCV | Real Time Road Lane Detection

Dolly_Vaishnav

Autonomous Driving Car is one of the most disruptive innovations in AI. Fuelled by Deep Learning algorithms, they are continuously driving our society forward and creating new opportunities in the mobility sector. An autonomous car can go anywhere a traditional car can go and does everything that an experienced human driver does. But it's very essential to train it properly. One of the many steps involved during the training of an autonomous driving car is lane detection, which is the preliminary step. Today, we are going to learn how to perform lane detection using videos.

Brief steps involved in Road Lane Detection

Road Lane Detection requires to detection of the path of self-driving cars and avoiding the risk of entering other lanes. Lane recognition algorithms reliably identify the location and borders of the lanes by analyzing the visual input. Advanced driver assistance systems (ADAS) and autonomous vehicle systems both heavily rely on them. Today we will be talking about one of these lane detection algorithms. The steps involved are:


Read and Decode video file into frames
Grayscale Conversion of image
Reduce noise by applying filter
Detecting edges
Mask the canny image
Find coordinates of road lanes
Fit the coordinates into the canny image
Edge detection is done

- **Capturing and decoding video file:** We will capture the video using VideoFileClip object and after the capturing has been initialized every video frame is decoded (i.e. converting into a sequence of images).
- **Grayscale conversion of image:** The video frames are in RGB format, RGB is converted to grayscale because processing a single channel image is faster than processing a three-channel colored image.
- **Reduce noise:** Noise can create false edges, therefore before going further, it's imperative to perform image smoothening. Gaussian blur is used to perform this process. Gaussian blur is a typical image filtering technique for lowering noise and enhancing image characteristics. The weights are selected using a Gaussian distribution, and each pixel is subjected to a weighted average that considers the pixels surrounding it. By reducing high-frequency elements and improving overall image quality, this blurring technique creates softer, more visually pleasant images.
- **Canny Edge Detector:** It computes gradient in all directions of our blurred image and traces the edges with large changes in intensity. For more explanation please go through this article: Canny Edge Detector
- **Region of Interest:** This step is to take into account only the region covered by the road lane. A mask is created here, which is of the same dimension as our road image. Furthermore, bitwise AND operation is performed between each pixel of our canny image and this mask. It ultimately masks the canny image and shows the region of interest traced by the polygonal contour of the mask.
- **Hough Line Transform:** In image processing, the Hough transformation is a feature extraction method used to find basic geometric objects like lines and circles. By converting the picture space into a parameter space, it makes it possible to identify shapes by accumulating voting points. We'll use the probabilistic Hough Line Transform in our algorithm. The Hough transformation has been extended to address the computational complexity with the probabilistic Hough transformation. In order to speed up processing while preserving accuracy in shape detection, it randomly chooses a selection of picture points and applies the Hough transformation solely to those points.
- **Draw lines on the Image or Video:** After identifying lane lines in our field of interest using Hough Line Transform, we overlay them on our visual input(video stream/image).

Dataset: To demonstrate the working of this algorithm we will be working on a video file of a road. You can download the dataset from this GitHub link – Dataset

> *Note: This code is implemented in google colab. If you are working on any other editor you might have make some alterations in code because colab has some dependency issues with OpenCV*

Steps to Implement Road Lane Detection

Step 1: Install OpenCV library in Python.

Python3

```
!pip install -q opencv-python
```

Step 2: Import the necessary libraries.

Python3

```
# Libraries for working with image processing
import numpy as np
import pandas as pd
import cv2
from google.colab.patches import cv2_imshow
# Libraries needed to edit/save/watch video clips
from moviepy import editor
import moviepy
```

Step 3: Define the driver function for our algorithm.

Python3

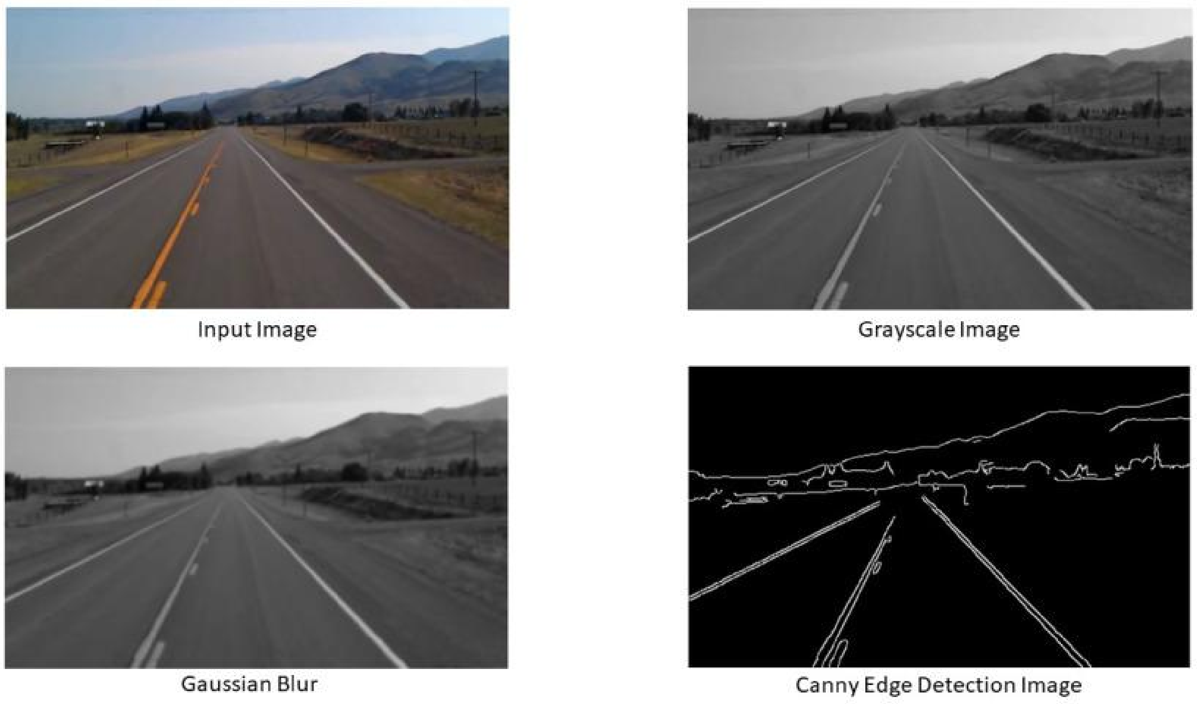
```
def process_video(test_video, output_video):
    """
    Read input video stream and produce a video file with detected lane lines.
    Parameters:
        test_video: location of input video file
        output_video: location where output video file is to be saved
    """
    # read the video file using VideoFileClip without audio
    input_video = editor.VideoFileClip(test_video, audio=False)
    # apply the function "frame_processor" to each frame of the video
    # will give more detail about "frame_processor" in further steps
    # "processed" stores the output video
    processed = input_video.fl_image(frame_processor)
    # save the output video stream to an mp4 file
    processed.write_videofile(output_video, audio=False)
```

Step 4: Define “frame_processor” function where all the processing happens on a frame to detect lane lines.

Python3

```python
def frame_processor(image):
    """
    Process the input frame to detect lane lines.
    Parameters:
        image: image of a road where one wants to detect lane lines
        (we will be passing frames of video to this function)
    """
    # convert the RGB image to Gray scale
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # applying gaussian Blur which removes noise from the image
    # and focuses on our region of interest
    # size of gaussian kernel
    kernel_size = 5
    # Applying gaussian blur to remove noise from the frames
    blur = cv2.GaussianBlur(grayscale, (kernel_size, kernel_size), 0)
    # first threshold for the hysteresis procedure
    low_t = 50
    # second threshold for the hysteresis procedure
    high_t = 150
    # applying canny edge detection and save edges in a variable
    edges = cv2.Canny(blur, low_t, high_t)
    # since we are getting too many edges from our image, we apply
    # a mask polygon to only focus on the road
    # Will explain Region selection in detail in further steps
    region = region_selection(edges)
    # Applying hough transform to get straight lines from our image
    # and find the lane lines
    # Will explain Hough Transform in detail in further steps
    hough = hough_transform(region)
    #lastly we draw the lines on our resulting frame and return it as output
    result = draw_lane_lines(image, lane_lines(image, hough))
    return result
```

Output:



Canny Edge Detection output

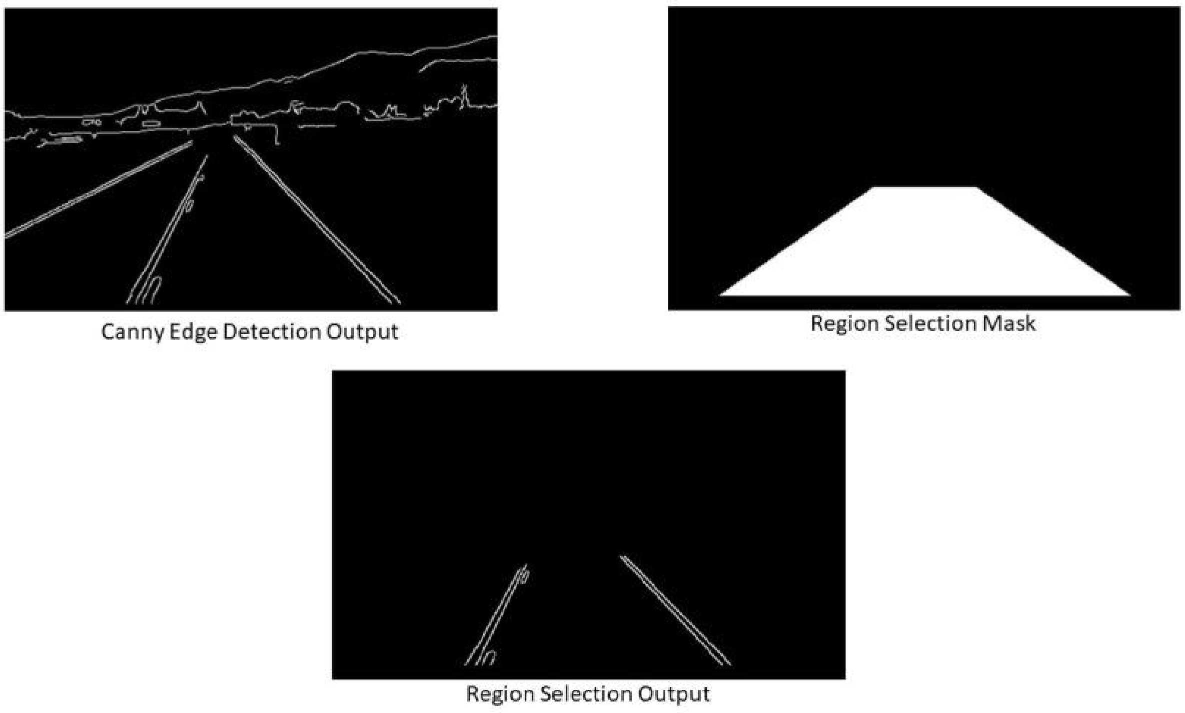
Step 5: Region Selection

Till now we have converted frames from RGB to Grayscale, applied Gaussian Blur to reduce noise and used canny edge detection. Next we will select the region where we want to detect road lanes.

Python3

```python
def region_selection(image):
    """
    Determine and cut the region of interest in the input image.
    Parameters:
        image: we pass here the output from canny where we have
        identified edges in the frame
    """
    # create an array of the same size as of the input image
    mask = np.zeros_like(image)
    # if you pass an image with more then one channel
    if len(image.shape) > 2:
        channel_count = image.shape[2]
        ignore_mask_color = (255,) * channel_count
    # our image only has one channel so it will go under "else"
    else:
          # color of the mask polygon (white)
        ignore_mask_color = 255
    # creating a polygon to focus only on the road in the picture
    # we have created this polygon in accordance to how the camera was placed
    rows, cols = image.shape[:2]
    bottom_left  = [cols * 0.1, rows * 0.95]
    top_left     = [cols * 0.4, rows * 0.6]
    bottom_right = [cols * 0.9, rows * 0.95]
    top_right    = [cols * 0.6, rows * 0.6]
    vertices = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)
    # filling the polygon with white color and generating the final mask
    cv2.fillPoly(mask, vertices, ignore_mask_color)
    # performing Bitwise AND on the input image and mask to get only the edges on the road
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image
```

Output:



Region Selection Output

Step 6: Now we will be identifying straight lines in the output image from the above function using Probabilistic Hough Transform

Python3

```
def hough_transform(image):
    """
    Determine and cut the region of interest in the input image.
    Parameter:
        image: grayscale image which should be an output from the edge detector
    """
    # Distance resolution of the accumulator in pixels.
    rho = 1
    # Angle resolution of the accumulator in radians.
    theta = np.pi/180
    # Only lines that are greater than threshold will be returned.
    threshold = 20
    # Line segments shorter than that are rejected.
    minLineLength = 20
    # Maximum allowed gap between points on the same line to link them
    maxLineGap = 500
    # function returns an array containing dimensions of straight lines
    # appearing in the input image
    return cv2.HoughLinesP(image, rho = rho, theta = theta, threshold = threshold,
                           minLineLength = minLineLength, maxLineGap = maxLineGap)
```

Output:

```
[[[284 180 382 278]]

[[281 180 379 285]]

[[137 274 183 192]]

[[140 285 189 188]]

[[313 210 388 285]]

[[139 285 188 188]]

[[132 282 181 194]]

[[146 285 191 196]]

[[286 187 379 284]]]
```

Step 7: Plotting Lines on video frames

Now that we have received the coordinates using Hough Transform, we will plot them on our original image(frame) but as we can see that we are getting coordinates of more than 2 lines so we will first find slope of left and right lane and then overlay them over the original image.

We have define 4 functions here to help draw left and right lane on our input frame:

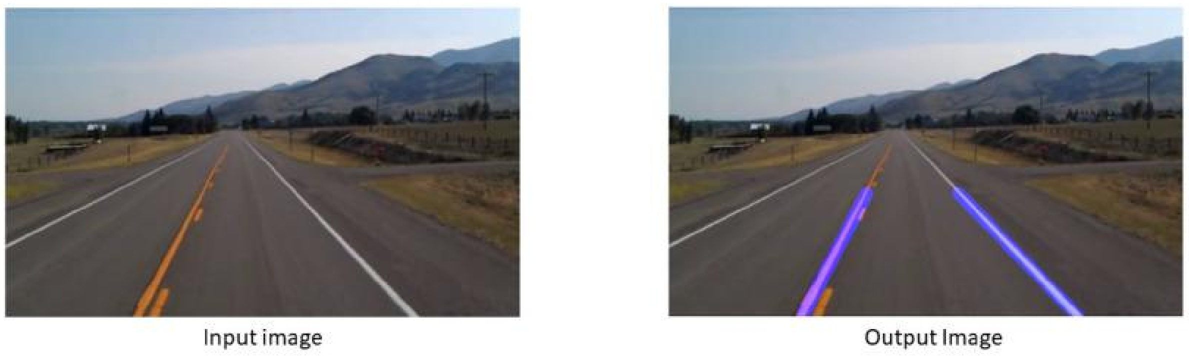
1. **Average_Slope_Intercept:** This function takes in the hough transform lines and calculate their slope and intercept. If the slope of a line is negative then it belongs to left lane else the line belongs to the right lane. Then we calculate the weighted average slope and intercept of left lane and right lanes.
2. **Pixel_Points:** By using slope, intercept and y-values of the line we find the x values for the line and returns the x and y coordinates of lanes as integers.
3. **Lane_Lines:** The function where Average_Slope_Intercept and Pixel Points are called and coordinates of right lane and left lane are calculated.
4. **Draw_Lane_Lines:** This function draws the left lane and right lane of the road on the input frame. Returns the output frame which is then stored in the variable "processed" in our driver function "process_video".

Python3

```
def average_slope_intercept(lines):
    """
    Find the slope and intercept of the left and right lanes of each image.
    Parameters:
        lines: output from Hough Transform
    """
    left_lines    = [] #(slope, intercept)
    left_weights  = [] #(length,)
    right_lines   = [] #(slope, intercept)
    right_weights = [] #(length,)

    for line in lines:
        for x1, y1, x2, y2 in line:
            if x1 == x2:
                continue
            # calculating slope of a line
            slope = (y2 - y1) / (x2 - x1)
            # calculating intercept of a line
            intercept = y1 - (slope * x1)
            # calculating length of a line
            length = np.sqrt(((y2 - y1) ** 2) + ((x2 - x1) ** 2))
            # slope of left lane is negative and for right lane slope is positive
            if slope < 0:
                left_lines.append((slope, intercept))
                left_weights.append((length))
            else:
                right_lines.append((slope, intercept))
                right_weights.append((length))
    #
    left_lane  = np.dot(left_weights,  left_lines) / np.sum(left_weights)  if len(left_weig
    right_lane = np.dot(right_weights, right_lines) / np.sum(right_weights) if len(right_we
    return left_lane, right_lane

def pixel_points(y1, y2, line):
    """
    Converts the slope and intercept of each line into pixel points.
        Parameters:
            y1: y-value of the line's starting point.
            y2: y-value of the line's end point.
            line: The slope and intercept of the line.
    """
    if line is None:
        return None
    slope, intercept = line
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    y1 = int(y1)
    y2 = int(y2)
    return ((x1, y1), (x2, y2))

def lane_lines(image, lines):
    """
    Create full lenght lines from pixel points.
        Parameters:
            image: The input test image.
            lines: The output lines from Hough Transform.
    """
    left_lane, right_lane = average_slope_intercept(lines)
    y1 = image.shape[0]
    y2 = y1 * 0.6
    left_line  = pixel_points(y1, y2, left_lane)
    right_line = pixel_points(y1, y2, right_lane)
    return left_line, right_line
```

```
def draw_lane_lines(image, lines, color=[255, 0, 0], thickness=12):
    """
    Draw lines onto the input image.
        Parameters:
            image: The input test image (video frame in our case).
            lines: The output lines from Hough Transform.
            color (Default = red): Line color.
            thickness (Default = 12): Line thickness.
    """
    line_image = np.zeros_like(image)
    for line in lines:
        if line is not None:
            cv2.line(line_image, *line,  color, thickness)
    return cv2.addWeighted(image, 1.0, line_image, 1.0, 0.0)
```

Output:



Road Lane Line Detection Output on an image

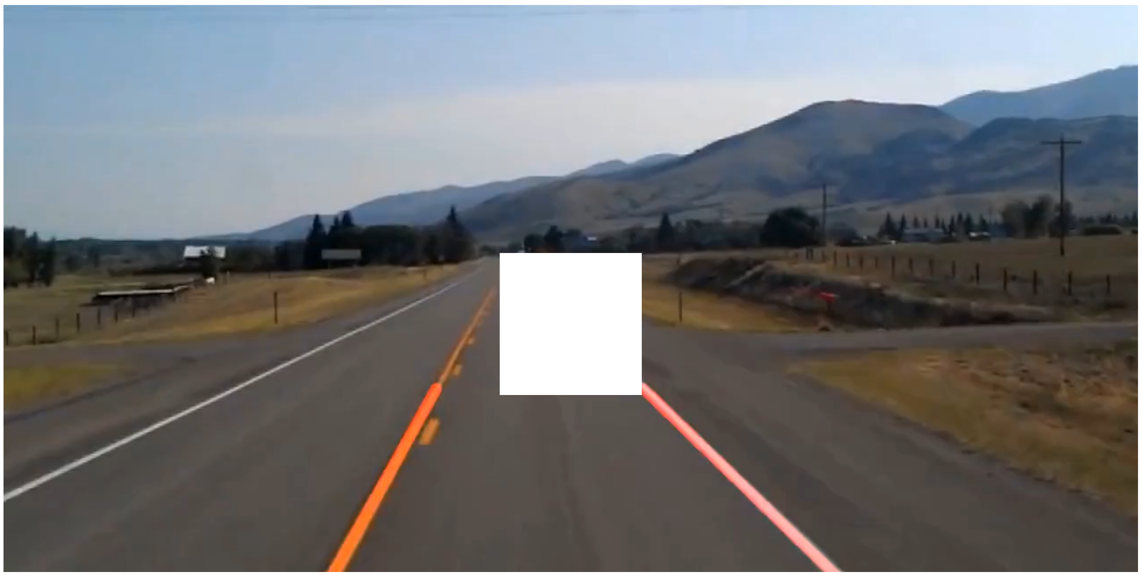
Complete Code for Real-time Road Lane Detection

Python3

```
import numpy as np
import pandas as pd
import cv2
from google.colab.patches import cv2_imshow
#Import everything needed to edit/save/watch video clips
from moviepy import editor
import moviepy

def region_selection(image):
    """
    Determine and cut the region of interest in the input image.
    Parameters:
        image: we pass here the output from canny where we have
        identified edges in the frame
    """
    # create an array of the same size as of the input image
    mask = np.zeros_like(image)
    # if you pass an image with more then one channel
    if len(image.shape) > 2:
        channel_count = image.shape[2]
        ignore_mask_color = (255,) * channel_count
    # our image only has one channel so it will go under "else"
    else:
          # color of the mask polygon (white)
        ignore_mask_color = 255
    # creating a polygon to focus only on the road in the picture
    # we have created this polygon in accordance to how the camera was placed
    rows, cols = image.shape[:2]
    bottom_left  = [cols * 0.1, rows * 0.95]
    top_left     = [cols * 0.4, rows * 0.6]
    bottom_right = [cols * 0.9, rows * 0.95]
    top_right    = [cols * 0.6, rows * 0.6]
    vertices = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)
    # filling the polygon with white color and generating the final mask
    cv2.fillPoly(mask, vertices, ignore_mask_color)
    # performing Bitwise AND on the input image and mask to get only the edges on the road
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image

def hough_transform(image):
    """
    Determine and cut the region of interest in the input image.
    Parameter:
        image: grayscale image which should be an output from the edge detector
    """
    # Distance resolution of the accumulator in pixels.
    rho = 1
    # Angle resolution of the accumulator in radians.
    theta = np.pi/180
    # Only lines that are greater than threshold will be returned.
    threshold = 20
    # Line segments shorter than that are rejected.
    minLineLength = 20
    # Maximum allowed gap between points on the same line to link them
    maxLineGap = 500
    # function returns an array containing dimensions of straight lines
    # appearing in the input image
    return cv2.HoughLinesP(image, rho = rho, theta = theta, threshold = threshold,
                           minLineLength = minLineLength, maxLineGap = maxLineGap)

def average_slope_intercept(lines):
    """
    Find the slope and intercept of the left and right lanes of each image.
    Parameters:
        lines: output from Hough Transform
    """
    left_lines    = [] #(slope, intercept)
    left_weights  = [] #(length,)
    right_lines   = [] #(slope, intercept)
    right_weights = [] #(length,)

    for line in lines:
        for x1, y1, x2, y2 in line:
            if x1 == x2:
                continue
            # calculating slope of a line
            slope = (y2 - y1) / (x2 - x1)
            # calculating intercept of a line
            intercept = y1 - (slope * x1)
            # calculating length of a line
            length = np.sqrt(((y2 - y1) ** 2) + ((x2 - x1) ** 2))
            # slope of left lane is negative and for right lane slope is positive
            if slope < 0:
                left_lines.append((slope, intercept))
                left_weights.append((length))
            else:
                right_lines.append((slope, intercept))
                right_weights.append((length))
    #
    left_lane  = np.dot(left_weights,  left_lines) / np.sum(left_weights)  if len(left_weig
    right_lane = np.dot(right_weights, right_lines) / np.sum(right_weights) if len(right_we
    return left_lane, right_lane

def pixel_points(y1, y2, line):
    """
    Converts the slope and intercept of each line into pixel points.
        Parameters:
            y1: y-value of the line's starting point.
            y2: y-value of the line's end point.
            line: The slope and intercept of the line.
    """
    if line is None:
        return None
    slope, intercept = line
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    y1 = int(y1)
    y2 = int(y2)
    return ((x1, y1), (x2, y2))

def lane_lines(image, lines):
    """
    Create full lenght lines from pixel points.
        Parameters:
            image: The input test image.
            lines: The output lines from Hough Transform.
    """
    left_lane, right_lane = average_slope_intercept(lines)
    y1 = image.shape[0]
    y2 = y1 * 0.6
    left_line  = pixel_points(y1, y2, left_lane)
    right_line = pixel_points(y1, y2, right_lane)
    return left_line, right_line


def draw_lane_lines(image, lines, color=[255, 0, 0], thickness=12):
    """
    Draw lines onto the input image.
        Parameters:
            image: The input test image (video frame in our case).
            lines: The output lines from Hough Transform.
            color (Default = red): Line color.
            thickness (Default = 12): Line thickness.
    """
    line_image = np.zeros_like(image)
    for line in lines:
        if line is not None:
            cv2.line(line_image, *line,  color, thickness)
    return cv2.addWeighted(image, 1.0, line_image, 1.0, 0.0)

def frame_processor(image):
    """
    Process the input frame to detect lane lines.
    Parameters:
        image: image of a road where one wants to detect lane lines
        (we will be passing frames of video to this function)
    """
    # convert the RGB image to Gray scale
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # applying gaussian Blur which removes noise from the image
    # and focuses on our region of interest
    # size of gaussian kernel
    kernel_size = 5
    # Applying gaussian blur to remove noise from the frames
    blur = cv2.GaussianBlur(grayscale, (kernel_size, kernel_size), 0)
    # first threshold for the hysteresis procedure
    low_t = 50
    # second threshold for the hysteresis procedure
    high_t = 150
    # applying canny edge detection and save edges in a variable
    edges = cv2.Canny(blur, low_t, high_t)
    # since we are getting too many edges from our image, we apply
    # a mask polygon to only focus on the road
    # Will explain Region selection in detail in further steps
    region = region_selection(edges)
    # Applying hough transform to get straight lines from our image
    # and find the lane lines
    # Will explain Hough Transform in detail in further steps
    hough = hough_transform(region)
    #lastly we draw the lines on our resulting frame and return it as output
    result = draw_lane_lines(image, lane_lines(image, hough))
    return result

# driver function
def process_video(test_video, output_video):
    """
    Read input video stream and produce a video file with detected lane lines.
    Parameters:
        test_video: location of input video file
        output_video: location where output video file is to be saved
    """
    # read the video file using VideoFileClip without audio
    input_video = editor.VideoFileClip(test_video, audio=False)
    # apply the function "frame_processor" to each frame of the video
    # will give more detail about "frame_processor" in further steps
    # "processed" stores the output video
    processed = input_video.fl_image(frame_processor)
    # save the output video stream to an mp4 file
    processed.write_videofile(output_video, audio=False)
```

```
# calling driver function
process_video('input.mp4','output.mp4')
```

Final Output:



00:00 00:26

Conclusion

We covered one of many ways for detecting road lanes using Canny Edge Detector and Hough Transform. Some other ways for road lane detection use complex neural networks and sensor data. This kind of advanced techniques and algorithms are currently used in Tesla for autonomous driving. The algorithm discussed in this article is the first step in understanding the basic working of autonomous vehicle. We will be coming up with more articles to help you understand difficult concepts like autonomous driving easily.

Last Updated : 24 May, 2023 11

Similar Reads

1. Real-Time Edge Detection using OpenCV in Python | Canny edge detection method
2. Multiple Color Detection in Real-Time using Python-OpenCV
3. Real time object color detection using OpenCV
4. Python | Corner detection with Harris Corner Detection method using OpenCV
5. Python | Corner Detection with Shi-Tomasi Corner Detection Method using OpenCV
6. OpenCV - Facial Landmarks and Face Detection using dlib and OpenCV