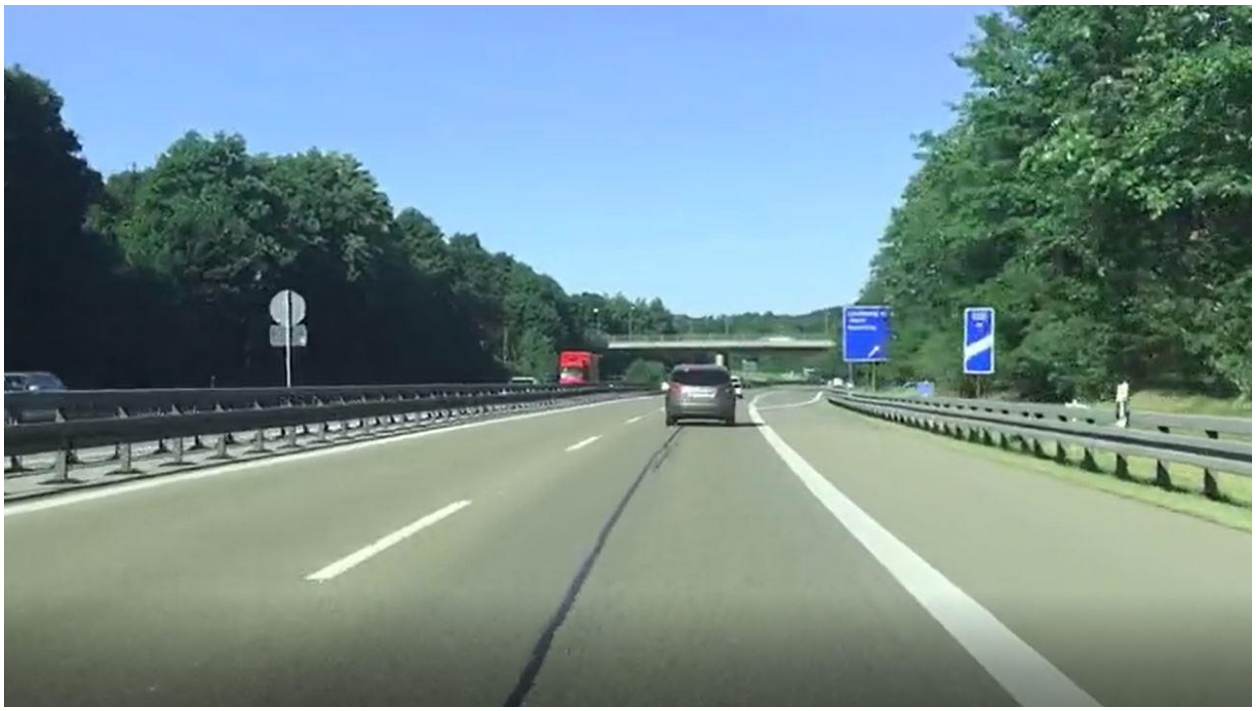
Explanation

To implement Hough Transformation for lane detection

1. **Static Image conversion into gray scale**
   A small frame from the video is saved as a jpg image and the function is used to read the image as shown below.

   ```
   def read_image(image_path):
       """This function reads any image from a given path"""
       image = cv2.imread(image_path)
       return image
   ```



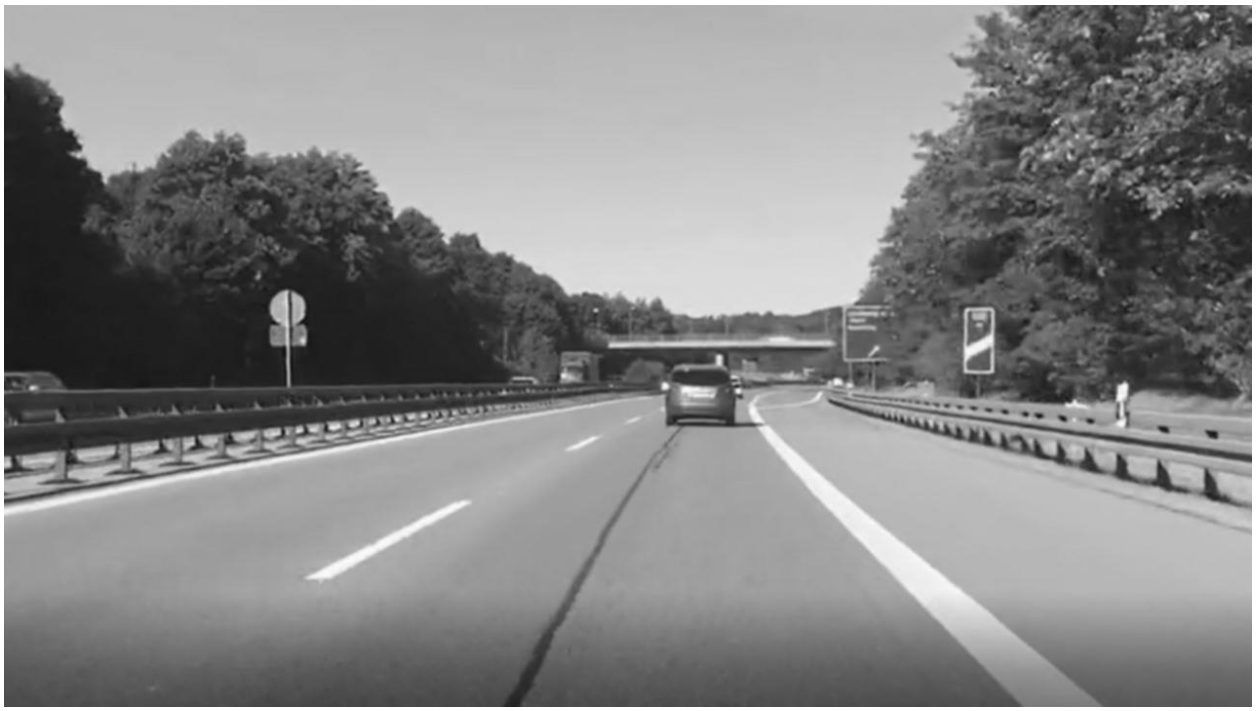
   This image frame is then converted into gray scale. This is because the Hough transformation works better with a gray scale image that has to sharp light intensity contrast. This gray scale conversion is performed with the gray image conversion function

2. **Gray scale image conversion**

   ```
   def gray_image(image):
       return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
   ```

3. Filtering the Gray Image with Gaussian Filter

The Hough transformation works best with images that are binary i.e., black, and white and also with less sharpness. Thus, we pass the image into a gaussian filter which reduces the sharpness or intensity on the gray scale image. This is done using the function.



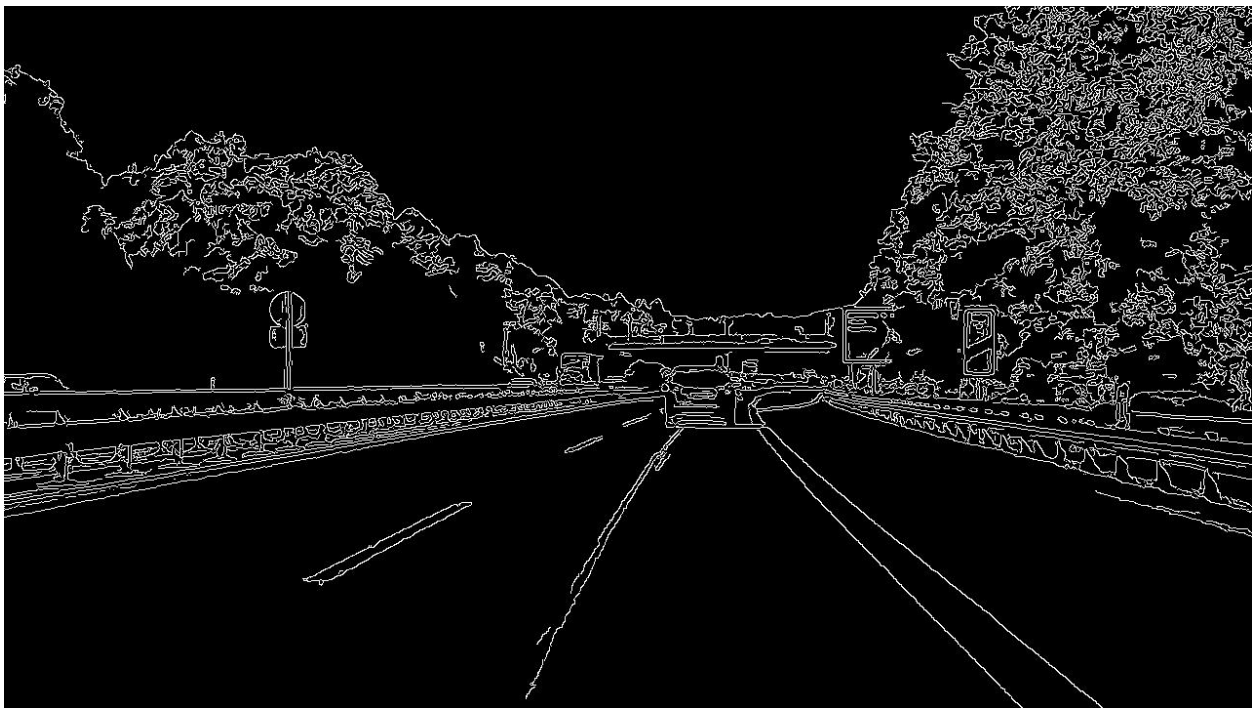
```
def gaussian_filter (image, filter_size, sigma):
    return cv2.GaussianBlur(image, (filter_size, filter_size), sigma)
```

4. Edge identification using Canny Edge detection function.

Once we have our filtered image with less noise, we detect the edges in the picture image using the canny edge detection with the canny function

```
def canny_edge_detection(image, low_threshold, high_threshold):
    return cv2.Canny(image, low_threshold, high_threshold)
```

The result is seen in the image below



This image would be the basis to implement the Hough transformation

5. Defining our area of interest

From the canny image, we are interested in the road lane itself, thus we need to define this region so that every other region which is of no interest are not considered. This is done using the function below.

```
def ROI_masking(image):
    height = image.shape[0]
    width = image.shape[1]
    image =  np.array(image, dtype=np.float32)
    polygon_coords = np.array([(0, height), (0, round(height/1.25)), (round(width/2),round(height/2)),(width, height)])
    mask  = np.zeros((height, width))
```

```
masked_image = cv2.fillPoly(mask, [polygon_coords], 255)
masked_image = np.array(masked_image, dtype=np.float32)
return cv2.bitwise_and(image, masked_image)
```

This function works by:

- *first creating an image with similar shape known as a mask with shape of input with all zeros*
- *Then a polygon is impressed on that image using opencv library function.*
- *Then a bitwise_AND operation is performed on the input image and masked image to get region of interest. The bitwise AND operation only takes true in regions that are absolutely true for masked image and the original image.*

6. Drawing the Hough Lines

Once we have identified our area of interest, we then draw the Hough lines on that area of interest using the Hough transformation function which converts the image of area of interest into the needed format then the in-built HoughLinesP return all the lines of interest identified in the area of interest image.

```
def hough_transformation(reg_of_int_img):
    image = cv2.convertScaleAbs(reg_of_int_img)
    return cv2.HoughLinesP(image, 1, np.pi/180, 50, None, 0, 1000)
```

7. Determine the average slope intercept

The average slope intercept is obtained using the function below

```
def average_slope_intercept(image, lines):
    left_lines = []
    right_lines = []

    #following for loop is to get slope and intercept
    for line in lines:
        x1, y1, x2, y2 = line[0][:]
        parameters = np.polyfit((x1,x2), (y1,y2), 1)
        slope = parameters[0]
        intercept = parameters[1]
        if slope < 0:
            left_lines.append((slope, intercept))
        else:
            right_lines.append((slope, intercept))

    #Left Line COordinates
    left_line_coordinates = []
    for line in left_lines:
        if line[0] > -0.1:
```

```
            continue
        left_line_coordinates.append(get_coordinates(image, line))

    #Right line coordinates
    right_line_coordinates = []
    for line in right_lines:
        if line[0] < 0.2:
            continue
        right_line_coordinates.append(get_coordinates(image, line))

    # getting left lines and right lines using cosine distance remove the crack line that could be found on
the road
    final_left_lines = []
    final_right_lines = []
    cosine_distance = []

    # Condition if both curves have the same slope
    if len(left_line_coordinates) == 0 or len(right_line_coordinates) == 0:
        if len(left_line_coordinates) == 0:
            lines = right_line_coordinates
        else:
            lines = left_line_coordinates

        if len(lines) <= 2:
            return np.array(lines)

        # Calculating the cosine distance between the left line and right line
        for l in lines:
            for m in lines:
                cosine_distance.append(spatial.distance.cosine(l, m))
        cosine_distance = [item for item in cosine_distance if not(math.isnan(item))]
        threshold = np.percentile(cosine_distance,85)

# Filter the lines based on threshold and cosine distance with 85 percentiles
        for left in lines:
            for right in lines:
                dist = spatial.distance.cosine(left, right)

                if dist > threshold and len(final_left_lines) == 0:
                    final_left_lines.append(list(left))
                    final_right_lines.append(list(right))
                    continue

                if dist > threshold:
```

```
            if list(left) not in final_left_lines and list(left) not in final_right_lines:
                final_left_lines.append(list(left))
            if list(right) not in final_right_lines and list(right) not in final_right_lines:
                final_right_lines.append(list(right))

    # To find the average of the lines
    left_line = np.average(np.array(final_left_lines), axis=0)
    right_line = np.average(np.array(final_right_lines), axis=0)
    return np.array([left_line, right_line])

else:
    # To keep the farthest lines and removing crack lines
    for left in left_line_coordinates:
        for right in right_line_coordinates:
            cosine_distance.append(spatial.distance.cosine(left, right))
    cosine_distance = [item for item in cosine_distance if not(math.isnan(item))]
    threshold = np.percentile(cosine_distance,85)

    # Threshold filtering similar to NVM
    for left in left_line_coordinates:
        for right in right_line_coordinates:
            dist = spatial.distance.cosine(left, right)

            if dist > threshold and len(final_left_lines) == 0:
                final_left_lines.append(list(left))
                final_right_lines.append(list(right))
                continue

            if dist > threshold:
                if list(left) not in final_left_lines:
                    final_left_lines.append(list(left))
                if list(right) not in final_right_lines:
                    final_right_lines.append(list(right))

    left_line = np.average(np.array(final_left_lines), axis=0)
    right_line = np.average(np.array(final_right_lines), axis=0)

    return np.array([left_line, right_line])
```

The function in summary tries to keep those lines that are representing lanes and further aggregate all the other lines the Hough line function returned.

In detail, the ROI_image and Hough lines are passed as inputs, then a separation of the lines representing left lane with the right lane is performed. To achieve this, we try to fit a line on each of

the points and obtain a slop and intercept (y=mx+c). Depending on what the slope value is, we decide if a line is on the left side or on the right-hand side.
Furthermore, this slope and intercept are then used to get the coordinates of the lines.

8. **Impressing Lane lines to our image**
The output from the average slope intercept is then used to draw the lines on the image suing a weighted summed average of the slope and intercepts of left- and right-hand side lines. The function is shown below

```
def draw_lane_lines(image, lines, color=[0, 0, 255], thickness=10):
    line_image = np.zeros_like(image)
    for line in lines:
        x1, y1, x2, y2 =line
        if line is not None:
            cv2.line(line_image, (int(x1),int(y1)), (int(x2),int(y2)), color, thickness)
    return cv2.addWeighted(image, 0.8, line_image, 1.0, 1.0)
```

9. **Use a recursive function to implement all the other functions**
We create a separate recursive function that implements the other functions above to return our single output. This function helps modularize the code thus making any repetitive function call avoided. The function is found below.

```
def lanes(image):
    #gray image
    gray_images = gray_image(image)

    #Gaussian Filtering or smoothing
    gaussian_filtered_image = gaussian_filter(gray_images, 5, 5)

    #Canny edge detection
    canny_edge_image = canny_edge_detection(gaussian_filtered_image, 50, 150)

    #region of interest
    ROI_image = ROI_masking(canny_edge_image)

    #To get hough lines
    hough_lines = hough_transformation(ROI_image)

    #to get left and right lines
    lines = average_slope_intercept(ROI_image, hough_lines)
    threshold = spatial.distance.cosine(lines[0], lines[1])

    #to draw lanes using above lines
    draw_lanes = draw_lane_lines(image, lines,[0, 0, 255], 10)
```

```
    return threshold, draw_lanes
```

10. Final Impression of our lanes on the video file

We finally impress these lines into our video using cv2.VideoCapture for reading the video and cv2.VideoWriter to generate the output video.

```
# Reading the input video
video = cv2.VideoCapture('Lane_video.mp4')

# Check for the fps of video
fps = video.get(5)

# Checking for the total number of frames in the video
frame_count = video.get(7)

# Getting the frame size data
frame_width = int(video.get(3))
frame_height = int(video.get(4))
frame_size = (frame_width,frame_height)

# Impressing lane marked images into a video file
output = cv2.VideoWriter('Road Lane Tracking.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 20, frame_size)

# loading of each frame
while(video.isOpened()):
    ret, frame = video.read()
    if ret == True:
        threshold, final_image = lanes(frame)
        if 0.2 <= threshold <= 0.303:
            output.write(final_image)
            cv2.imshow("Road Lane Tracking",final_image)
            k = cv2.waitKey(50)
            if k == 113:
                break
        else:
            continue
    else:
        break
```

The final result is seen below