My PipeLine

1. **Preprocess the initial image**

- First convert the initial image to gray scale
- Then reduce the noise of the image by using **gaussian blur algorithm** with a specified **`kernel_size`** , a larger kernel size means the averaging or smoothing of shades is over a larger area

2. **Get the edges from preprocessed image**

- To get this, we used the **canny transform algorithm,** we detect strong edges (strong gradient) above a **`canny_hi`** (high threshold), and reject pixels, below the **`canny_low`** (low threshold). The pixels between the low and high will be included as long as they are at a strong edge. Pixel values are between 0 and 255, the value difference between two pixels will be in this range, so this is the reasonable range of the thresholds. The recommended low to high ratio of the thresholds is 1:2, to 1:3

3. **Block out everything except the region of interest**

- We should only consider the pixels where we expect the lane lines to be
- To get the region of interest (roi), we have to specify the corners (vertices) of a four-sided polygon (bottom_left, top_left, top_right, bottom_right), instead of specifying pixels I used fractions of the image's dimensions so I can eyeball it and calculate is from there
- We block or mask out all other edges that is not part of our region of interest

4. **Get the lines from the edges**

- We use the **hough transform algorithm** to get lines from the edges we detected. Here are the parameters we tweaked:
- **Rho, theta_coeff** - distance (pixels) and angular (degrees, converted to radians inside) resolutions of our grid in hough space, starting from one, we can scale this value up to be more flexible in what constitutes a line
- **Min_vote** - the number of intersections in a given grid cell a candidate line needs to have to make it into the output
- **Min_line_length** - the minimum length of the line in pixels that we can accept as an output
- **Max_line_gap** - the maximum distance in pixels between segments that we allow to be connected in a single line

5. **Draw the lines or extrapolate the lines**

6. **Overlap/overlay the drawn/extrapolated lines to the initial image**

```
param = {
   #blur parameters
  'kernel_size': 5,

   #canny transform parameters
  'canny_lo': 100,
  'canny_hi': 200,

  #region of interest parameters
  'ax_coef': 10.0/25,
  'bx_coef': 14.0/25,
  'cy_coef': 13.0/20,
  'dy_coef': 13.0/20,
  'maxy_coef': 1.0,
  'maxx_coef': 1.0,
  'startx_coef': 0.0,

  #hough parameters
  'rho': 1,
  'theta_coef': 1,
  'min_votes': 30,
  'min_line_length': 20,
  'max_line_gap': 20
}
```

```
def process_image(image):

  raw_image = np.copy(image)
  gray_image = gray(image)
  blur_image = reduce_noise(gray_image, param['kernel_size'])

  edge_image = get_edges(blur_image, param['canny_lo'], param['canny_hi'])

  x, y = image.shape[1], image.shape[0]
  vertices = get_vertices(x, y, param['ax_coef'], param['bx_coef'],
                                param['cy_coef'], param['dy_coef'],
                                param['maxx_coef'], param['maxy_coef'],
                                param['startx_coef'])
  roi = get_roi(edge_image, vertices)
  mask_image = mask(edge_image, roi)

  lines = get_lines(mask_image, param['rho'], param['theta_coef'],
                            param['min_votes'], param['min_line_length'],
                            param['max_line_gap'])

  #line_image = draw_lines(lines, raw_image)
  #result = overlap(line_image, raw_image)

  line_image2 = extrapolate_lines(lines, raw_image)
  result = overlap(line_image2, raw_image)
  return result
```

IMPORTANT NOTE: Please look at the test_images folder as I have saved a screenshot for each step of my pipeline for each test image.

Extrapolating Lines (draw lines improvement)

- The key here is the equation of the line which is **y = slope * x + intercept**
- The hough transform gives us small lines, we can extrapolate these small lines to make a left and right lane.
- We find the minimum and maximum y coordinates to get the points at the minimum and maximum y coordinates from the the points of the small lines
- First we separate the small lines into two groups. One with a positive slope (the left line), and one with a negative slope (the right line), the lines slant towards each other. We group the x, y and slope for the left and right lane.
- We take the average x coordinate, y coordinate, slope and intercept for each group.
- We derive the upper and lower x coordinate for each lane based on the computed values and the equation of the line
- We draw the line on a blank image and return the image

```
# equation of a line: y = slope*x + intercept
# left lane has a positive slope, right lane has a negative slope

def extrapolate_lines(lines, image, color=[255, 0, 0], thickness = 10,
                      positvie_thresh = 0, negative_thresh = 0):

  imshape = image.shape
  image = np.copy(image)*0

  #initialize minimum and maximum y coordinate
  minimum_y = image.shape[0]
  maximum_y = image.shape[0]

  #initialize groups of values into empty lists
  left_slopes = []
  left_xs = []
  left_ys = []
  right_slopes = []
  right_xs = []
  right_ys = []

  # segregate the small line segments into the left lane group or right lane group
  if lines is not None:
    for line in lines:
      for x1,y1,x2,y2 in line:

        # get the slope and intercept of the line (as defined by two points) using the polyfit function
        slope, intercept = np.polyfit((x1,x2), (y1,y2), 1)

        if (slope > positive_thresh): #if positive slope, put value to left lane group
          left_slopes += [slope]
          left_xs += [x1, x2]
          left_ys += [y1, y2]
        elif (slope < negative_thresh): #if negative slope, put value to right lane group
          right_slopes += [slope]
          right_xs += [x1, x2]
          right_ys += [y1, y2]

        # update the minimum y_coordinate based on values seen
        minimum_y = min(min(y1, y2), minimum_y)
```

```
  #average all the values in each group to get the slope, x, and y
  left_slope = np.mean(left_slopes)
  left_x = np.mean(left_xs)
  left_y = np.mean(left_ys)
  right_slope = np.mean(right_slopes)
  right_x = np.mean(right_xs)
  right_y = np.mean(right_ys)

  #derive the intercept using the equation of the line and average value
  left_intercept = left_y - (left_slope * left_x)
  right_intercept = right_y - (right_slope * right_x)

  if ((len(left_slopes) > 0) and (len(right_slopes) > 0)): #make sure we have points in each group
    #derive the x coordinate using the equation of the lines and derived values
    upper_left_x = int((minimum_y - left_intercept) / left_slope)
    lower_left_x = int((maximum_y - left_intercept) / left_slope)
    upper_right_x = int((minimum_y - right_intercept) / right_slope)
    lower_right_x = int((maximum_y - right_intercept) / right_slope)

    #draw the line based on two points
    cv2.line(image, (upper_left_x, minimum_y), (lower_left_x, maximum_y), color, thickness)
    cv2.line(image, (upper_right_x, minimum_y), (lower_right_x, maximum_y), color, thickness)

  return image
```

Reflection

Performance on white video, yellow video, Extra challenge

- You can see the performance of my pipeline in white.mp4, yellow.mp4, extra.mp4 in the root folder
- You can also see the performance for the extra challenge of the following steps of my pipeline (located in the root folder
- after the canny transform (**1-edge_extra. Mp4**)
- the region of interest overlayed (**2-roi_extra.mp4**)
- The small lines drawn at (**3-small-lines-extra.mp4**)
- IMPORTANT NOTE: Please look at the test_images folder as I have saved a screenshot for each step of my pipeline for each test image.
- **For the extra challenge, I made sure that the region of interest is correct, and also I noticed that because when it is sunny the shadows made by the trees are caught as edges by the canny transform and its slope was almost horizontal, what I did was the rejected the lines with slopes that were almost horizontal because that was just noise and would mess up my calculations. Horizontal lines don't make sense based on the angle of the camera**

Potential Shortcomings of my Pipeline

- It would not work if the camera angle was different and the region of interest would be different
- It would not work if there were animals or people crossing the street and are in between the two lanes, because this would be considered as edges by the canny transform so we would have to filter that out
- It would not work if there was something like a car directly in the lane as this would mess up the extrapolation of the lane since there would be spurious lines
- It would not work if the difference between the color of the lanes were too minimal, it would not be detected by the canny transform parameters
- Parameters can be tweaked for even better performance

Possible Improvements on My Pipeline

- Parameters can be tweaked for even better performance
- If there was abrupt changes between two frames, we can reject the result of the second frame because it does not make sense.
- Considering the following two consecutive frames, this does not make sense because of the abrupt change so we should disregard the second frame and consider it as an error/anomaly/miscalculation.

Appendix:

Getting the vertices to define the region of interest

```
def get_vertices(x, y, axc, bxc, cyc, dyc, maxyc = 1.0, maxxc = 1.0, startxc = 0.0):
  ax = int(axc*x)
  bx = int(bxc*x)
  cy = int(cyc*y)
  dy = int(dyc*y)
  maxy = int(maxyc*y)
  maxx = int(maxxc*x)
  startx = int(startxc*x)
  bottom_left = (startx, maxy)
  top_left = (ax, cy)
  top_right = (bx, dy)
  bottom_right = (maxx, maxy)
  vertices = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)
  return vertices
```

Process and Parameters on Extra Challenge

```
p = {
   #blur parameters
  'kernel_size': 3,

   #canny transform parameters
  'canny_lo': 50,
  'canny_hi': 150,

  #region of interest parameters
  'ax_coef': 0.41,
  'bx_coef': 0.60,
  'cy_coef': 0.65,
  'dy_coef': 0.65,
  'maxy_coef': 0.9,
  'maxx_coef': 0.85,
  'startx_coef': 0.15,

  #hough parameters
  'rho': 1,
  'theta_coef': 1,
  'min_votes': 20,
  'min_line_length': 50,
  'max_line_gap': 20
}
```

```
def draw(lines, image, color=[255, 0, 0], thickness = 2, thresh = 10):
  ''' draw the lines on a blank image'''
  lined_image = np.copy(image)*0

  if lines is not None:
    for line in lines:
      for x1,y1,x2,y2 in line:
        slope, intercept = np.polyfit((x1,x2), (y1,y2), 1)
        if abs(slope) > thresh:
          cv2.line(lined_image, (x1, y1), (x2, y2), color, thickness)

  return lined_image
```

```
def process(image):

  raw_image = np.copy(image)

  gray_image = gray(image)
  blur_image = reduce_noise(gray_image, p['kernel_size'])

  edge_image = get_edges(blur_image, p['canny_lo'], p['canny_hi'])

  #return convert_to_color(edge_image) #check if canny parameters detect edges of lanes

  x = image.shape[1]
  y = image.shape[0]
  vertices = get_vertices(x, y, p['ax_coef'], p['bx_coef'],
                                p['cy_coef'], p['dy_coef'],
                                p['maxy_coef'], p['maxx_coef'], p['startx_coef'])

  roi = get_roi(edge_image, vertices)
  mask_image = mask(edge_image, roi)

  #return overlap(convert_to_color(roi), raw_image) #check if roi is good

  lines = get_lines(mask_image, p['rho'], p['theta_coef'],
                                p['min_votes'], p['min_line_length'],
                                p['max_line_gap'], )

  #line_image = draw(lines, raw_image, thresh = 0.5)
  #result = overlap(line_image, raw_image)

  line_image2 = extrapolate_lines(lines, raw_image,
                                  positive_thresh = 0.5, negative_thresh = -0.5)
  result = overlap(line_image2, raw_image)

  return result
```

Performance on Test Images