Project: Finding Lane Lines on the Road

Develop a pipeline to identify lane lines on the road. You must apply it on a series of individual images, provided in the *test_images* folder.

Once you have a result that looks roughly like the image *line-segments-example* in the examples folder (also shown below), you'll need to try to average and/or extrapolate the line segments you've detected to map out the full extent of the lane lines.

The tools you have are color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection. You are also free to explore and try other techniques that were not presented. Your goal is piece together a pipeline to detect the line segments in the image, then average/extrapolate them and draw them onto the image for display (as below).

Your output should look something like this (above) after detecting line segments using the helper functions below

Your goal is to connect/average/extrapolate line segments to get output like this

Import Packages

In [175]:

```
#importing some useful packages
import matplotlib.pyplot as plt
import numpy as np
import cv2
%matplotlib inline
```

Read in an Image

In [176]:

```
#reading in an image
image = cv2.imread('test_images/solidWhiteRight.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#printing out some stats and plotting
print('This image is:', type(image), 'with dimensions:', image.shape)
plt.imshow(image)  # if you wanted to show a single color channel image called 'gray
```

```
This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)
```

Out[176]:

```
<matplotlib.image.AxesImage at 0x1374a9da0>
```

Ideas for Lane Detection Pipeline

Some OpenCV functions that might be useful for this project are:

`cv2.inRange()` for color selection
`cv2.fillPoly()` for regions selection
`cv2.line()` to draw lines on an image given endpoints
`cv2.addWeighted()` to coadd / overlay two images `cv2.cvtColor()` to grayscale or change color
`cv2.imwrite()` to output images to file
`cv2.bitwise_and()` to apply a mask to an image

Helper Functions

Below are some helper functions to help get you started.

In [177]:

```
import math

def grayscale(img):
    """Applies the Grayscale transform
    This will return an image with only one color channel
    but NOTE: to see the returned image as grayscale
    (assuming your grayscaled image is called 'gray')
    you should call plt.imshow(gray, cmap='gray')"""
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Or use BGR2GRAY if you read an image with cv2.imread()
    # return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def canny(img, low_threshold, high_threshold):
    """Applies the Canny transform"""
    return cv2.Canny(img, low_threshold, high_threshold)

def gaussian_blur(img, kernel_size):
    """Applies a Gaussian Noise kernel"""
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)

def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    `vertices` should be a numpy array of integer points.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with depending on the
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
```

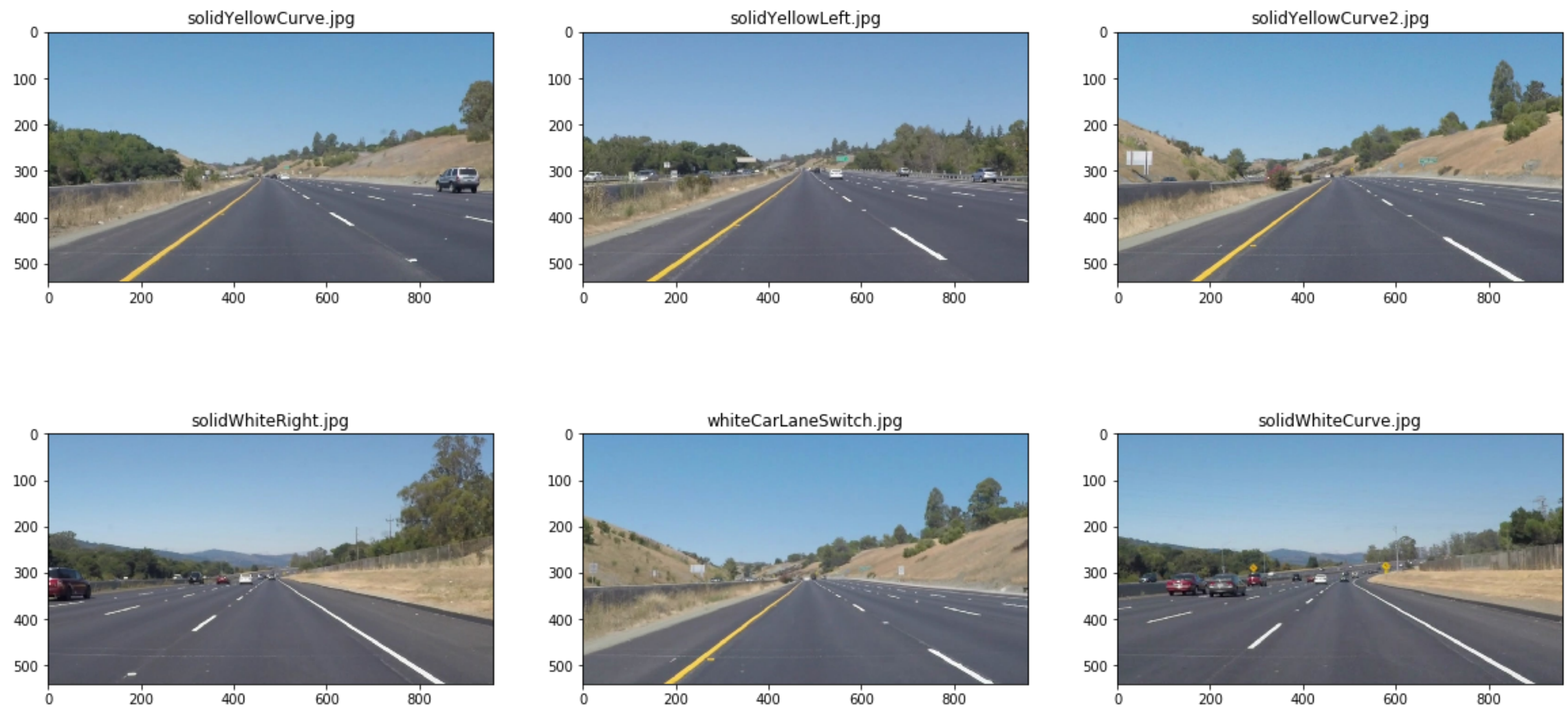
```
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, np.array([vertices], dtype=np.int32), ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image


def draw_lines(img, lines, color=[255, 0, 0], thickness=4):
    """
    This function draws `lines` with `color` and `thickness`.
    Lines are drawn on the image inplace (mutates the image).
    If you want to make the lines semi-transparent, think about combining
    this function with the weighted_img() function below
    """
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(img, (x1, y1), (x2, y2), color, thickness)

def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    """
    `img` should be the output of a Canny transform.

    Returns an image with hough lines drawn.
    """
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=
    line_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
    draw_lines(line_img, lines)
    return line_img

def weighted_img(img, initial_img, alpha=0.8, beta=1., gamma=0.):
    """
    `img` is the output of the hough_lines(), An image with lines drawn on it.
    Should be a blank image (all black) with lines drawn on it.

    `initial_img` should be the image before any processing.

    The result image is computed as follows:

    initial_img * α + img * β + γ
    NOTE: initial_img and img must be the same shape!
    """
    return cv2.addWeighted(initial_img, alpha, img, beta, gamma)
```

Test Images

Build your pipeline to work on the images in the directory "test_images"

In [178]:

```python
import os

path = "test_images/"
files = os.listdir(path)
images = []

f, plots = plt.subplots((len(files)+3-1)//3, 3, figsize=(20,10))
plots = [plot for sublist in plots for plot in sublist]

for file, plot in zip(files, plots):
    image = cv2.cvtColor(cv2.imread(os.path.join(path, file)), cv2.COLOR_BGR2RGB)
    plot.set_title(file)
    plot.imshow(image)
    images.append((image, file))
```



Build a Lane Finding Pipeline

Build the pipeline and run your solution on all test_images.

Try tuning the various parameters, especially the low and high Canny thresholds as well as the Hough lines parameters.

In [179]:

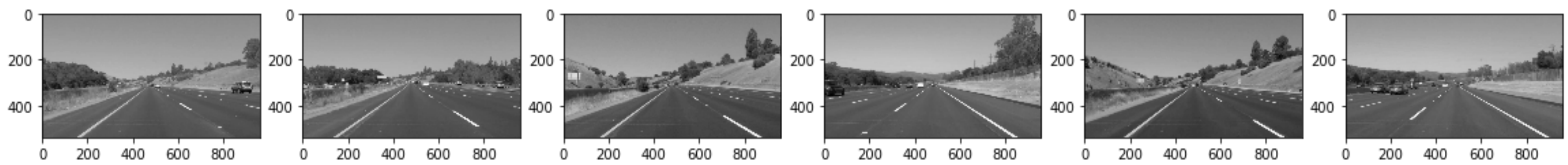
```
# TODO: Build your pipeline that will draw lane lines segments on the test_images

# Set images to individual variables
img1 = cv2.cvtColor(cv2.imread(os.path.join(path, files[0])), cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(cv2.imread(os.path.join(path, files[1])), cv2.COLOR_BGR2RGB)
img3 = cv2.cvtColor(cv2.imread(os.path.join(path, files[2])), cv2.COLOR_BGR2RGB)
img4 = cv2.cvtColor(cv2.imread(os.path.join(path, files[3])), cv2.COLOR_BGR2RGB)
img5 = cv2.cvtColor(cv2.imread(os.path.join(path, files[4])), cv2.COLOR_BGR2RGB)
img6 = cv2.cvtColor(cv2.imread(os.path.join(path, files[5])), cv2.COLOR_BGR2RGB)
```

In [180]:

```
# Convert to grayscale

img_gray1 = grayscale(img1)
img_gray2 = grayscale(img2)
img_gray3 = grayscale(img3)
img_gray4 = grayscale(img4)
img_gray5 = grayscale(img5)
img_gray6 = grayscale(img6)

f, (ax1,ax2,ax3,ax4,ax5,ax6) = plt.subplots(1, 6, figsize=(20,10))
ax1.imshow(img_gray1, cmap='gray')
ax2.imshow(img_gray2, cmap='gray')
ax3.imshow(img_gray3, cmap='gray')
ax4.imshow(img_gray4, cmap='gray')
ax5.imshow(img_gray5, cmap='gray')
ax6.imshow(img_gray6, cmap='gray')
```

Out[180]:

```
<matplotlib.image.AxesImage at 0x132a21cf8>
```

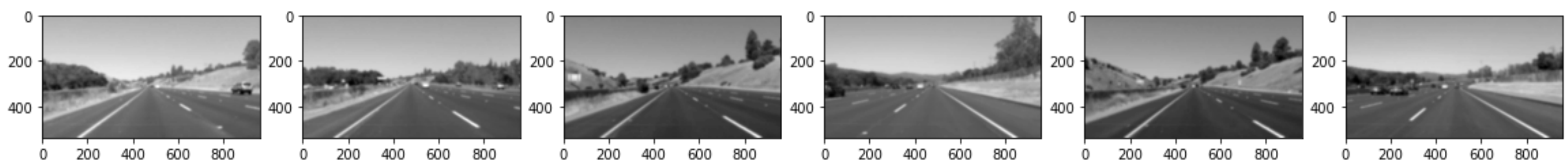


In [181]:

```
# Apply Gaussian Blur

img_gauss1 = gaussian_blur(img_gray1,25)
img_gauss2 = gaussian_blur(img_gray2,25)
img_gauss3 = gaussian_blur(img_gray3,25)
img_gauss4 = gaussian_blur(img_gray4,25)
img_gauss5 = gaussian_blur(img_gray5,25)
img_gauss6 = gaussian_blur(img_gray6,25)


f, (ax1,ax2,ax3,ax4,ax5,ax6) = plt.subplots(1, 6, figsize=(20,10))
ax1.imshow(img_gauss1, cmap='gray')
ax2.imshow(img_gauss2, cmap='gray')
ax3.imshow(img_gauss3, cmap='gray')
ax4.imshow(img_gauss4, cmap='gray')
ax5.imshow(img_gauss5, cmap='gray')
ax6.imshow(img_gauss6, cmap='gray')
```

Out[181]:

```
<matplotlib.image.AxesImage at 0x13ed2bf28>
```



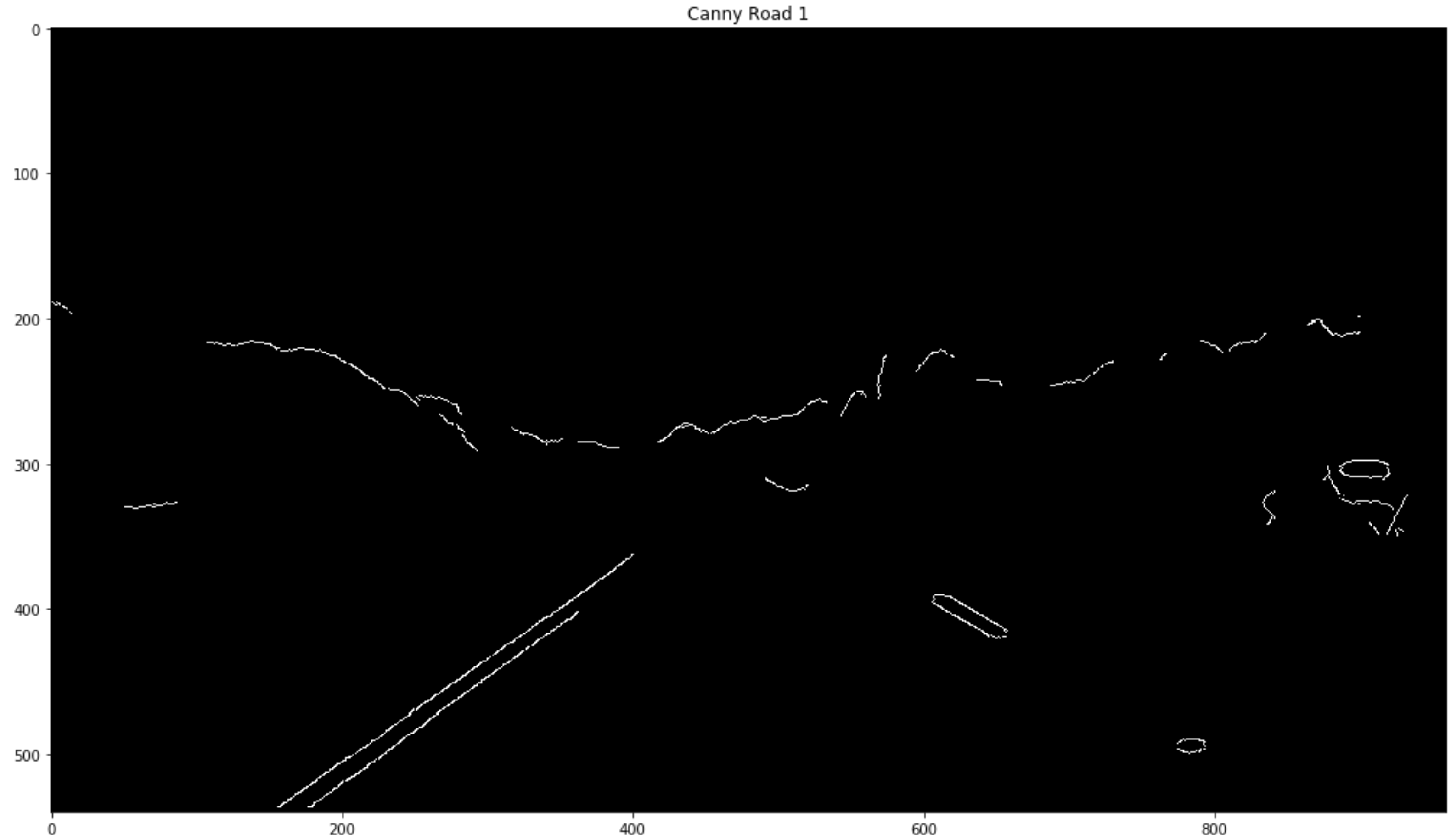
In [182]:

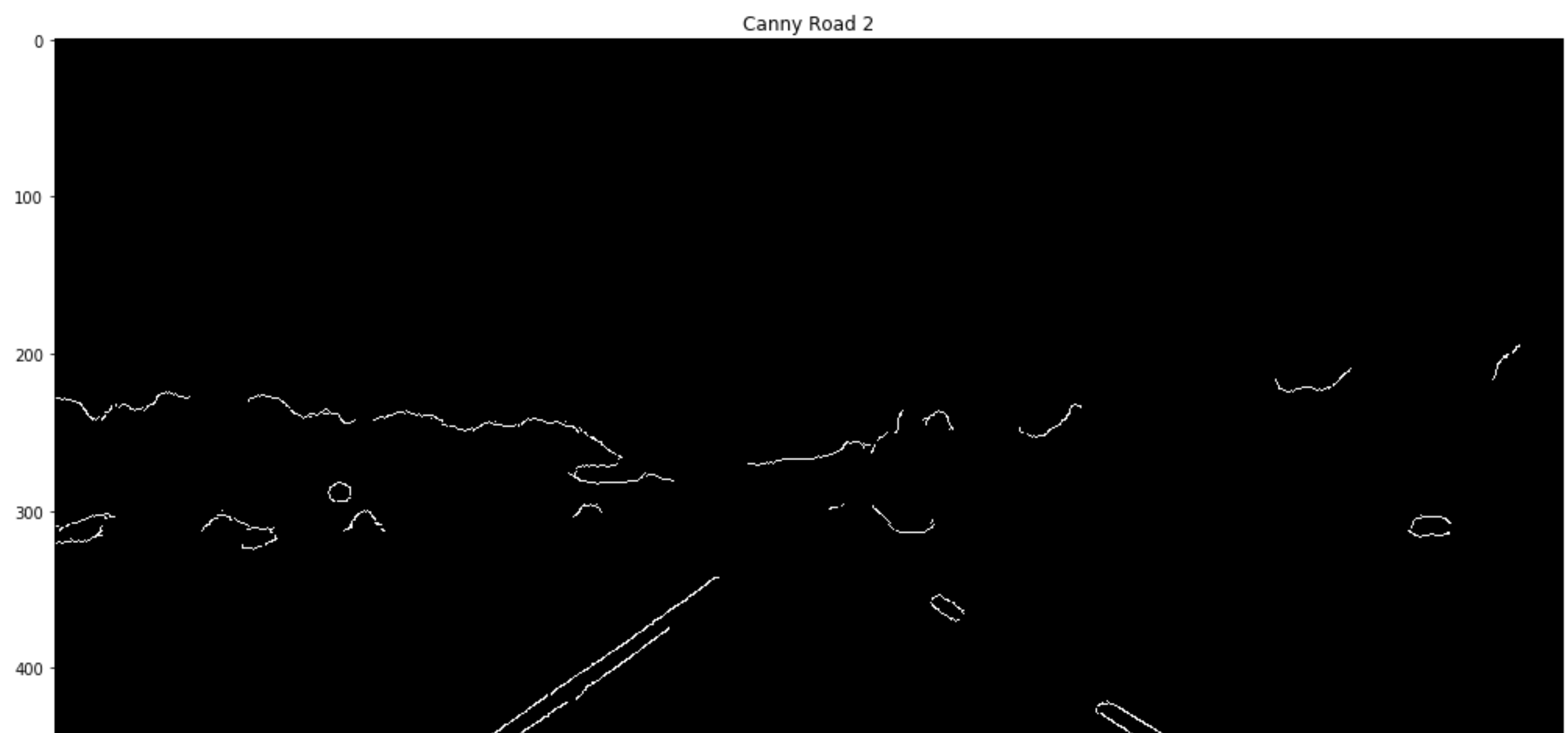
```
# Apply Canny
# define lower and upper thresholds for hysteresis
lower = 0
upper = 70

road_edges1 = canny(img_gauss1,lower,upper)
road_edges2 = canny(img_gauss2,lower,upper)
road_edges3 = canny(img_gauss3,lower,upper)
road_edges4 = canny(img_gauss4,lower,upper)
road_edges5 = canny(img_gauss5,lower,upper)
road_edges6 = canny(img_gauss6,lower,upper)

# plot the first canny edge detected image
f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 1')
ax1.imshow(road_edges1, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 2')
ax1.imshow(road_edges2, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 3')
ax1.imshow(road_edges3, cmap = 'gray')
```

```
ax1.imshow(road_edges3, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 4')
ax1.imshow(road_edges4, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 5')
ax1.imshow(road_edges5, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Canny Road 6')
ax1.imshow(road_edges6, cmap = 'gray')
```

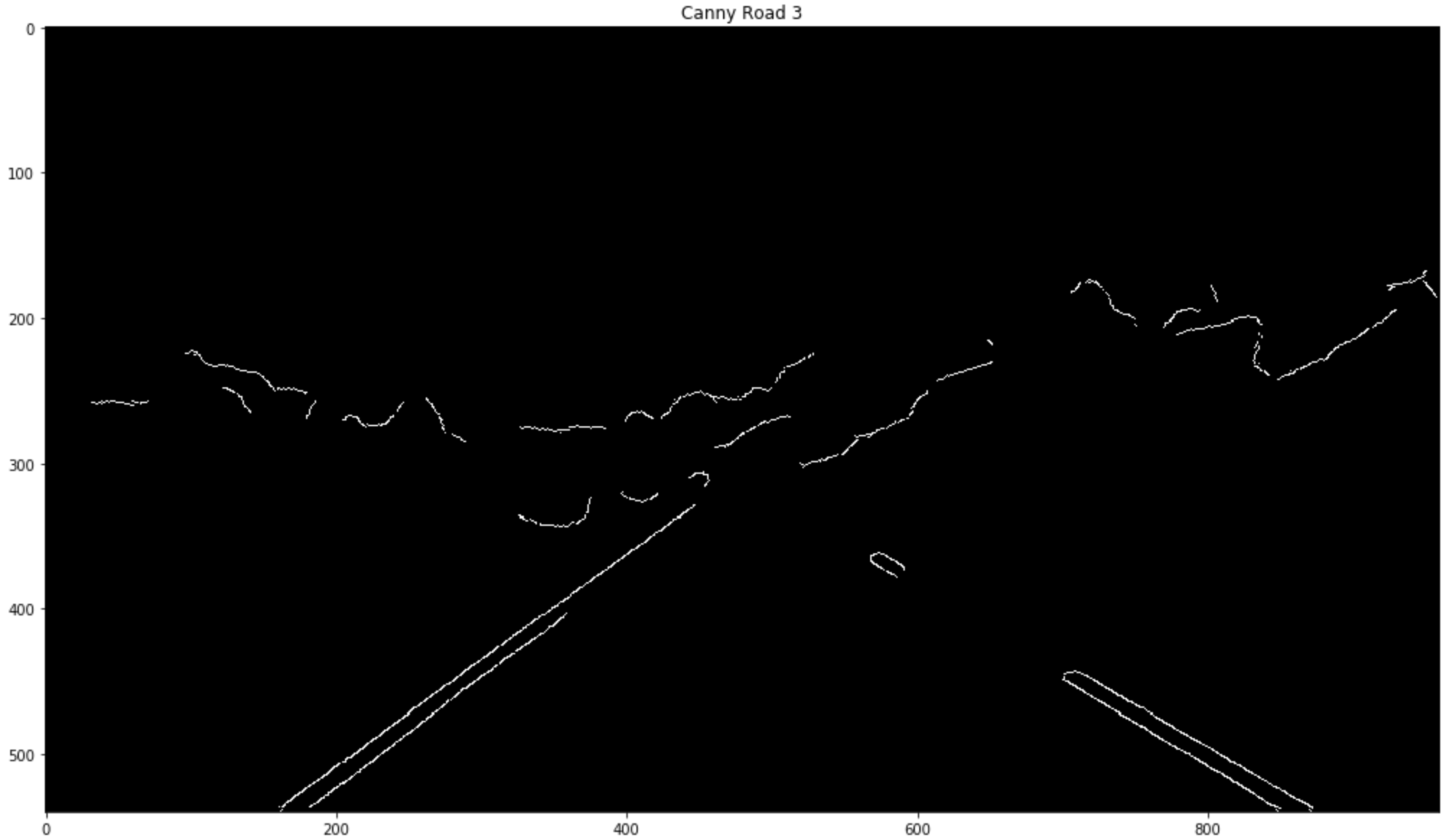
Out[182]:

```
<matplotlib.image.AxesImage at 0x13fb0c668>
```

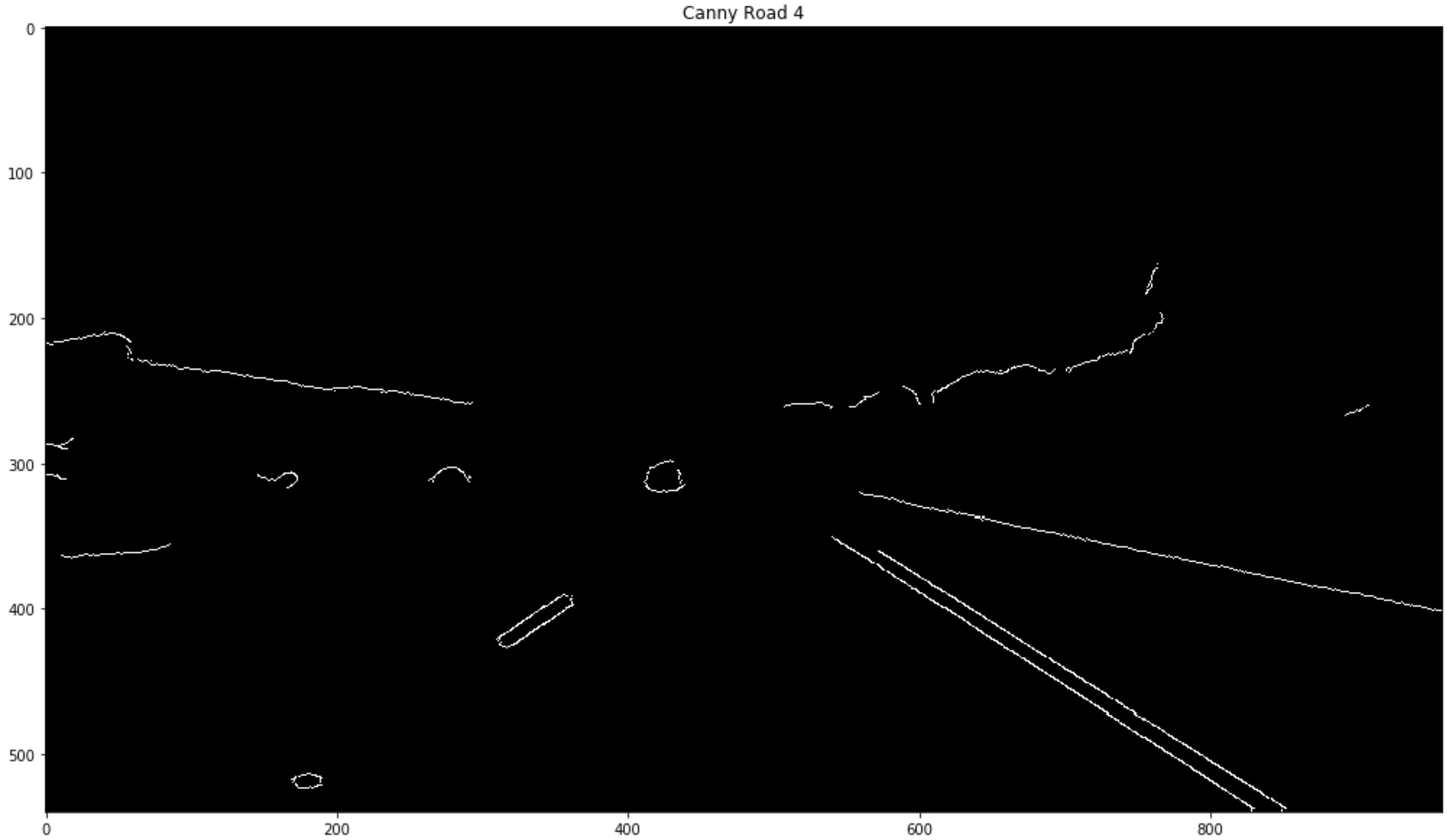




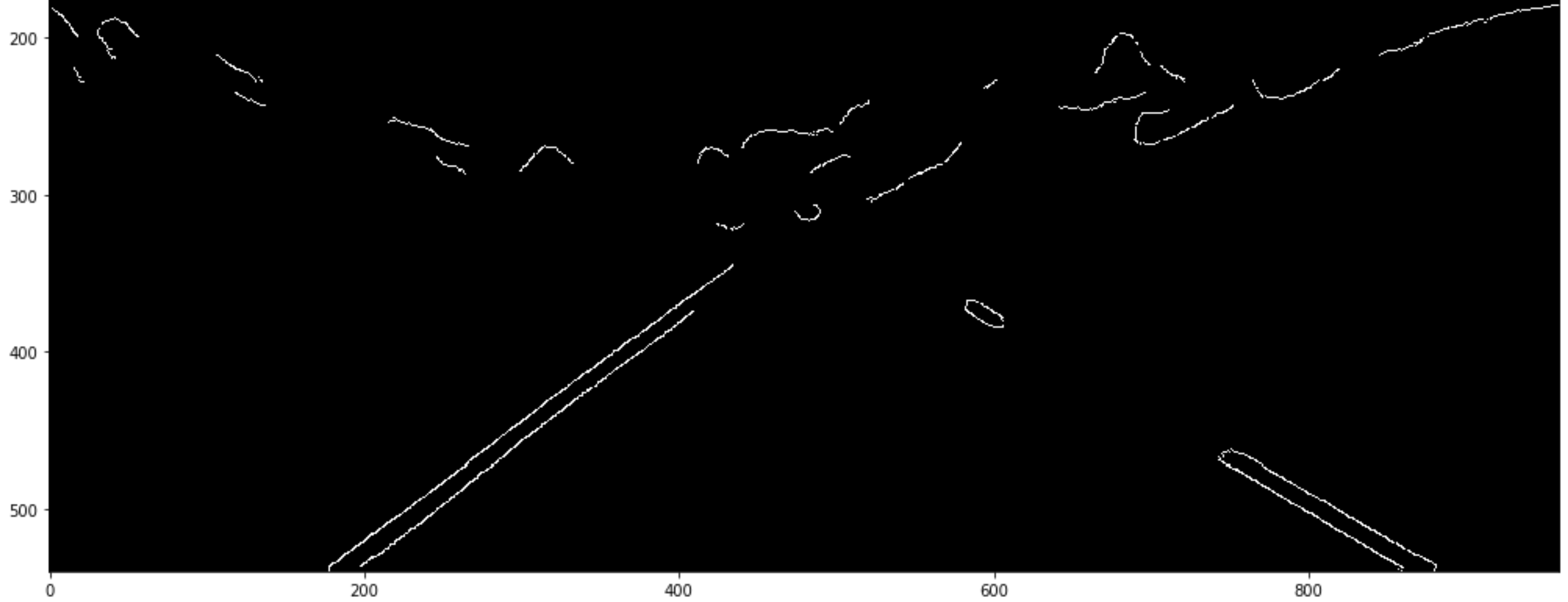
Canny Road 3



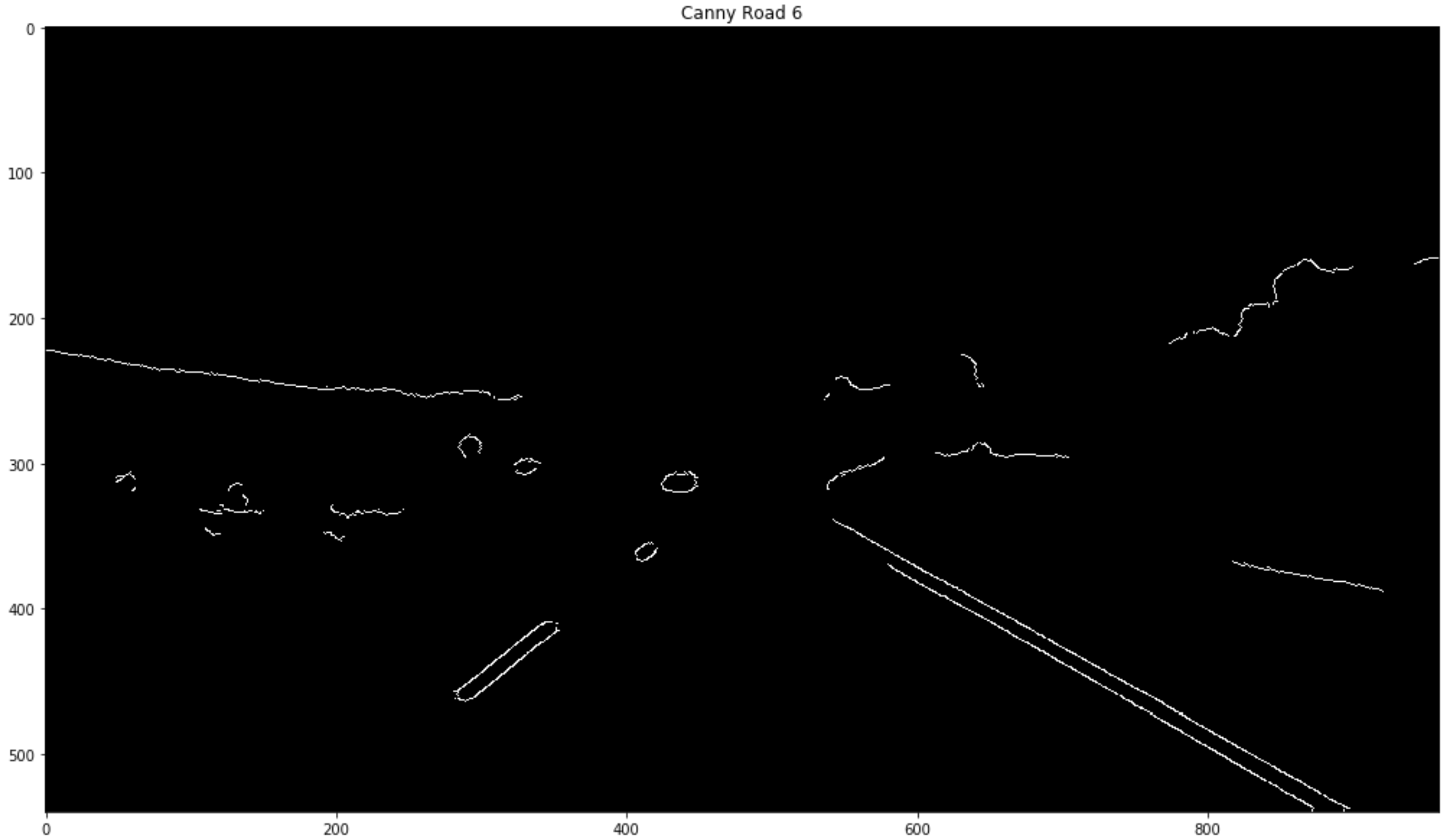
Canny Road 4



Canny Road 5


200
300
400
500
0
200
400
600
800

Canny Road 6

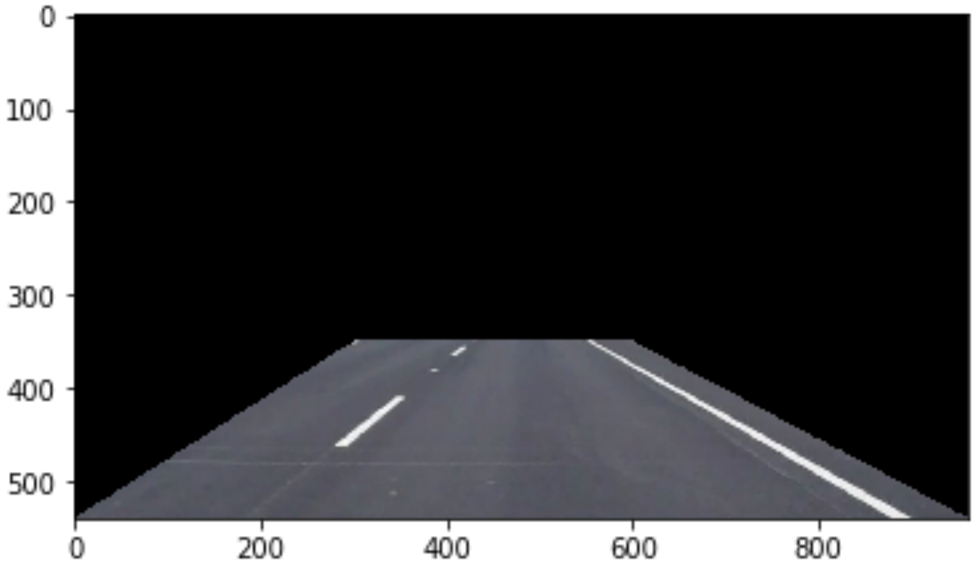

0
100
200
300
400
500
0
200
400
600
800

In [183]:

```
# Obtain the coordinates, to generate a proper region of interest
# We will use these coordinates later on, for now we just want to try them out with

mask_test = np.copy(img6)

coordinates = np.array([[[0,540],[960,540],[600,350], [300,350]]])

# Apply Region of Interest to crop
masked_test_img1 = region_of_interest(mask_test, coordinates)
plt.imshow(masked_test_img1)
```

Out[183]:

```
<matplotlib.image.AxesImage at 0x1302997b8>
```



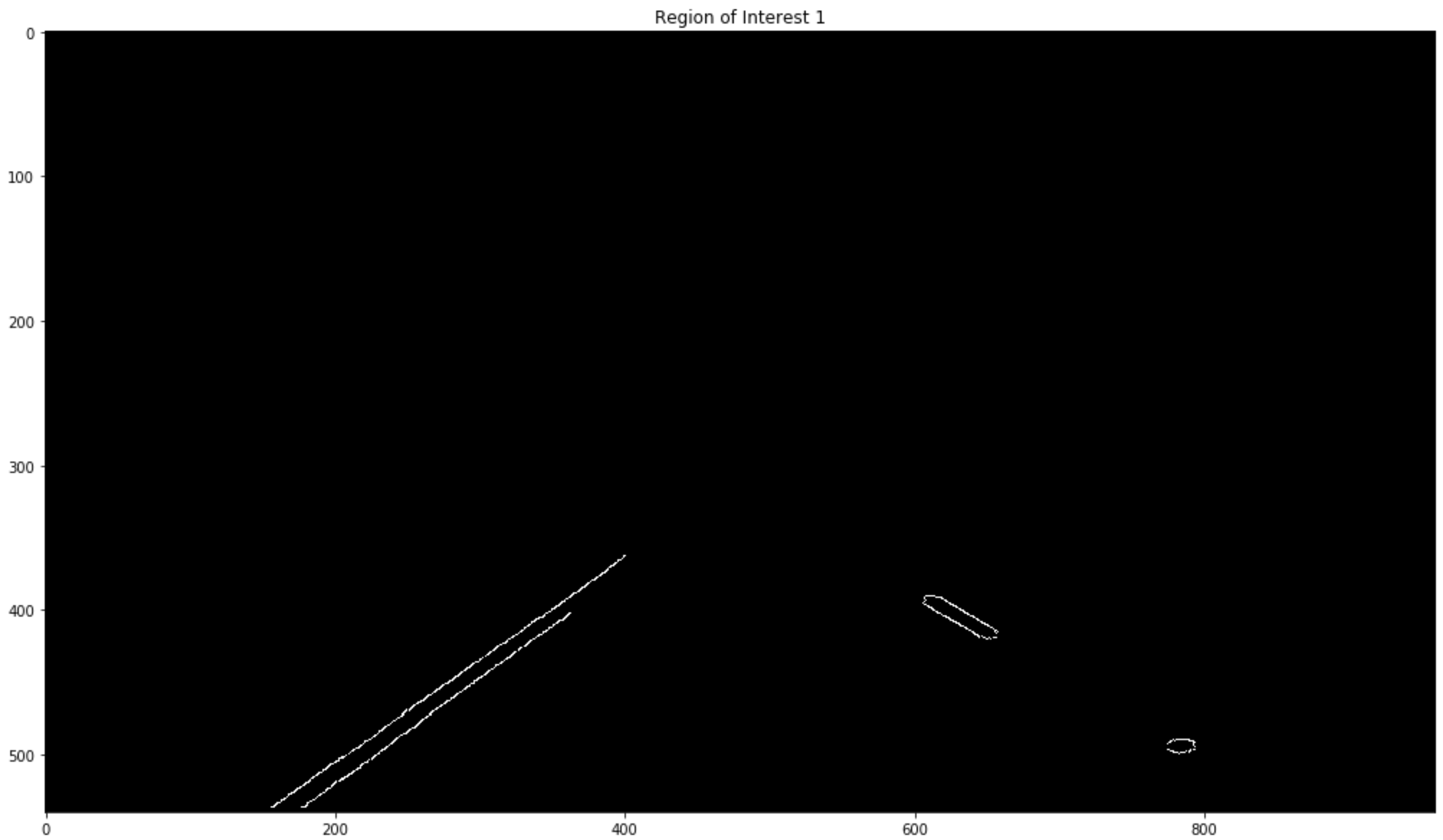
In [184]:

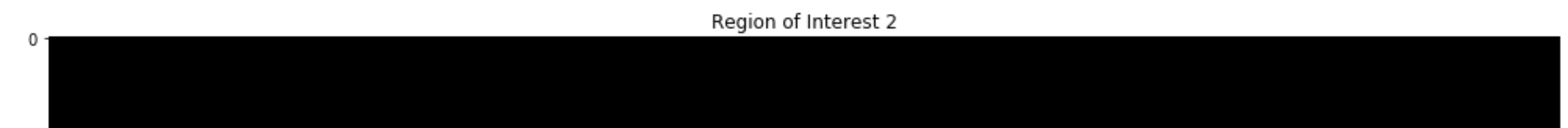
```
# Crop the image so we get only a region of interest with the coordinates we adquire

masked_img1 = np.copy(road_edges1)
masked_img2 = np.copy(road_edges2)
masked_img3 = np.copy(road_edges3)
masked_img4 = np.copy(road_edges4)
masked_img5 = np.copy(road_edges5)
masked_img6 = np.copy(road_edges6)





# Apply Region of Interest to crop
masked_img1 = region_of_interest(masked_img1, coordinates)
masked_img2 = region_of_interest(masked_img2, coordinates)
masked_img3 = region_of_interest(masked_img3, coordinates)
masked_img4 = region_of_interest(masked_img4, coordinates)
masked_img5 = region_of_interest(masked_img5, coordinates)
masked_img6 = region_of_interest(masked_img6, coordinates)
```

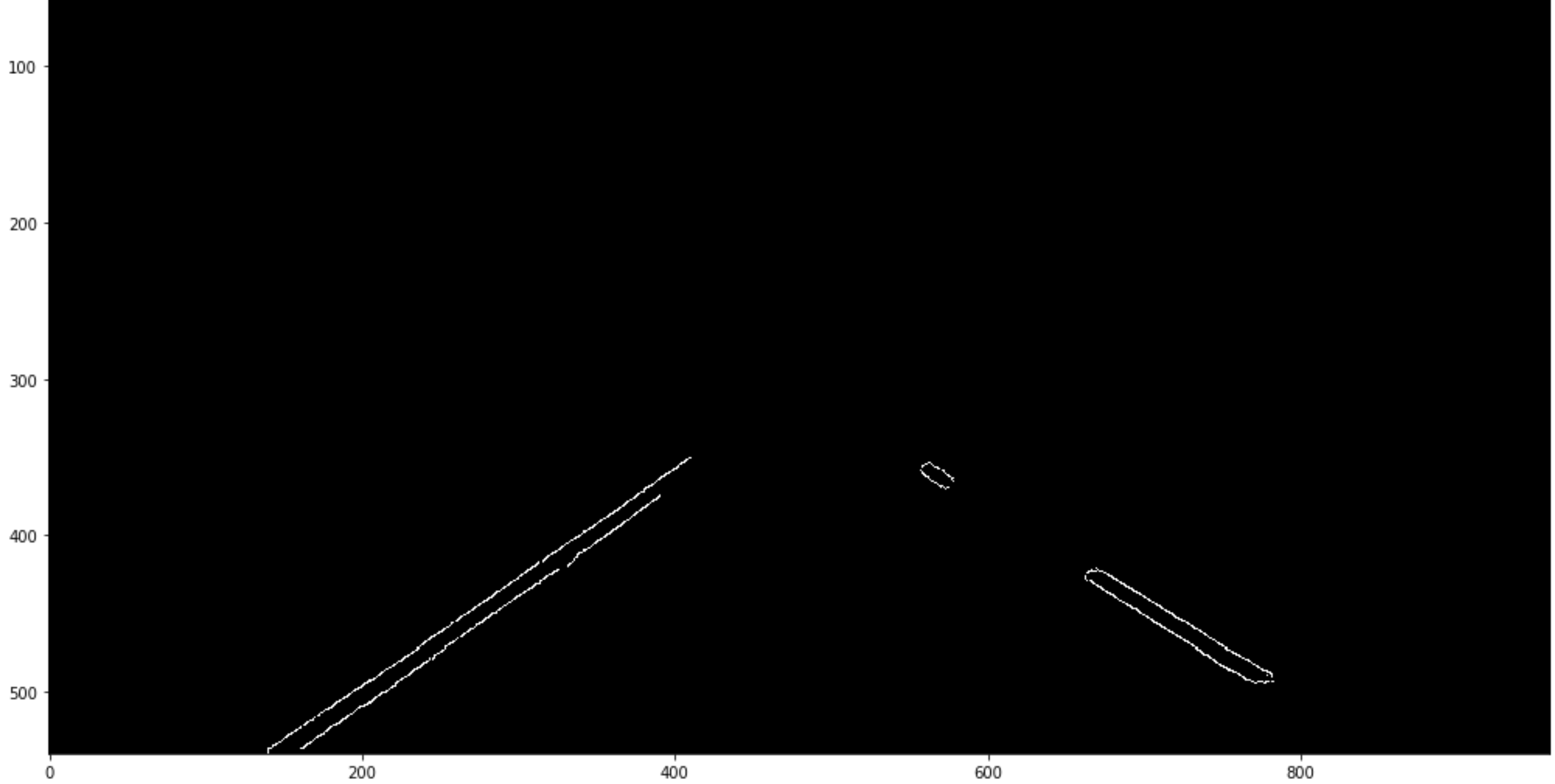
```
f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 1')
ax1.imshow(masked_img1, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 2')
ax1.imshow(masked_img2, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 3')
ax1.imshow(masked_img3, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 4')
ax1.imshow(masked_img4, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 5')
ax1.imshow(masked_img5, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Region of Interest 6')
ax1.imshow(masked_img6, cmap = 'gray')
```

Out[184]:

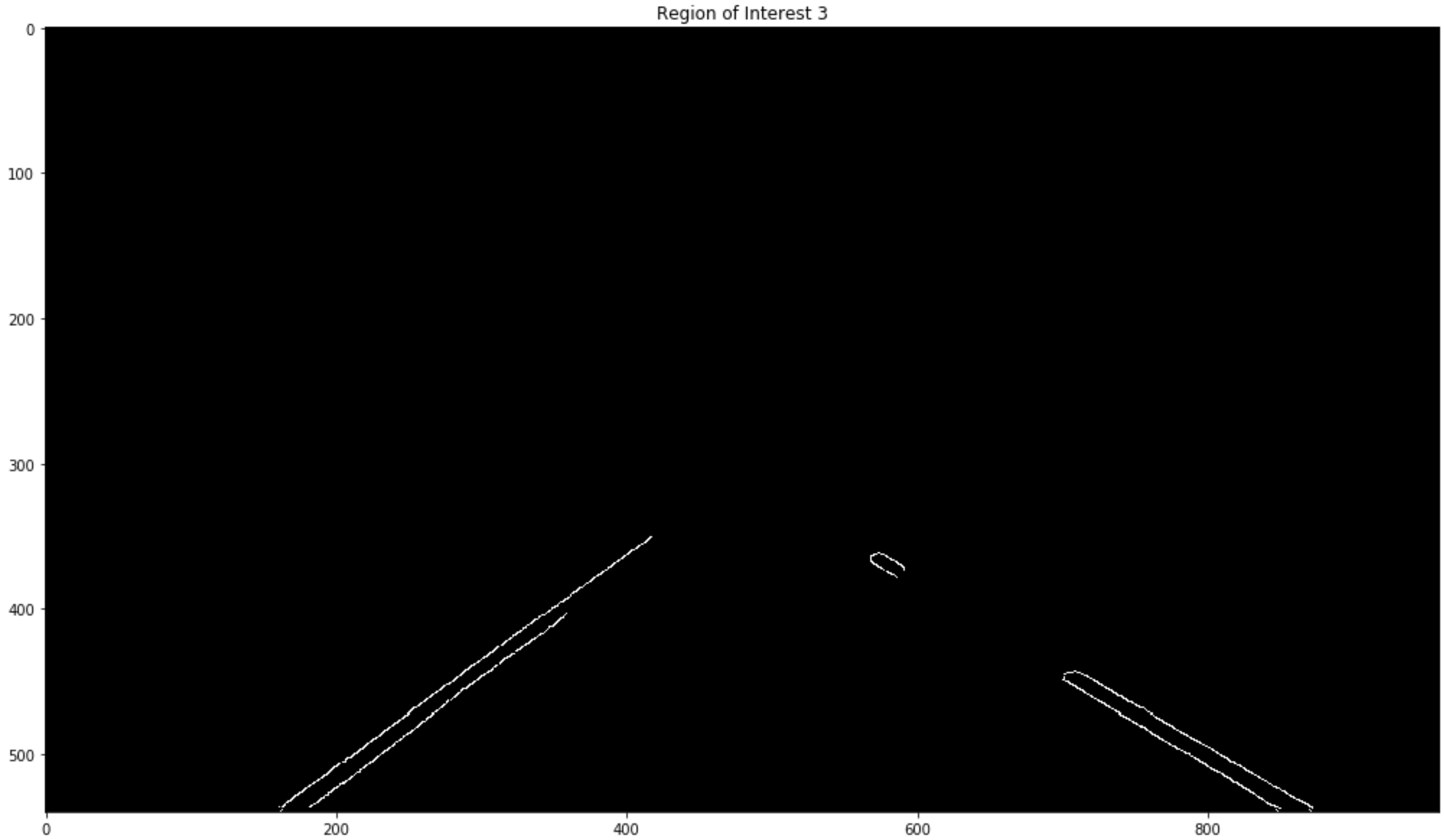
<matplotlib.image.AxesImage at 0x13ed171d0>



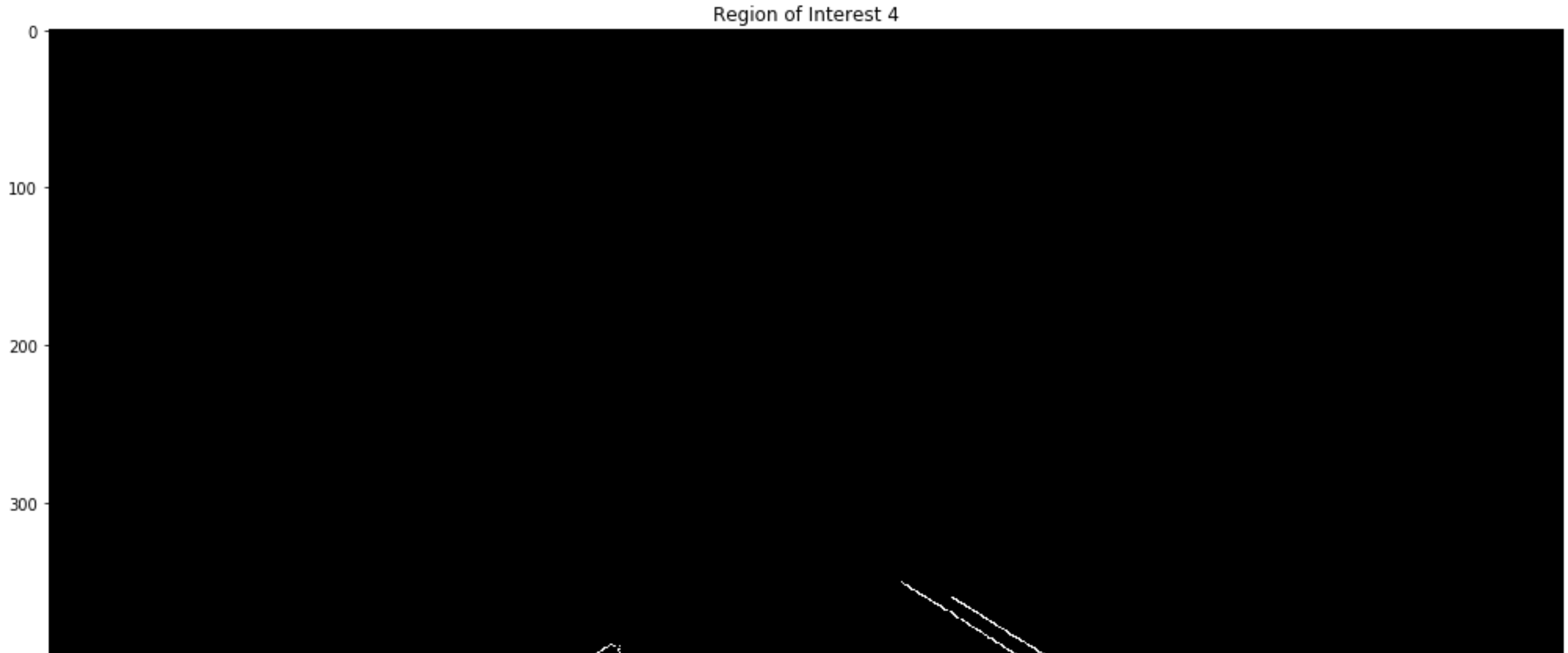


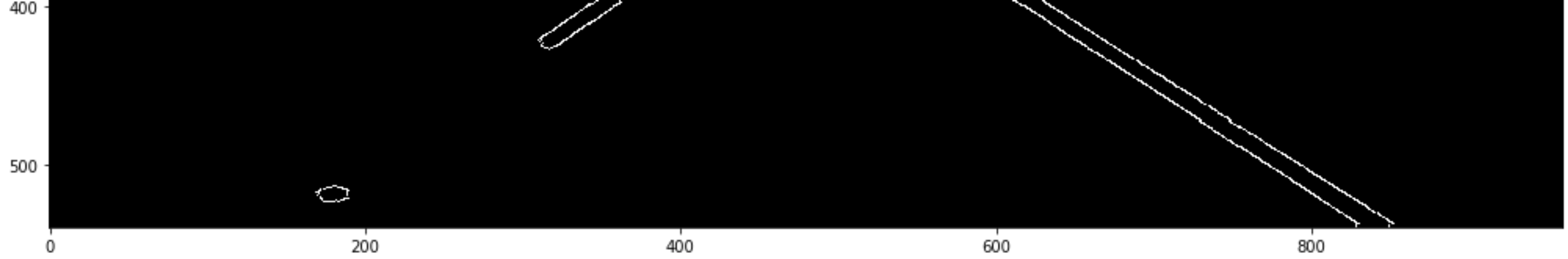


Region of Interest 3

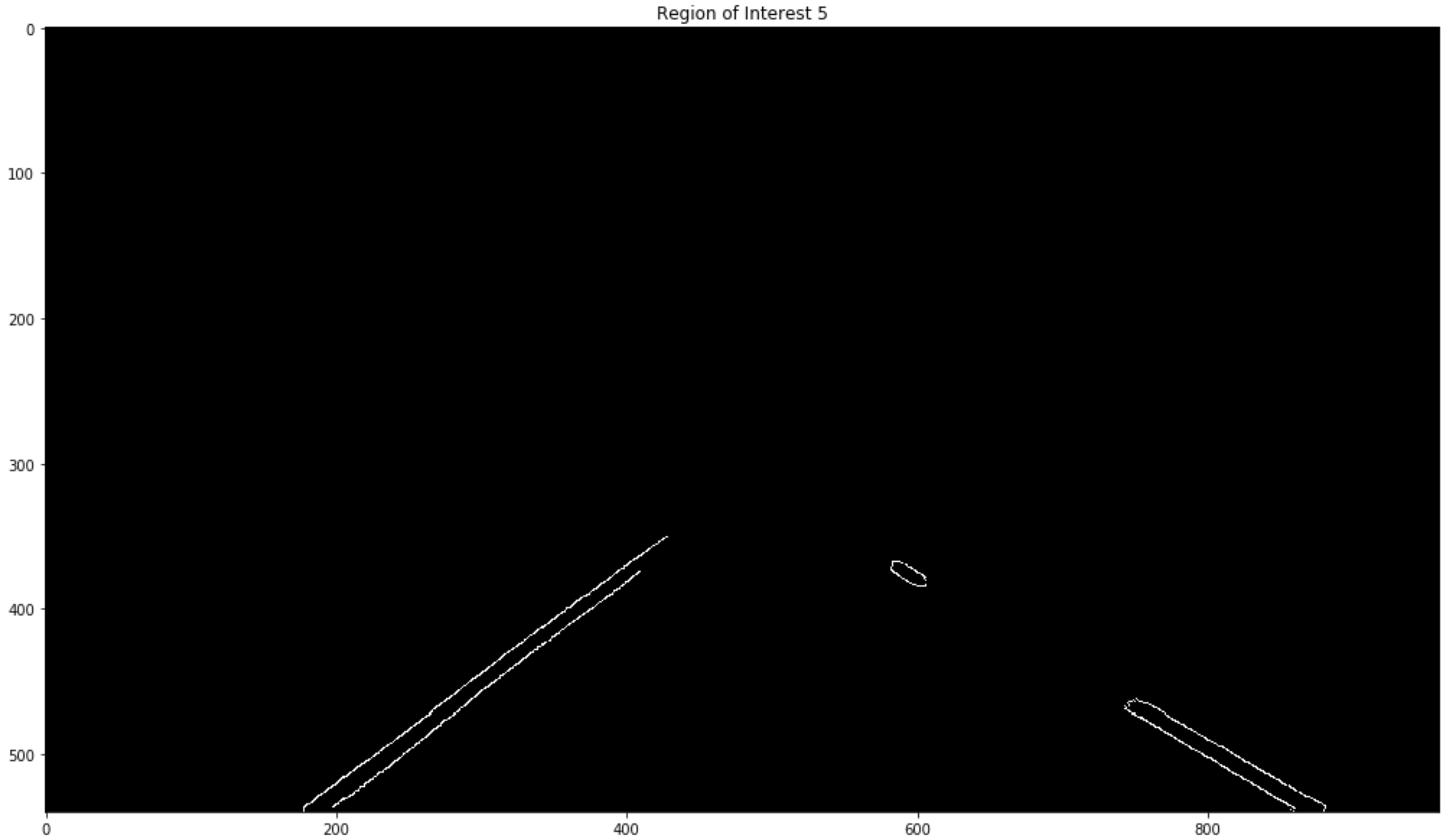


Region of Interest 4

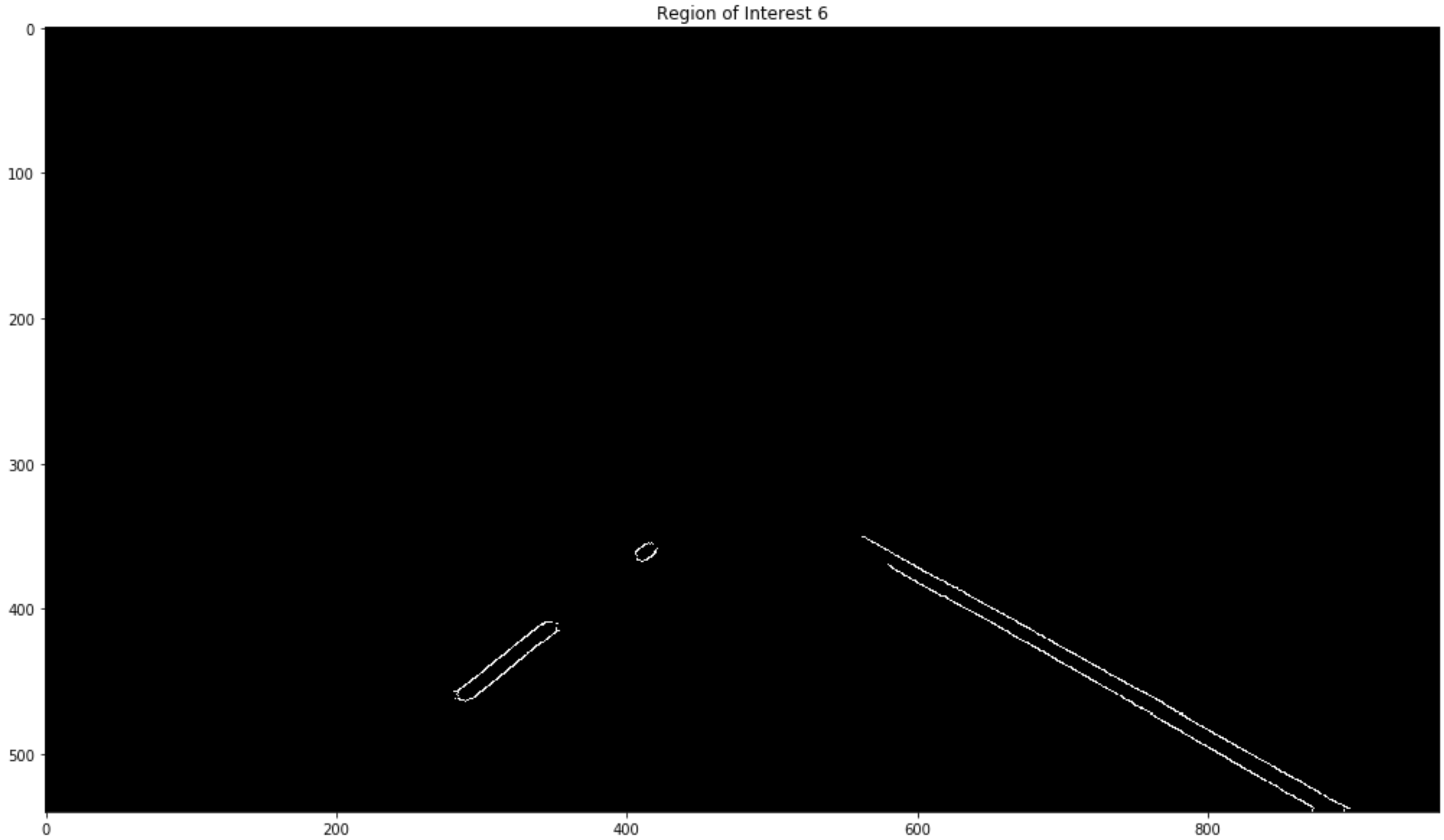




Region of Interest 5



Region of Interest 6



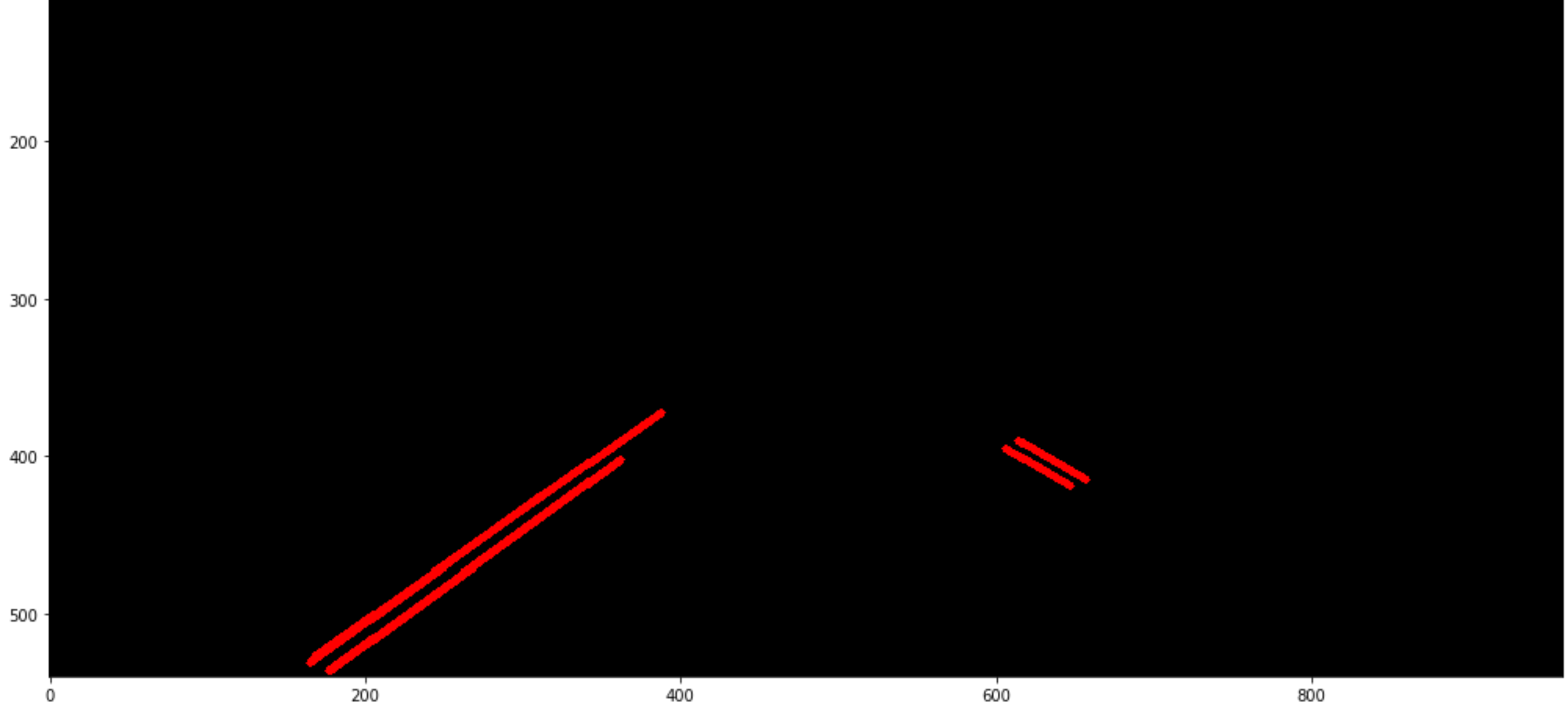
In [185]:

```
# Hough Lines
# define the Hough transform parameters
# make a blank the same size as our image to draw on
```

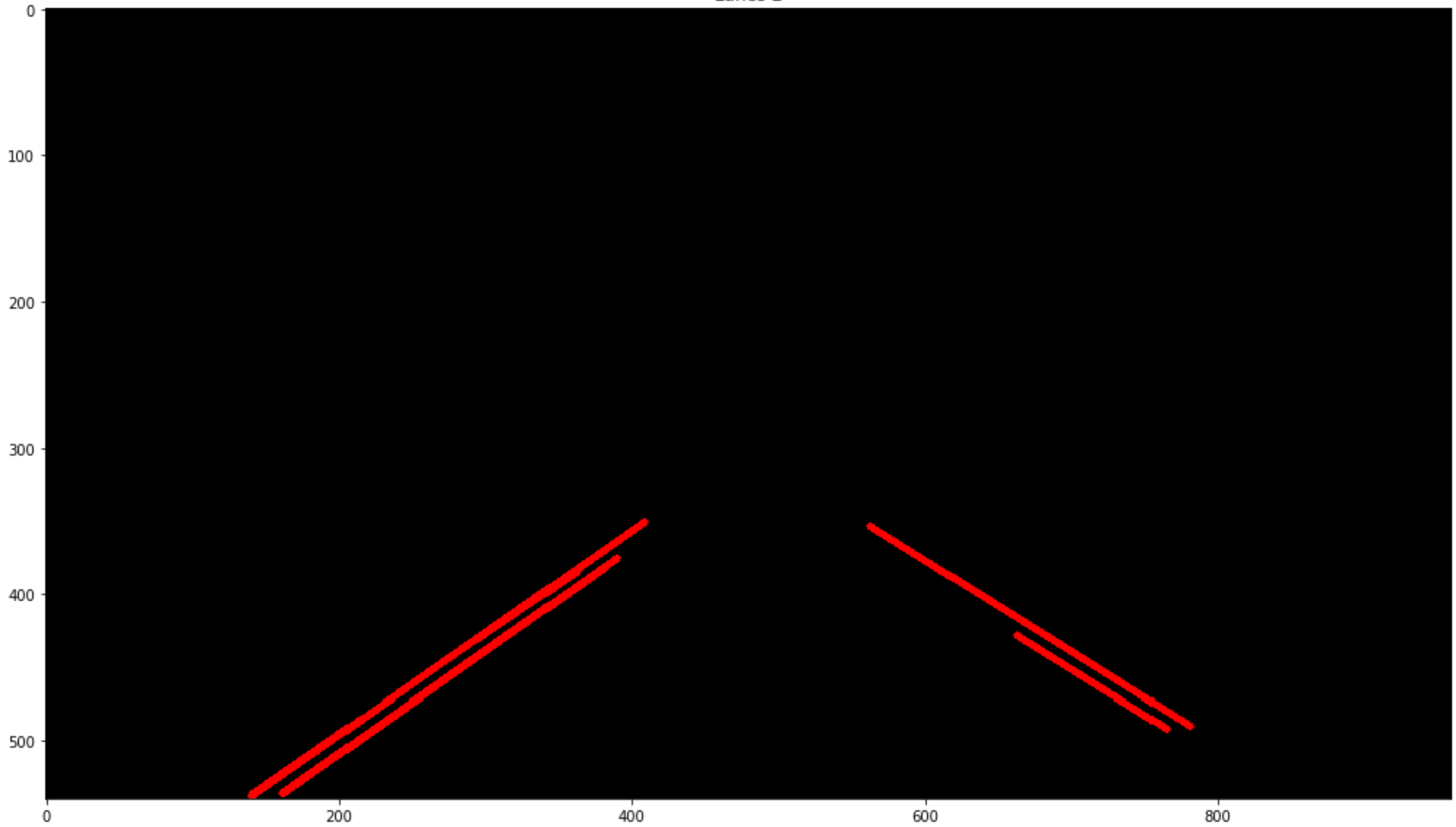
```python
rho = 2
theta = np.pi/180
threshold = 50
min_line_length = 20
max_line_gap = 100







line_image1 = hough_lines(masked_img1,rho,theta,threshold,min_line_length,max_line_
line_image2 = hough_lines(masked_img2,rho,theta,threshold,min_line_length,max_line_
line_image3 = hough_lines(masked_img3,rho,theta,threshold,min_line_length,max_line_
line_image4 = hough_lines(masked_img4,rho,theta,threshold,min_line_length,max_line_
line_image5 = hough_lines(masked_img5,rho,theta,threshold,min_line_length,max_line_
line_image6 = hough_lines(masked_img6,rho,theta,threshold,min_line_length,max_line_




f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 1')
ax1.imshow(line_image1, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 2')
ax1.imshow(line_image2, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 3')
ax1.imshow(line_image3, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 4')
ax1.imshow(line_image4, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 5')
ax1.imshow(line_image5, cmap = 'gray')

f, (ax1) = plt.subplots(1, 1, figsize=(20,10))
ax1.set_title('Lanes 6')
ax1.imshow(line_image6, cmap = 'gray')
```

Out[185]:

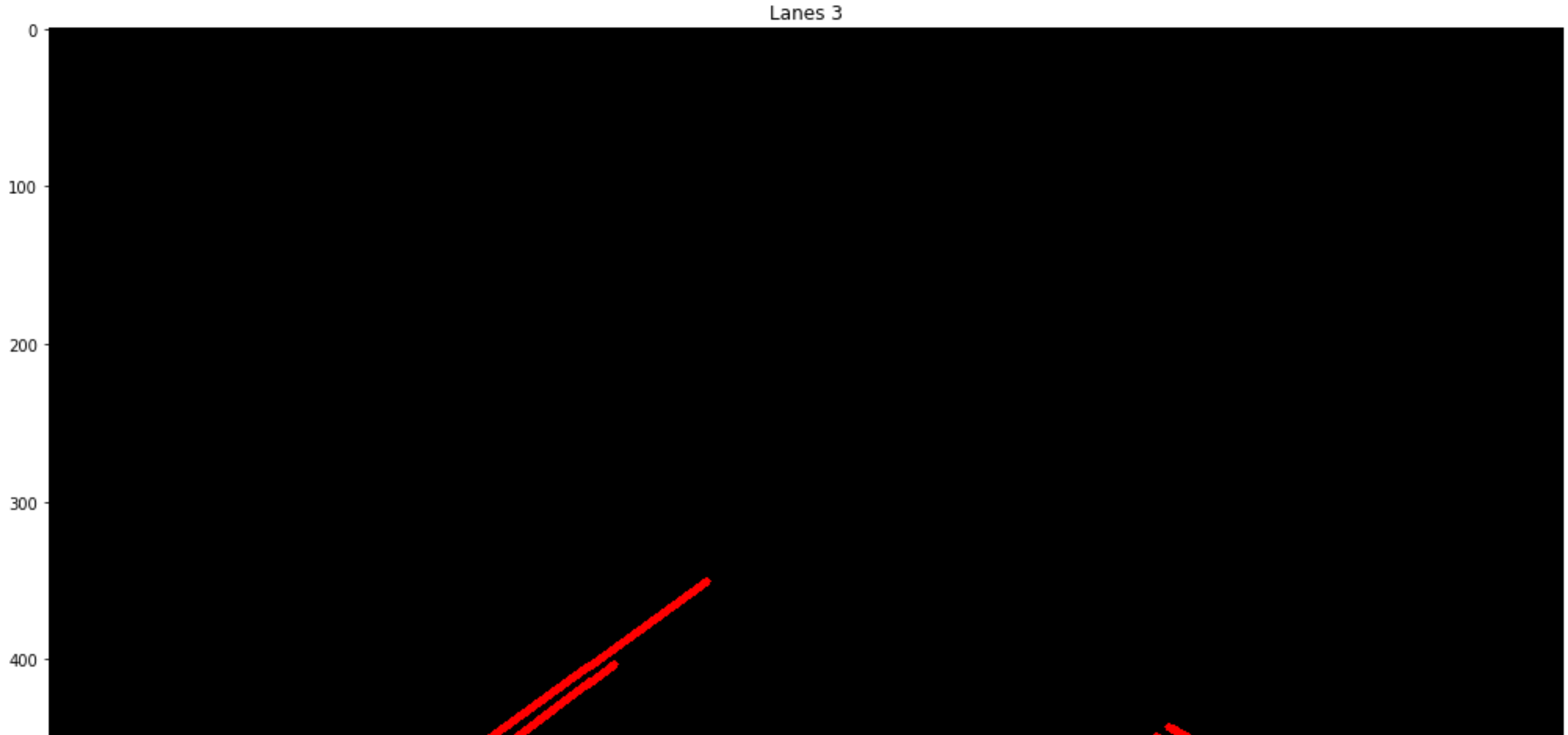
```
<matplotlib.image.AxesImage at 0x134620710>
```



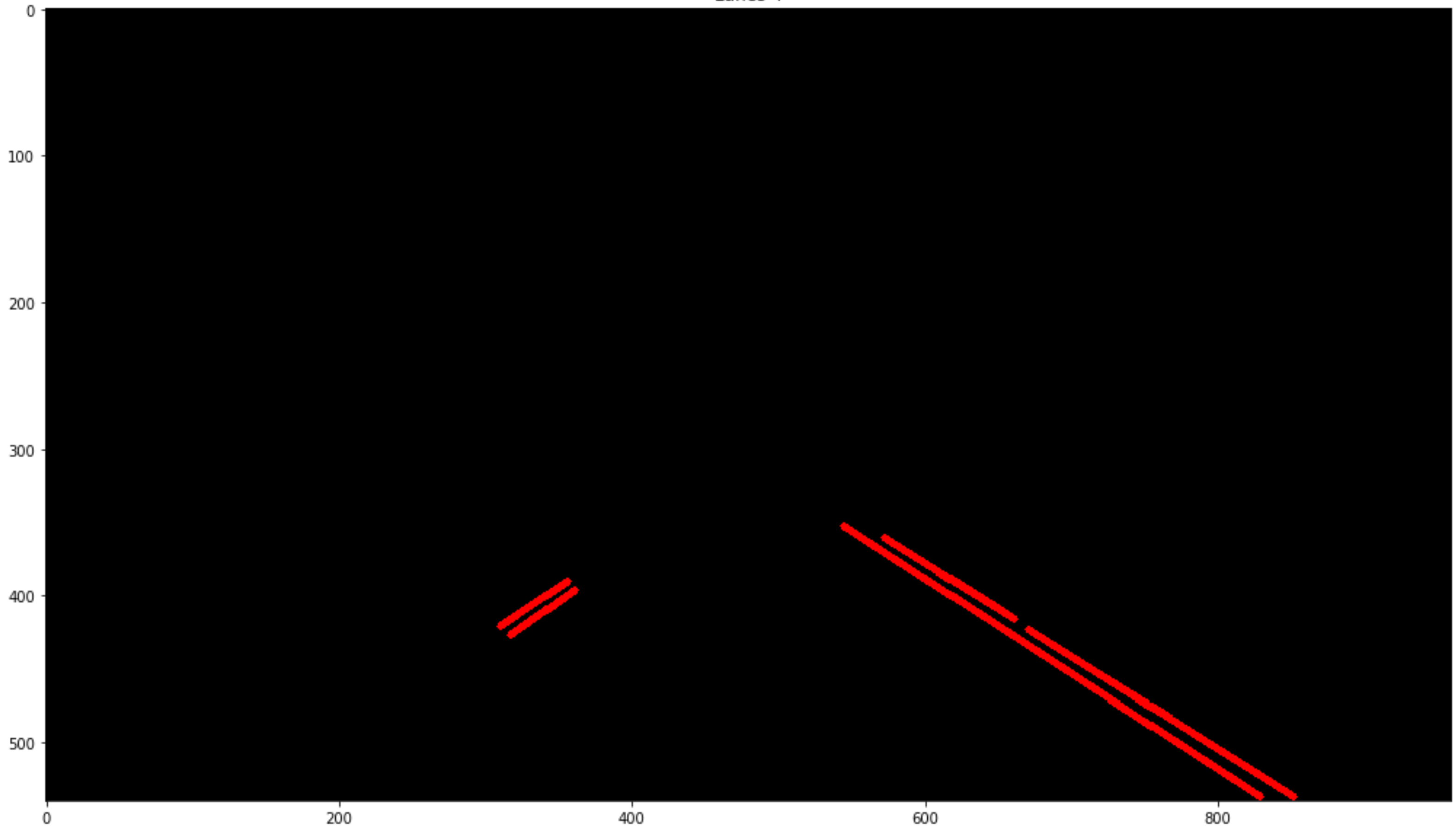
Lanes 2



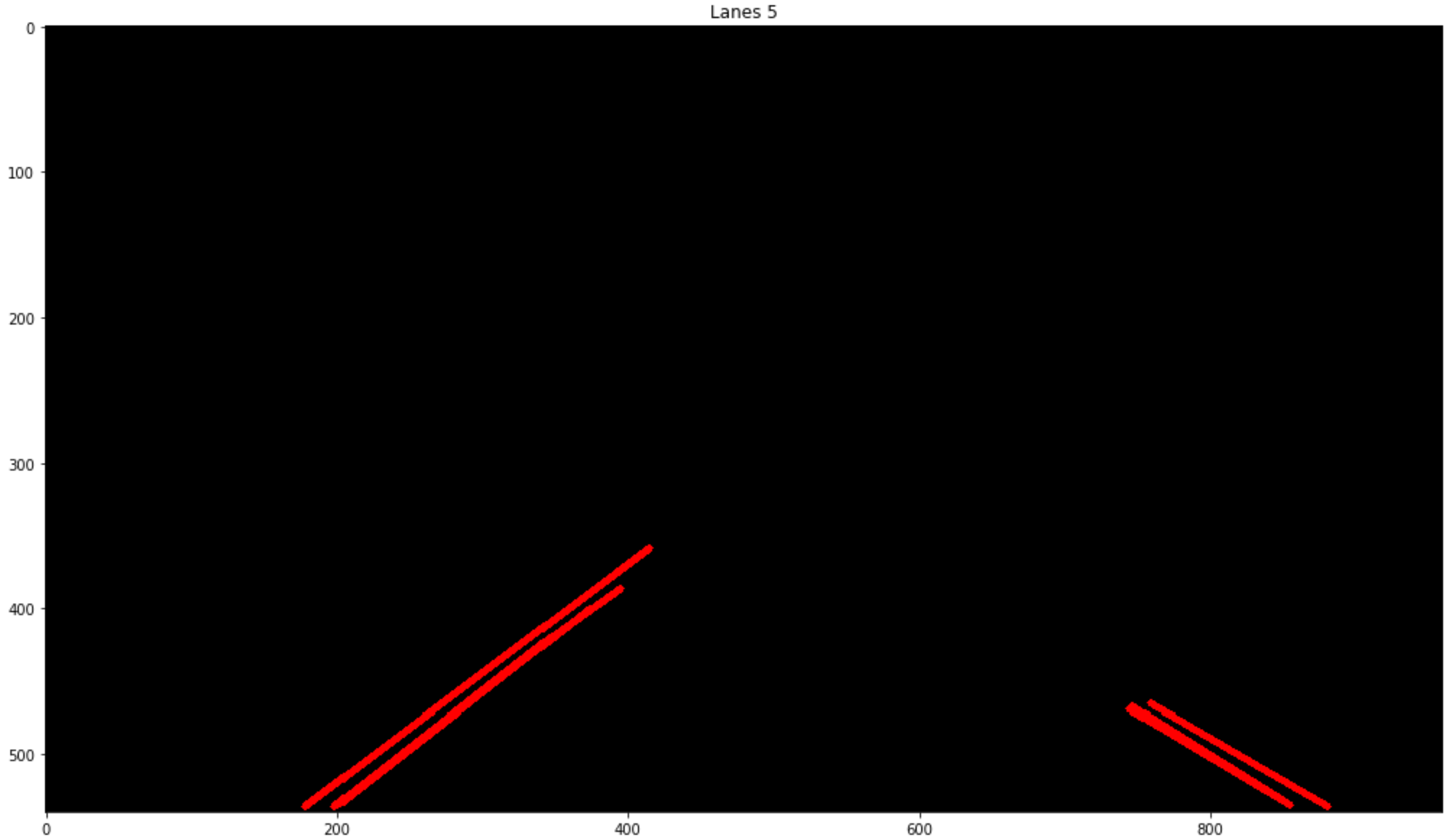
Lanes 3



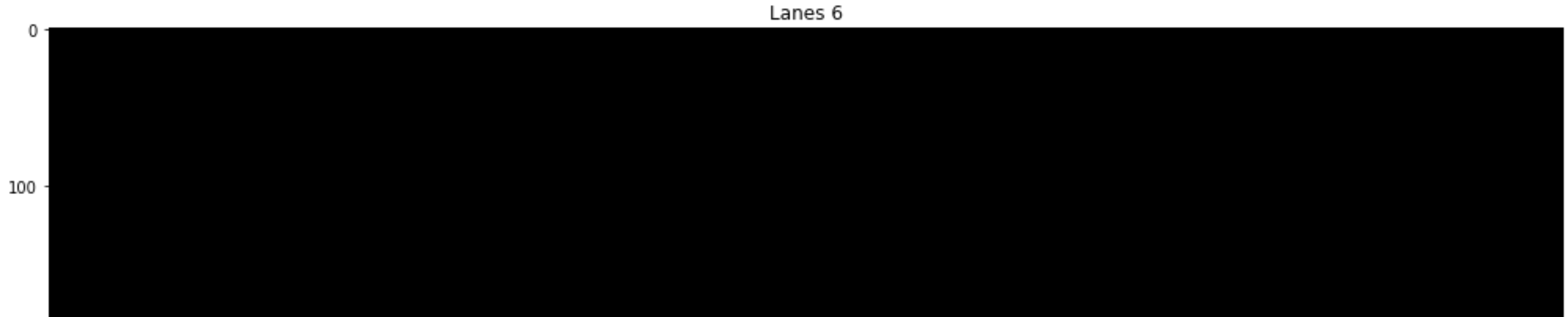
Lanes 4

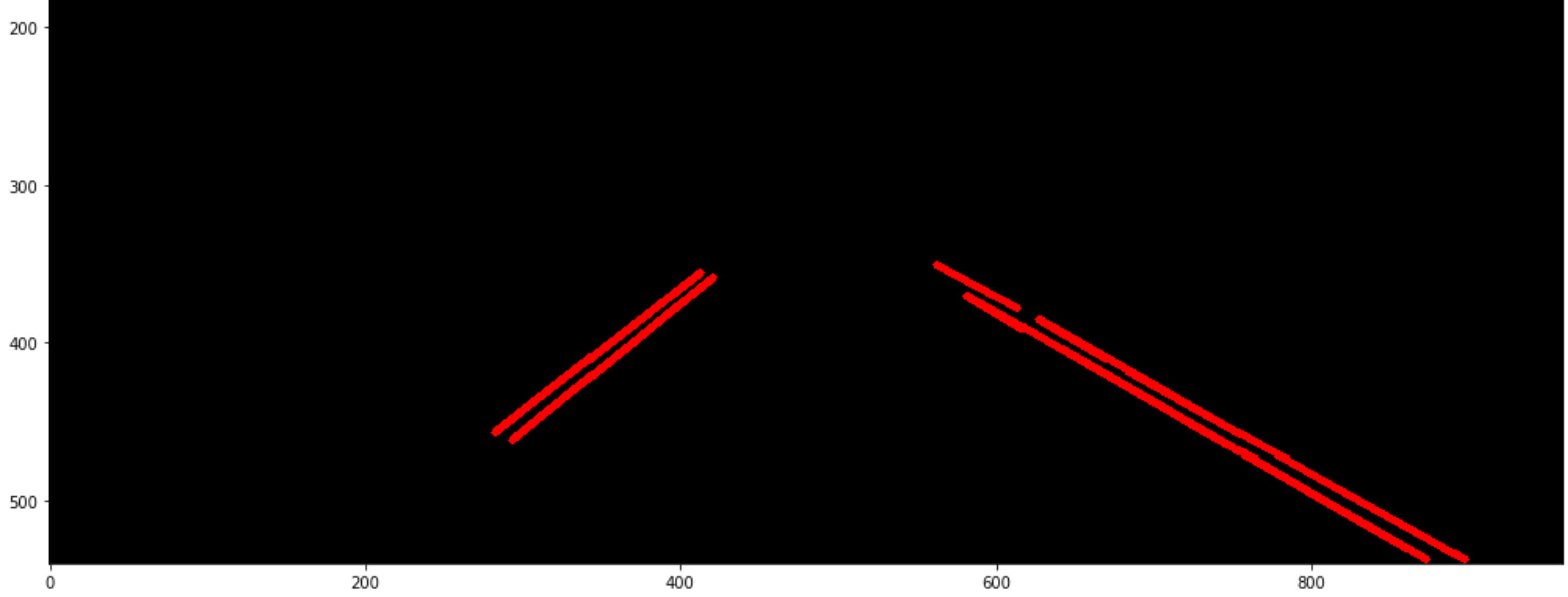


Lanes 5



Lanes 6





Improve the draw_lines() function

At this point, you should have the Hough line segments drawn onto the road. Extend your code to define a line to run the full length of the visible lane based on the line segments you identified with the Hough Transform. Try to average and/or extrapolate the line segments you've detected to map out the full extent of the lane lines. The output should draw a single, solid line over the left lane line and a single, solid line over the right lane line. The lines should start from the bottom of the image and extend out to the top of the region of interest.

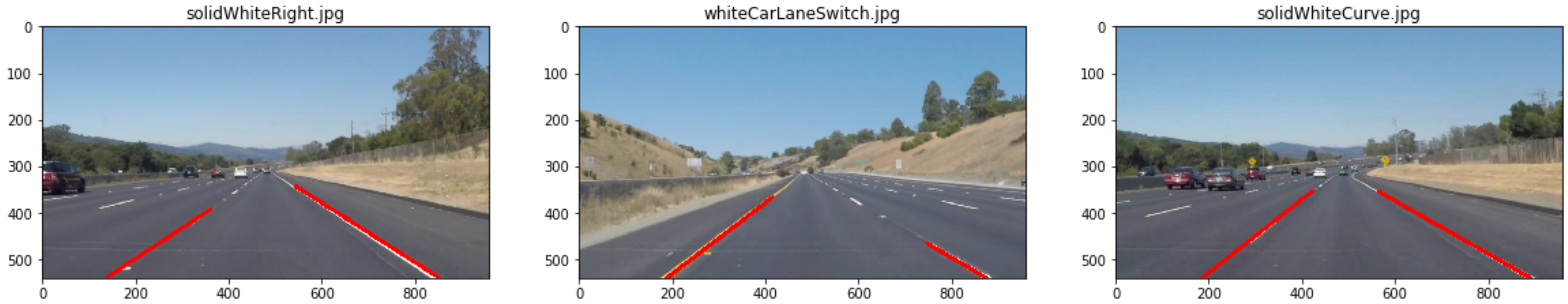
In [186]:

```
# TODO: Build your pipeline that will draw complete lane lines on the test_images
```

In [187]:

```
line_segments = []
line_segments.append(masked_img1)
line_segments.append(masked_img2)
line_segments.append(masked_img3)
line_segments.append(masked_img4)
line_segments.append(masked_img5)
line_segments.append(masked_img6)



f, plots = plt.subplots((len(files)+3-1)//3, 3, figsize=(20,10))
plots = [plot for sublist in plots for plot in sublist]

for original_img, image, plot in zip(images, line_segments, plots):
    final_image = np.copy(original_img[0])

    # Get Hough Lines
    lines = cv2.HoughLinesP(image, rho, theta, threshold, np.array([]), min_line_len

    # Right line
    xR = []
    yR = []
```

```
    # Left line
    xL = []
    yL = []

    #Fill the vectors
    for line in lines:
        for x1,y1,x2,y2 in line:
            degrees = math.degrees(math.atan2(x1-y1, x2-y2))
            if degrees < 0:
                xR += [x1,x2]
                yR += [y1,y2]
            else:
                xL += [x1,x2]
                yL += [y1,y2]

    # Fit polynomial to x,y points
    z = np.polyfit(xR,yR,1)
    m = z[0]
    b = z[1]

    # Calculate and draw right line from points
    x1 = int((final_image.shape[0] - b) / m)
    x2 = max(xR)
    # y = mx+b
    y1 = int(m * x1 + b)
    y2 = int(m * x2 + b)

    #Draw line
    cv2.line(final_image,(x1,y1),(x2,y2),(255,0,0),8)

    # Fit polynomial to x,y points
    z = np.polyfit(xL,yL,1)
    m = z[0]
    b = z[1]

    # Calculate and draw left line from points
    x1 = min(xL)
    x2 = int((final_image.shape[0] - b) / m)
    # y = mx+b
    y1 = int(m * x1 + b)
    y2 = int(m * x2 + b)

    #Draw line
    cv2.line(final_image,(x1,y1),(x2,y2),(255,0,0),8)

    plot.set_title(original_img[1])
    plot.imshow(final_image)
```



In [ ]:

Project: Finding Lane Lines on the Road

Develop a pipeline to identify lane lines on the road. You must apply it on a series of individual images, provided in the *test_images* folder.

Once you have a result that looks roughly like the image *line-segments-example* in the examples folder (also shown below), you'll need to try to average and/or extrapolate the line segments you've detected to map out the full extent of the lane lines.

The tools you have are color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection. You are also free to explore and try other techniques that were not presented. Your goal is piece together a pipeline to detect the line segments in the image, then average/extrapolate them and draw them onto the image for display (as below).

Your output should look something like this (above) after detecting line segments using the helper functions below

Your goal is to connect/average/extrapolate line segments to get output like this

Import Packages

In [175]:

```
```

Read in an Image

In [176]:

This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)

Out[176]:

<matplotlib.image.AxesImage at 0x1374a9da0>

Ideas for Lane Detection Pipeline

Some OpenCV functions that might be useful for this project are:

`cv2.inRange()` for color selection
`cv2.fillPoly()` for regions selection
`cv2.line()` to draw lines on an image given endpoints
`cv2.addWeighted()` to coadd / overlay two images `cv2.cvtColor()` to grayscale or change color
`cv2.imwrite()` to output images to file
`cv2.bitwise_and()` to apply a mask to an image

Helper Functions

Below are some helper functions to help get you started.

In [177]:

Test Images

Build your pipeline to work on the images in the directory "test_images"

In [178]:

Build a Lane Finding Pipeline

Build the pipeline and run your solution on all test_images.

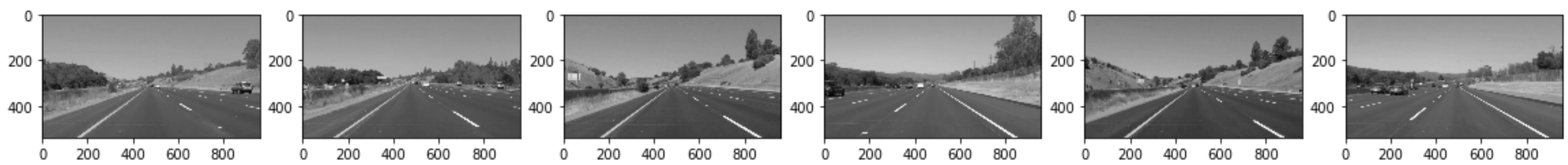
Try tuning the various parameters, especially the low and high Canny thresholds as well as the Hough lines parameters.

In [179]:

In [180]:

Out[180]:

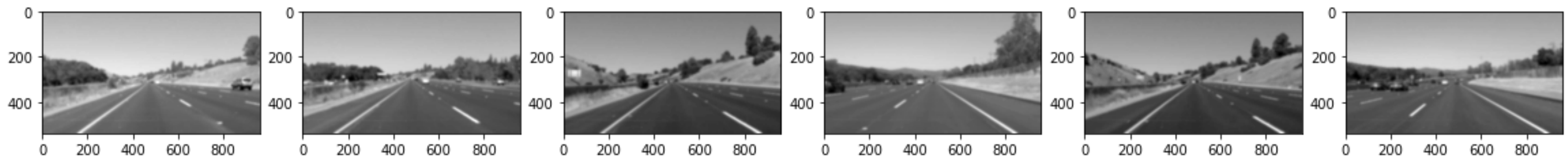
```
<matplotlib.image.AxesImage at 0x132a21cf8>
```



In [181]:

Out[181]:

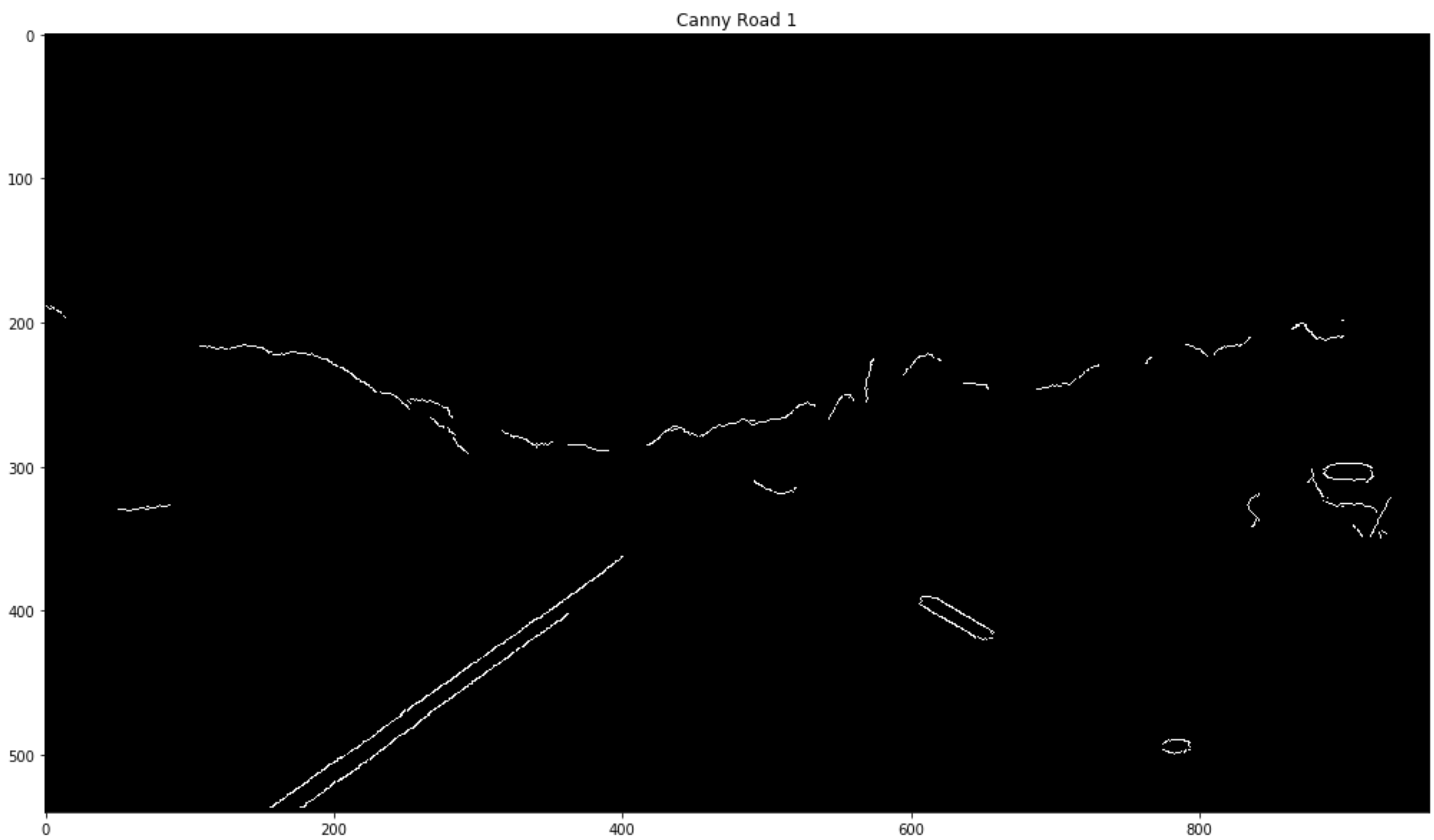
```
<matplotlib.image.AxesImage at 0x13ed2bf28>
```

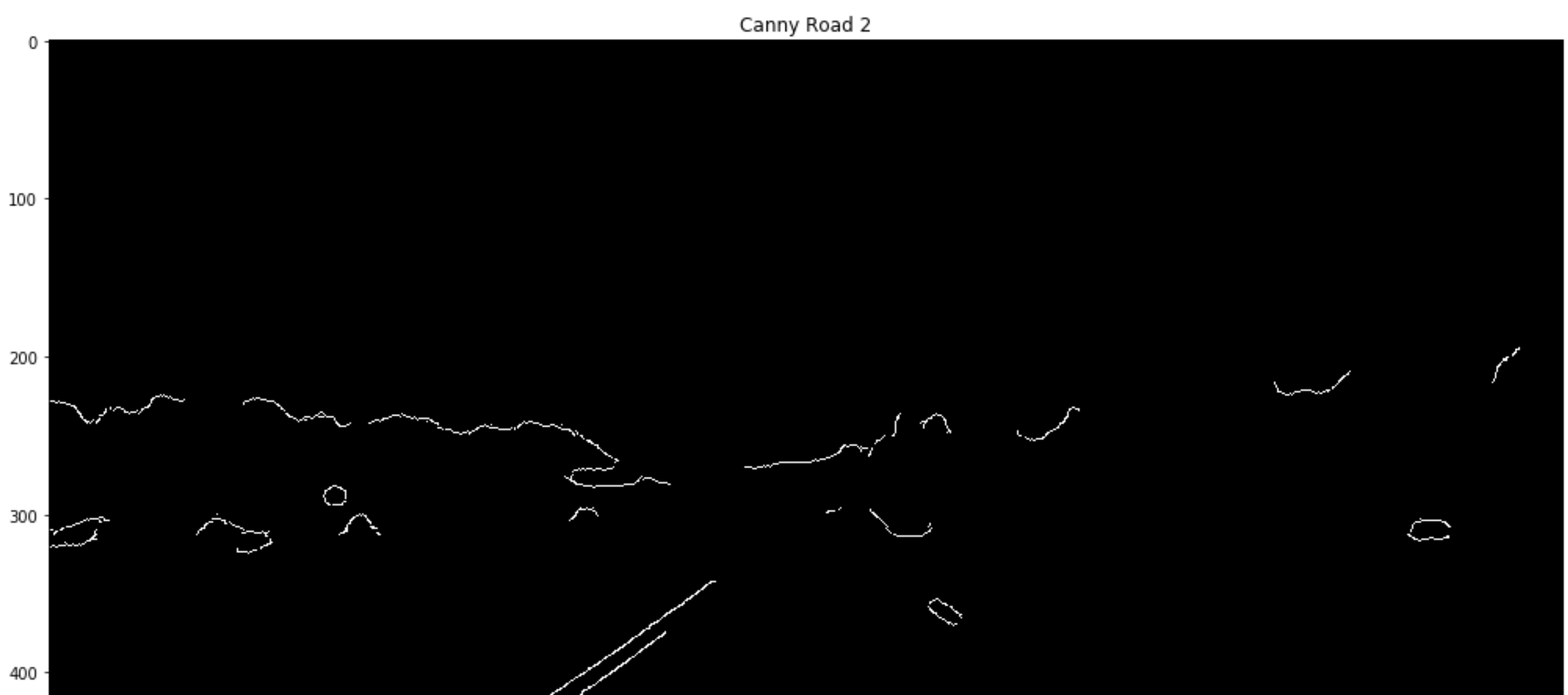


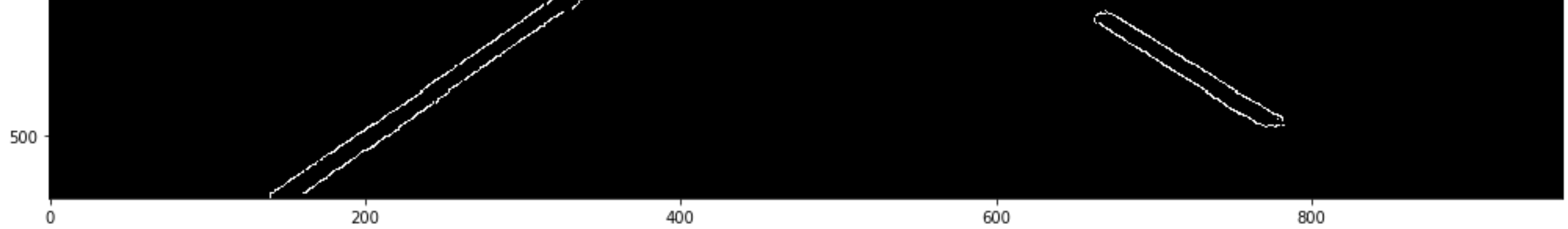
In [182]:

Out[182]:

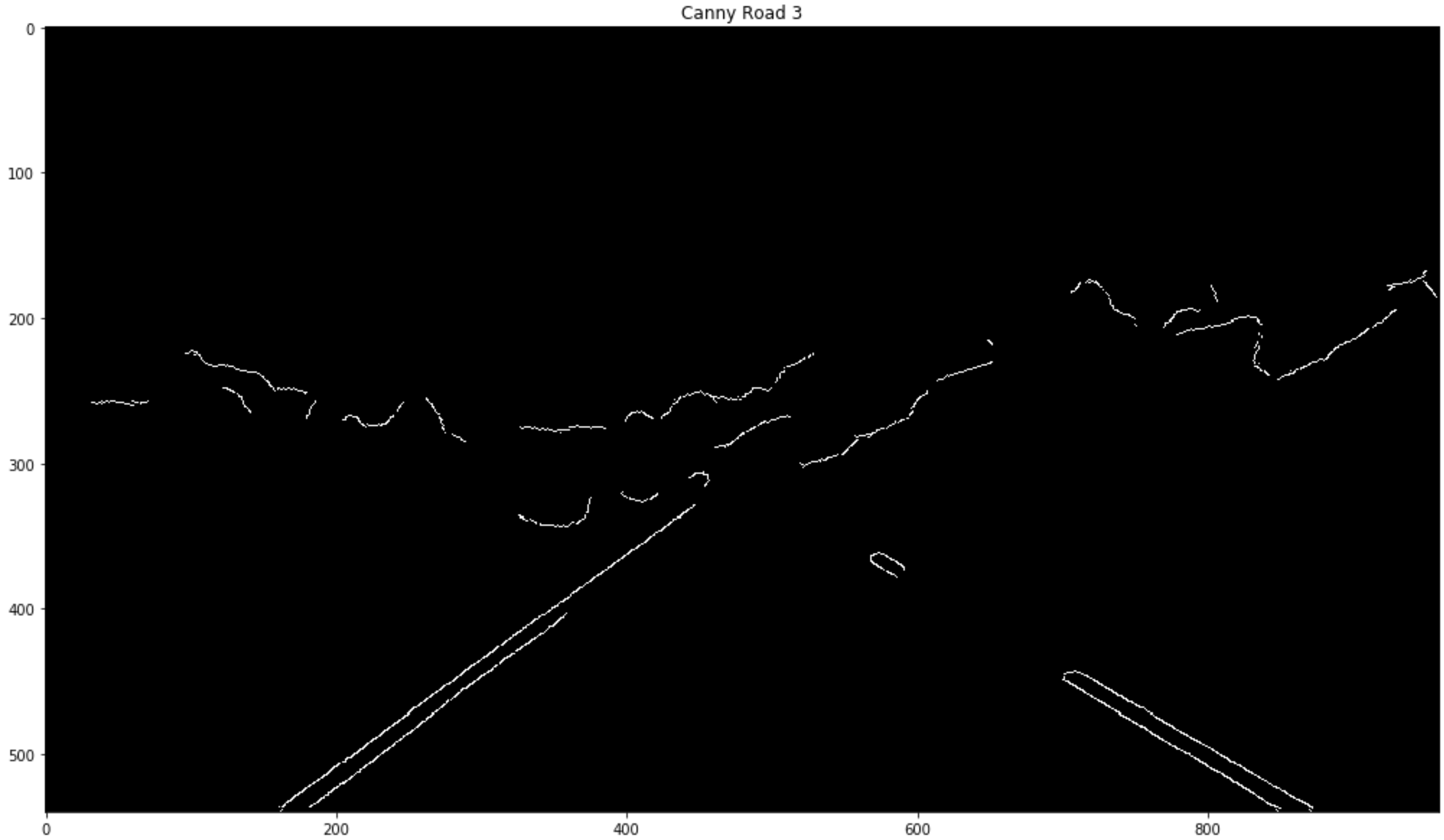
```
<matplotlib.image.AxesImage at 0x13fb0c668>
```



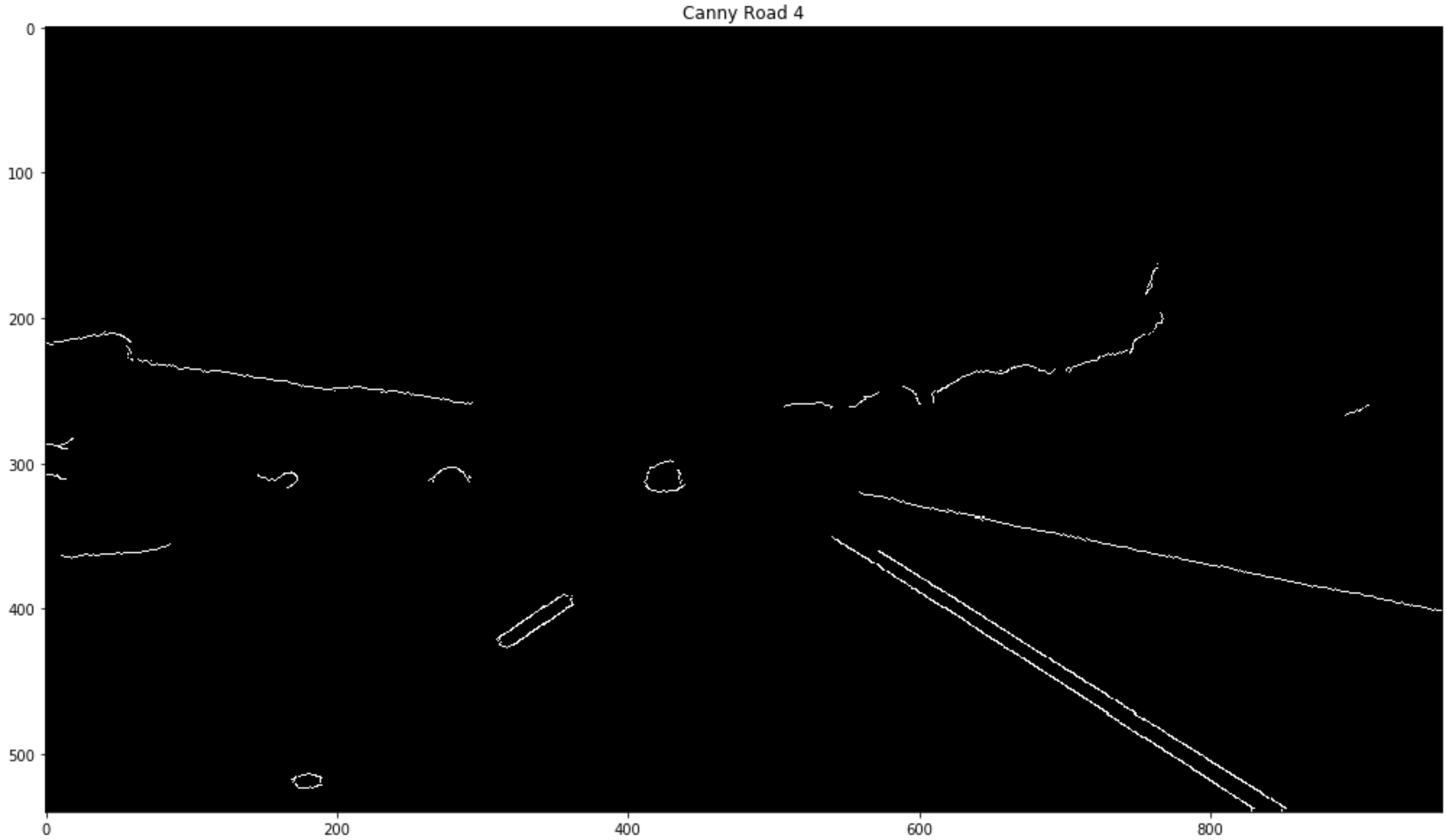




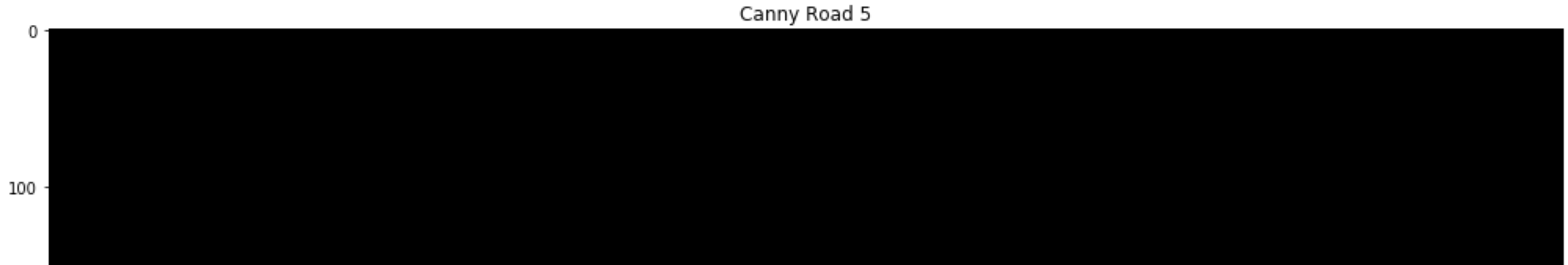
Canny Road 3

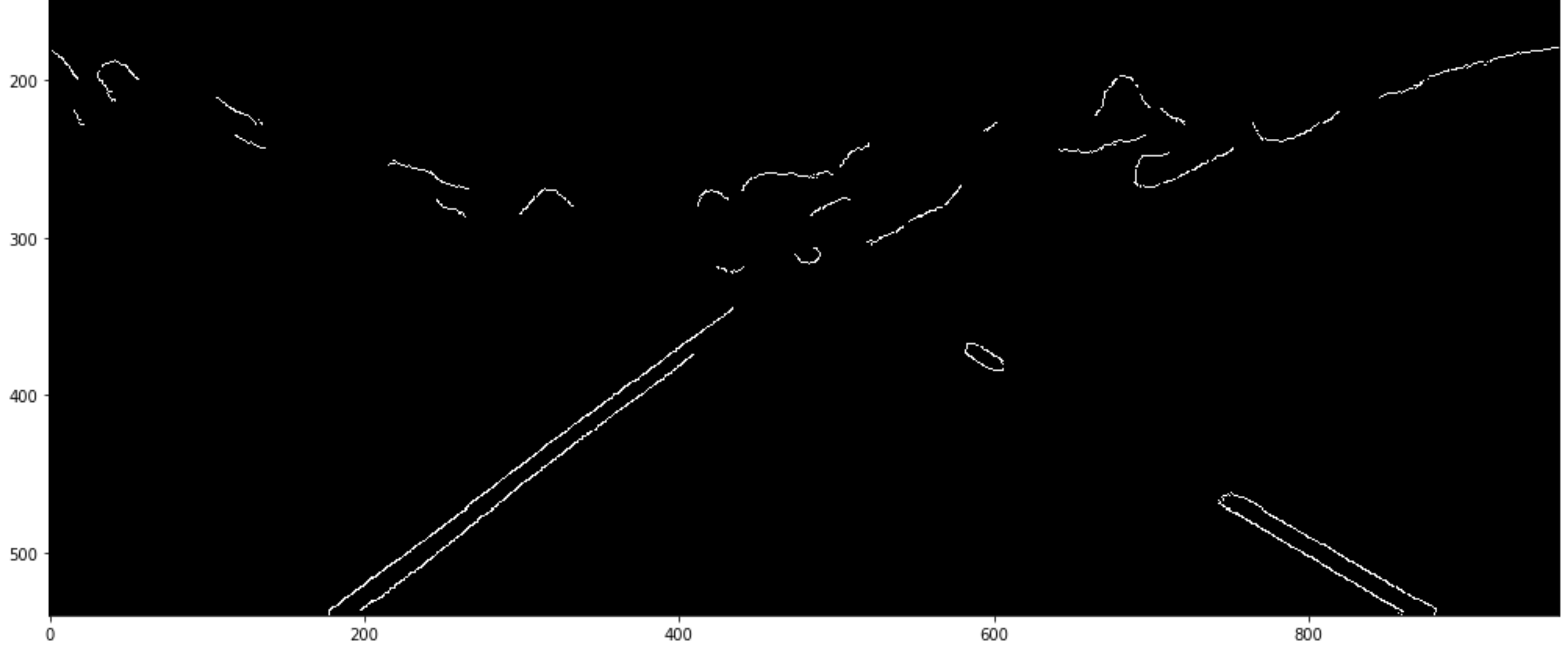


Canny Road 4



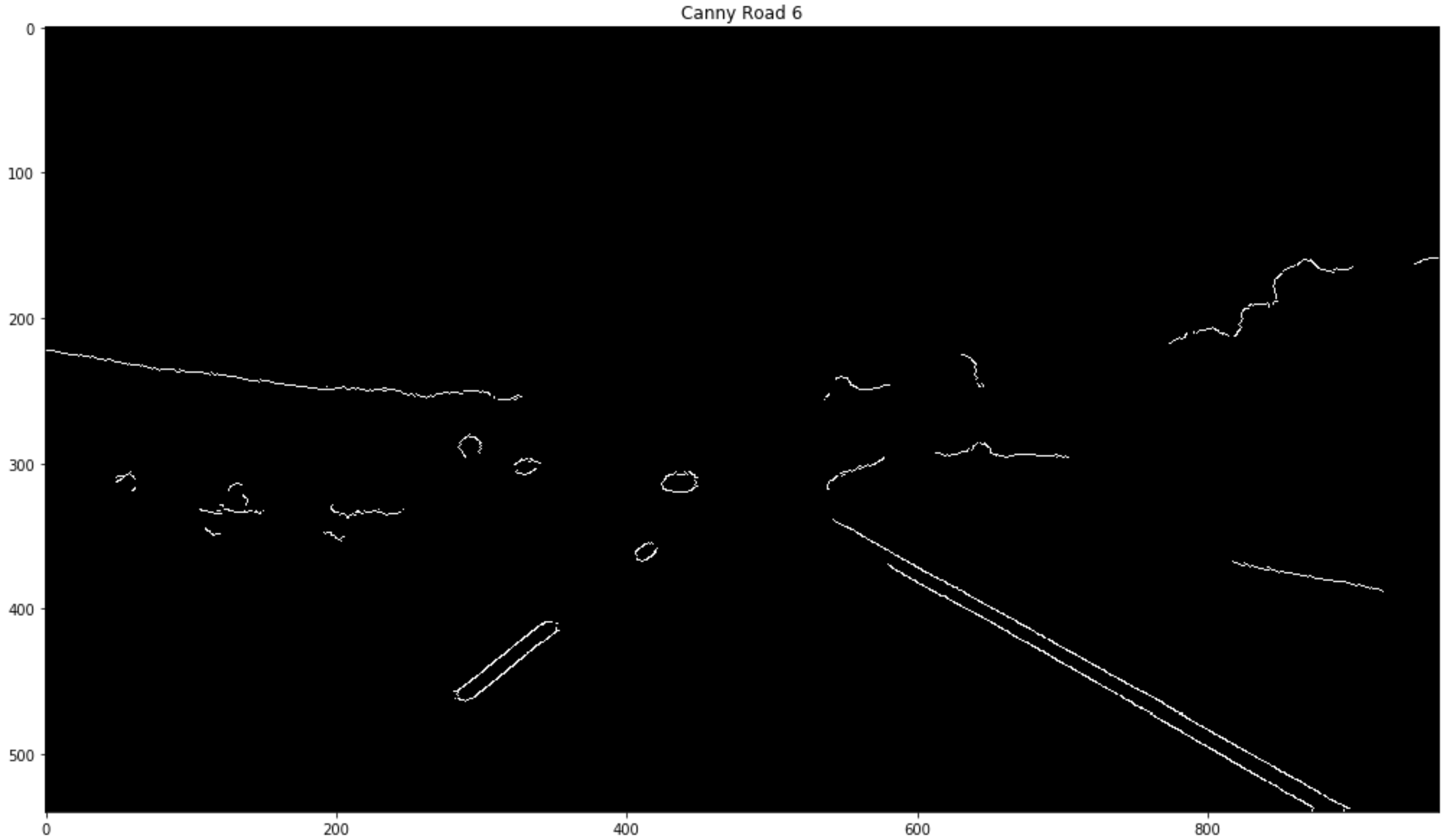
Canny Road 5





200
300
400
500
0
200
400
600
800

Canny Road 6

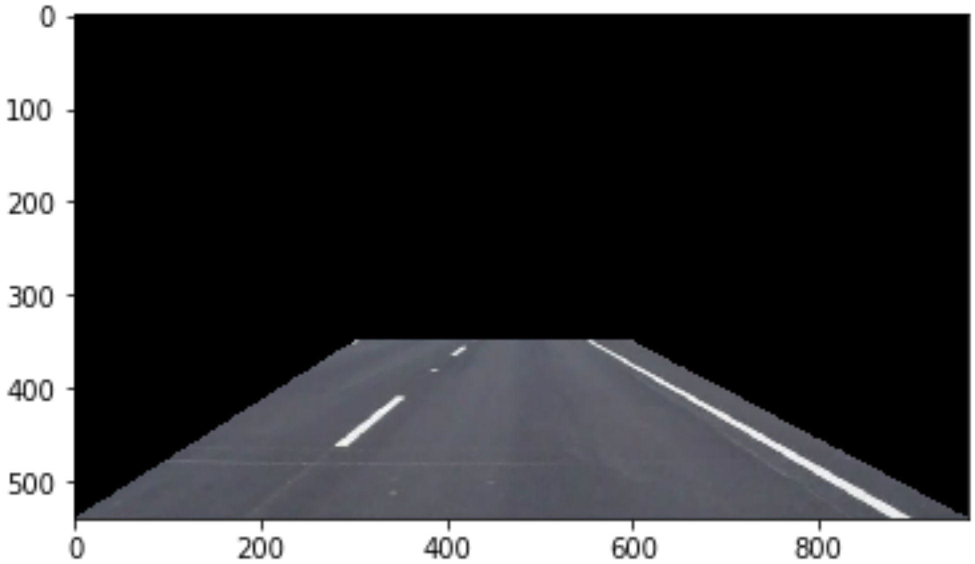


0
100
200
300
400
500
0
200
400
600
800

In [183]:

Out[183]:

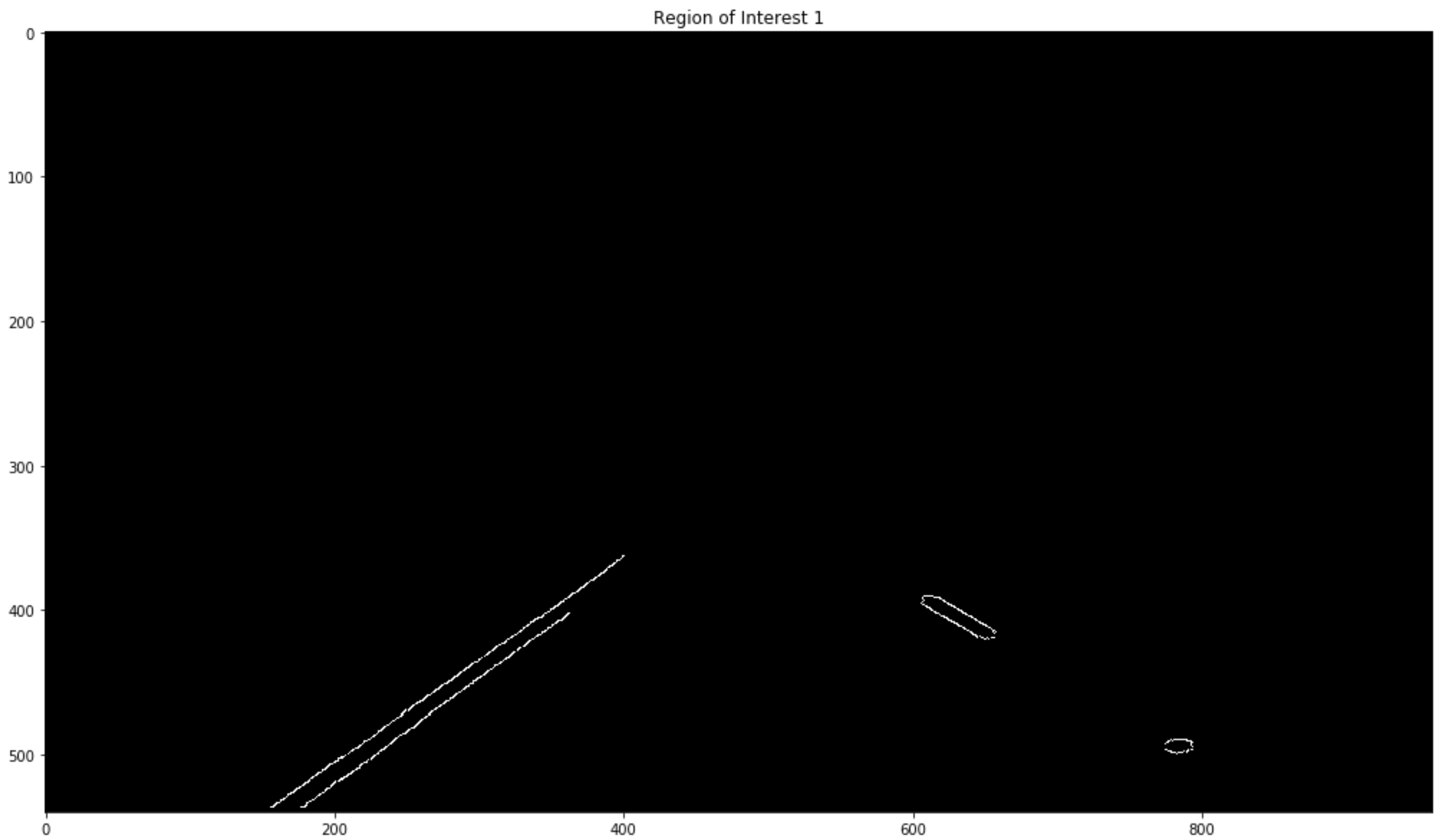
```
<matplotlib.image.AxesImage at 0x1302997b8>
```

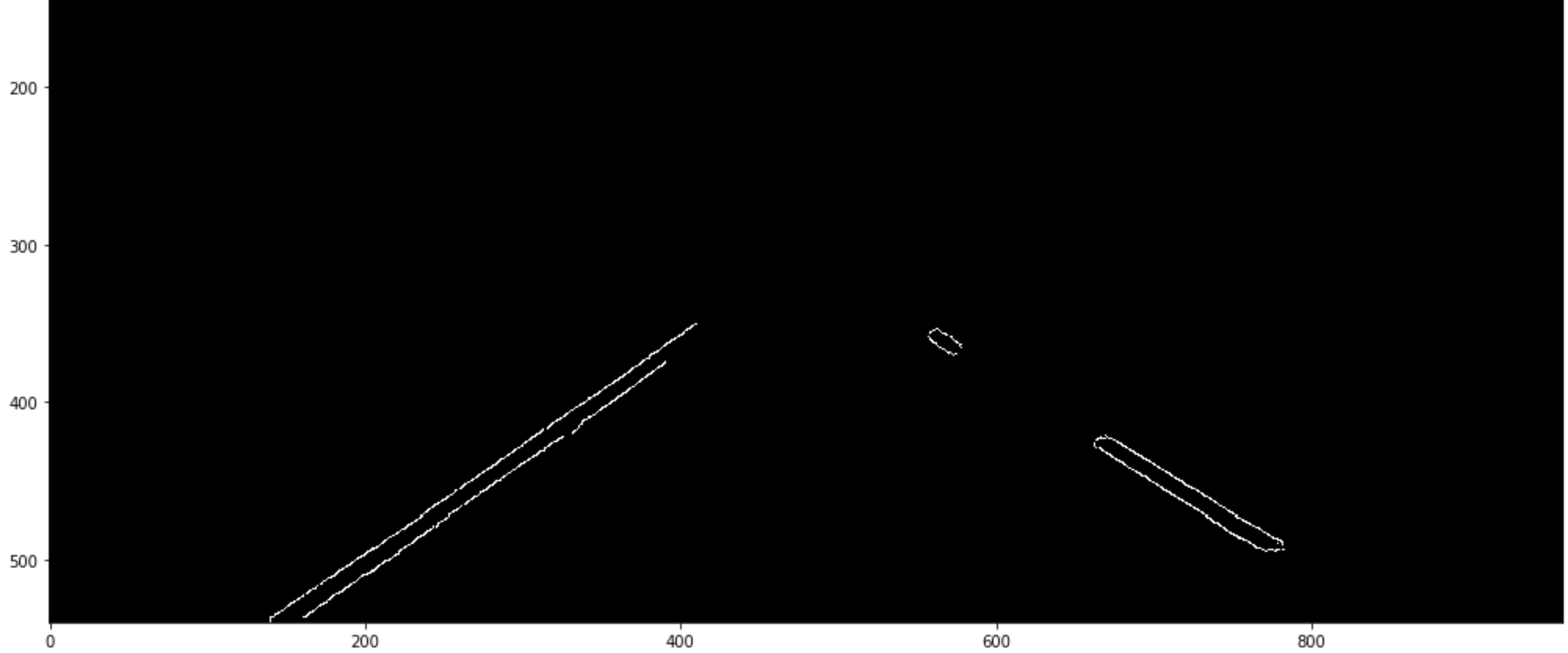


In [184]:

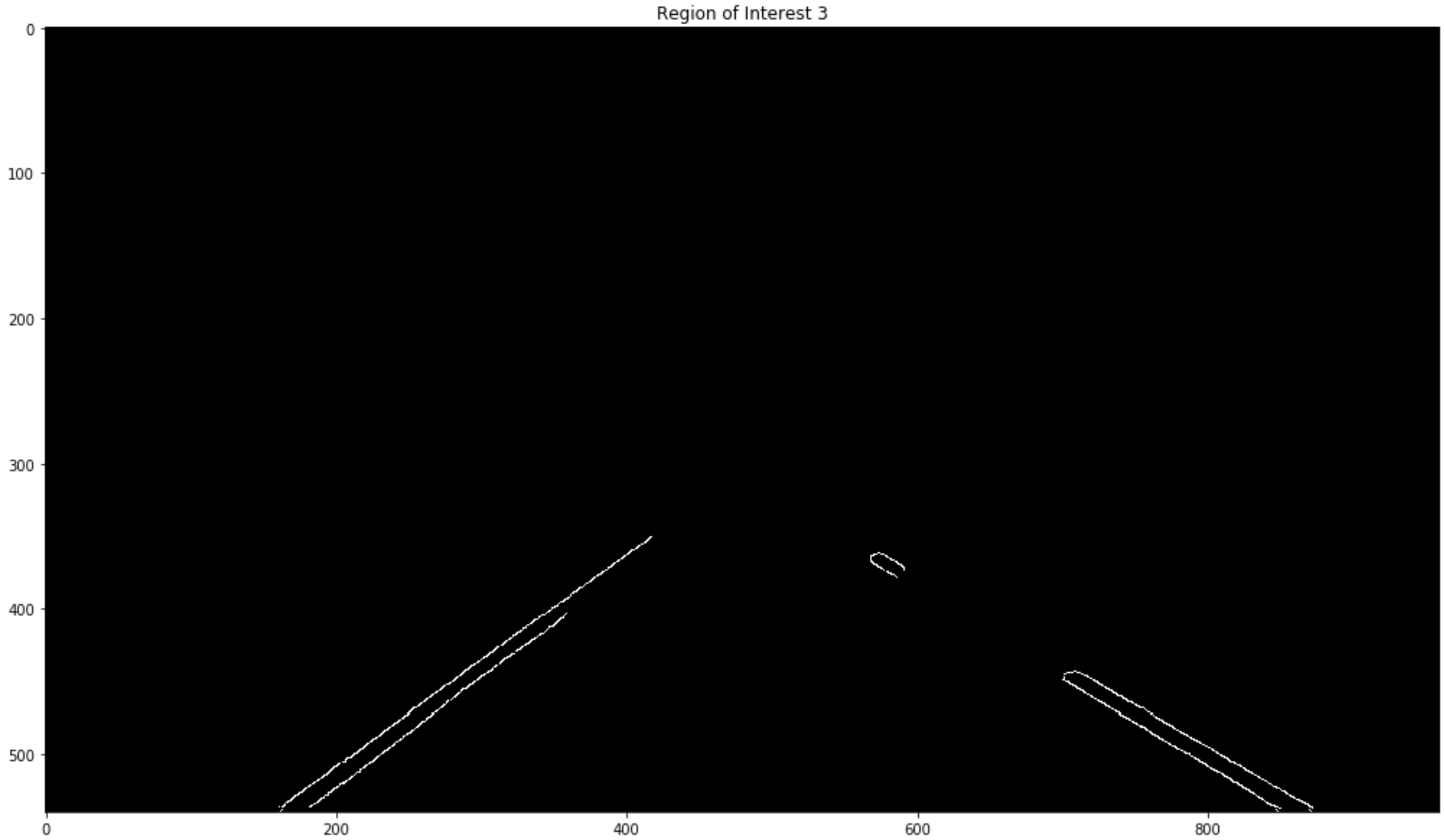
Out[184]:

```
<matplotlib.image.AxesImage at 0x13ed171d0>
```

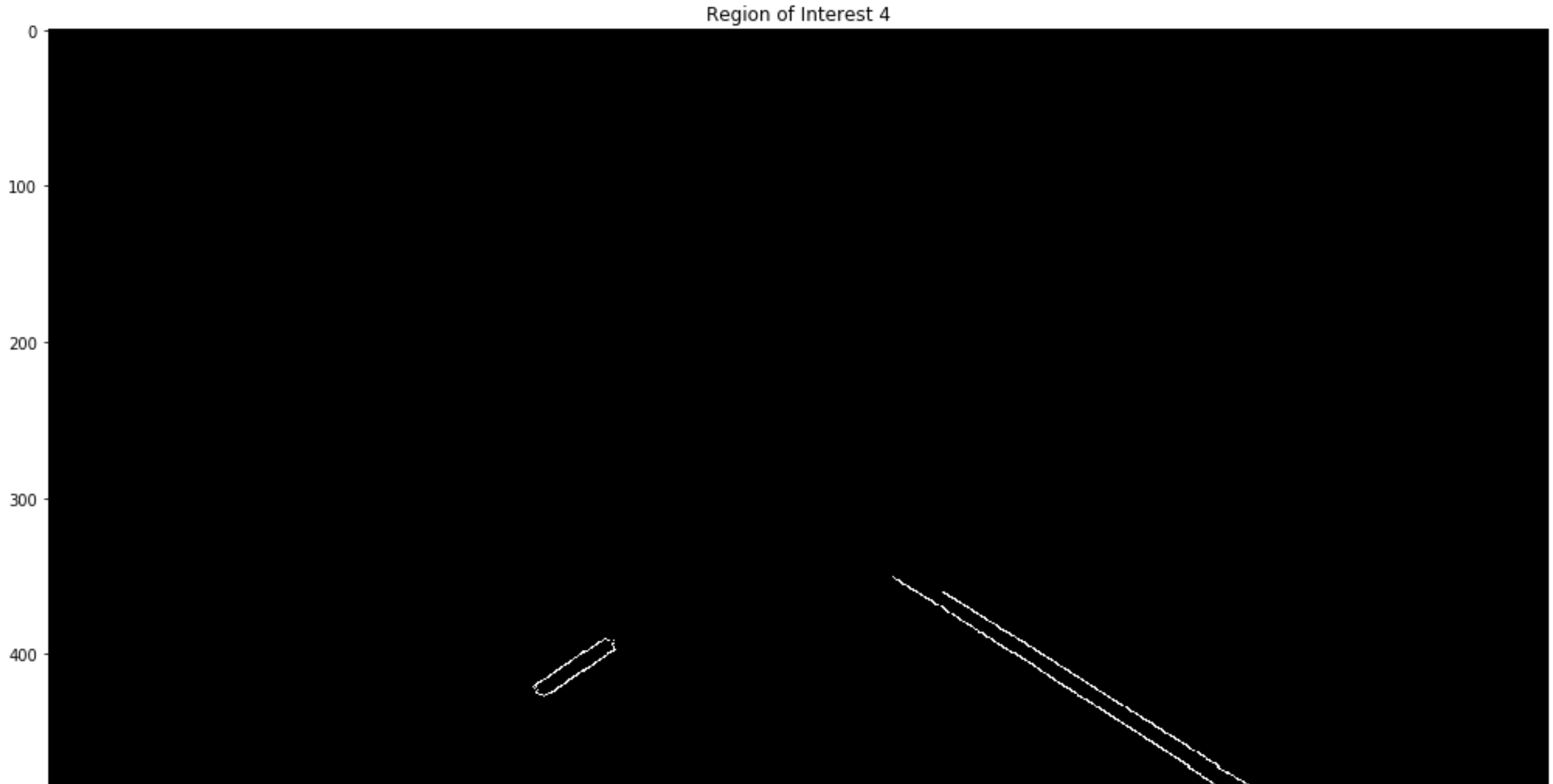




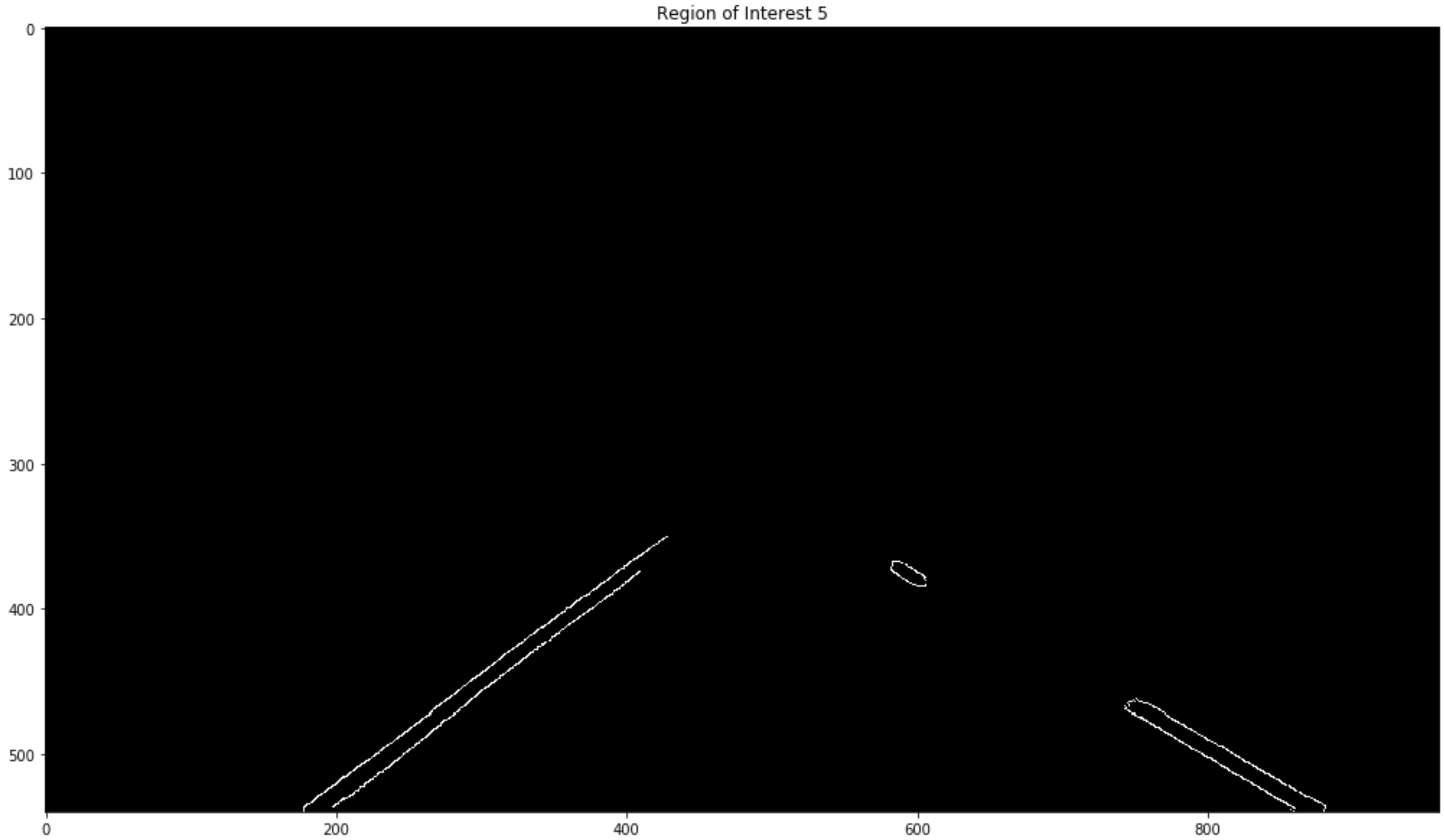
Region of Interest 3



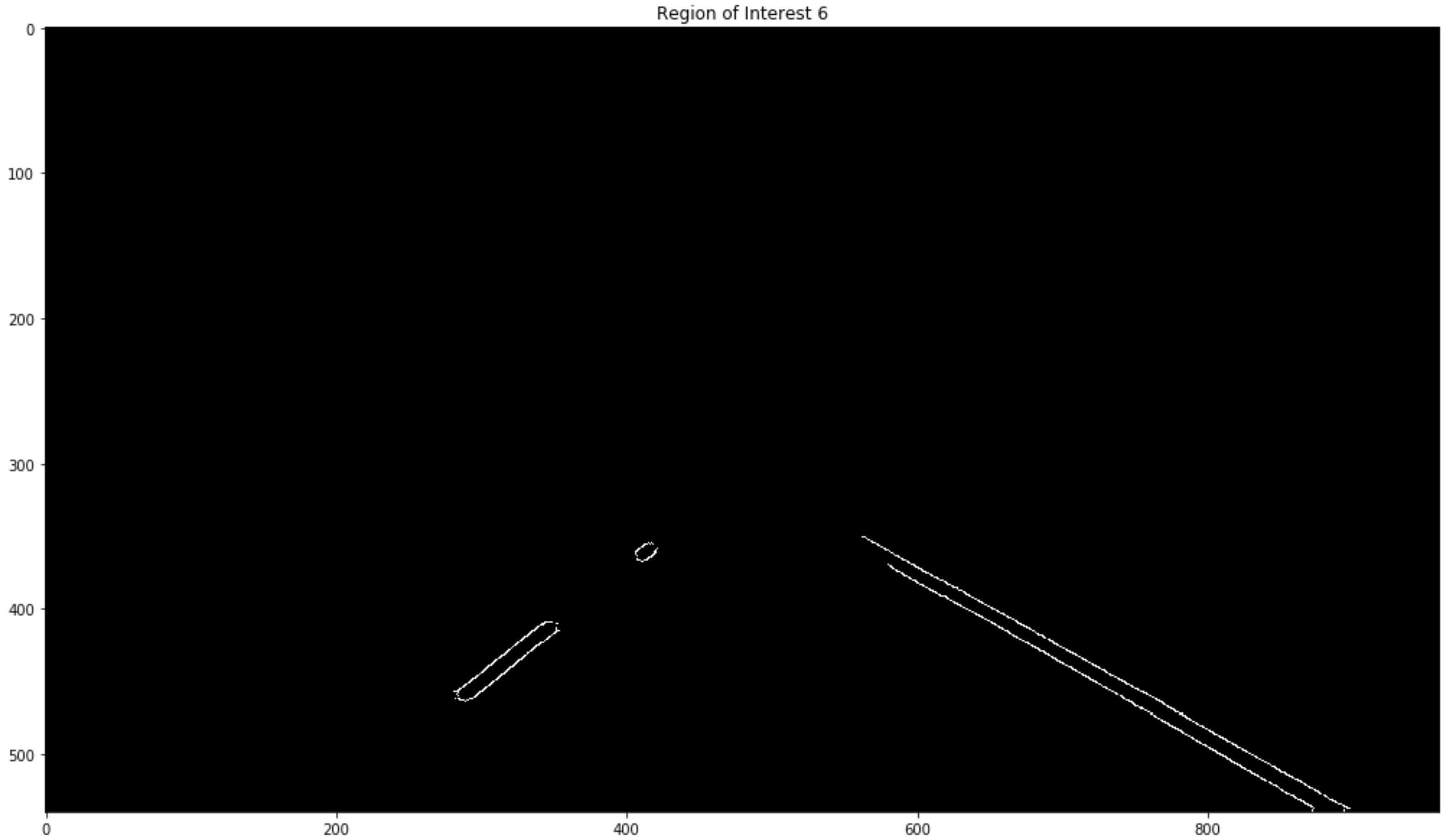
Region of Interest 4



Region of Interest 5



Region of Interest 6

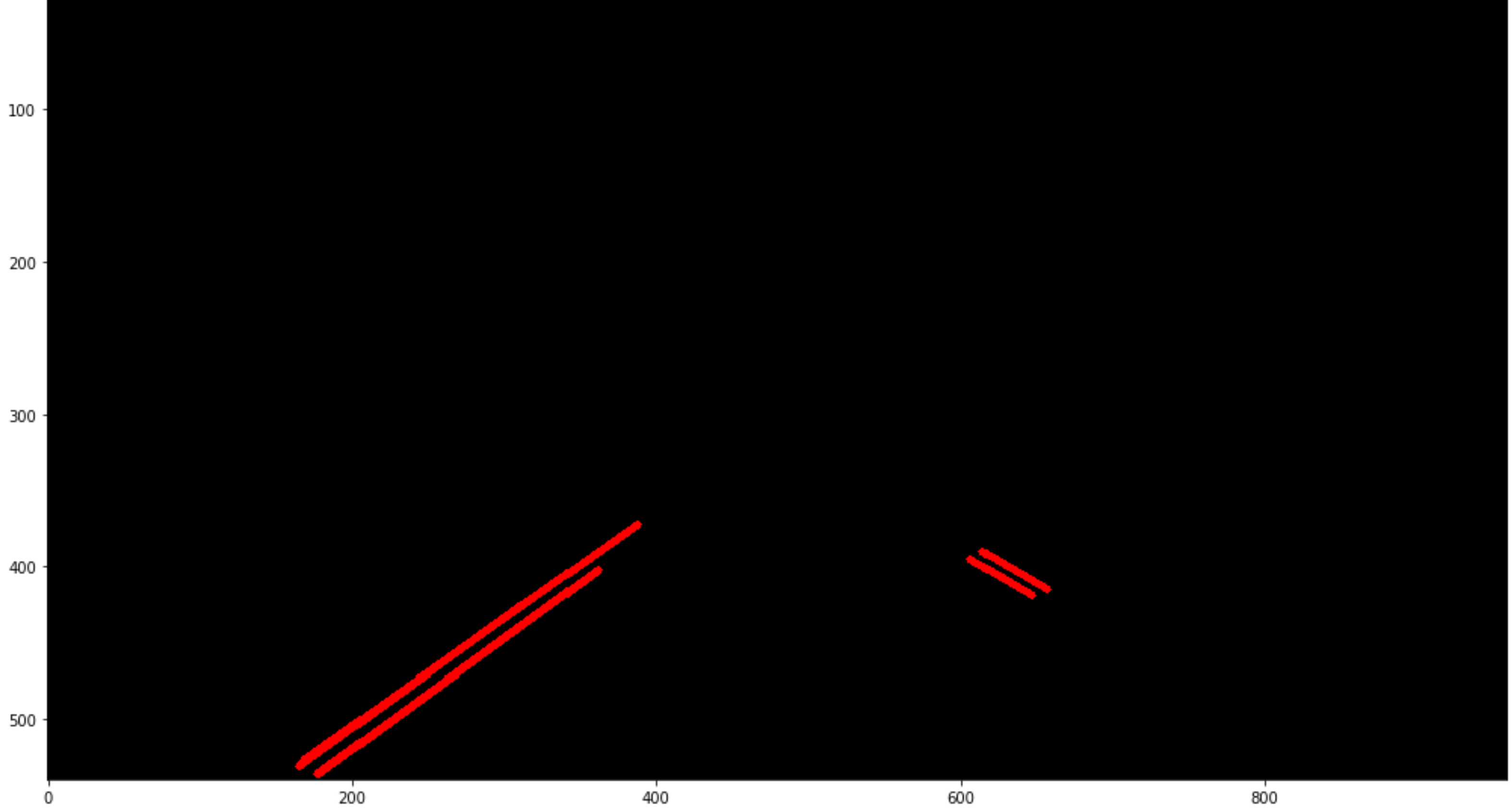


In [185]:

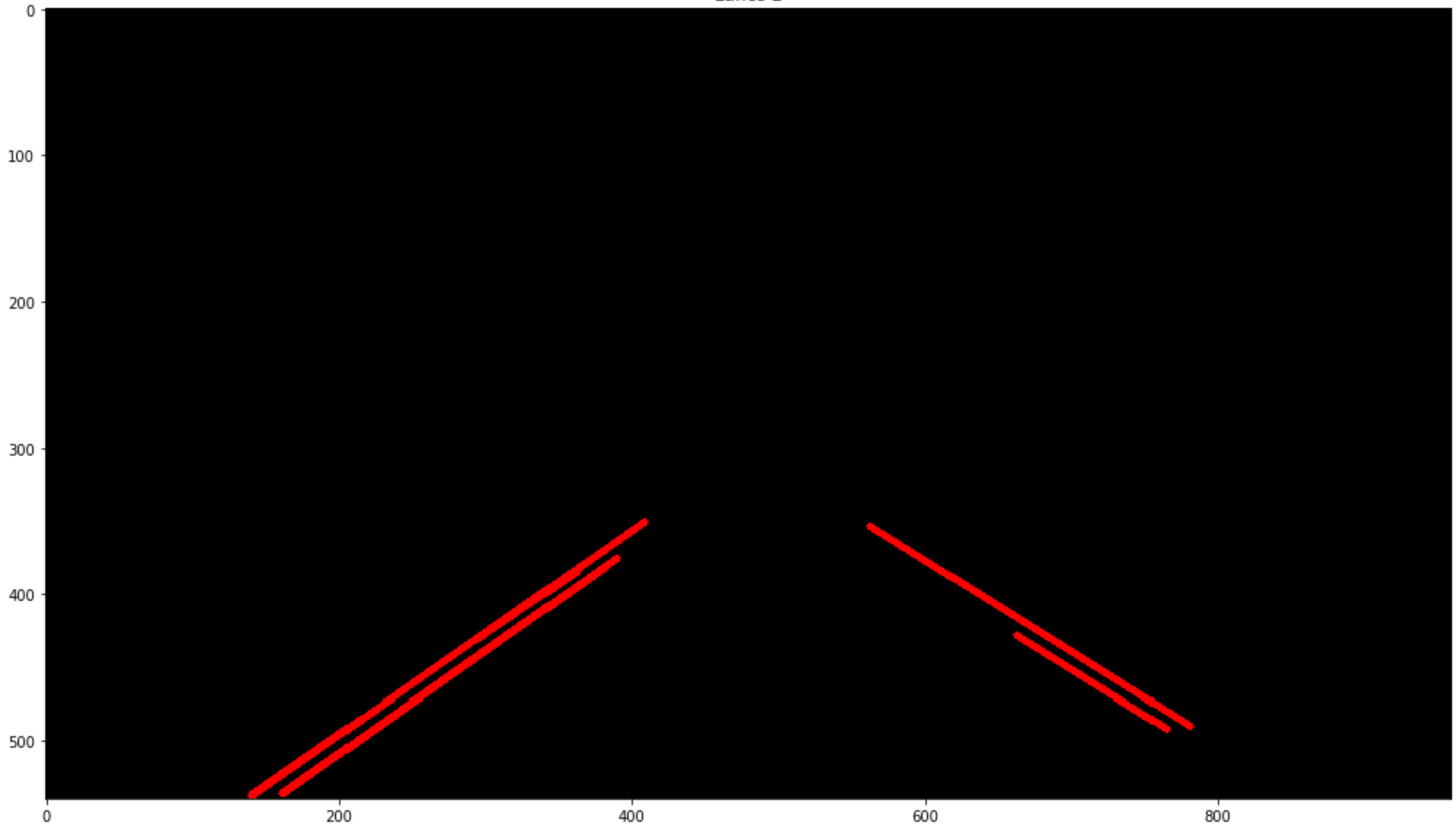
Out[185]:

<matplotlib.image.AxesImage at 0x134620710>

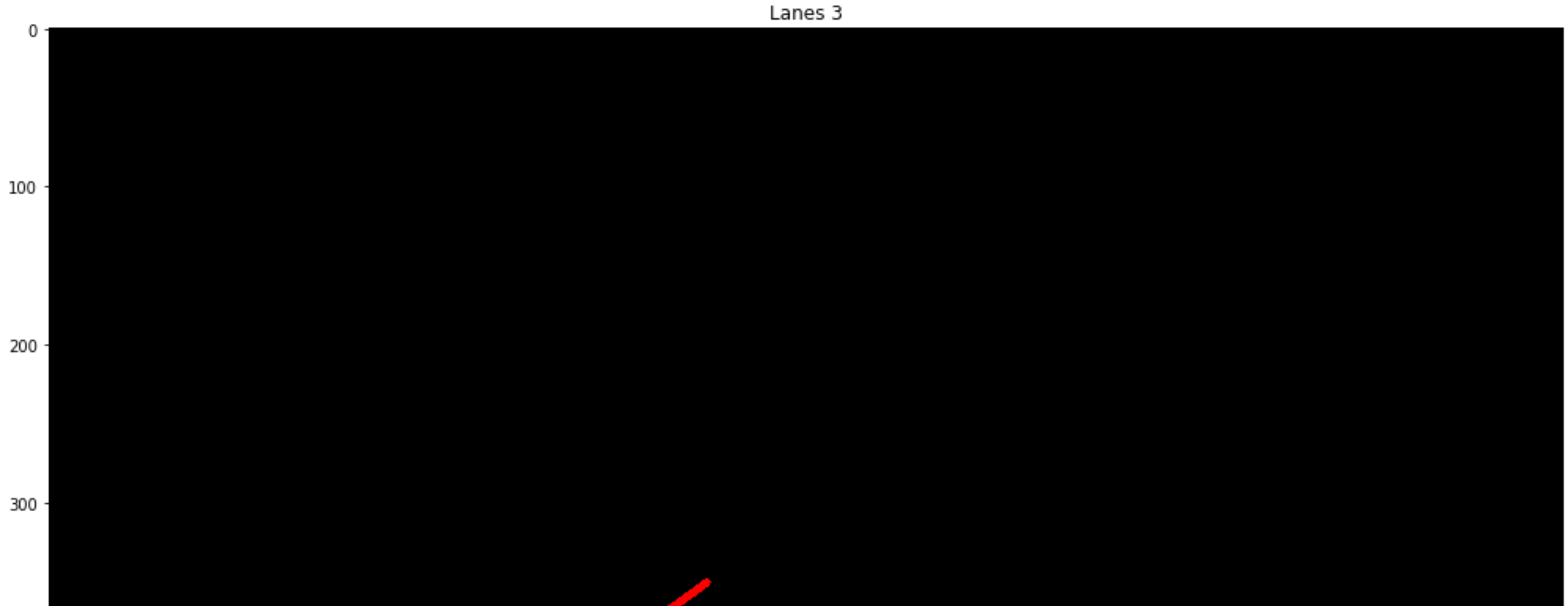
Lanes 1

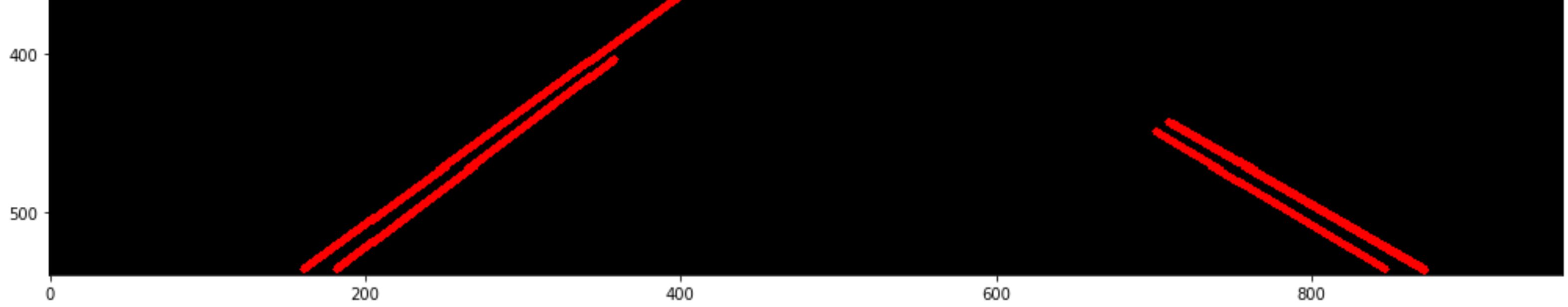


Lanes 2

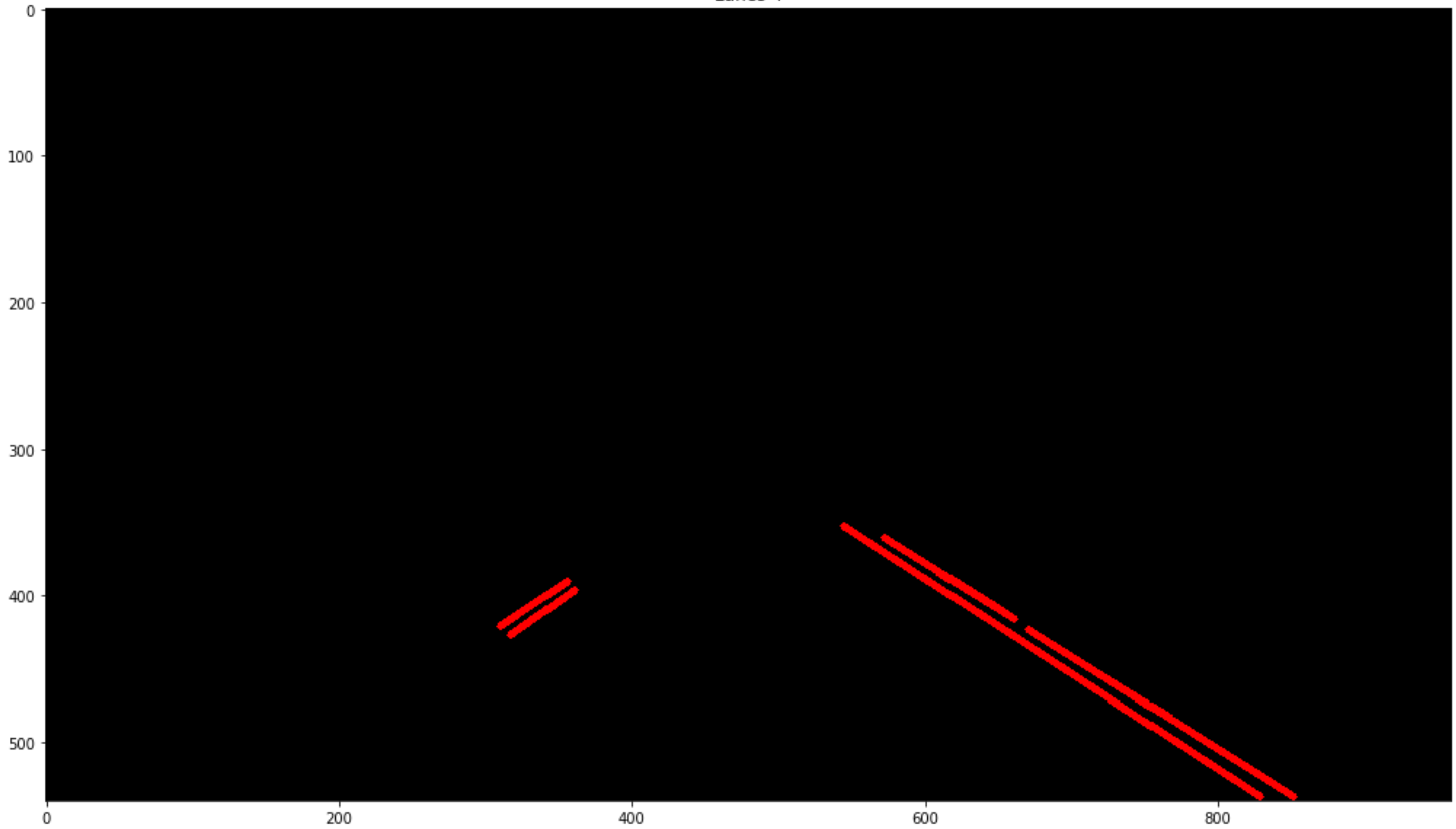


Lanes 3

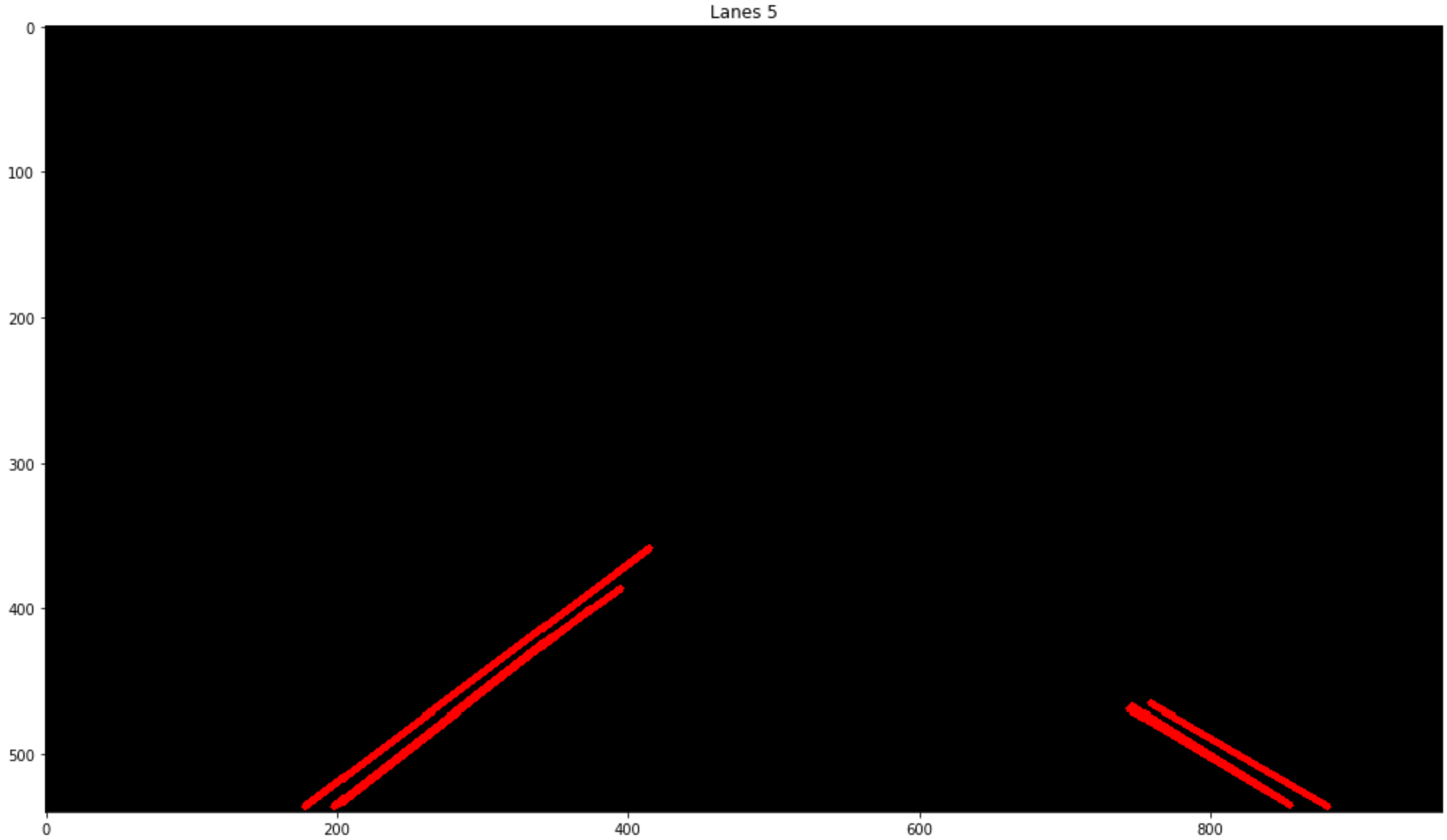




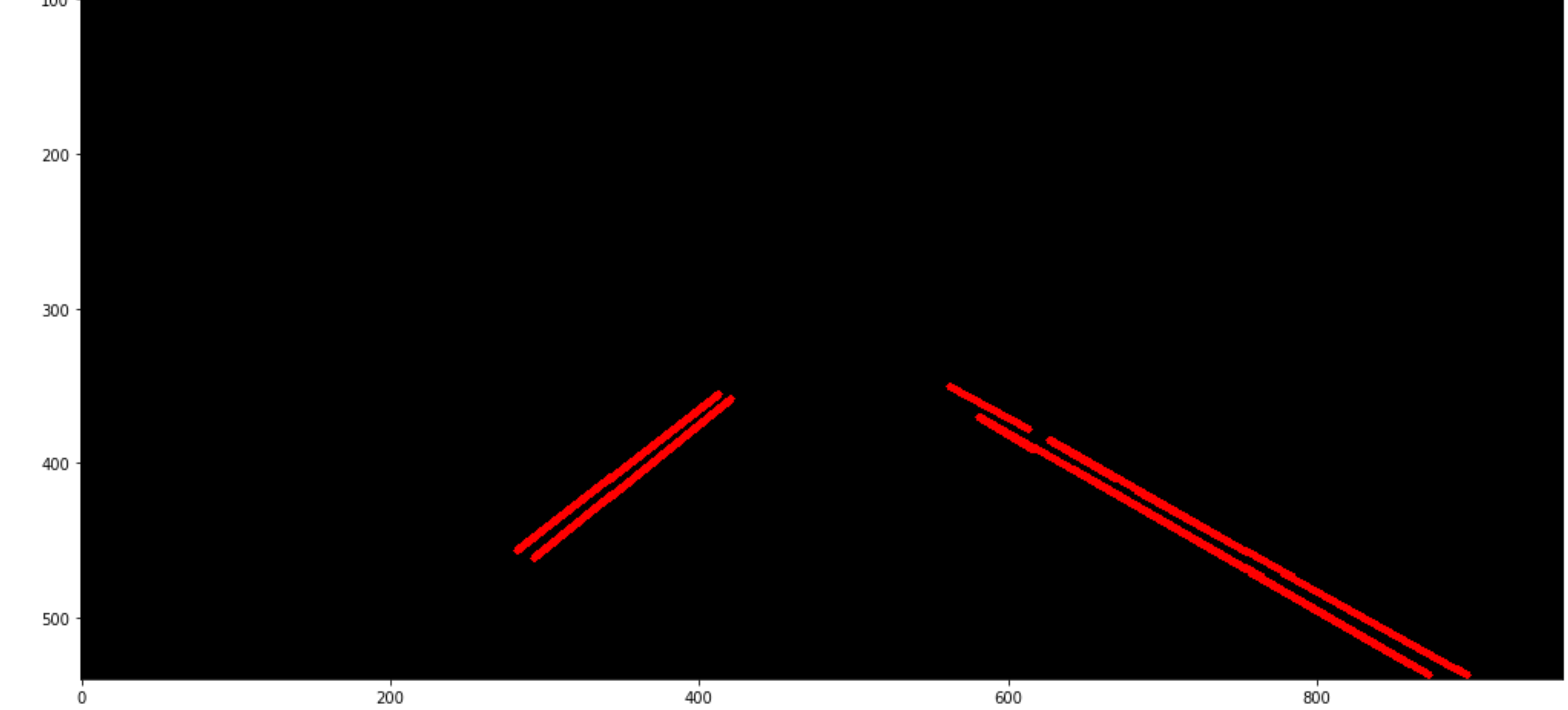
Lanes 4



Lanes 5



Lanes 6



Improve the draw_lines() function

At this point, you should have the Hough line segments drawn onto the road. Extend your code to define a line to run the full length of the visible lane based on the line segments you identified with the Hough Transform. Try to average and/or extrapolate the line segments you've detected to map out the full extent of the lane lines. The output should draw a single, solid line over the left lane line and a single, solid line over the right lane line. The lines should start from the bottom of the image and extend out to the top of the region of interest.

In [186]:

In [187]:

In [ ]:

```

```