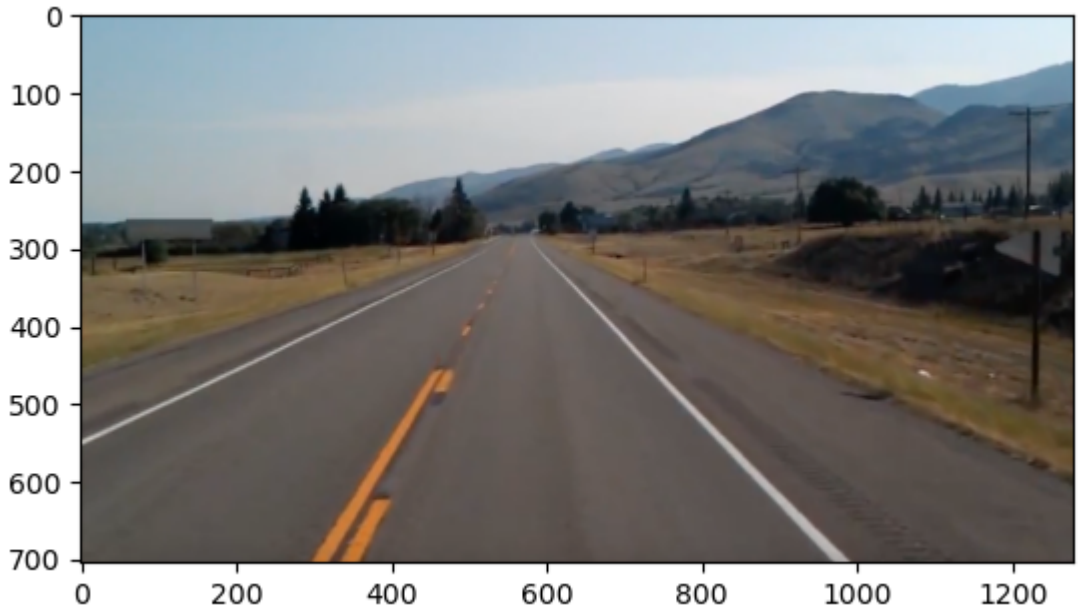
In [19]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, clear_output
import time
```

In [20]:
```
image = cv2.imread("D:/Downloads/Data sets/Lane lines/test_image.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
lane_image = np.copy(image_rgb)
```

In [21]:
```
plt.imshow(image_rgb)
```

Out[21]:
```
<matplotlib.image.AxesImage at 0x1b3682ced10>
```



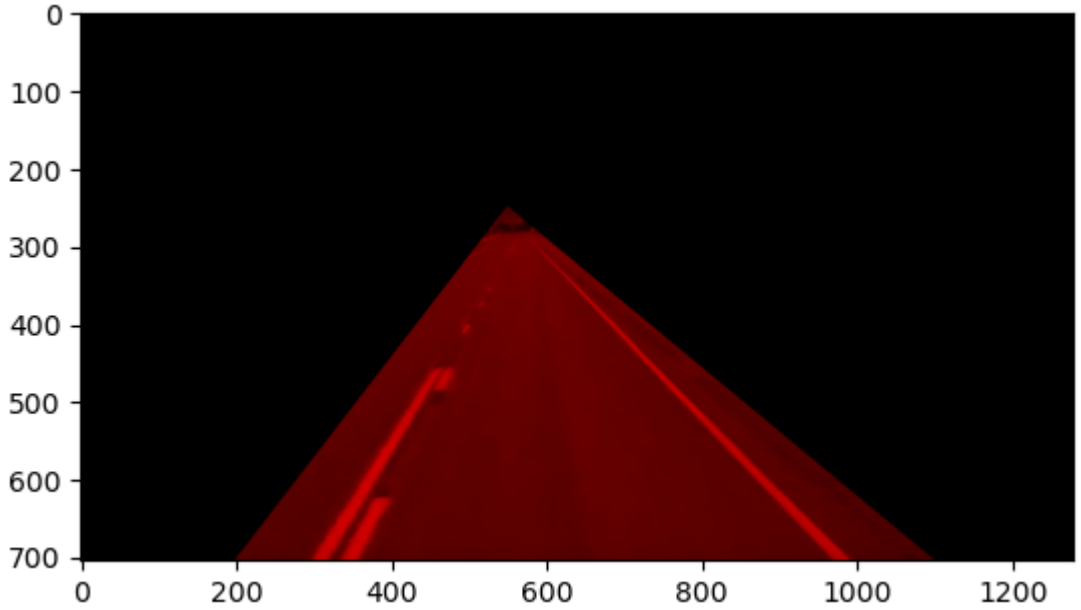
In [22]:
```
def canny_edge_detection(image_rgb):
        gray = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
        blur = cv2.GaussianBlur(gray,(5,5),0)
        canny = cv2.Canny(blur, 50, 150)
        return canny
```

In [23]:
```
 def region_of_interest(image_rgb):
        height = image.shape[0]
        triangle = np.array([(200, height), (1100, height), (550, 250)], dtype=np.int32)
        mask = np.zeros_like(image_rgb)
        cv2.fillPoly(mask, [triangle], 255)
        masked_image = cv2.bitwise_and(image_rgb, mask)  # Apply bitwise AND operation to create the masked image
        return masked_image  # Return the masked image
```

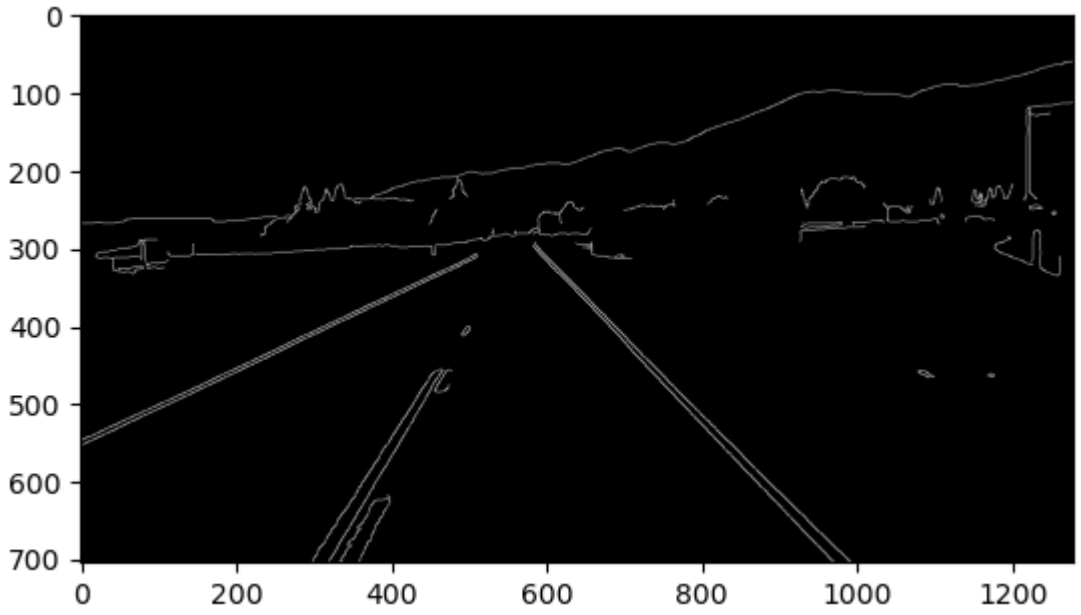
In [24]:
```
def average_slope_intercept(image_rgb, lines):
    left_fit = [] #contains coordinates of the averaged lines on the left
    right_fit = [] #contains coordinates of the averaged lines on the right
    for line in lines:
        x1, y1, x2, y1 = line.reshape(4)
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        intercept = parameters[1]
        if slope < 0:     # x would be negative value and y would be positive value
            left_fit.append((slope, intecept))
        else:
                right_fit.append((slope, intercept))
    left_fit_average = np.average(left_fit, axis = 0)
    right_fit_average = np.average(right_fit, axis = 0)
    print(left_fit_average, 'left')
    print(right_fit_average, 'right')
```

In [25]:
```
def display_lines(image_rgb, lines):
        line_image = np.zeros_like(image_rgb)
        if lines is not None:
            for line in lines:
                x1, y1, x2, y2 = line.reshape(4)
                cv2.line(line_image, (x1, y1), (x2, y2), (0, 0, 255), 10)  # Draw lines in red color with thickness 10
        return line_image
```

In [26]:
```
region_of_interest_np = region_of_interest(image_rgb)
plt.imshow(region_of_interest_np , cmap='gray')  # assuming you want to display as grayscale
plt.show()
```

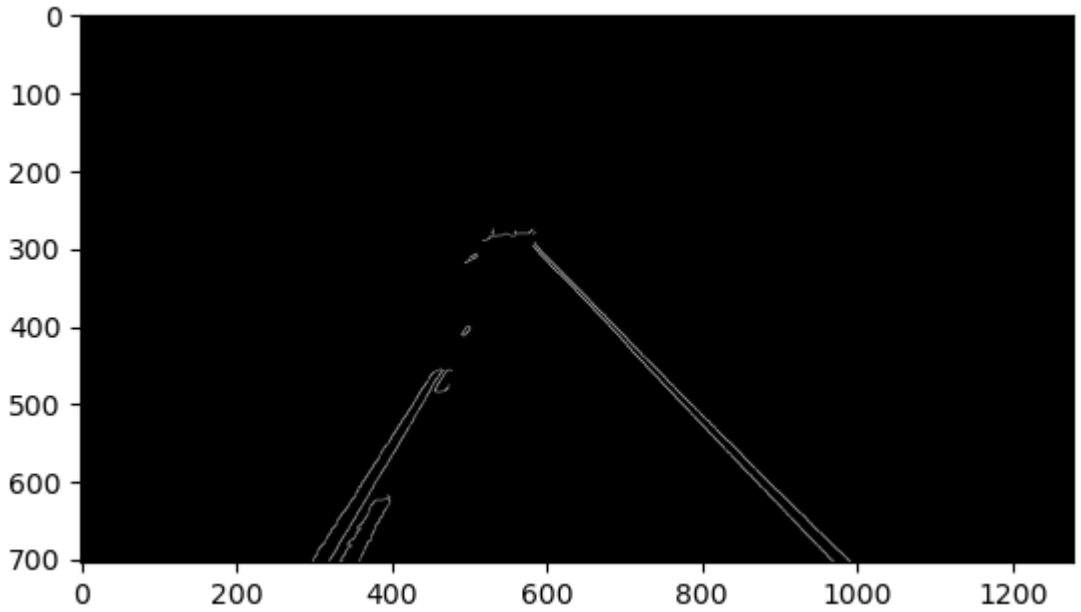


In [27]:
```
canny_image = canny_edge_detection(lane_image)
cropped_image = region_of_interest(canny_image)
plt.imshow(canny_image, cmap='gray')
plt.show()
```



In [28]:
```
cropped_image = region_of_interest(canny_image)
plt.imshow(cropped_image, cmap='gray')
```

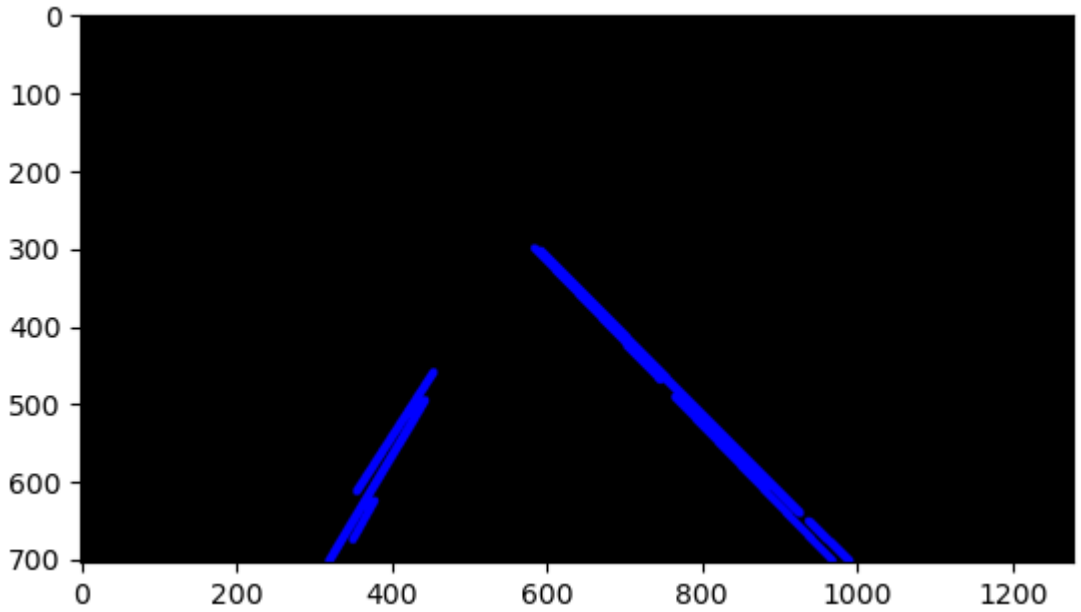
Out[28]: <matplotlib.image.AxesImage at 0x1b361f3ae50>



In [29]:
```
lines = cv2.HoughLinesP(cropped_image, 2,np.pi/180, 100, np.array([]),minLineLength=40,maxLineGap=5)
```

In [30]:
```
line_image = display_lines(lane_image, lines)
plt.imshow(line_image)
```

Out[30]: <matplotlib.image.AxesImage at 0x1b36831d690>

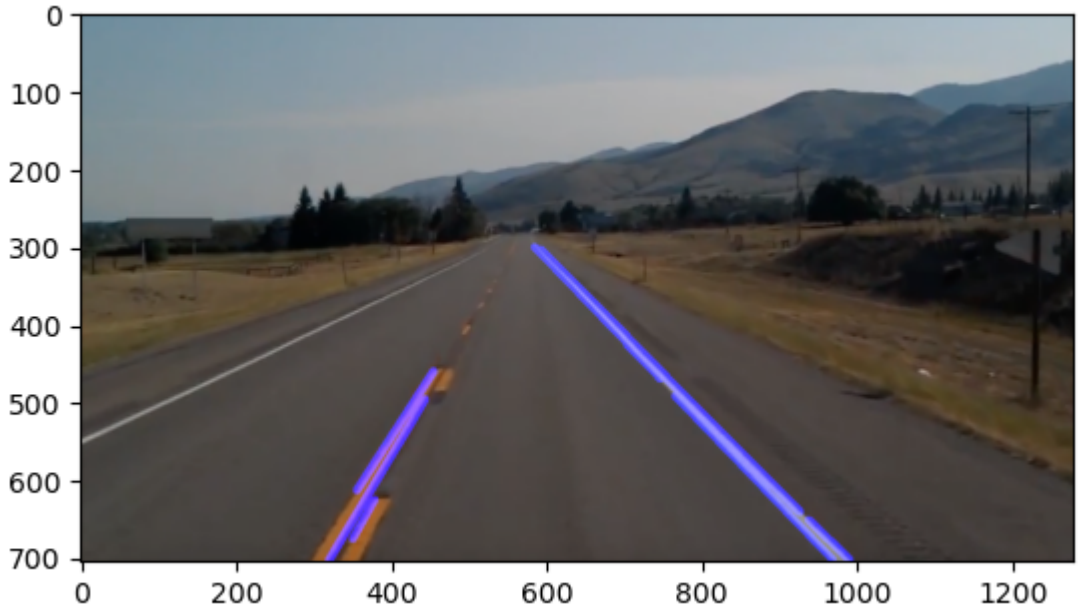


In [31]:
```
def average_slope_intercept(image_rgb, lines):
        left_fit = [] #contains coordinates of the averaged lines on the left
        right_fit = [] #contains coordinates of the averaged lines on the right
        for line in lines:
            x1, y1, x2, y2 = line.reshape(4)
            parameters = np.polyfit((x1, x2), (y1, y2), 1)
            slope = parameters[0]
            intercept = parameters[1]
            if slope < 0:
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))
        left_fit_average = np.average(left_fit, axis = 0)
        right_fit_average = np.average(right_fit, axis = 0)
        left_line = make_coordinates(image_rgb, left_fit_average)
        right_line = make_coordinates(image_rgb, right_fit_average)
        return np.array([left_line, right_line])
```

In [32]:
```
def make_coordinates(image_rgb, line_parameters):
    if line_parameters is not None:
        if isinstance(line_parameters, np.ndarray) and line_parameters.size == 2:
            slope, intercept = line_parameters
            y1 = image_rgb.shape[0] #represents the height.
            y2 = int(y1*(3/5))
            x1 = int((y1 - intercept)/slope)  # From straight line equation
            x2 = int((y2 - intercept)/slope)
            return np.array([x1, y1, x2, y2])
        else:
            # Handle the case where line_parameters is not in the expected format
            return None
    else:
        return None
```

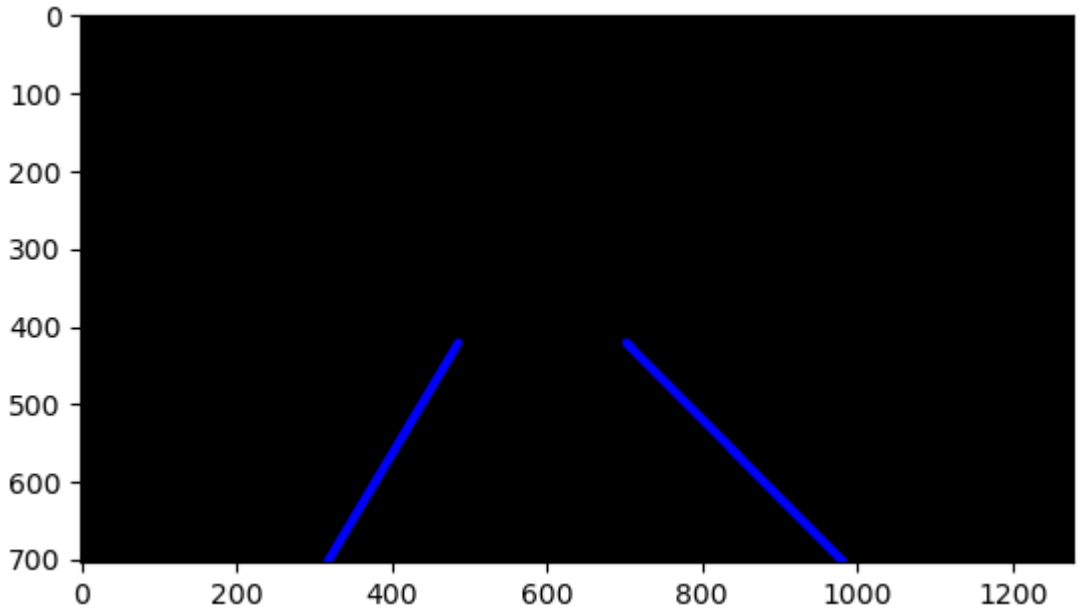
In [33]:
```
combo_image = cv2.addWeighted(lane_image, 0.8, line_image, 1,1) #Lane image is multiplied by 0.8 to make image more da
averaged_lines = average_slope_intercept(lane_image, lines)
plt.imshow(combo_image)
```

Out[33]:
```
<matplotlib.image.AxesImage at 0x1b361f86d10>
```



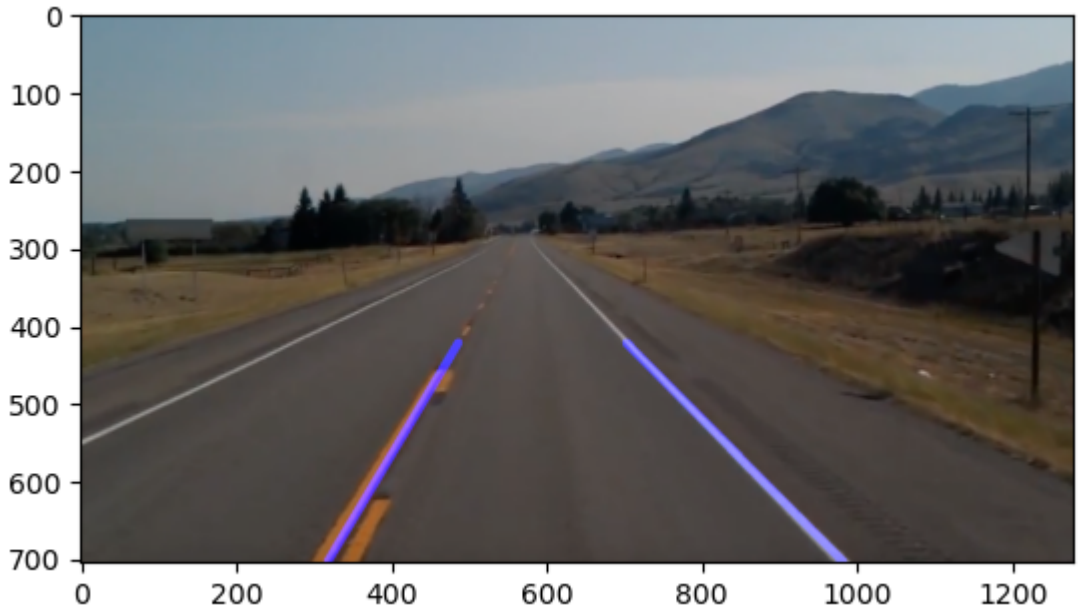
In [34]:
```
line_image = display_lines(lane_image, averaged_lines)
plt.imshow(line_image)
```

Out[34]: <matplotlib.image.AxesImage at 0x1b368366b50>



In [35]:
```
combo_image = cv2.addWeighted(lane_image, 0.8, line_image, 1,1)
plt.imshow(combo_image)
```

Out[35]: <matplotlib.image.AxesImage at 0x1b36832ed10>



In [36]:
```
# Load video
cap = cv2.VideoCapture("D:/Downloads/Data sets/Lane lines/test2.mp4")

try:
    while cap.isOpened():
        # Read a frame from the video
        ret, frame = cap.read()

        # Break the loop if there are no more frames
        if not ret:
            break

        # Perform necessary processing on the frame
        canny_image = canny_edge_detection(frame)
        cropped_image = region_of_interest(canny_image)
        lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 100, np.array([]), minLineLength=40, maxLineGap=5)
        averaged_lines = average_slope_intercept(frame, lines)
        line_image = display_lines(frame, averaged_lines)
        combo_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)

        # Display the processed frame
        clear_output(wait=True)
        plt.imshow(cv2.cvtColor(combo_image, cv2.COLOR_BGR2RGB))
        plt.axis('off')  # Turn off axis
        plt.show()

        # Add a small delay to slow down the video display
        time.sleep(0.03)

except KeyboardInterrupt:
    # Release the video capture when interrupted
    cap.release()
    print("Video stopped by user.")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 17
     15 cropped_image = region_of_interest(canny_image)
     16 lines = cv2.HoughLinesP(cropped_image, 2, np.pi/180, 100, np.array([]), minLineLength=40, maxLineGap=5)
---> 17 averaged_lines = average_slope_intercept(frame, lines)
     18 line_image = display_lines(frame, averaged_lines)
     19 combo_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)

Cell In[31], line 17, in average_slope_intercept(image_rgb, lines)
     15 left_line = make_coordinates(image_rgb, left_fit_average)
     16 right_line = make_coordinates(image_rgb, right_fit_average)
---> 17 return np.array([left_line, right_line])

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 1 dimensi
ons. The detected shape was (2,) + inhomogeneous part.
```

In [ ]:

In [ ]: