

# Finding lane lines for self-driving cars



Ross Kippenbrock  
PyData Berlin 2017

# About me

[github.com/rkipp1210](https://github.com/rkipp1210)  
@rosskipp

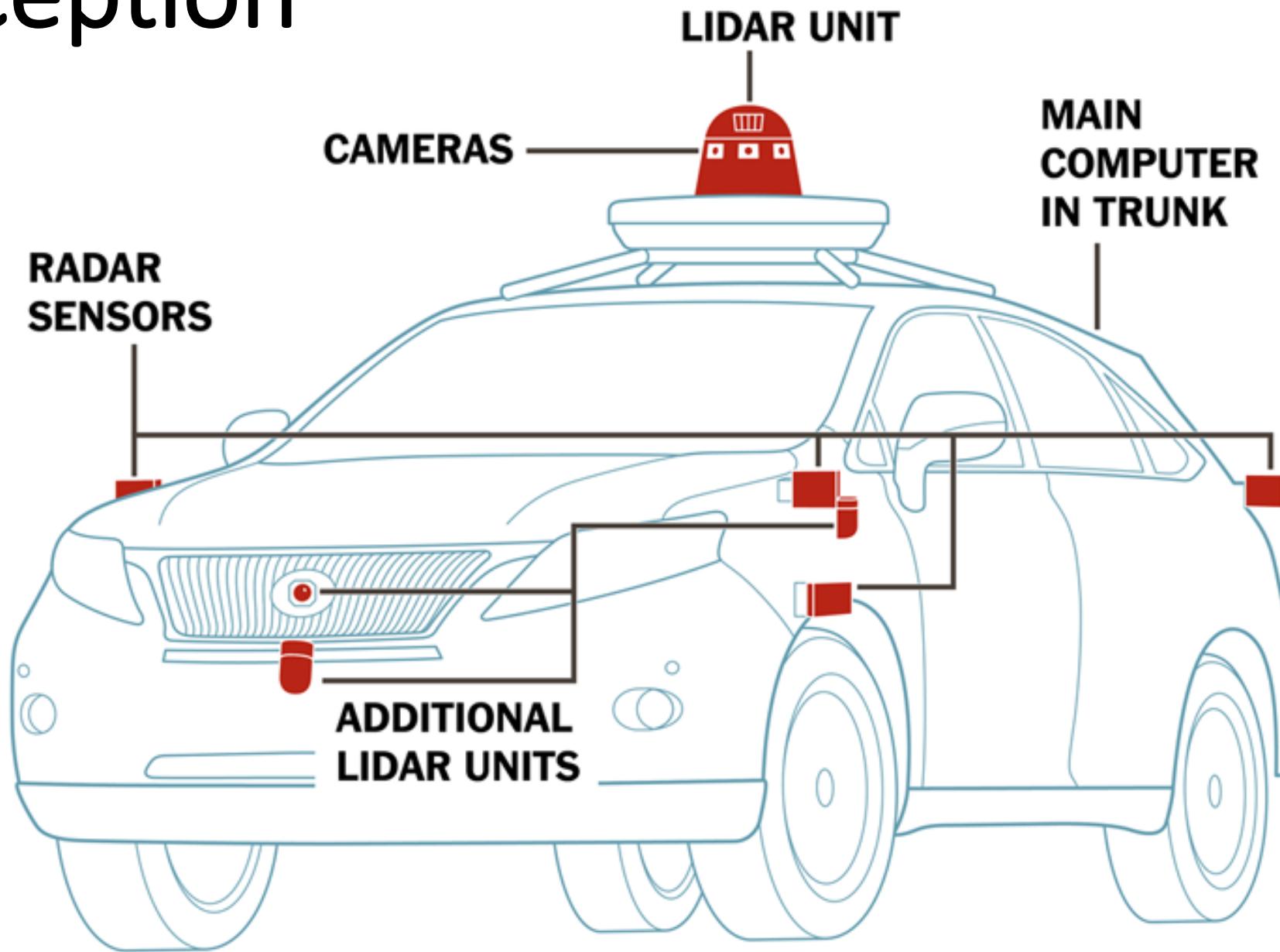


ŷhat  
alteryx

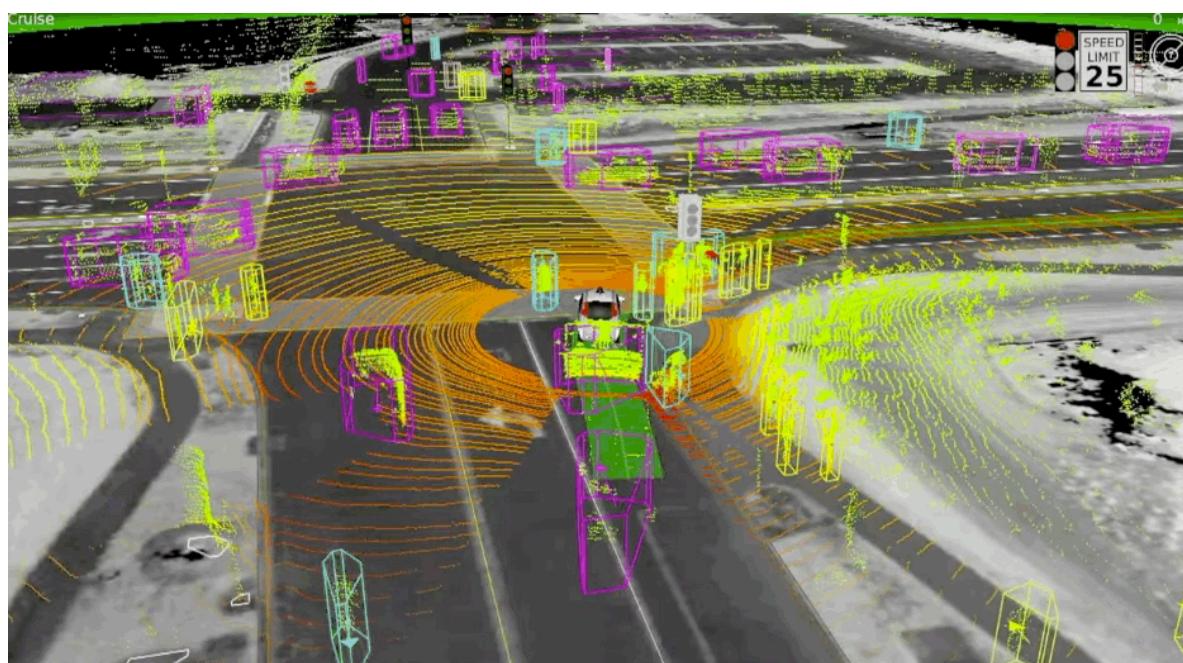
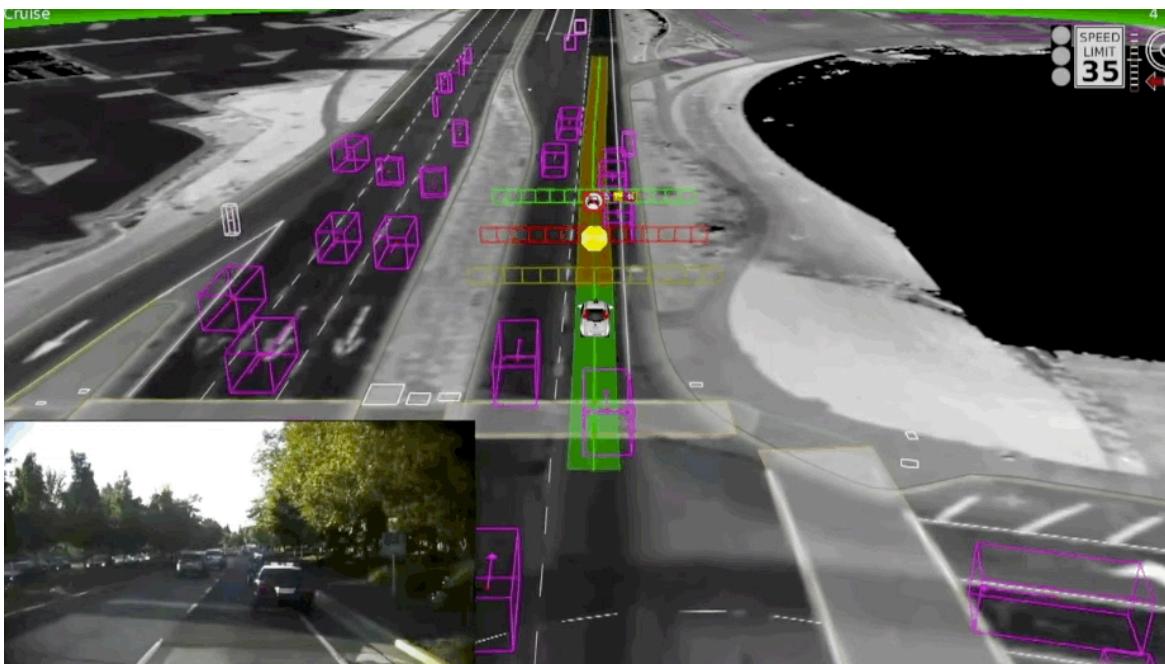
# Autonomous Cars need to do 3 things



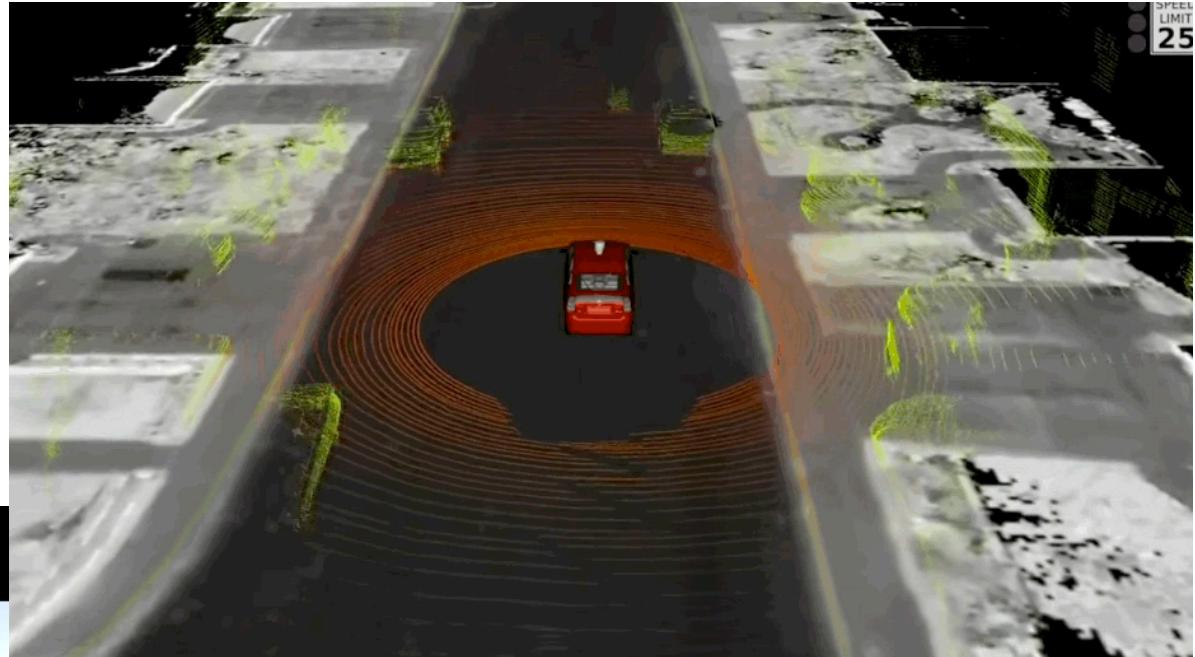
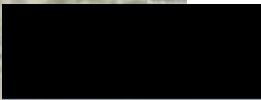
# Perception



# Decision



# Action



# Finding lane lines is perception and decision



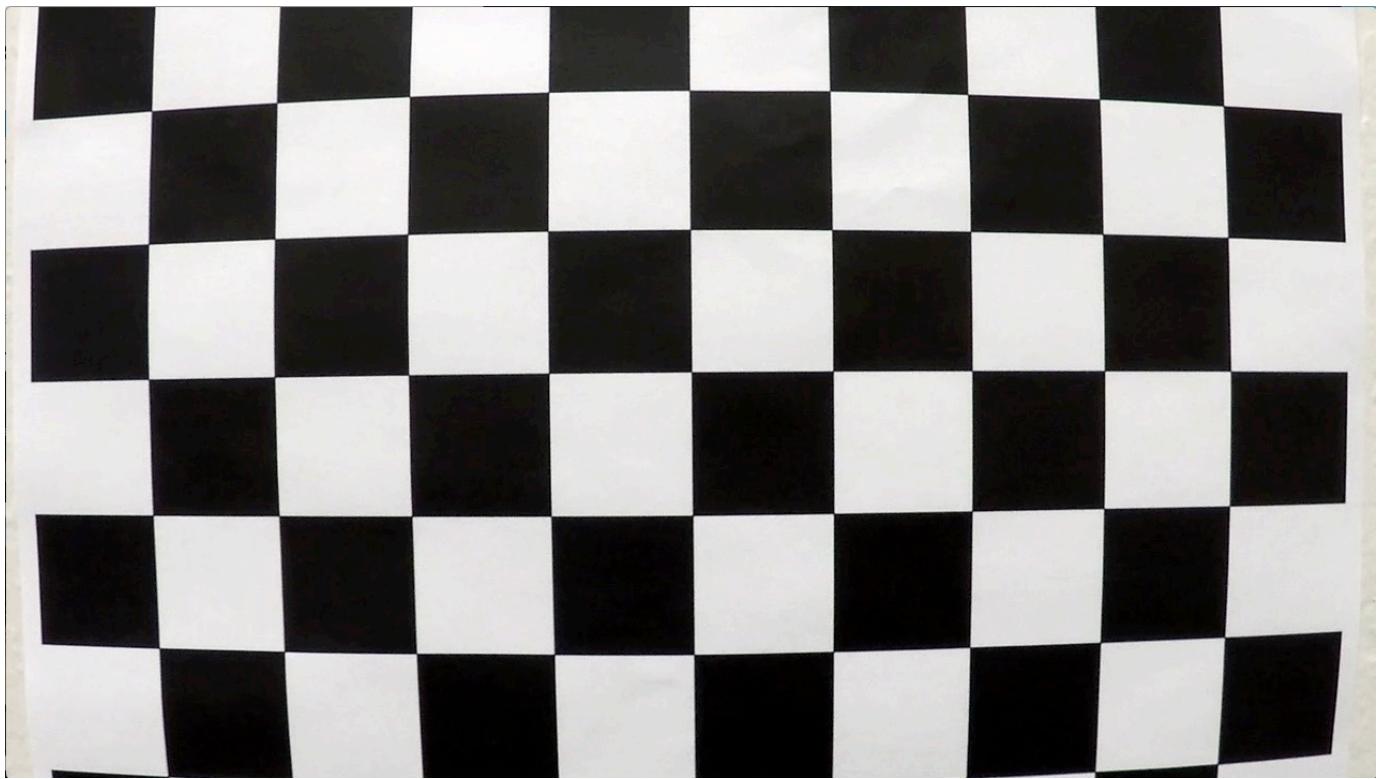
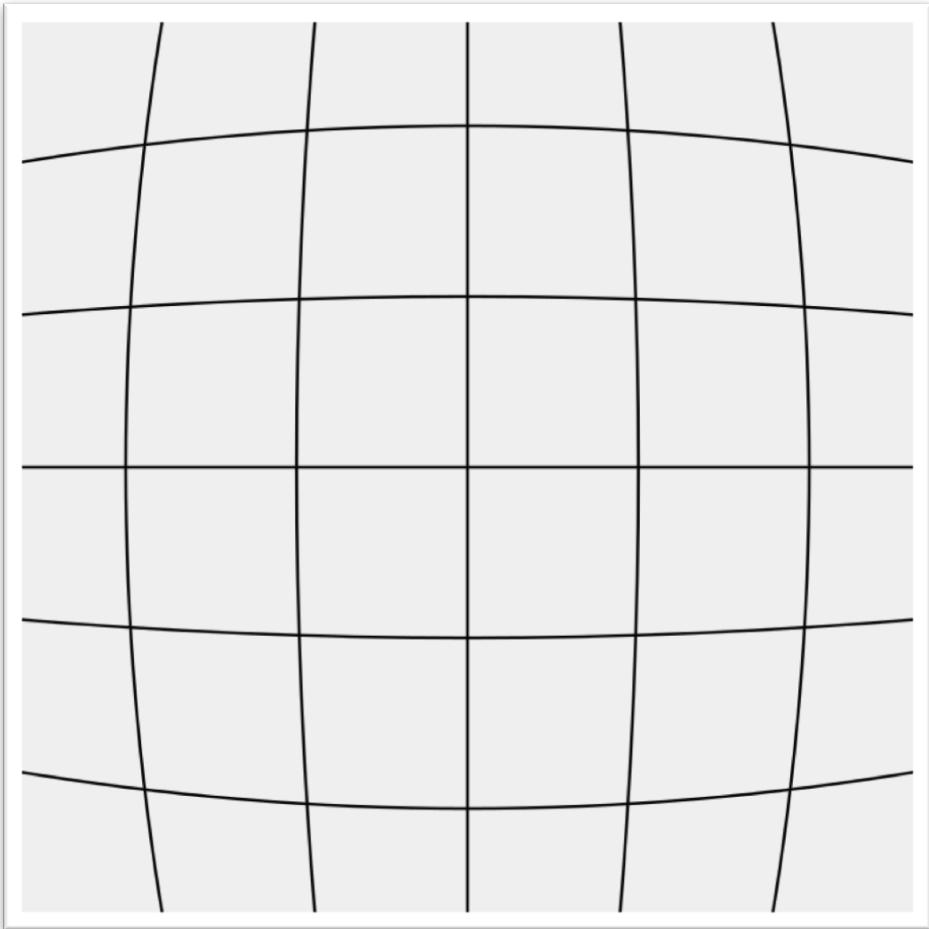
# What are we building?



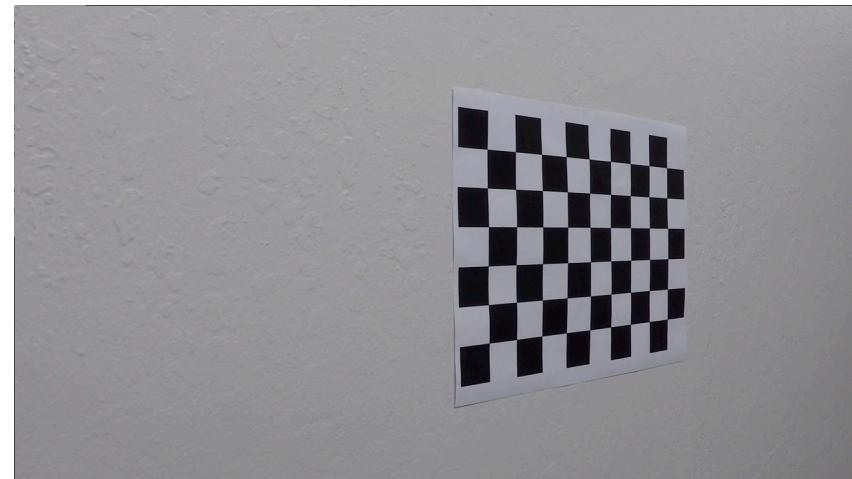
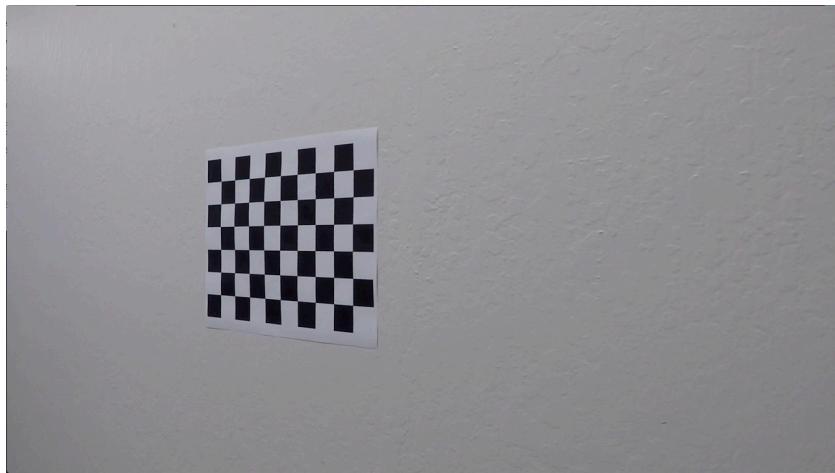
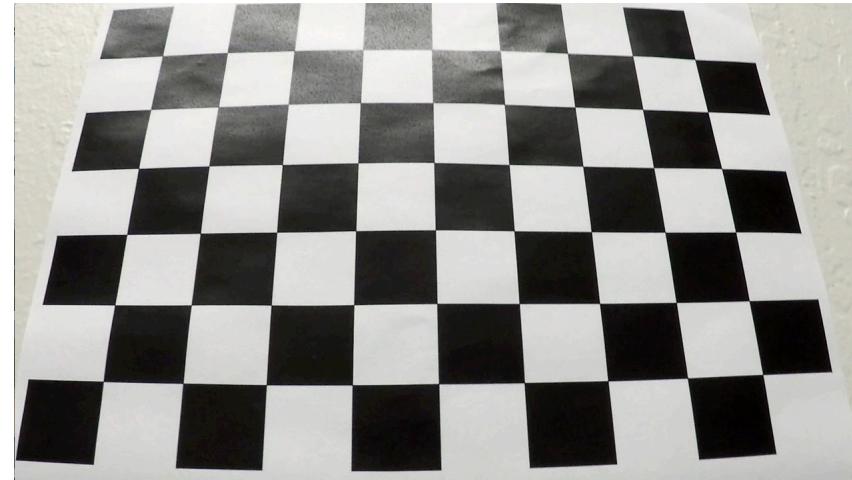
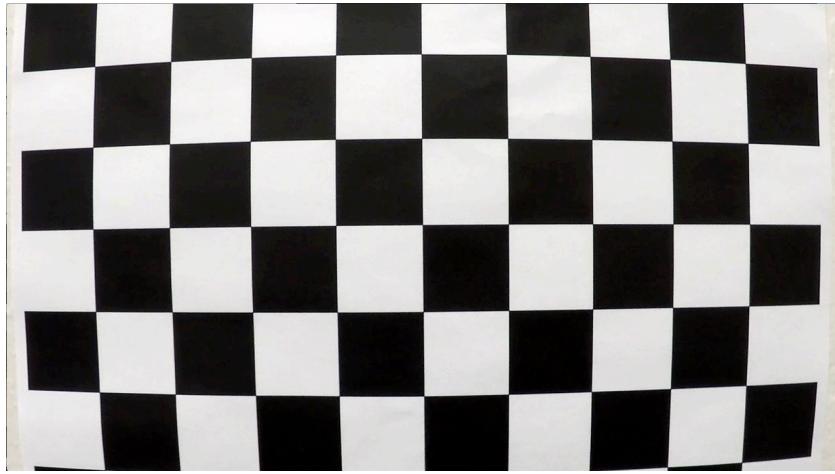
- 1.Undistort
- 2.Warp
- 3.Isolate Lanes
- 4.Curve Fit
- 5.Final Image

# Removing distortion

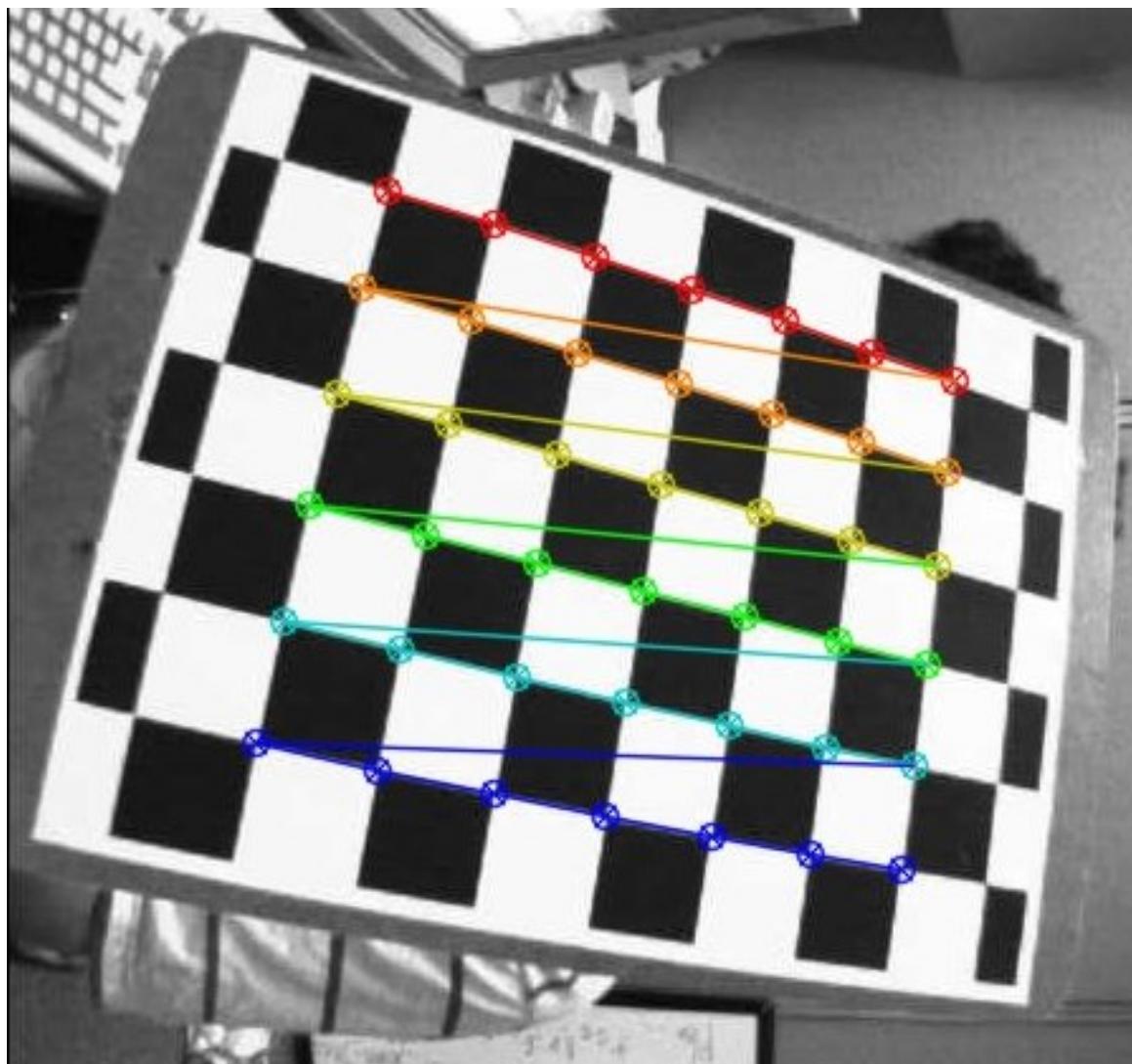
# Camera Lens Distortion



# OpenCV helps us correct for these distortions

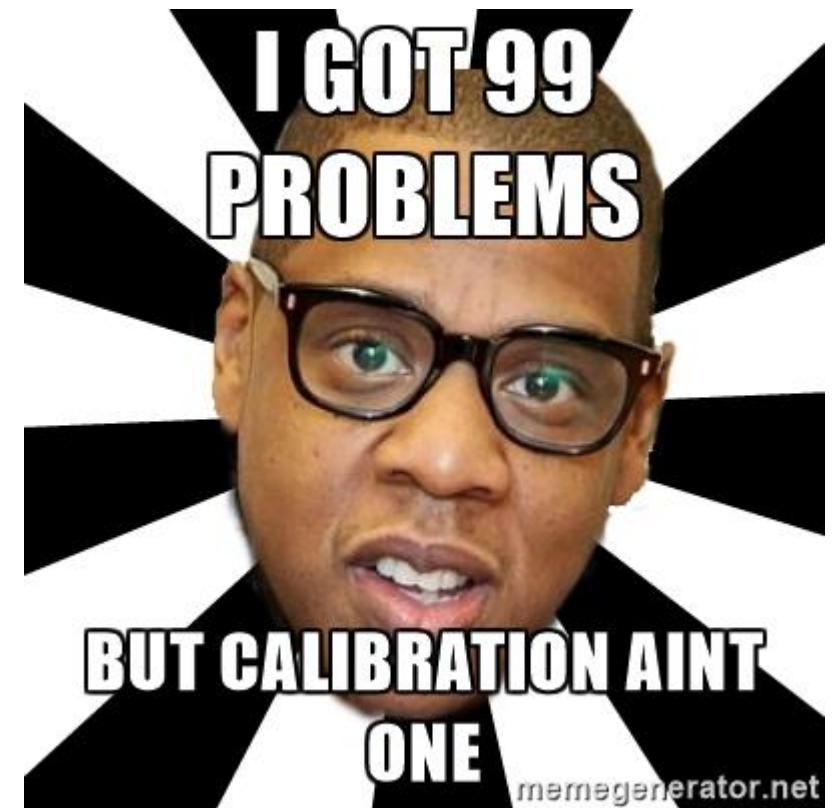


```
ret, corners = cv2.findChessboardCorners(gray, (9,6), None)
```



# Calibrate the camera

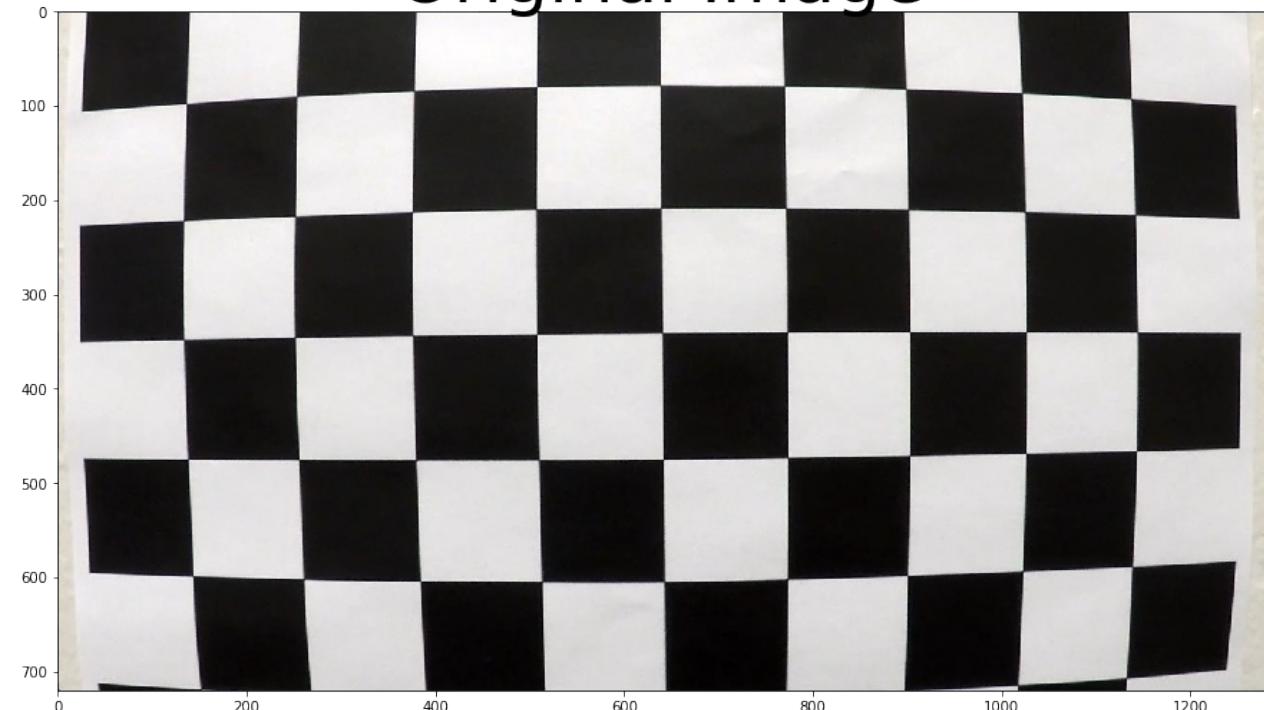
```
ret, mtx, dist, rvecs, tvecs =  
cv2.calibrateCamera(  
    objpoints,  
    imgpoints,  
    gray.shape[::-1],  
    None,  
    None  
)
```



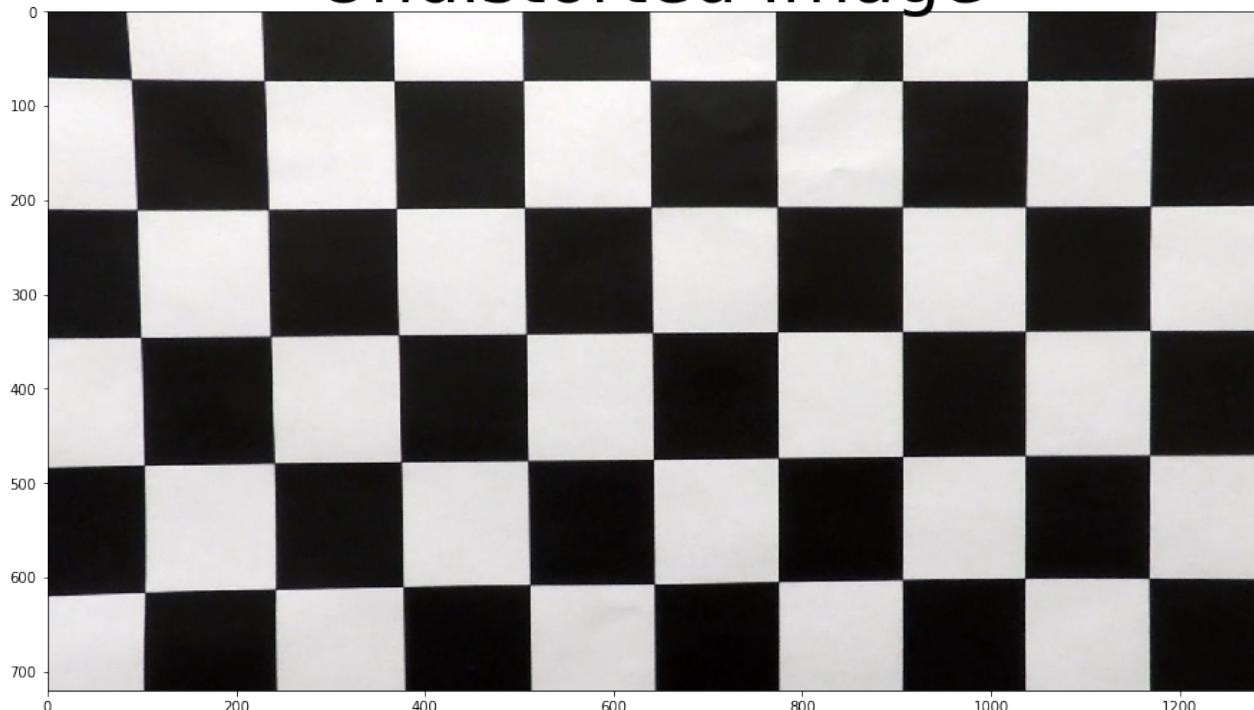
# Use the camera calibration to undistort images

```
undist = cv2.undistort(img, mtx, dist, None, mtx)
```

Original Image

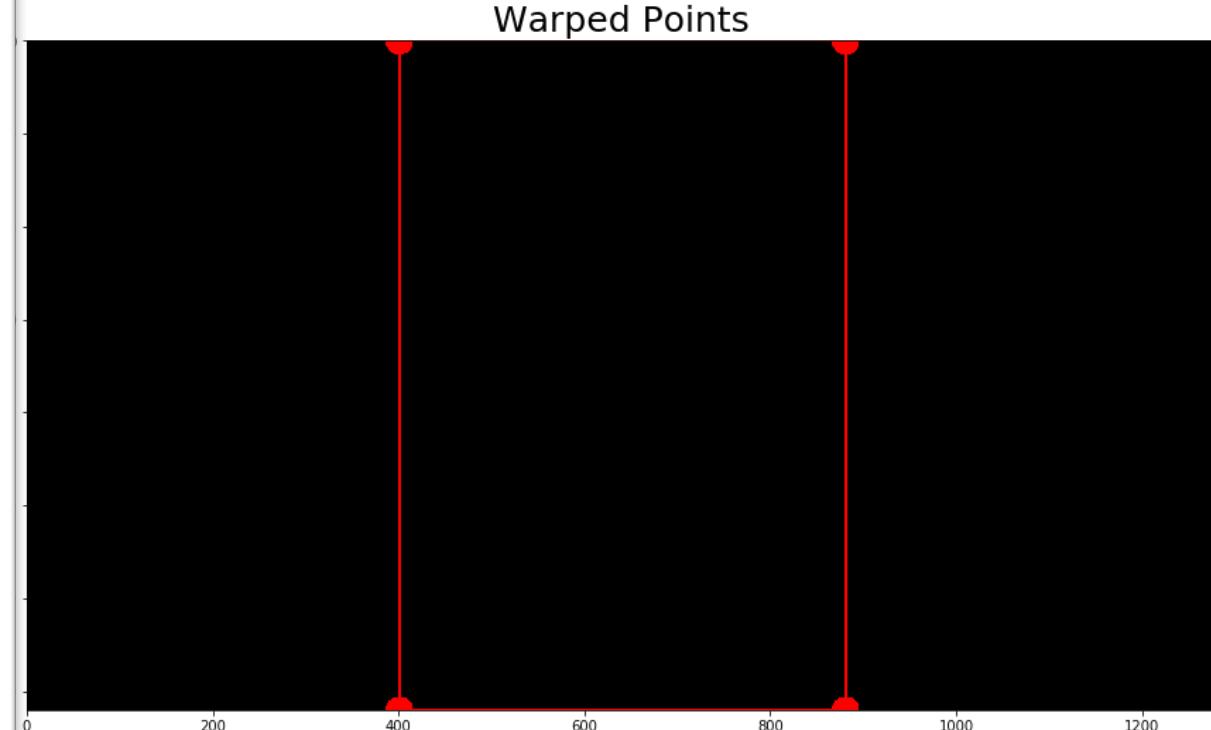
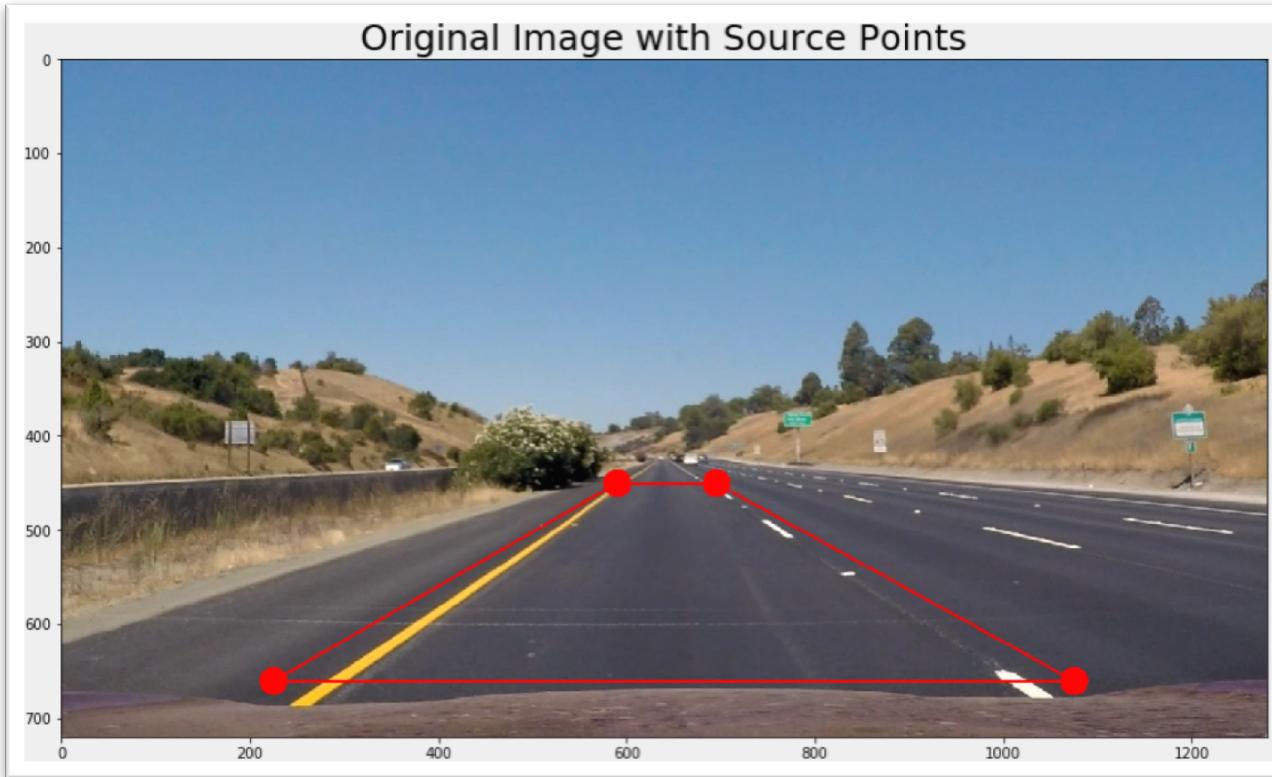


Undistorted Image



# Warping images

# Need a “birds eye view” for curve fitting



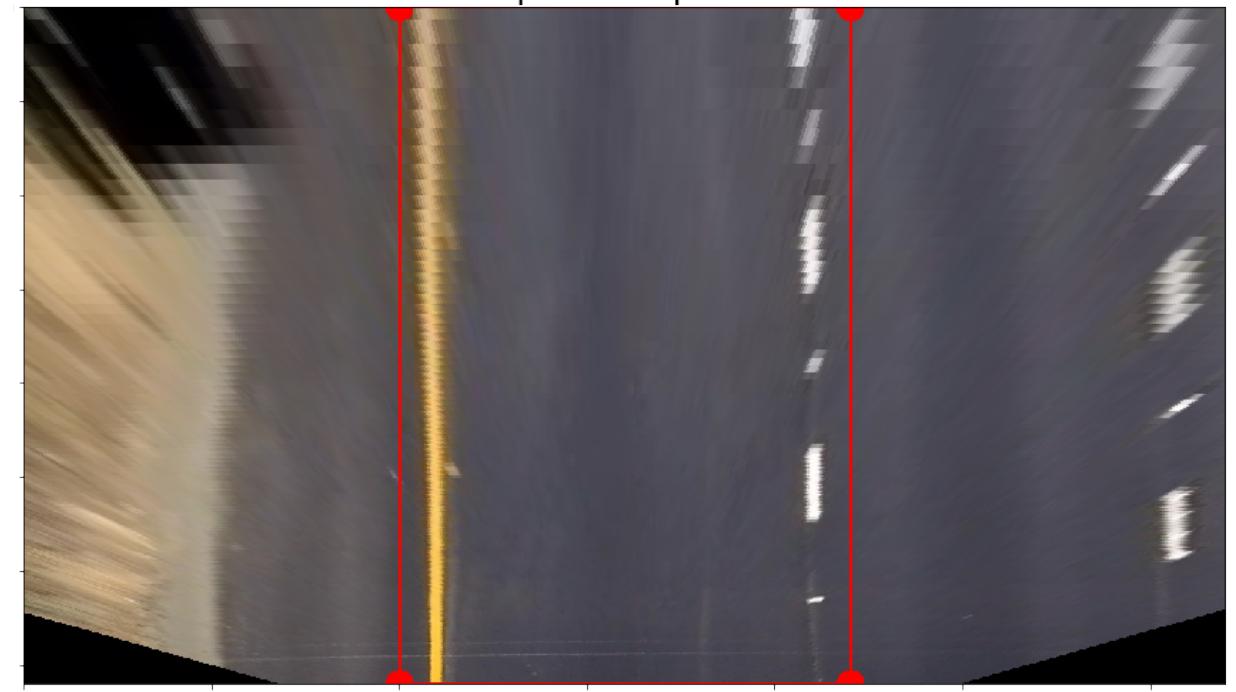
```
# create a transformation matrix  
M = cv2.getPerspectiveTransform(src, dst)
```

```
# apply the transform to the original image  
warped = cv2.warpPerspective(img, M, img_size)
```

Original Image with Source Points

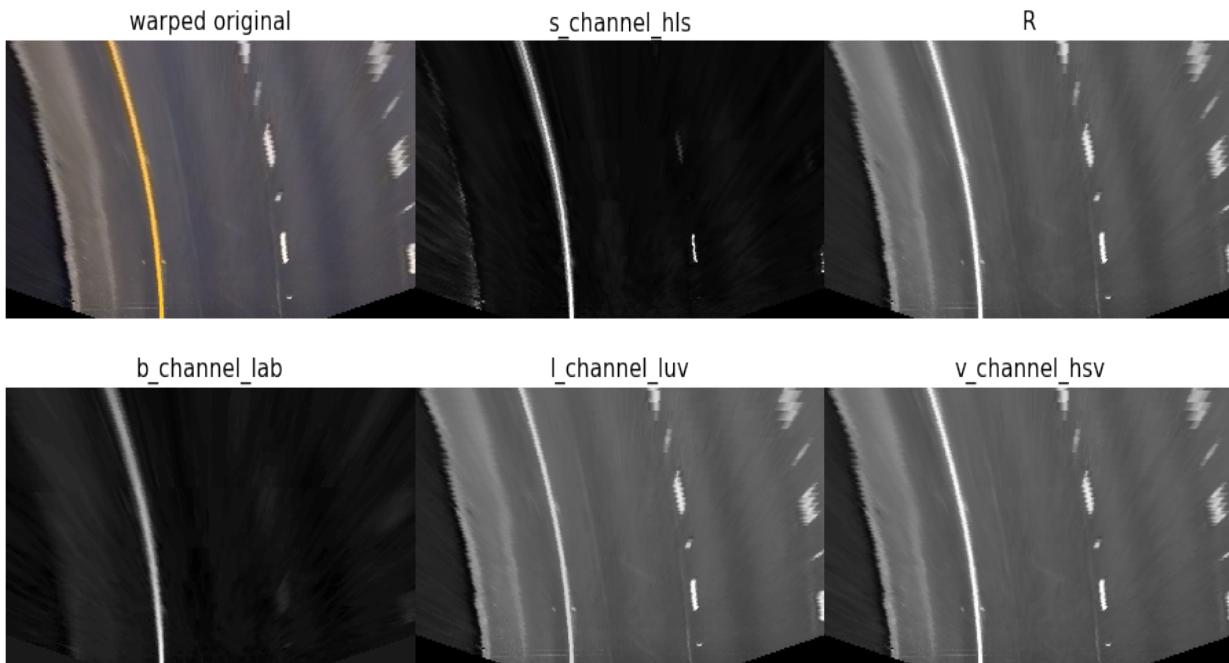


Warped Perspective

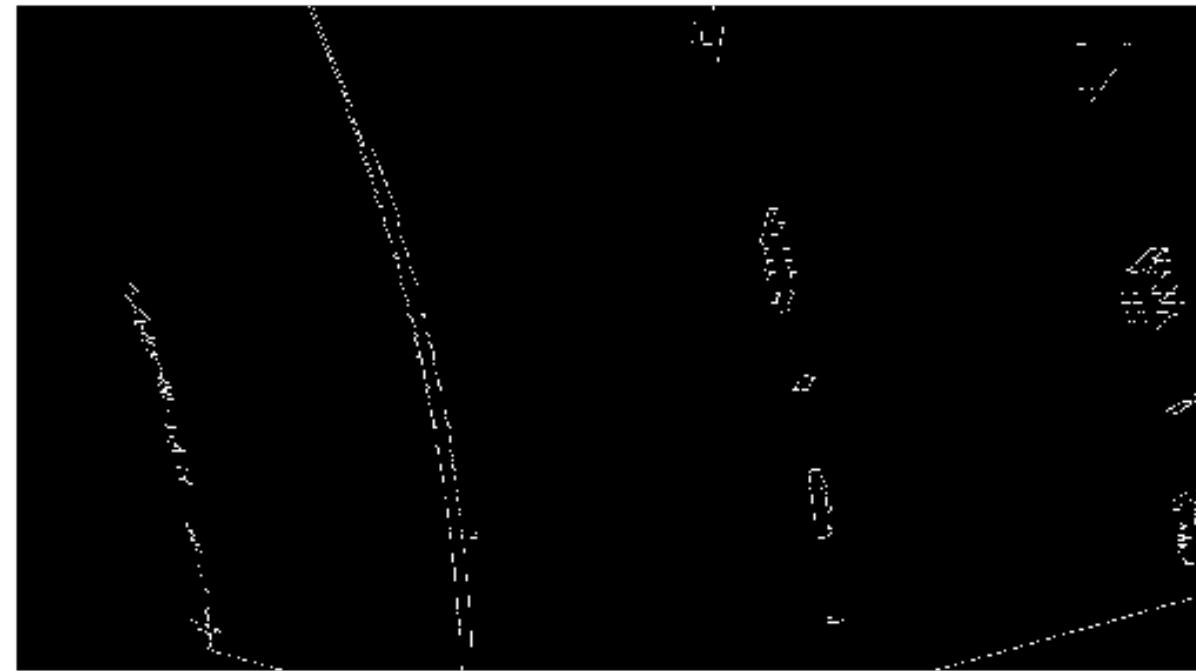


# Finding the lane lines

# Color Selection

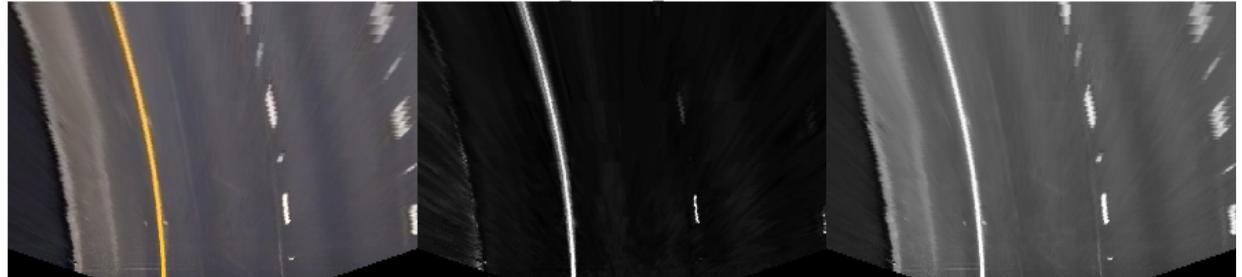


# Edge Detection

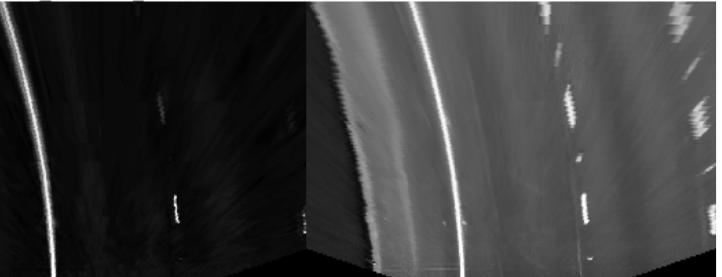


# Color Selection

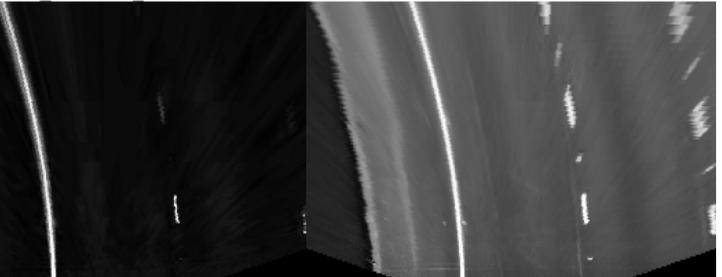
warped original



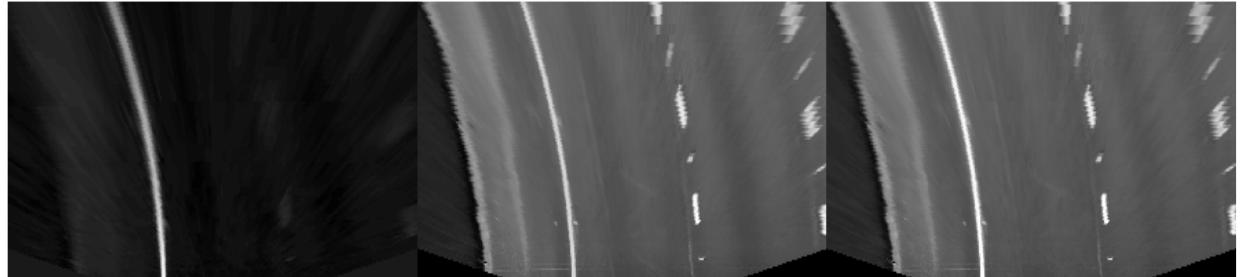
s\_channel\_hls



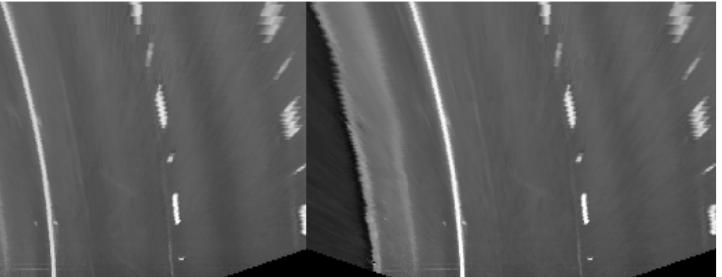
R



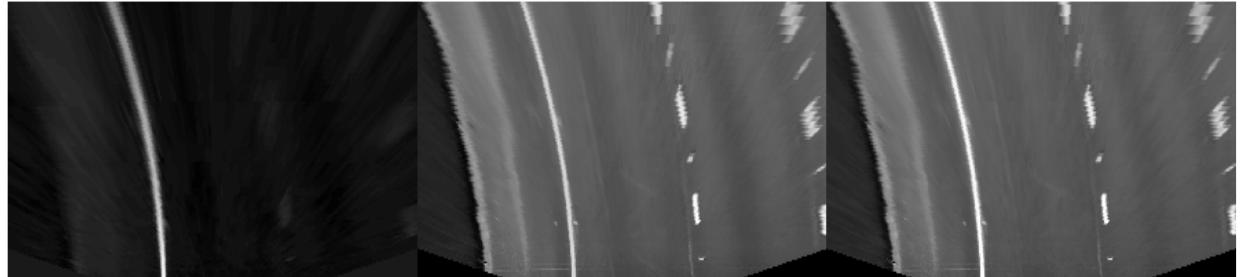
b\_channel\_lab



l\_channel\_luv



v\_channel\_hsv

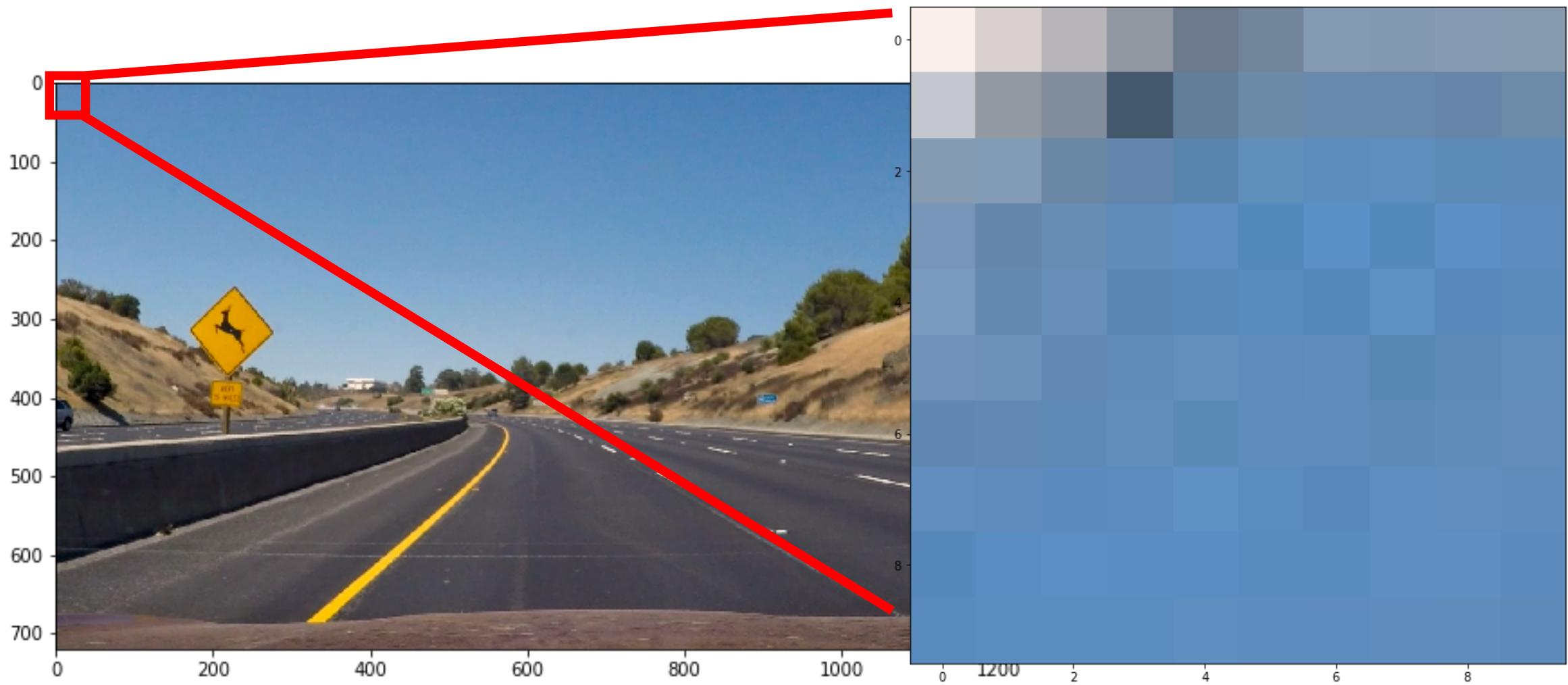


# Edge Detection



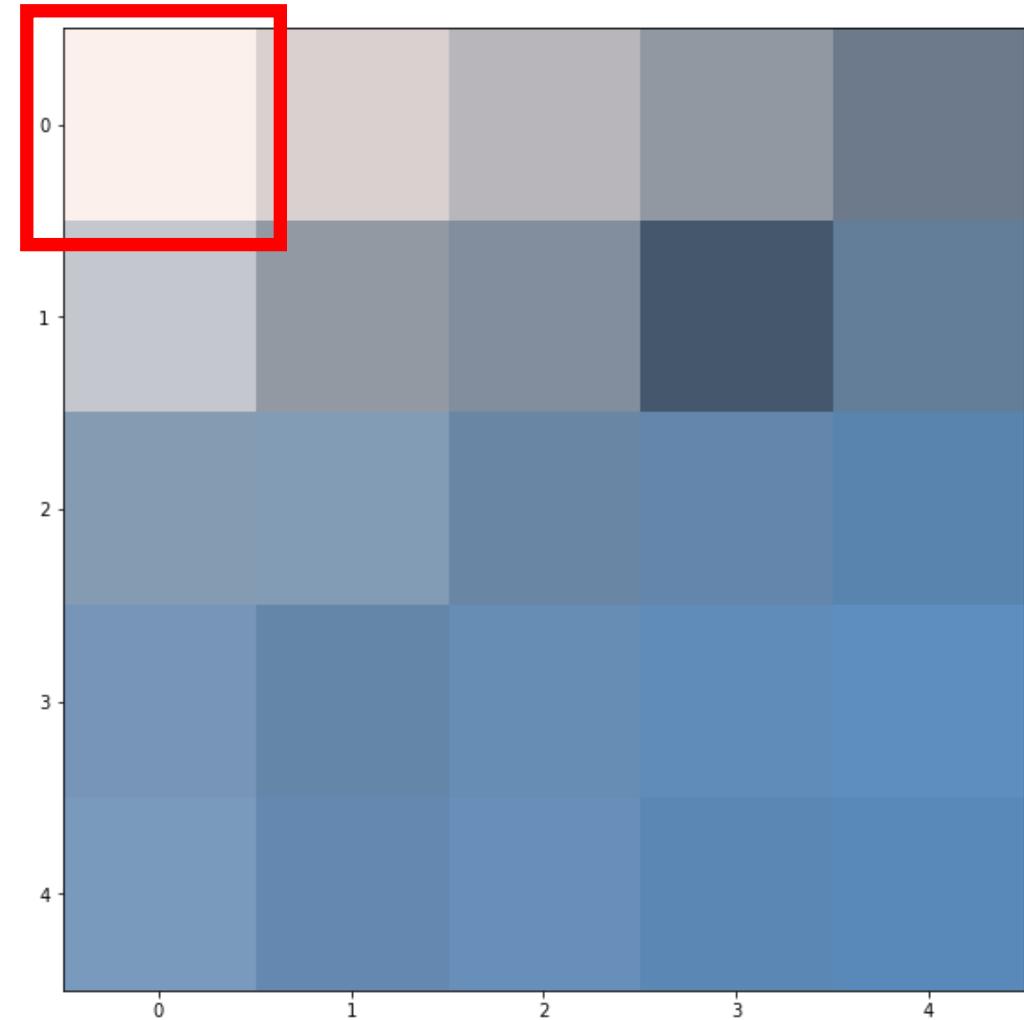
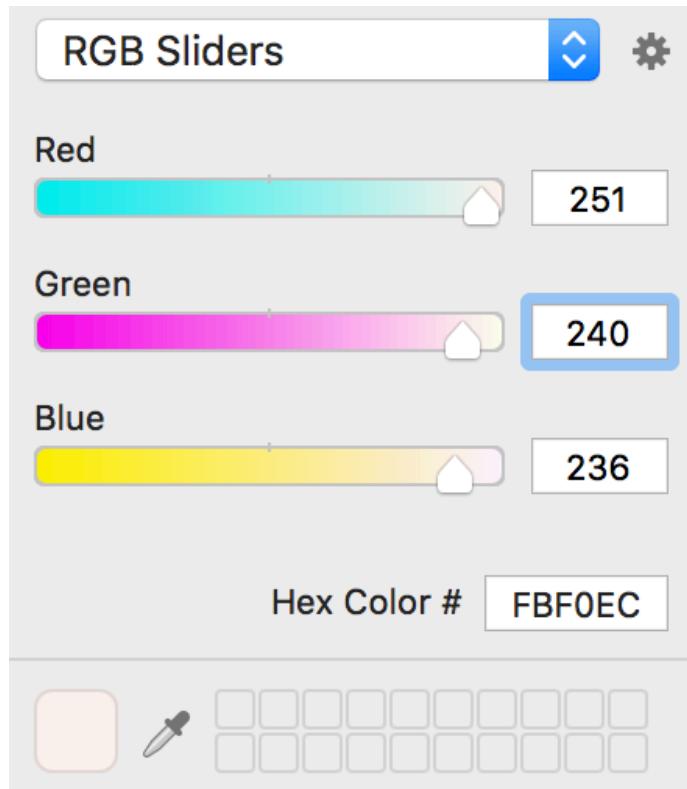
# Anatomy of an image

`img[0:10, 0:10, :]`



```
> img[0,0,:] # first pixel  
array([251, 240, 236])
```

Red      Green      Blue

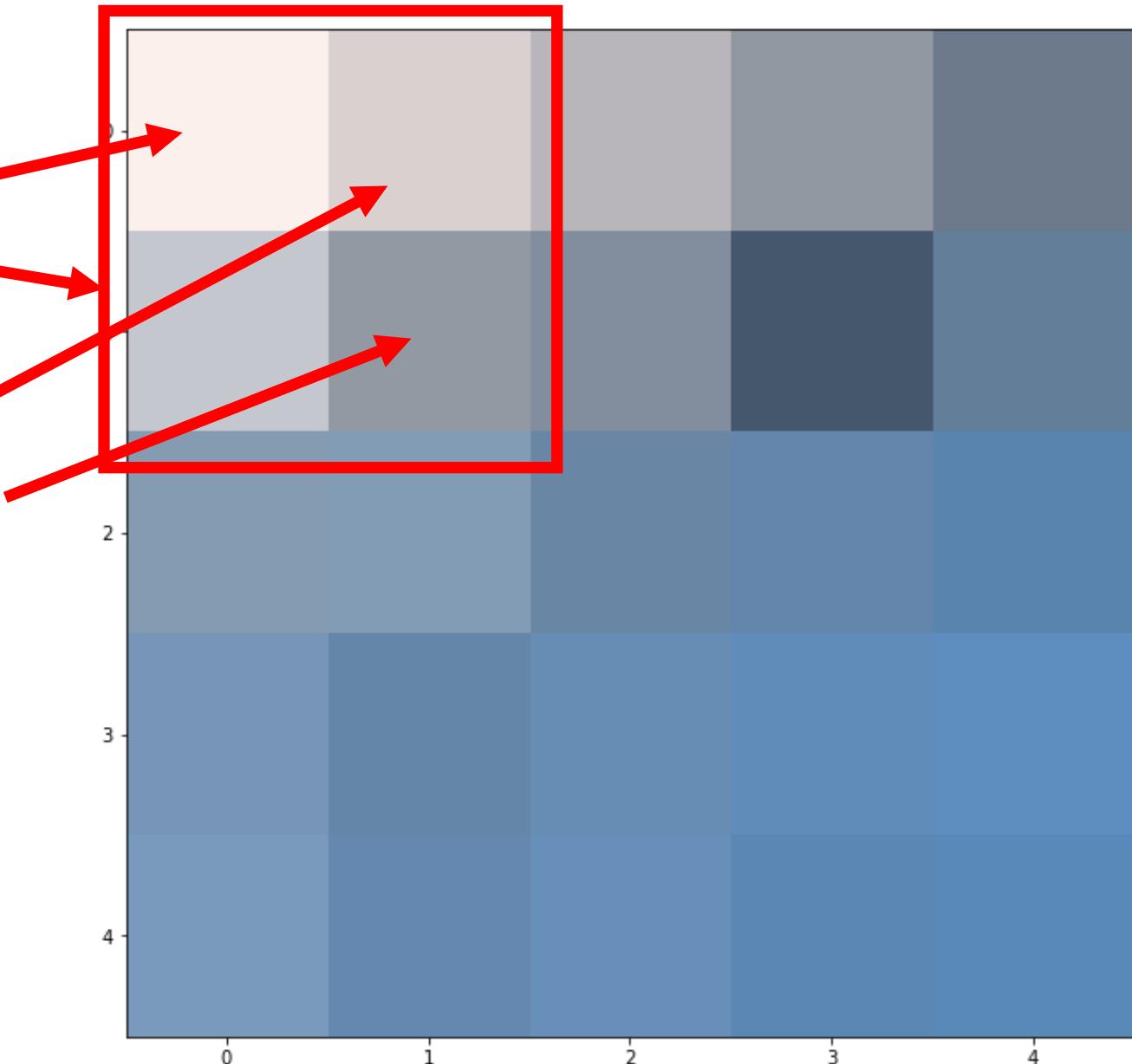


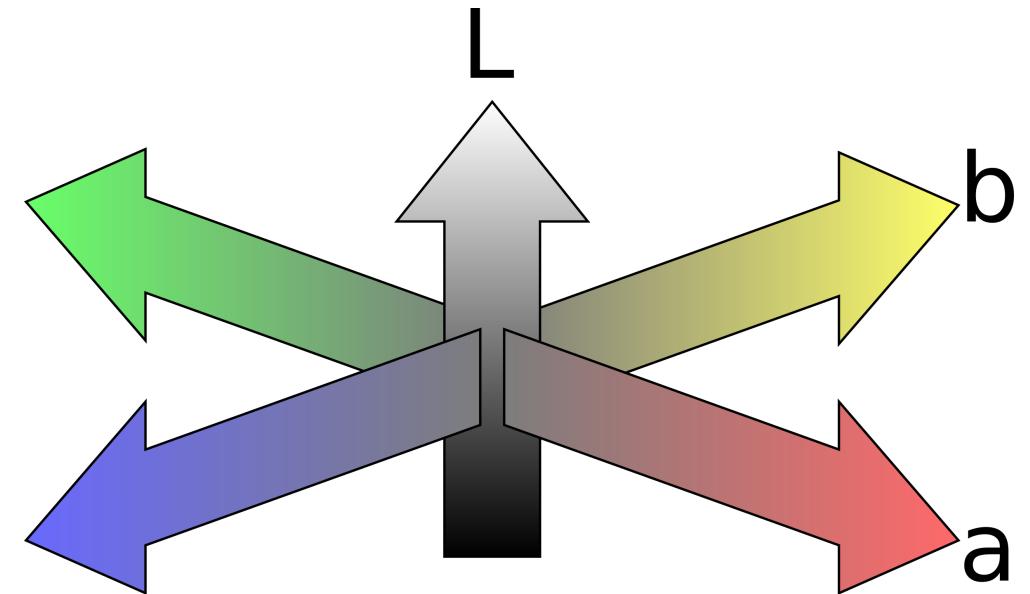
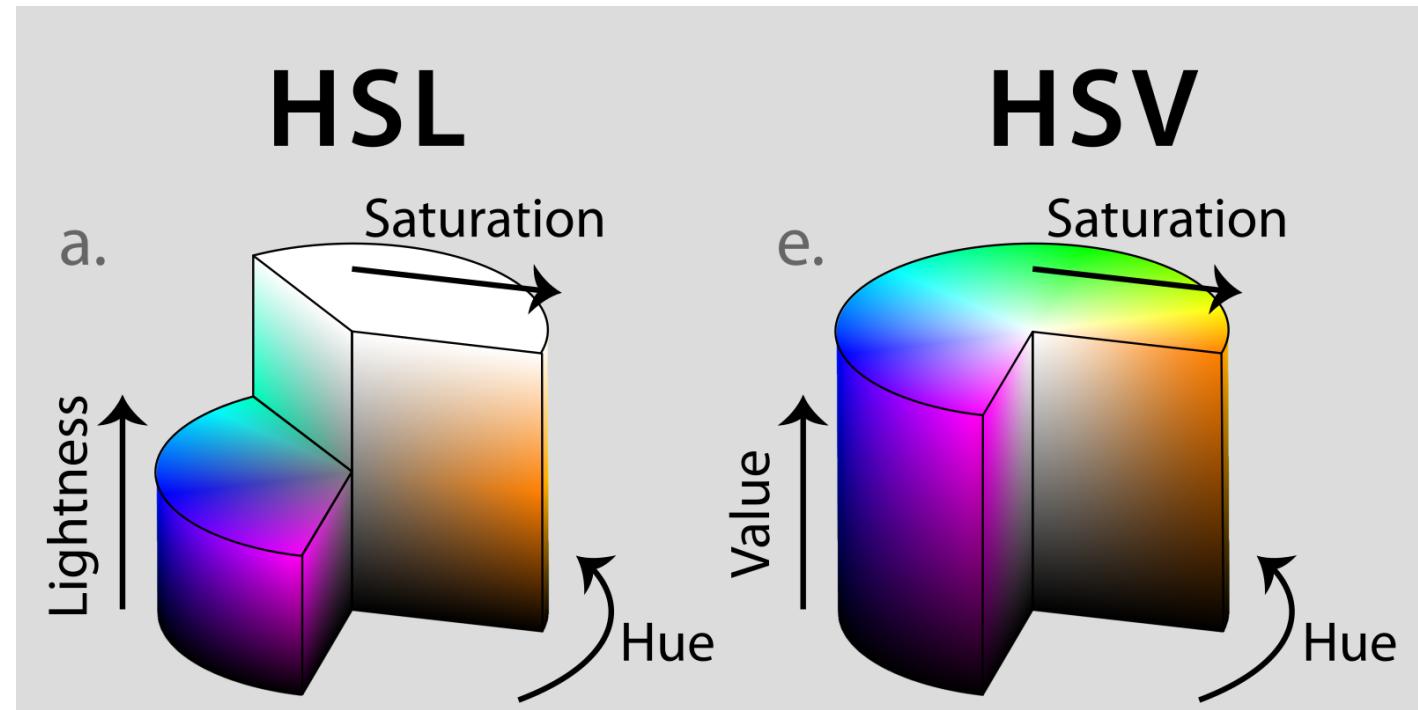
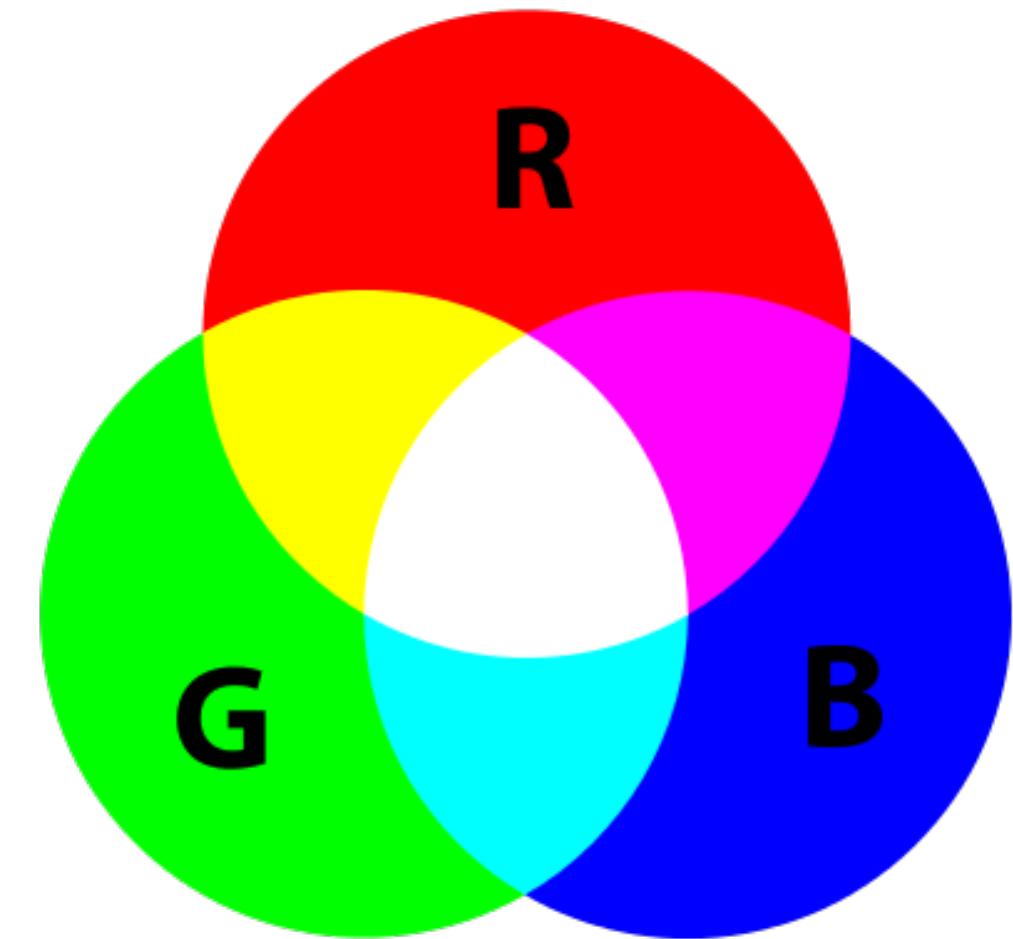
```
> img[0:2,0:2,:]
```

```
array([[[251, 240, 236],  
       [217, 209, 207]],
```

```
      [[196, 199, 206],  
       [146, 153, 163]]],
```

```
)
```





# OpenCV makes changing color space easy

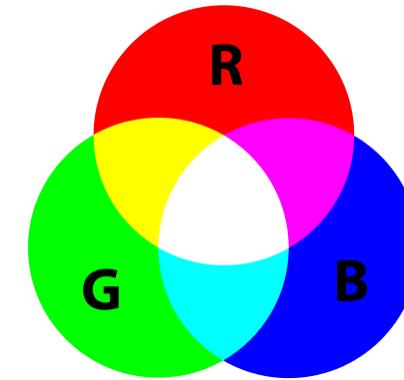
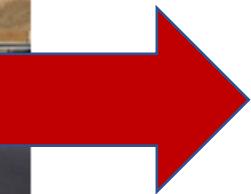
```
# convert RGB image to Hue Light Saturation (HLS)
HLS = cv2.cvtColor(RGB, cv2.COLOR_RGB2HLS)
```

```
> HLS[0,0,:]  
array([ 8, 244, 166])
```



```
> RGB[0,0,:]  
array([251, 240, 236])
```

# Isolate each color channel



```
red    = img[:, :, 0]  
green  = img[:, :, 1]  
blue   = img[:, :, 2]
```

Red Channel



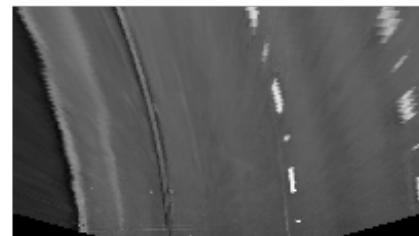
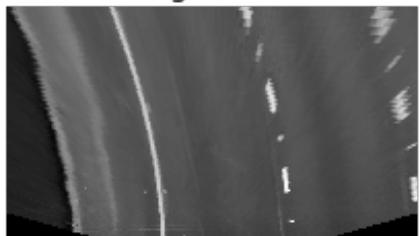
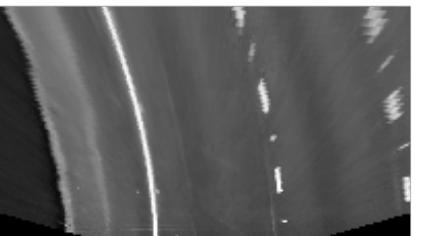
Green Channel



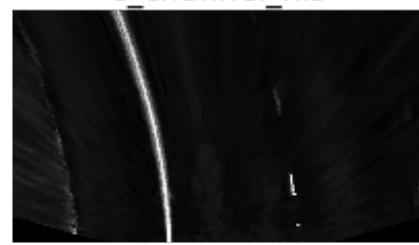
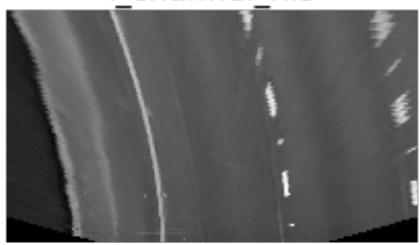
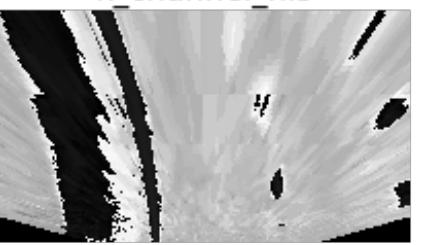
Blue Channel



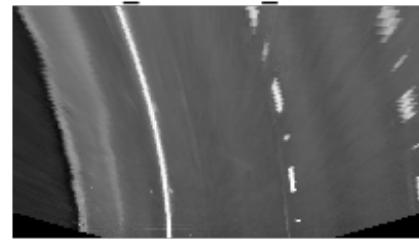
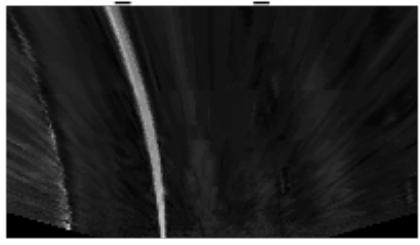
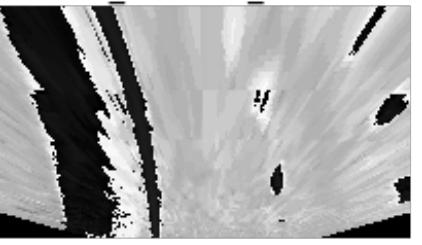
RGB



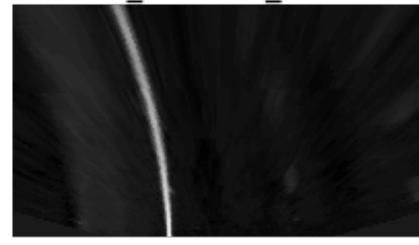
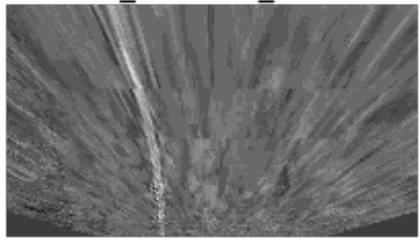
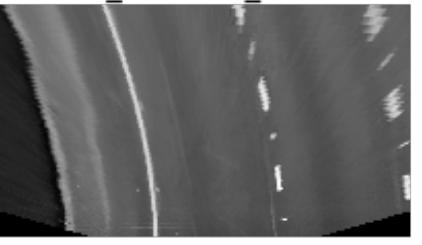
HLS



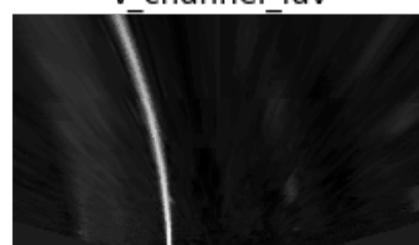
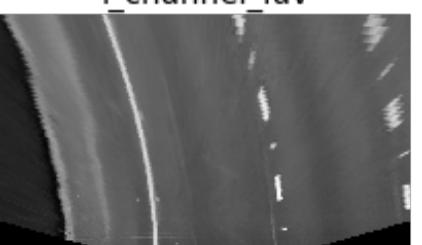
HSV



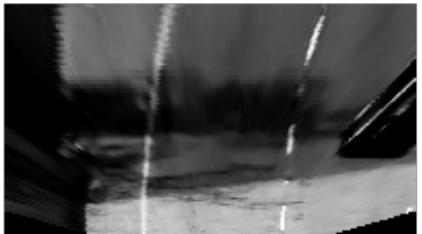
LAB



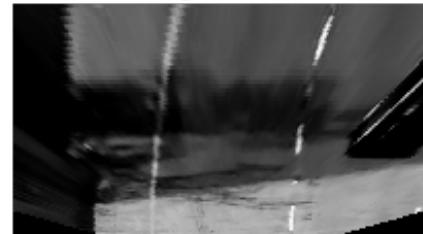
LUV



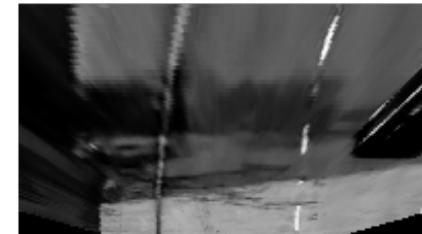
RGB



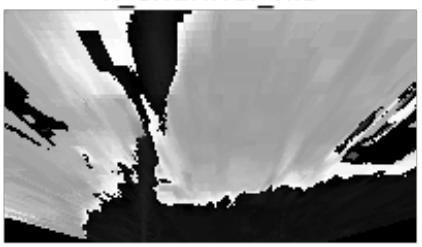
green



blue



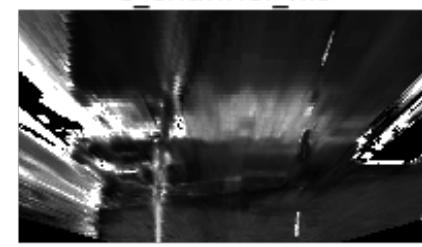
HLS



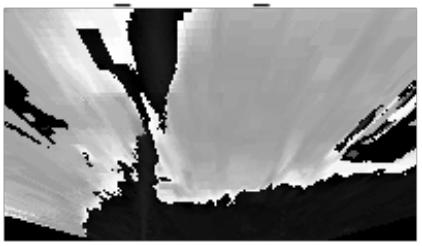
l\_channel\_hls



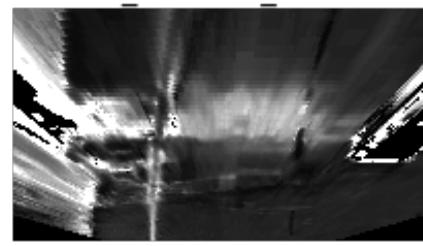
s\_channel\_hls



HSV



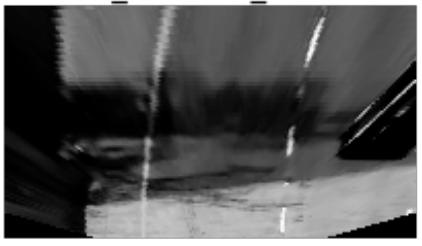
s\_channel\_hsv



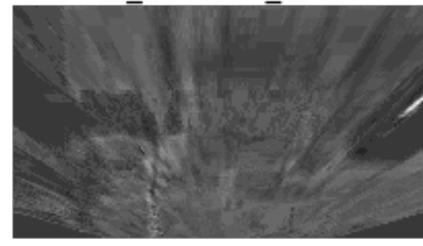
v\_channel\_hsv



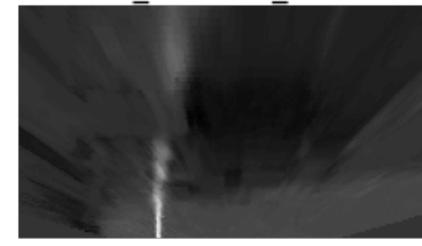
LAB



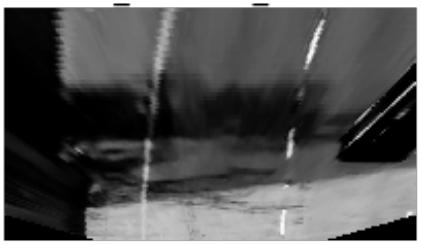
a\_channel\_lab



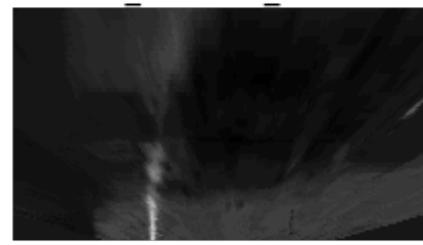
b\_channel\_lab



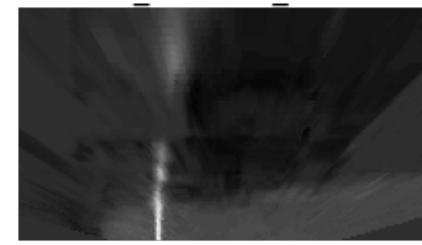
LUV



u\_channel\_luv



v\_channel\_luv

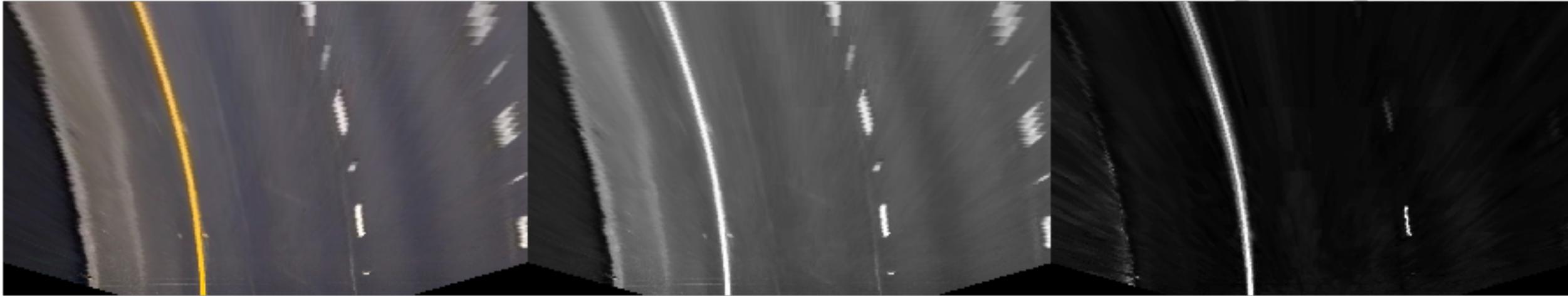


# The best ones...

warped original

red

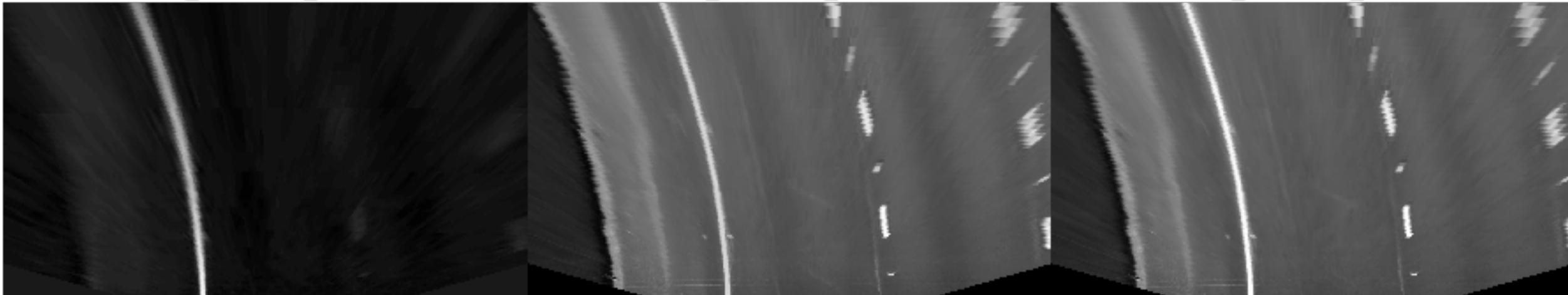
s\_channel\_hls



b\_channel\_lab

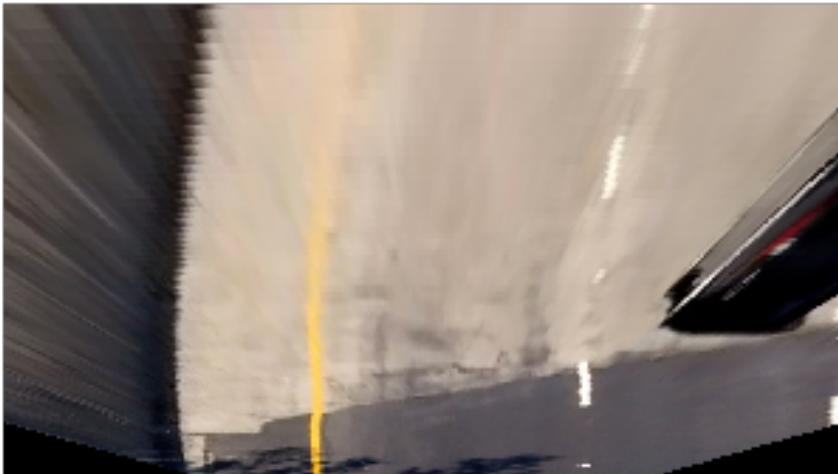
l\_channel\_luv

v\_channel\_hsv

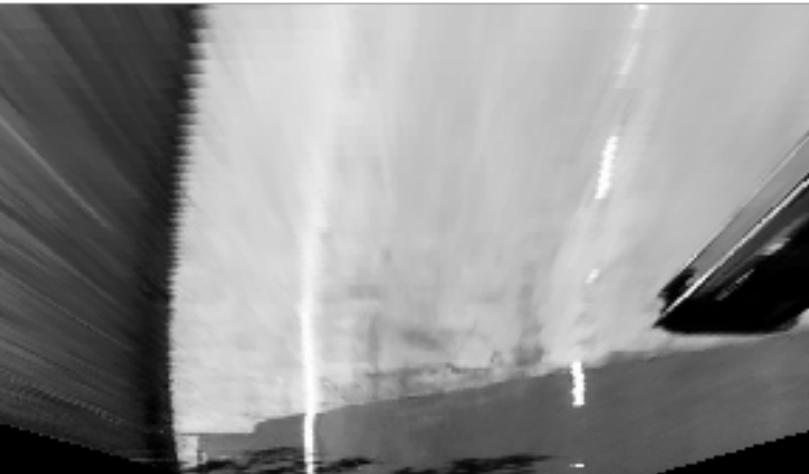


# But some images are harder than others

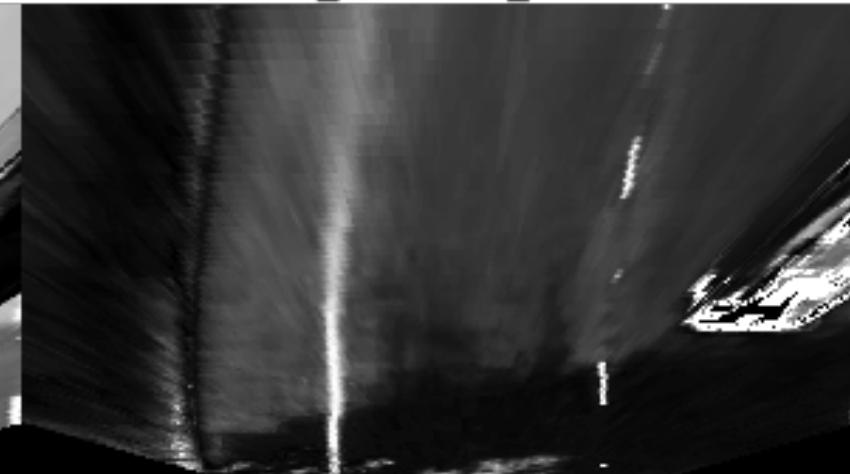
warped original



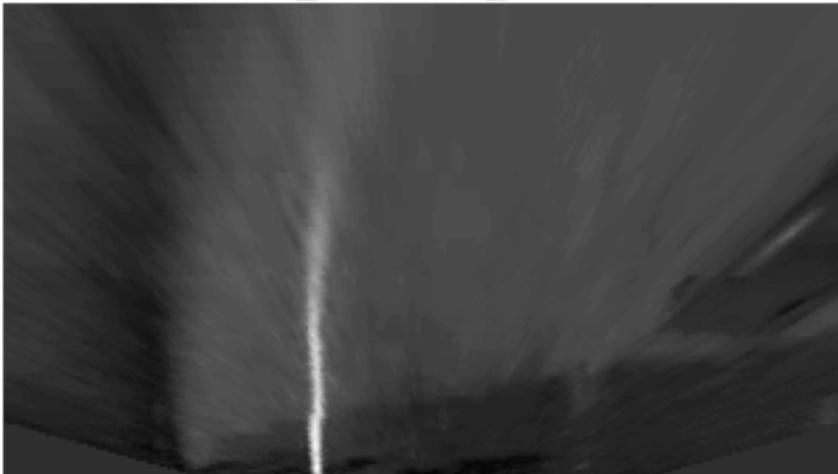
red



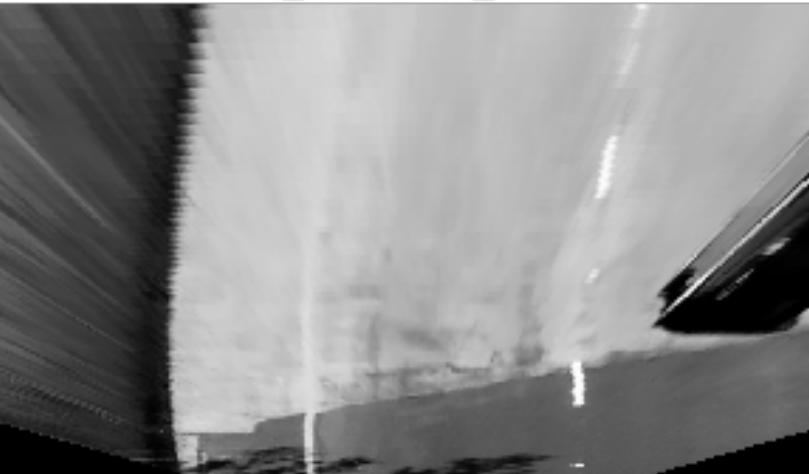
s\_channel\_hls



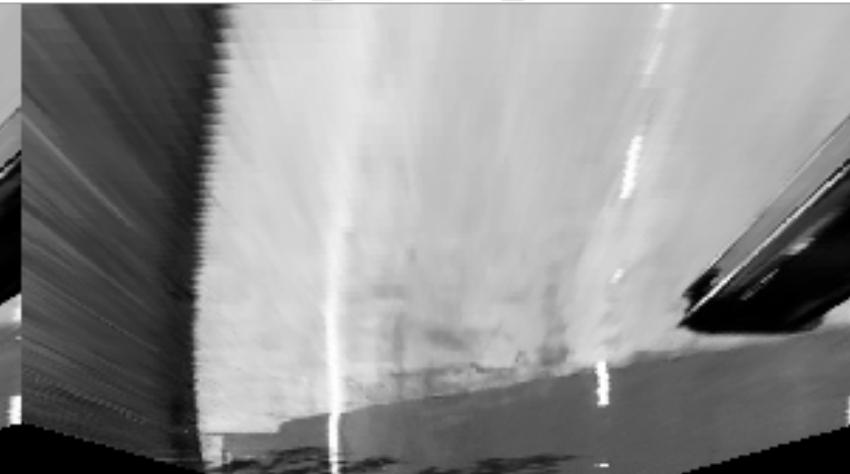
b\_channel\_lab



l\_channel\_luv



v\_channel\_hsv

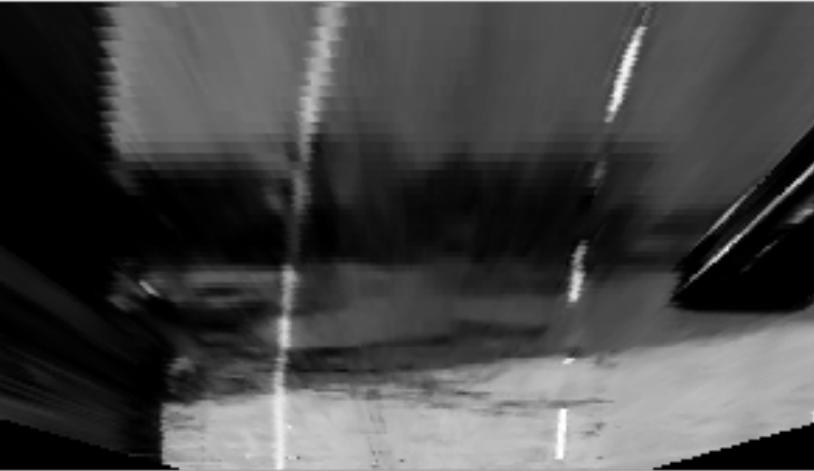


# And some are really hard

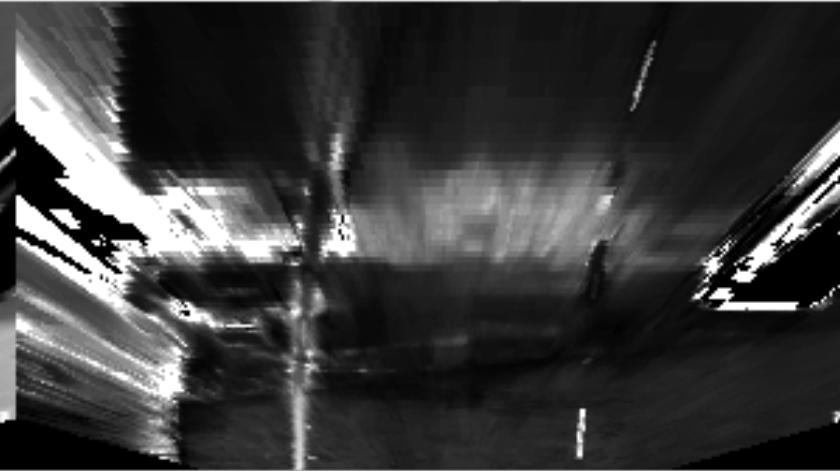
warped original



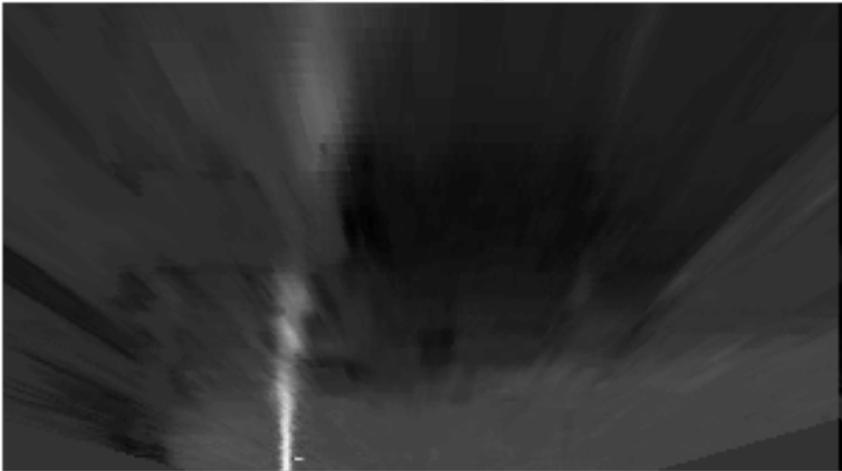
red



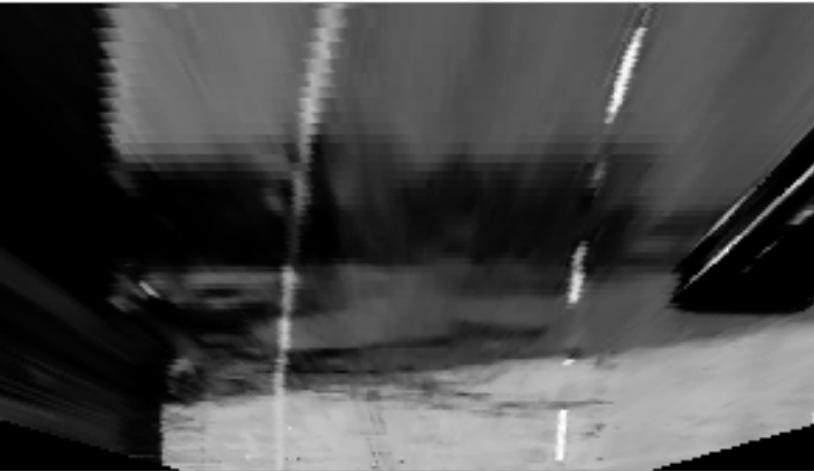
s\_channel\_hls



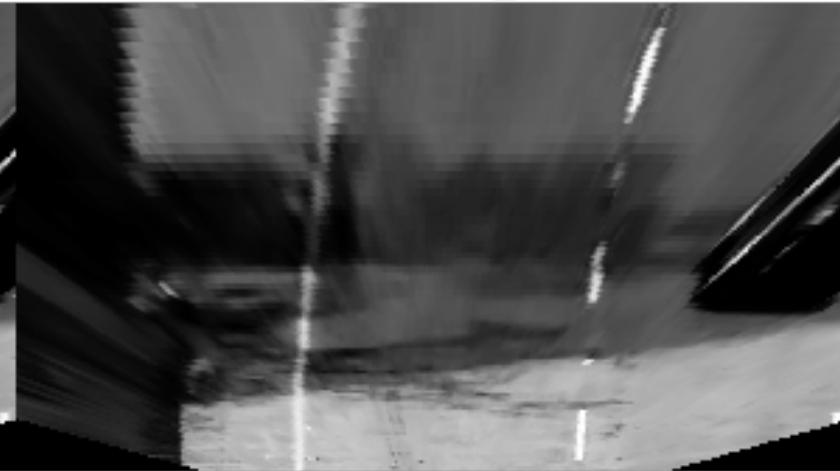
b\_channel\_lab



l\_channel\_luv

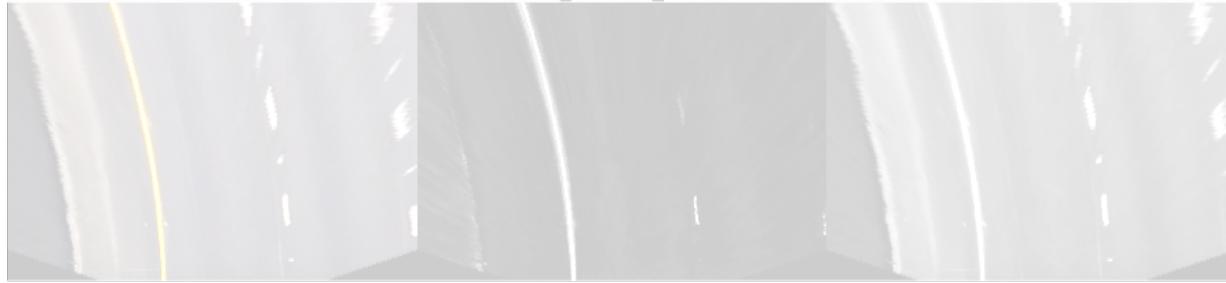


v\_channel\_hsv

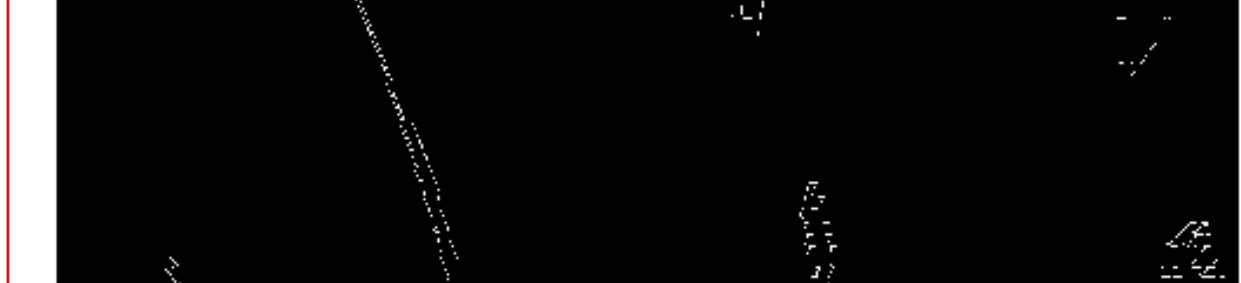


# Color Selection

warped original



s\_channel\_his



R

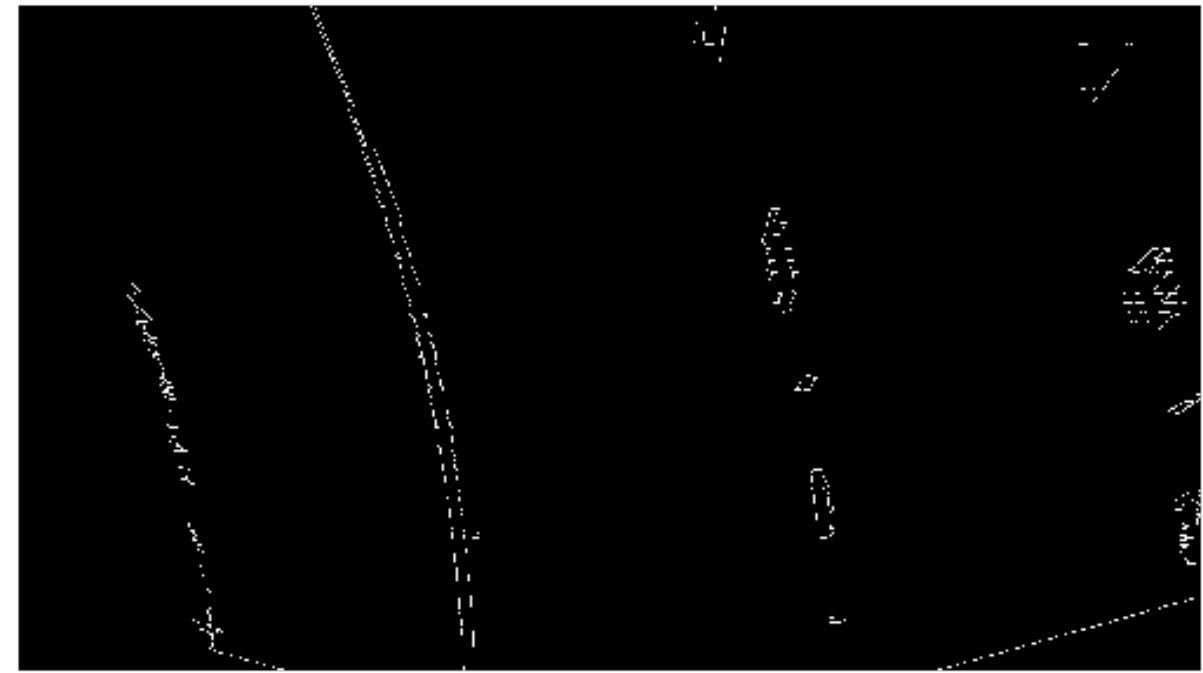


b\_channel\_lab

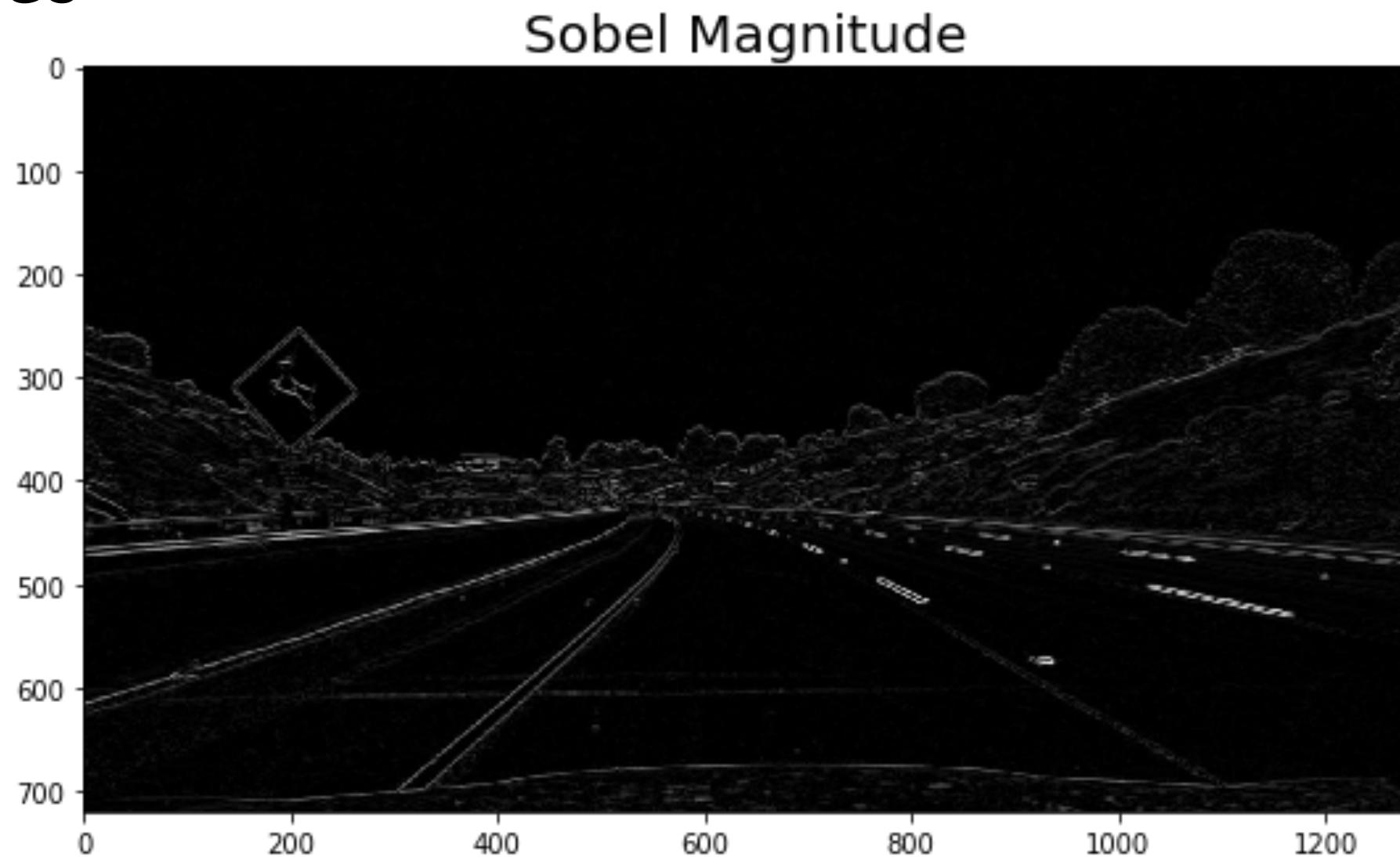
l\_channel\_luv

v\_channel\_hsv

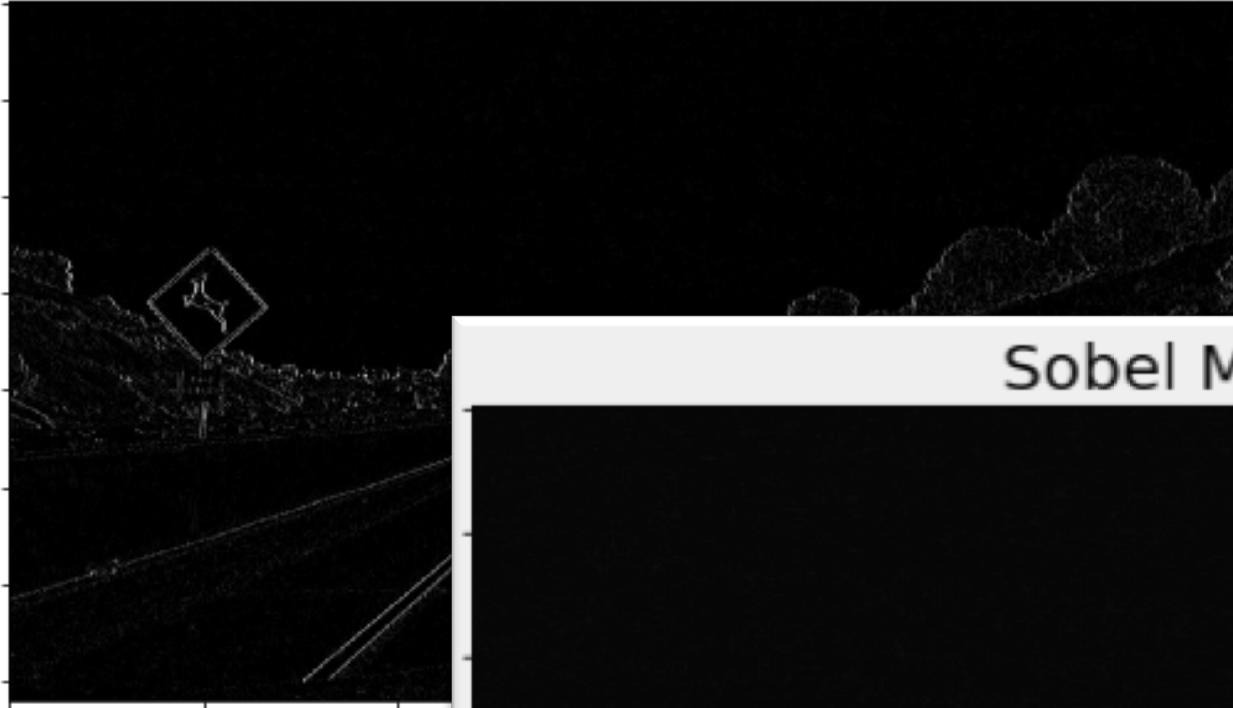
# Edge Detection



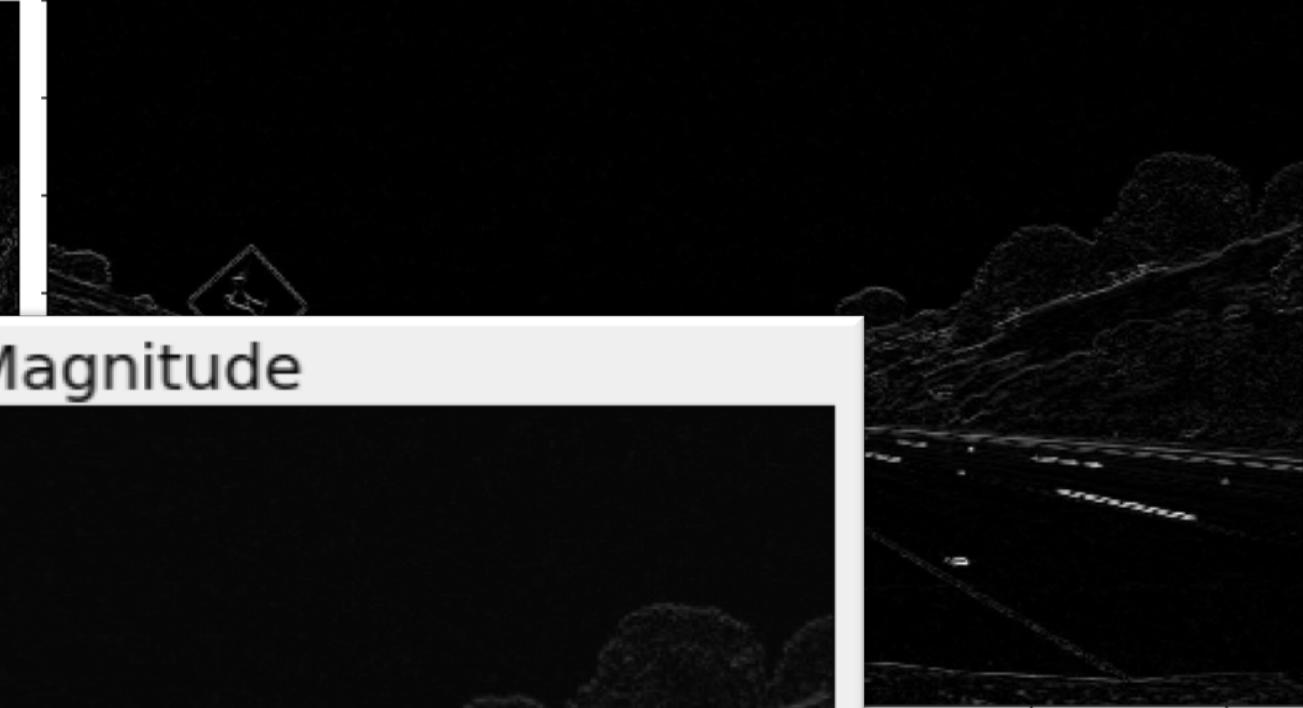
We can use built in OpenCV functions to find edges



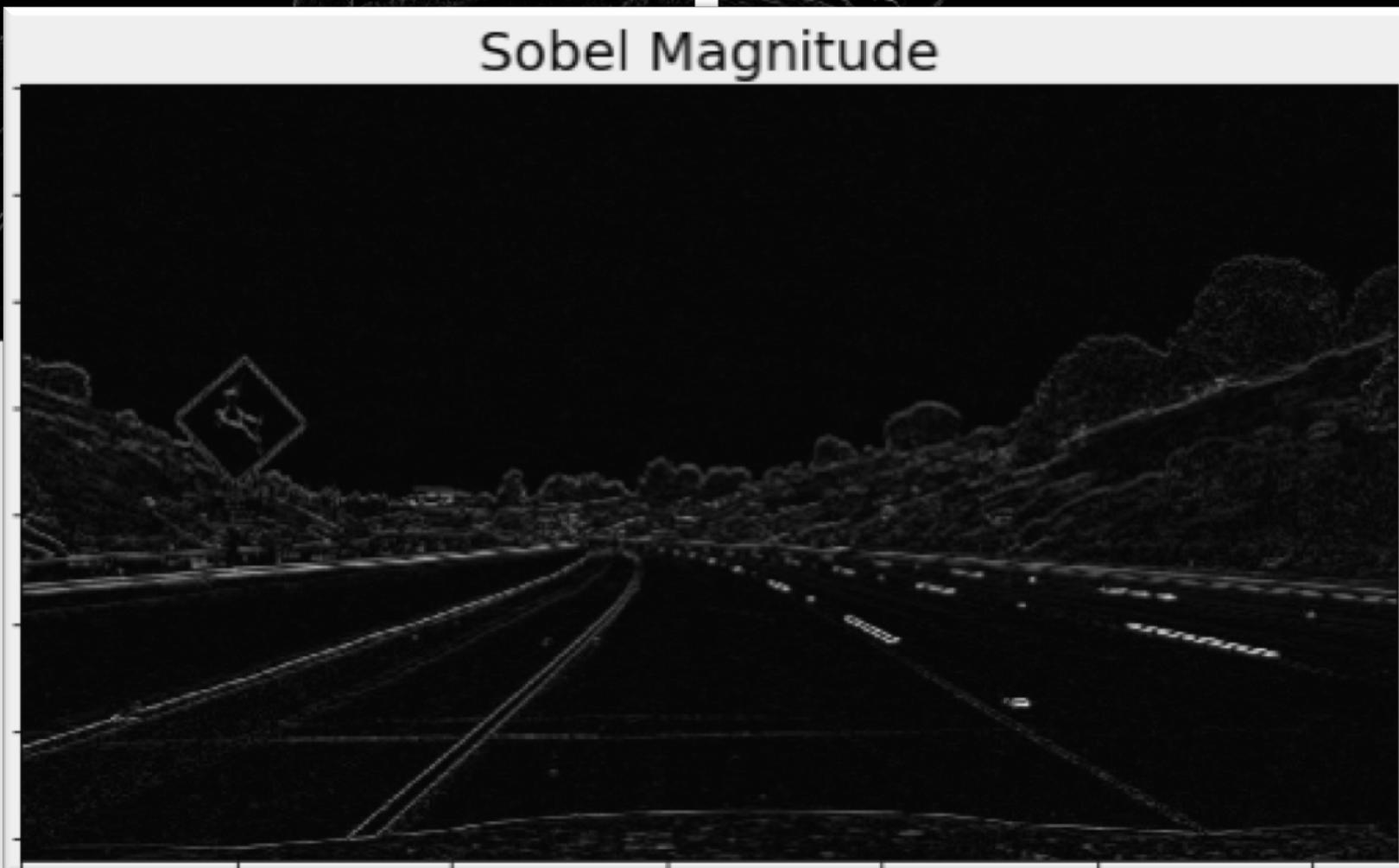
Sobel X



Sobel Y

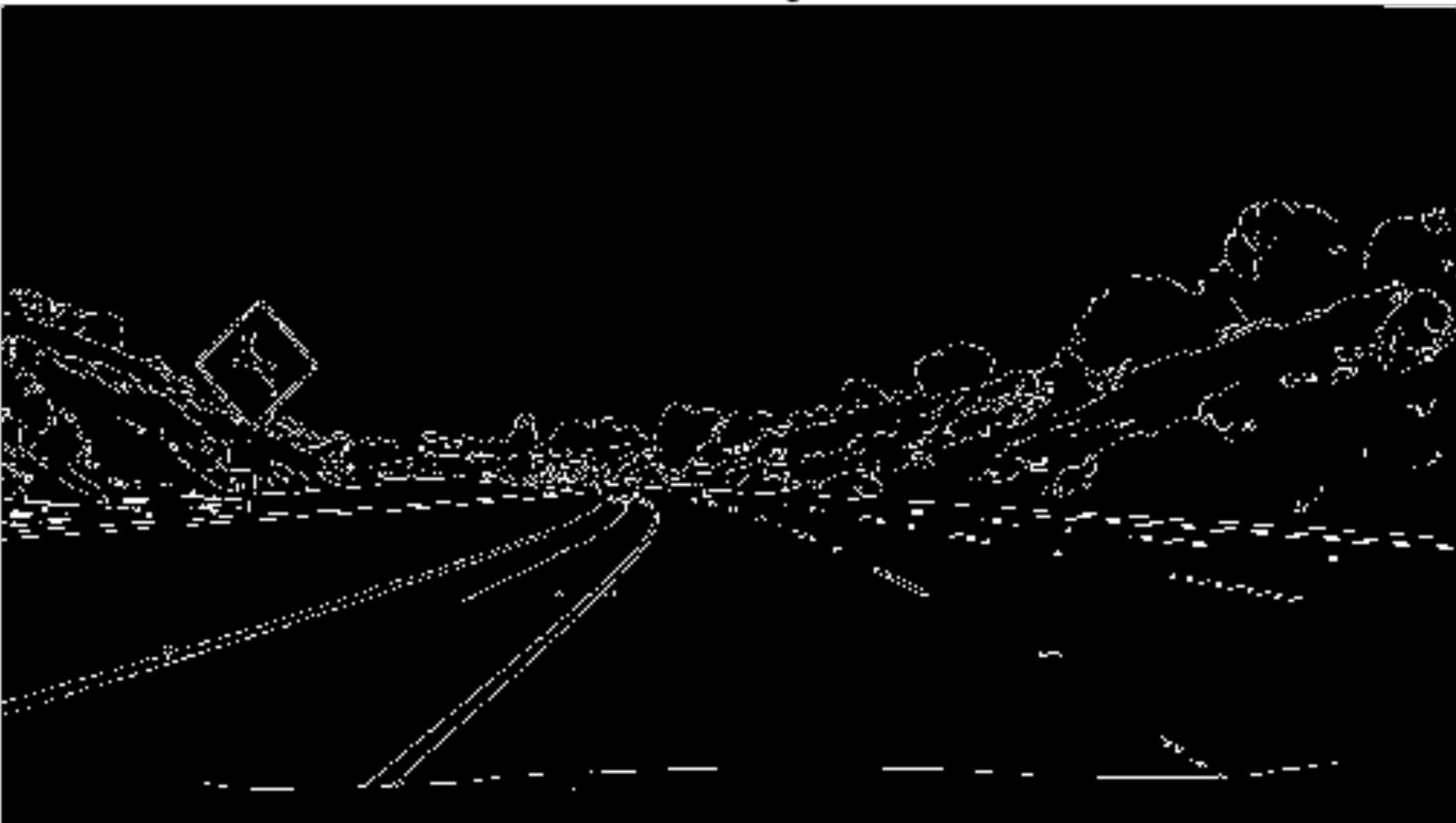


Sobel Magnitude



# Canny edge detection

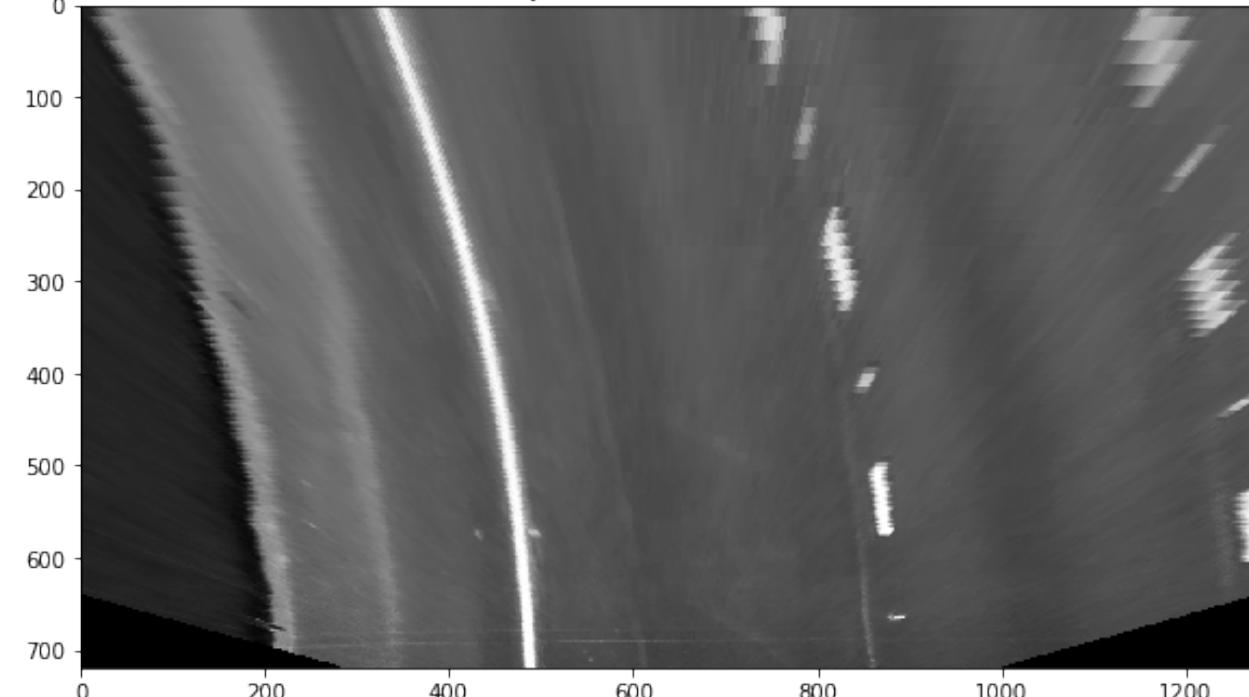
Canny



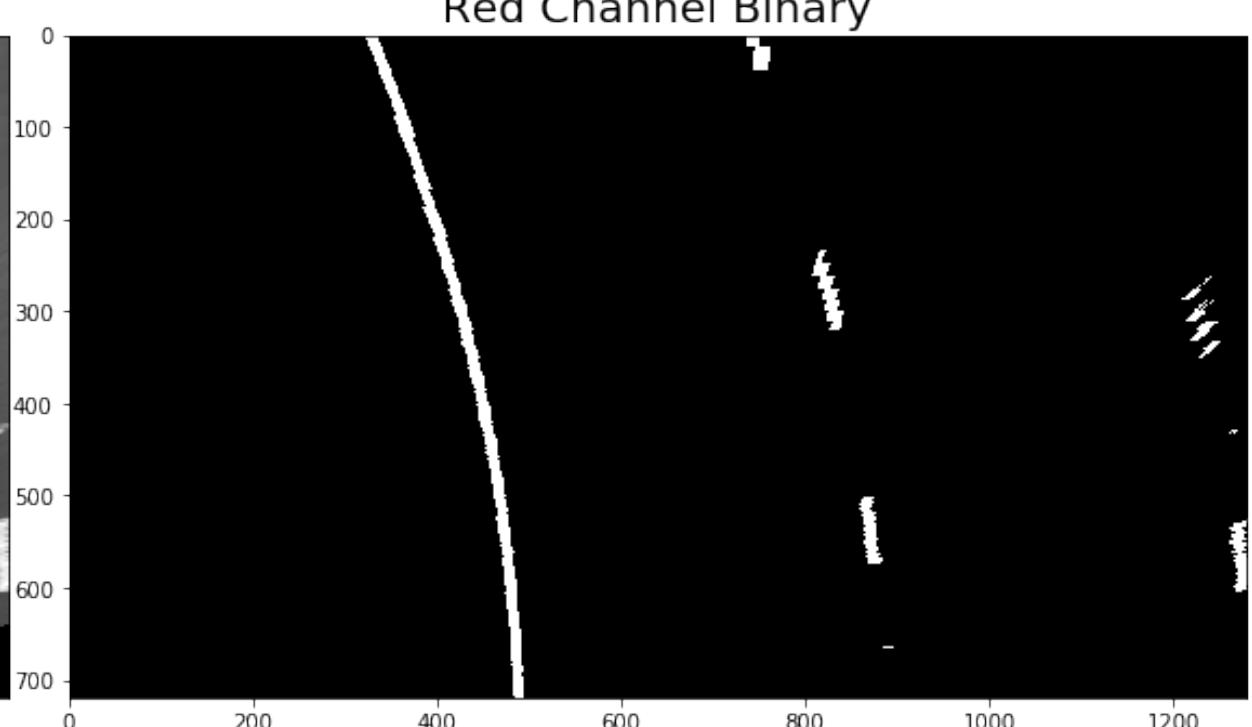
# Use binary thresholding to isolate “hot” pixels

```
channel = warped[:, :, 0]
thresh = (200, 255)
output = np.zeros_like(channel)
output[(channel >= thresh[0]) & (channel <= thresh[1])] = 1
```

Warped Red Channel



Red Channel Binary



# Combine binary threshold images

```
combined_binary = np.zeros_like(red)
combined_binary[ (red == 1) | (sobel == 1) ] = 1
```

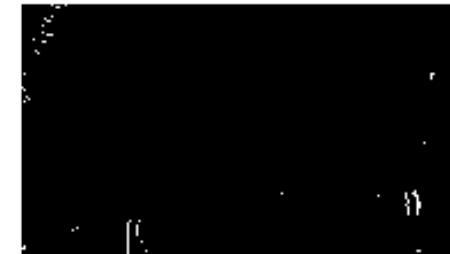
original ./test\_images/test4.jpg



RGB R Channel



Sobel



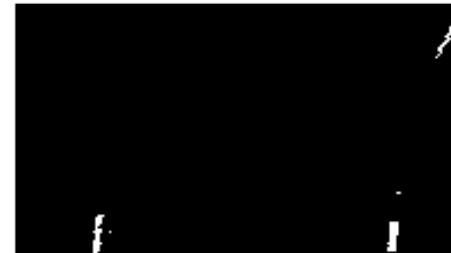
Final Binary



original ./test\_images/test5.jpg



RGB R Channel



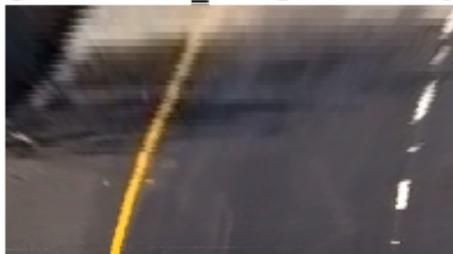
Sobel



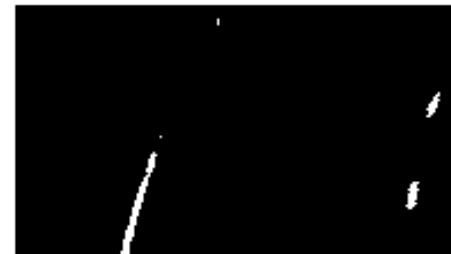
Final Binary



original ./test\_images/test6.jpg



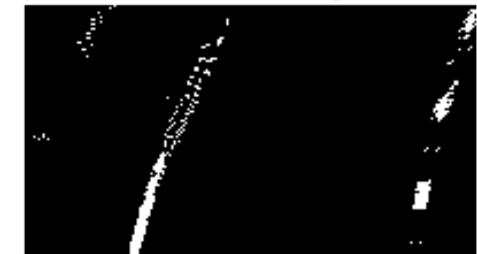
RGB R Channel



Sobel



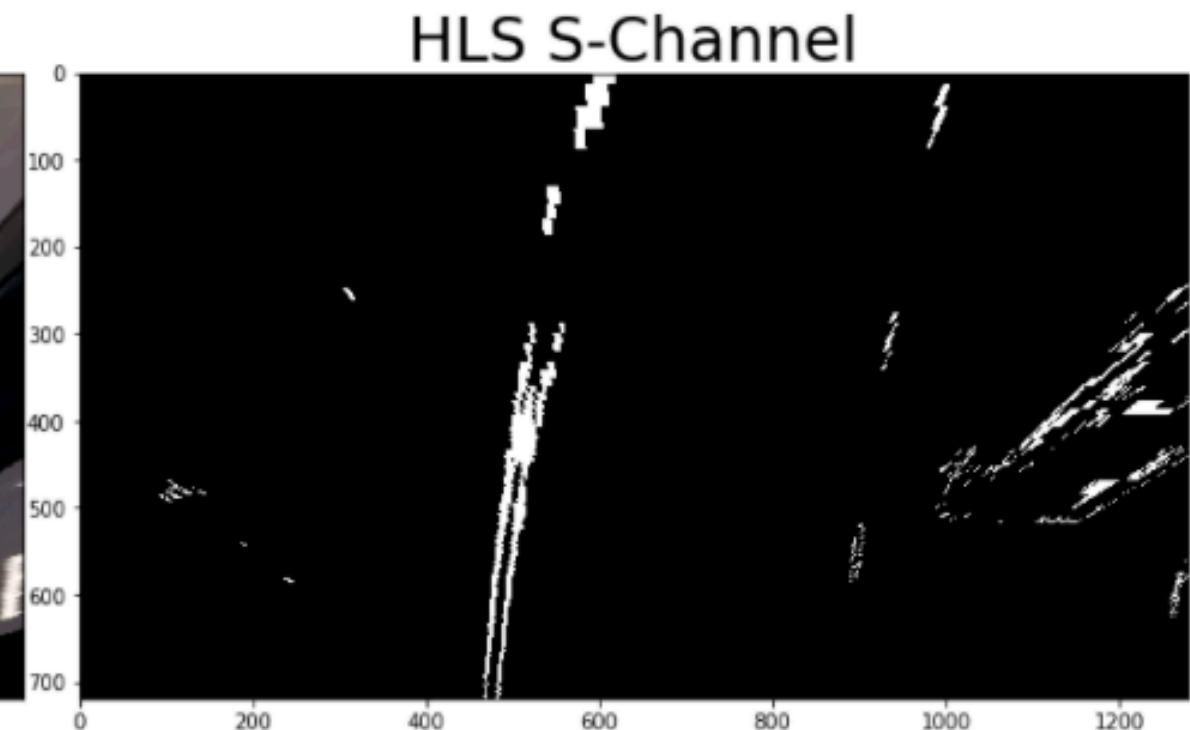
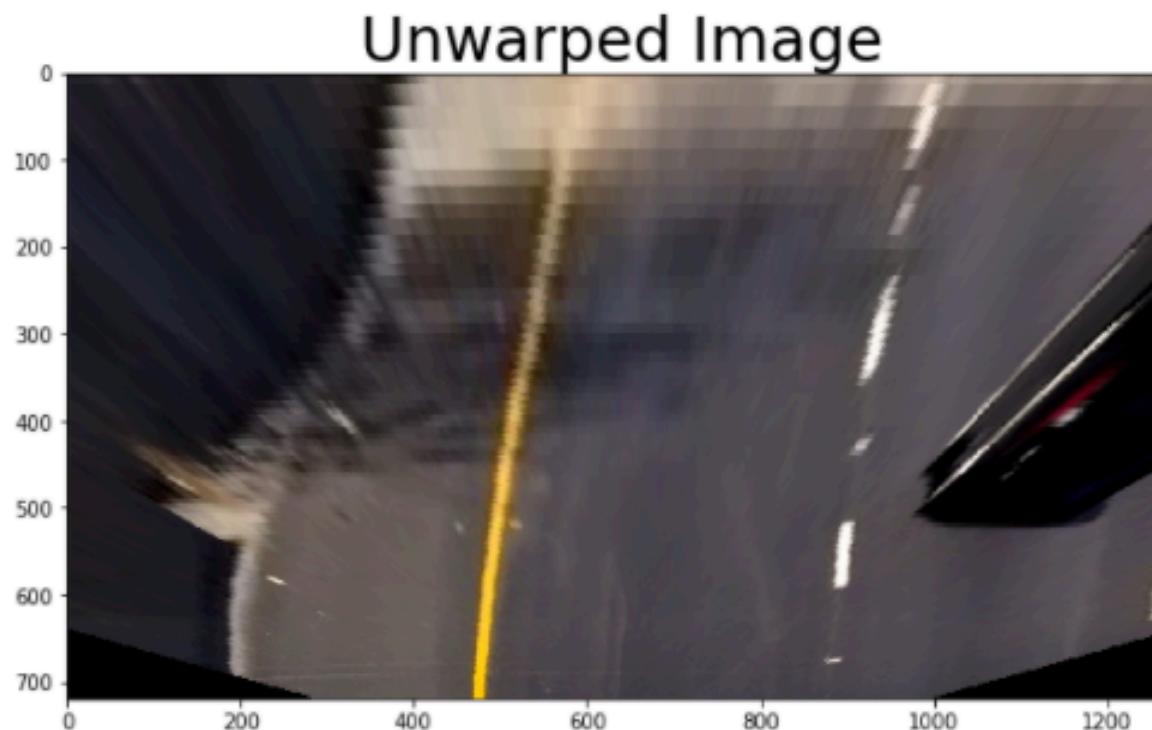
Final Binary



# Create a set of sliders to update the thresholds

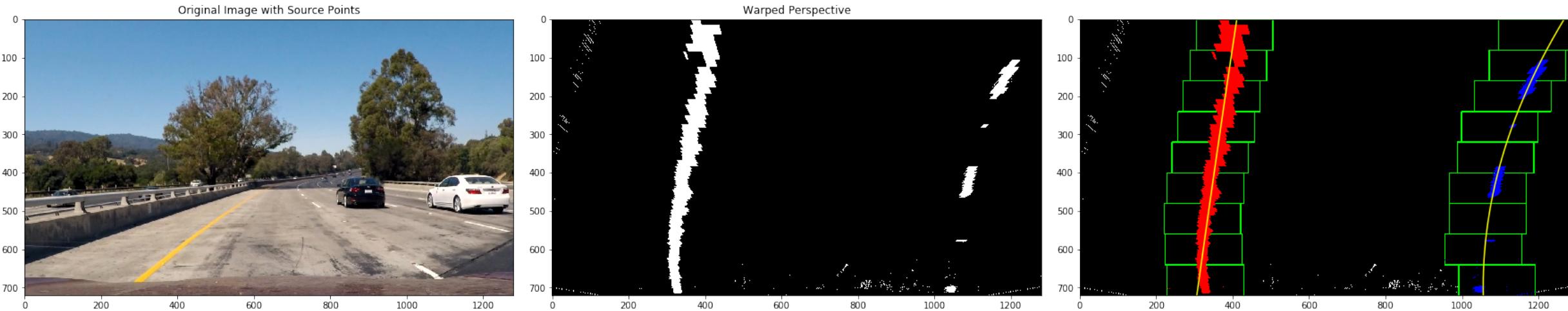
min\_thres  100  
h

max\_thre  200  
sh

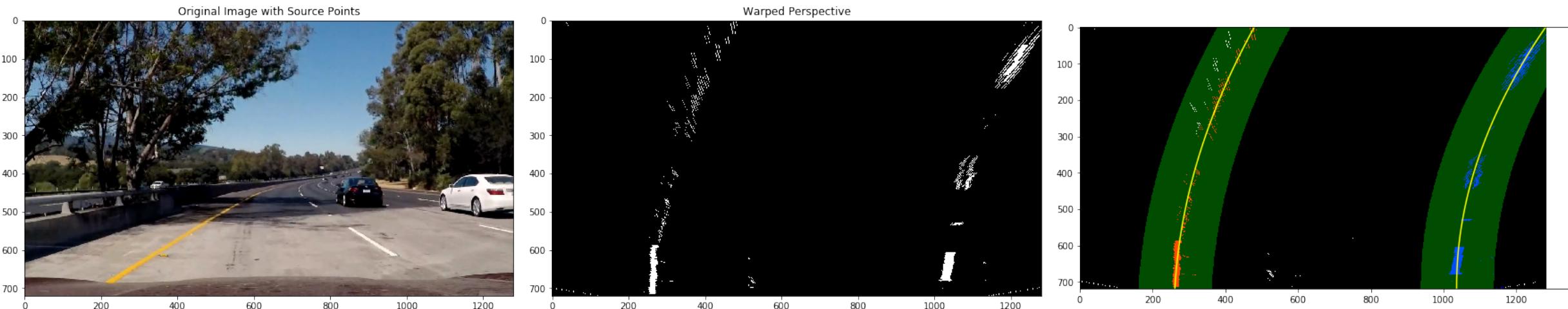


# Curve Fitting

# Method 1



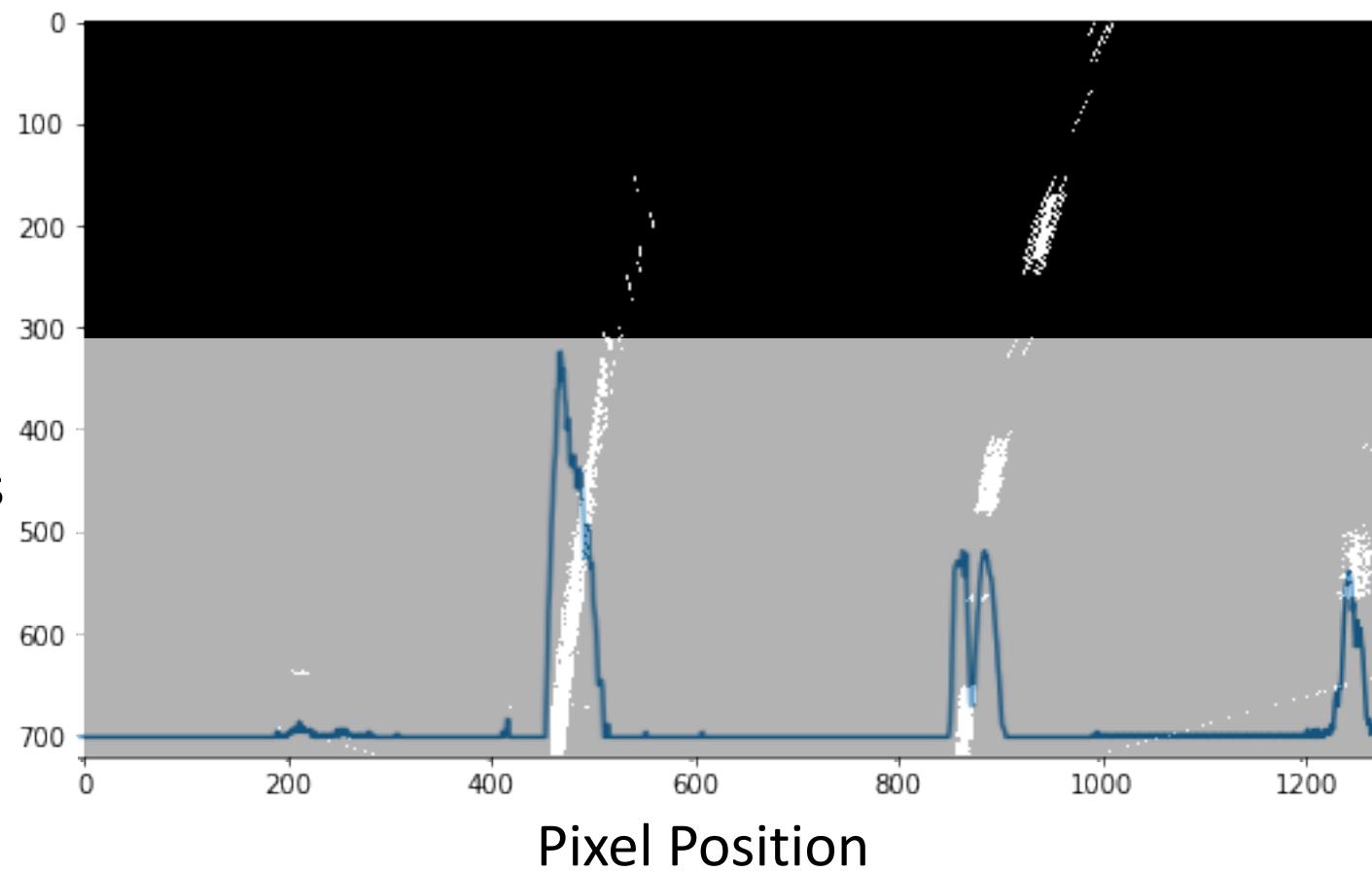
# Method 2



# Step 1: Find the start of the line

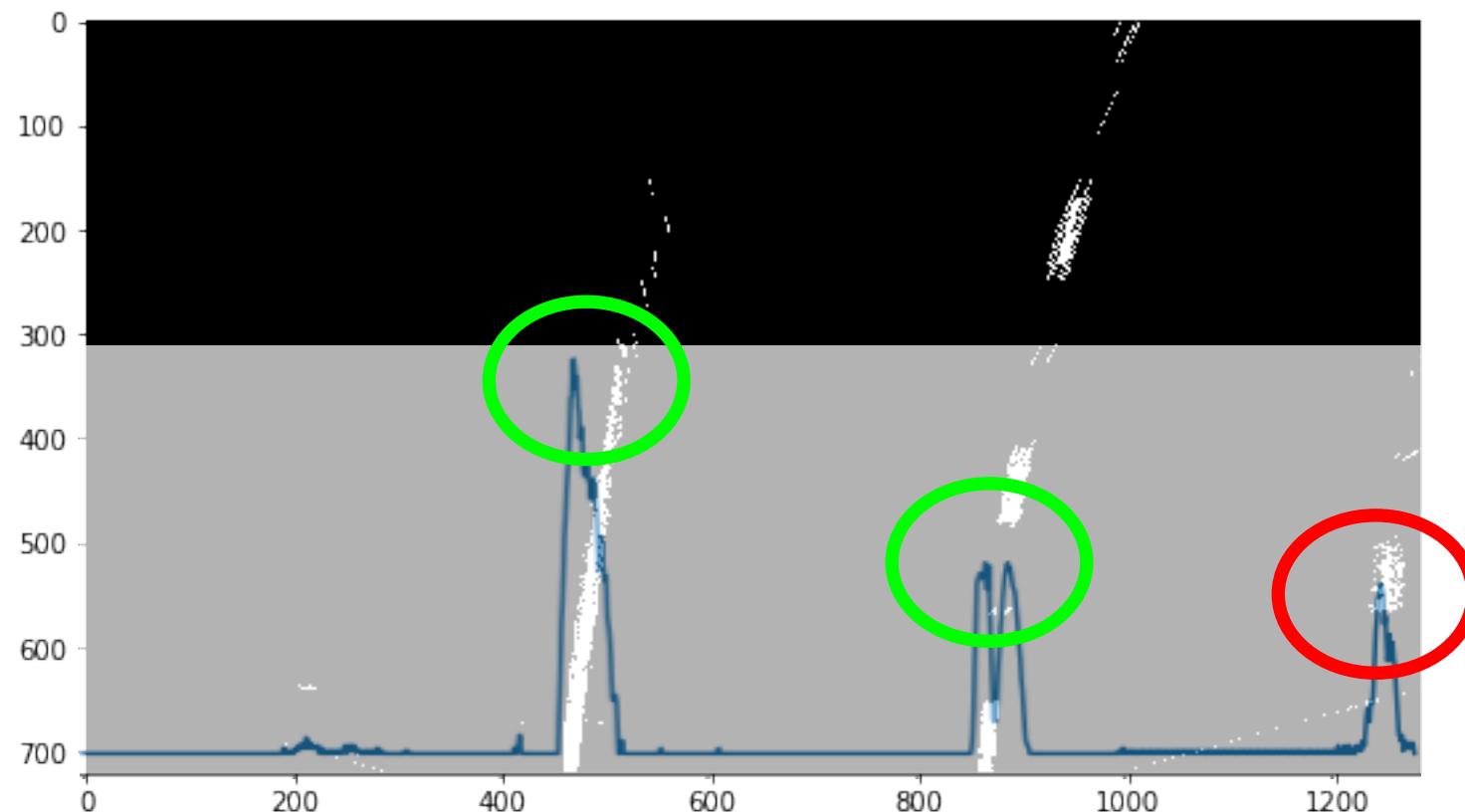
```
histogram = np.sum(img[img.shape[0]//2:,:,:], axis=0)
```

Sum white pixels  
at x location

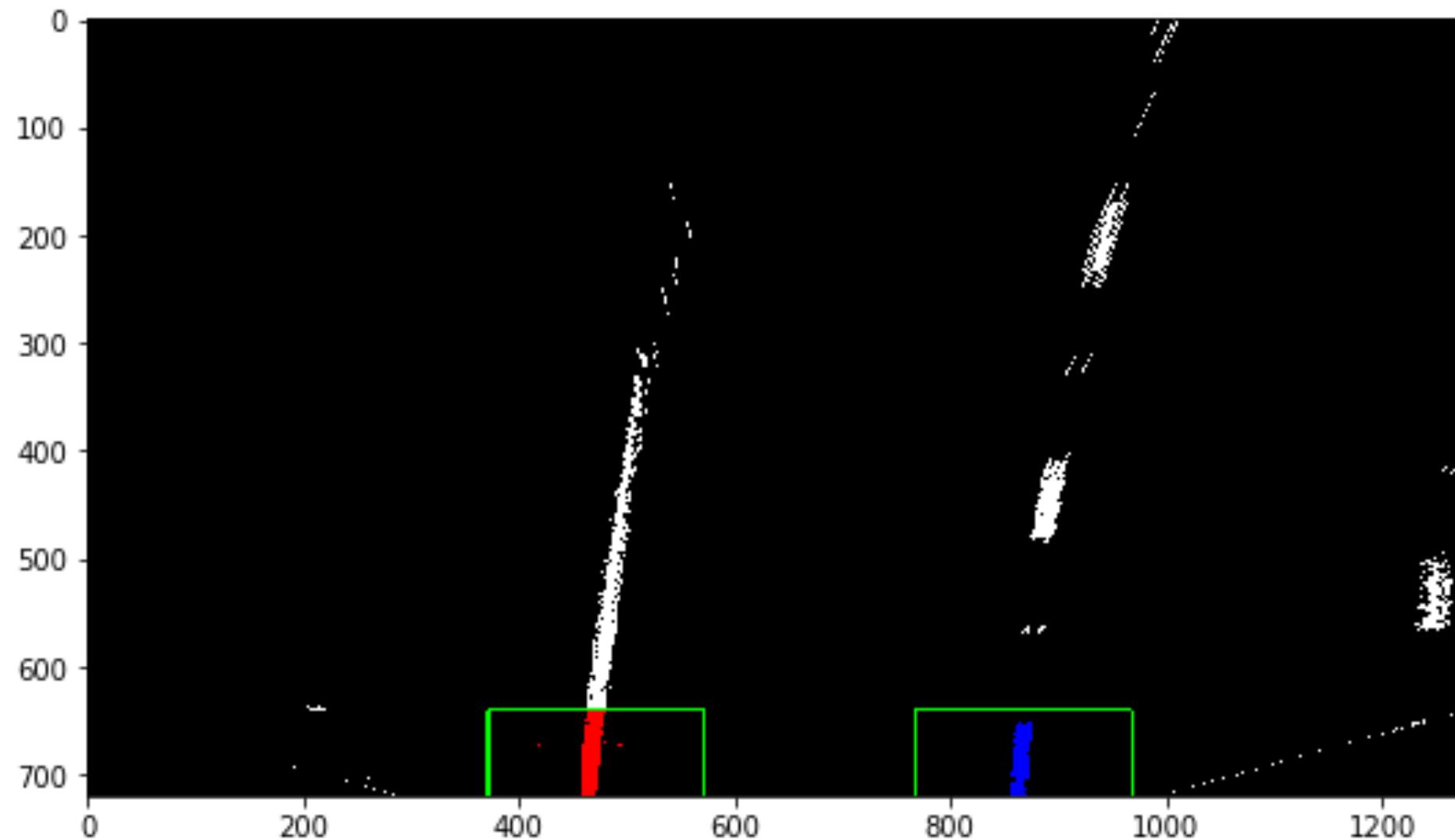


# Step 1: Find the start of the line

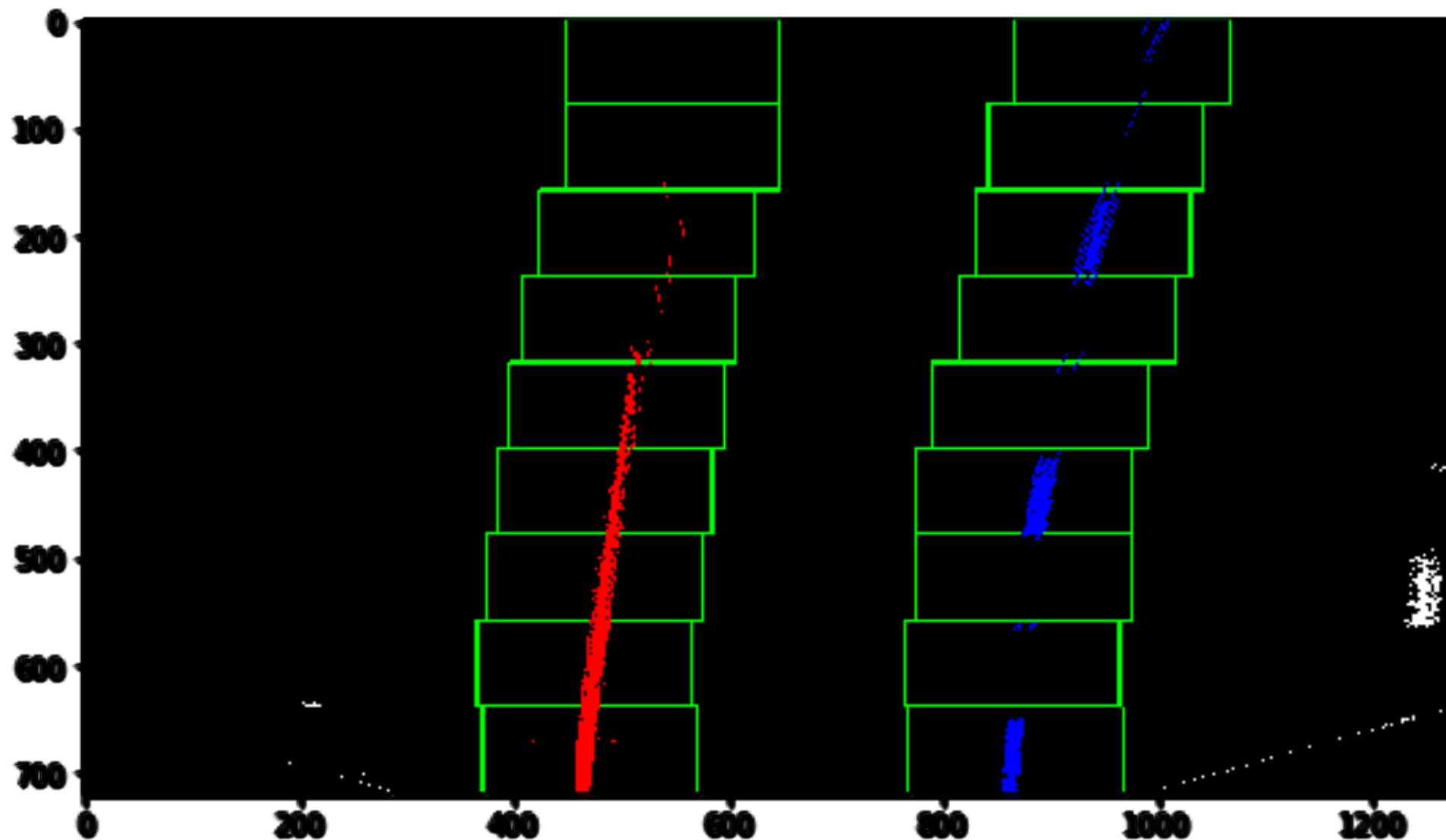
```
midpoint = np.int(histogram.shape[0]/2)  
leftx_base = np.argmax(histogram[:midpoint])  
rightx_base = np.argmax(histogram[midpoint:]) + midpoint
```



Step 2: Create windows around these maxima

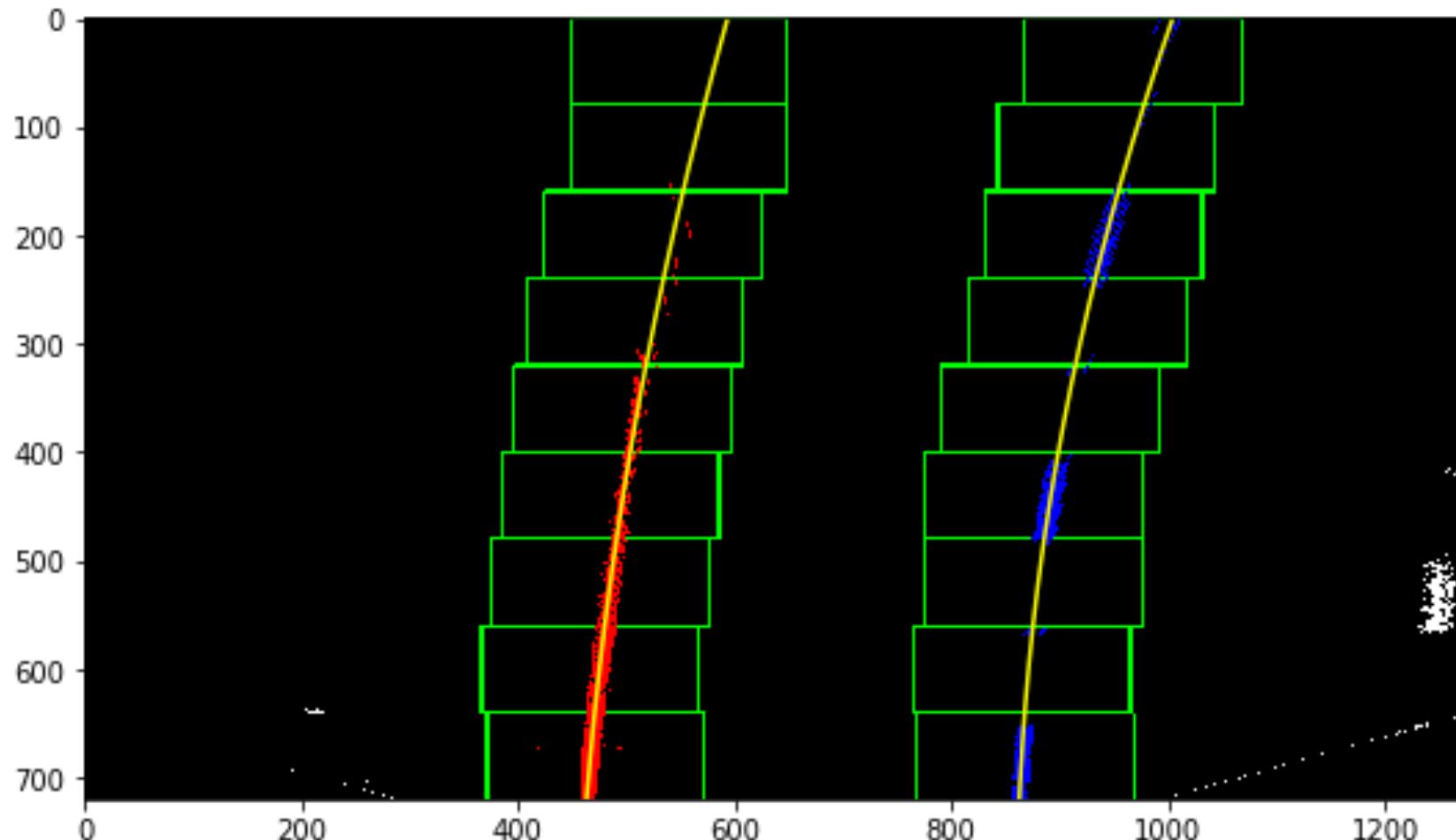


## Step 3: Stack and move the windows

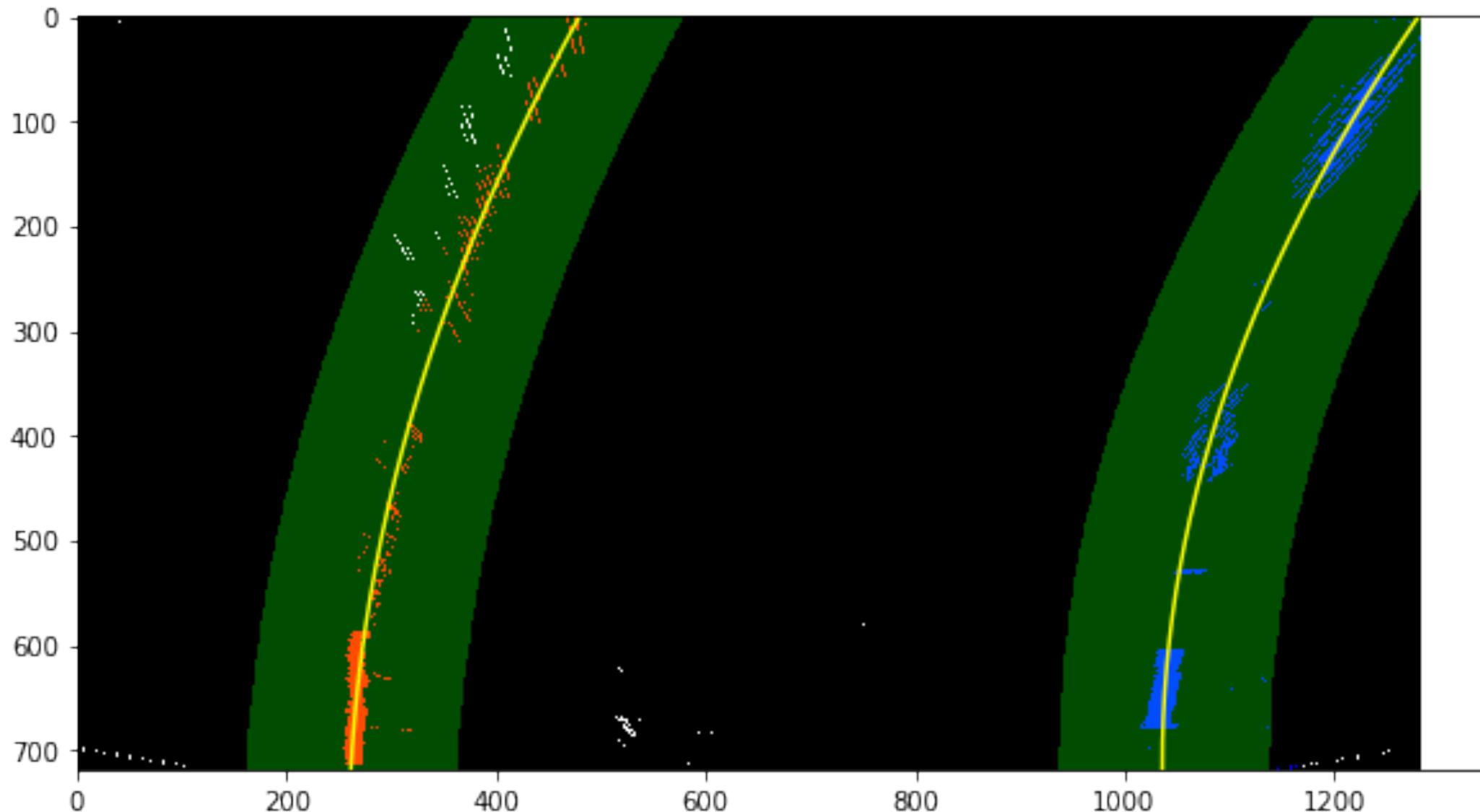


## Step 4: Polyfit all the points from windows

```
left_fit = np.polyfit(lefty, leftx, 2)  
right_fit = np.polyfit(righty, rightx, 2)
```

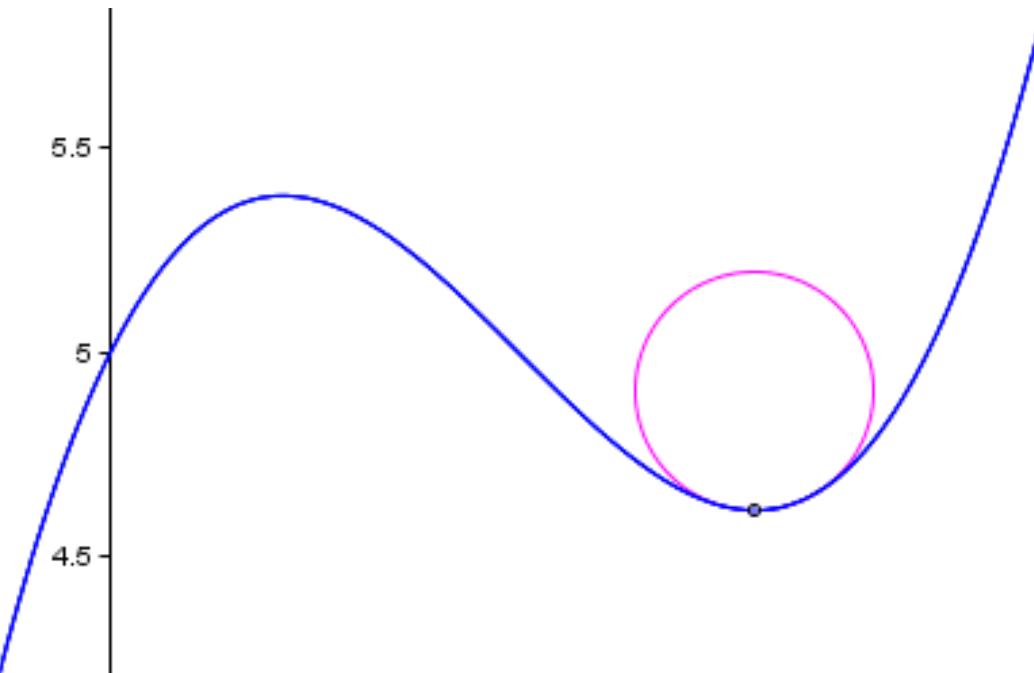


# Method 2 uses an existing fit



# Final Image

# Calculating radius of curvature



$$\text{Radius of curvature} = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

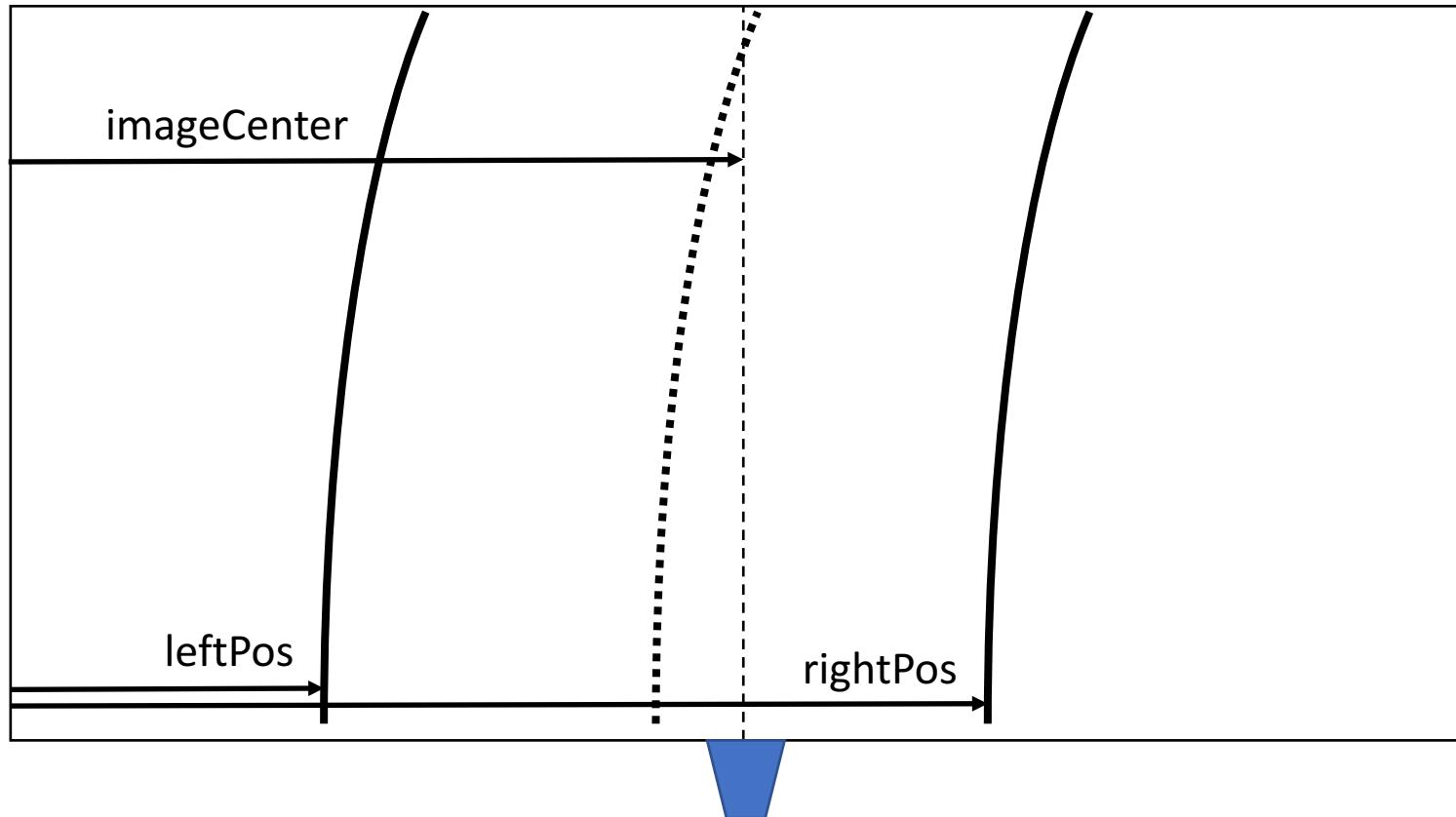
Polyfit equation:  $y = Ax^2 + Bx + C$   
fit = [A, B, C]

```
curve_rad = ((1 + (2*fit[0]*y_eval + fit[1])**2)**1.5) /  
np.absolute(2*fit[0])
```

# Distance to lane center

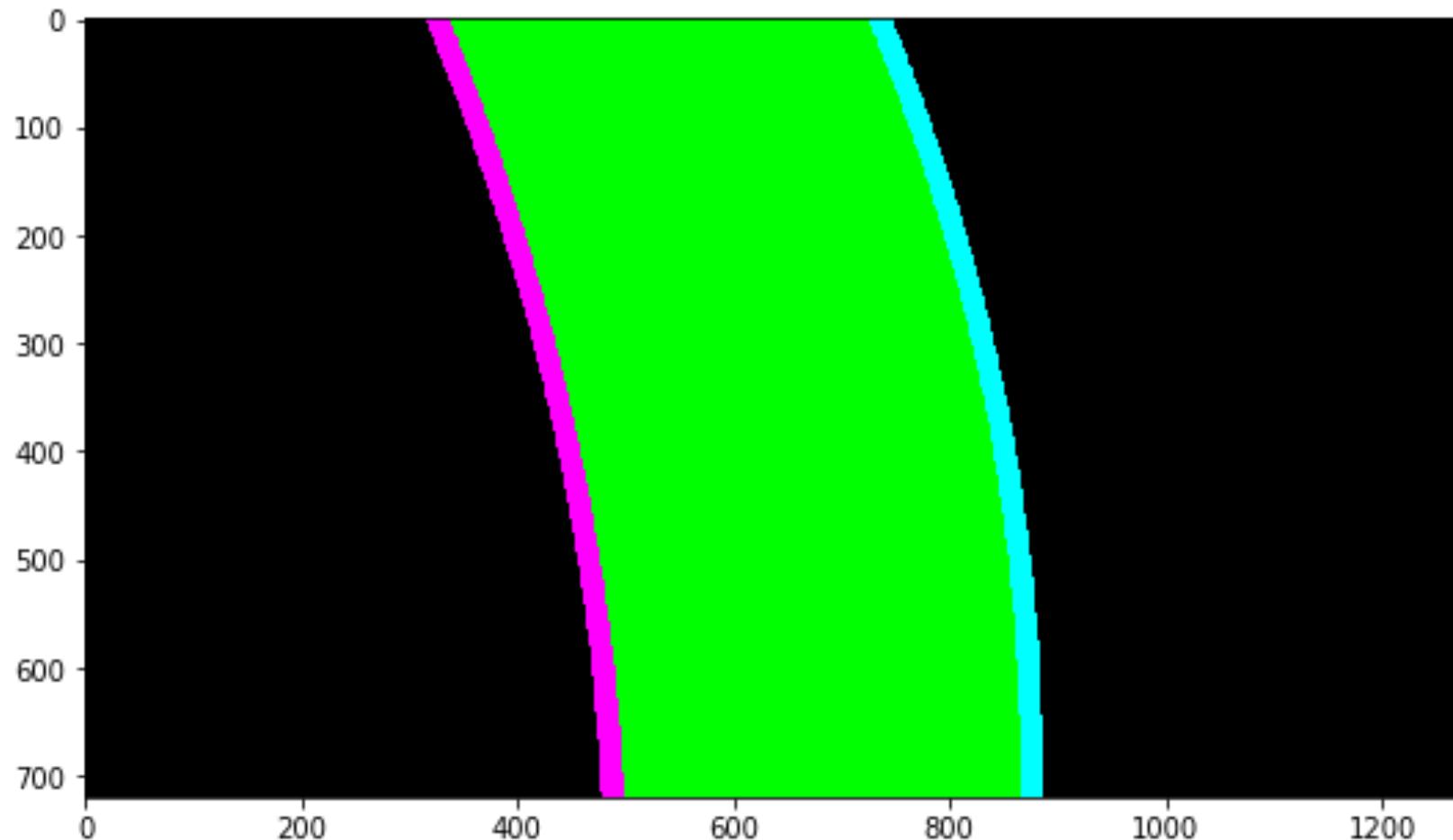
```
laneCenter = (rightPos - leftPos) / 2 + leftPos  
distanceToCenter = laneCenter - imageCenter  
If "+" then right of center
```

(0, 0)



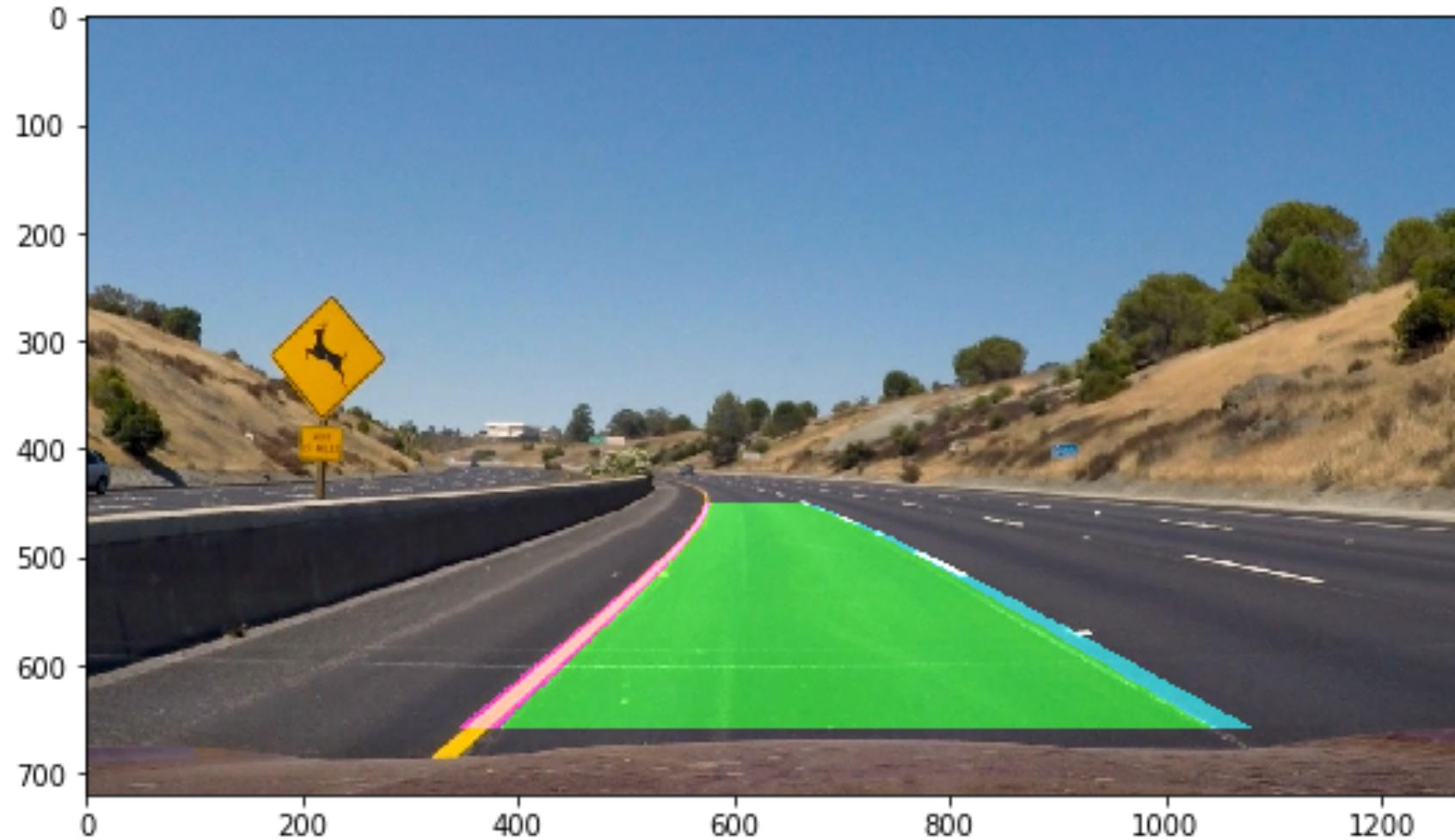
# Plotting the lines and the drive area

```
cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))  
cv2.polylines(color_warp, np.int32([pts_left]), ...)
```



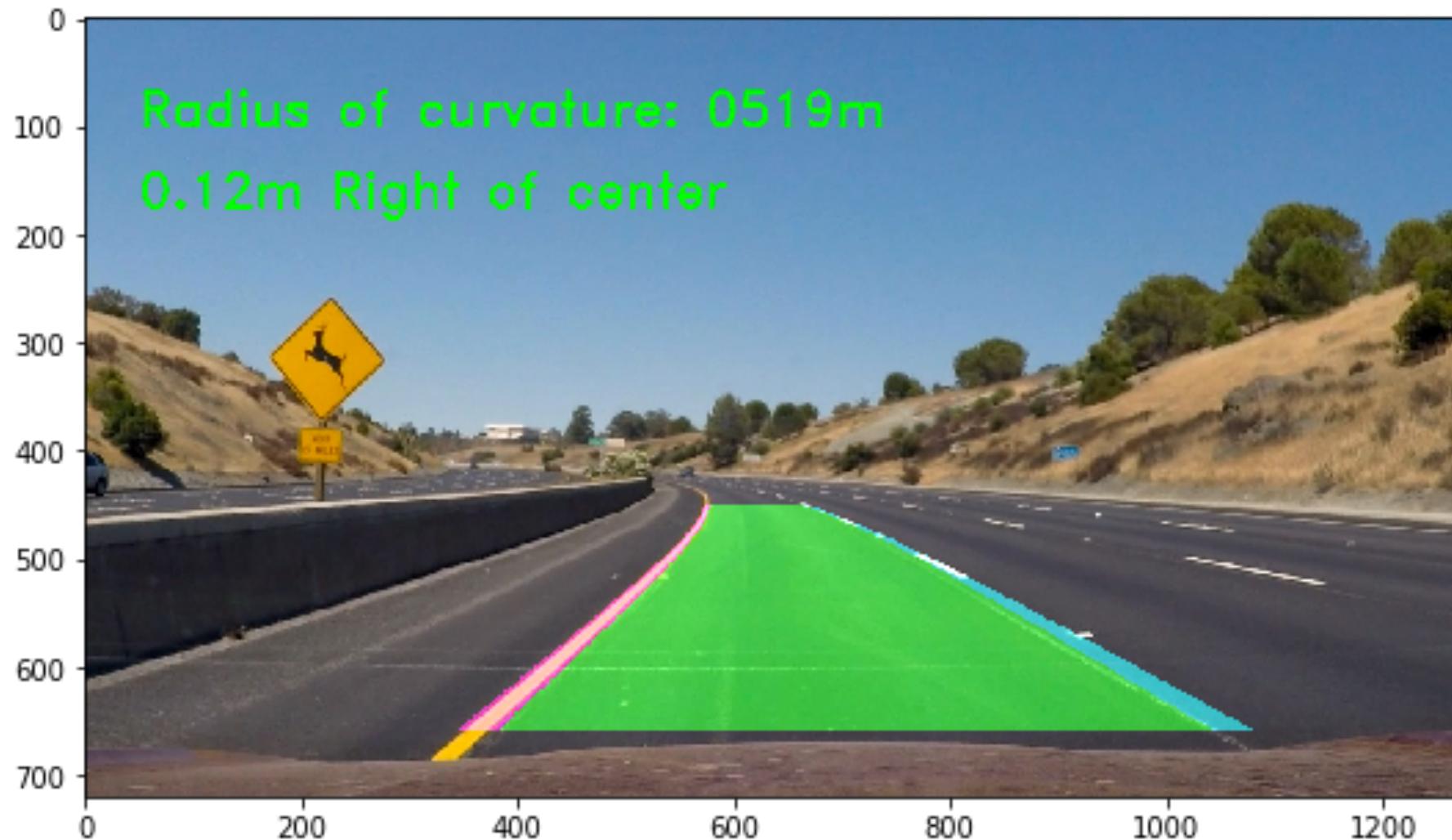
# Unwarp the image and add to the original

```
result = cv2.addWeighted(img, 1, newwarp, 0.5, 0)
```



# Add the radius and center distance

```
cv2.putText(img, text, (50,100), font, 1.5, (0,255, 0), 2, cv2.LINE_AA)
```



# Building more

- Image warping
- Dynamic color channels
- Speed
- Smoothing

# Questions?

[github.com/rkipp1210/pydata-berlin-2017](https://github.com/rkipp1210/pydata-berlin-2017)



# Sources

- <http://www.intmath.com/applications-differentiation/8-radius-curvature.php>
- Test images and videos come from the udacity open source self-driving car repo: <https://github.com/udacity/self-driving-car>
- <https://www.nytimes.com/interactive/2016/12/14/technology/how-self-driving-cars-work.html>
- Gifs from Giphy