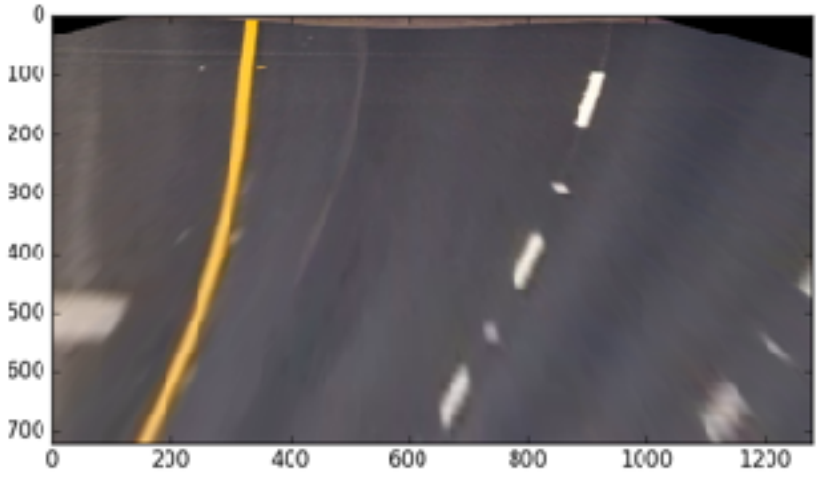
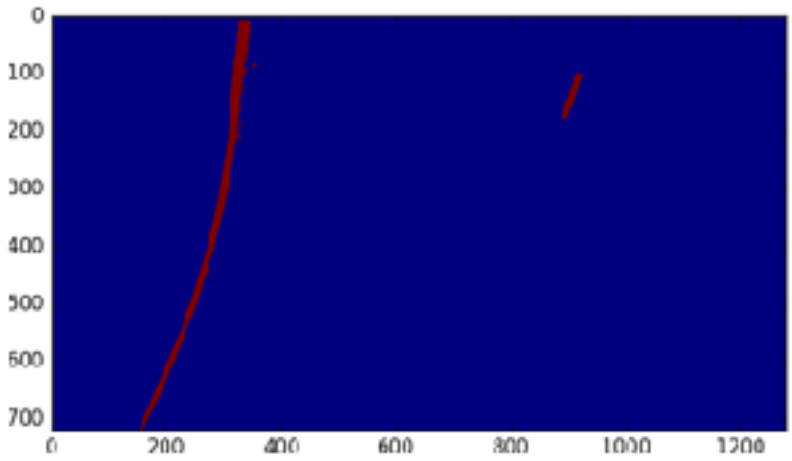
Image Processing Pipeline explanation

Step 1:
First the image is imported and then undistorted. This uses the calibration images supplied and applies the values mtx and dist. Below is the result.

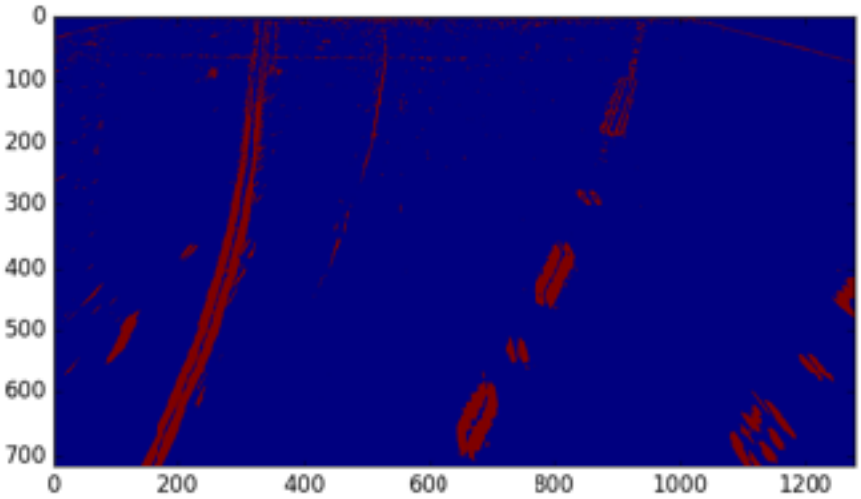
Step 2:
Then the image is warped into a “birds eye view.” This uses specific target points to warp the image accordingly. Below is the result.



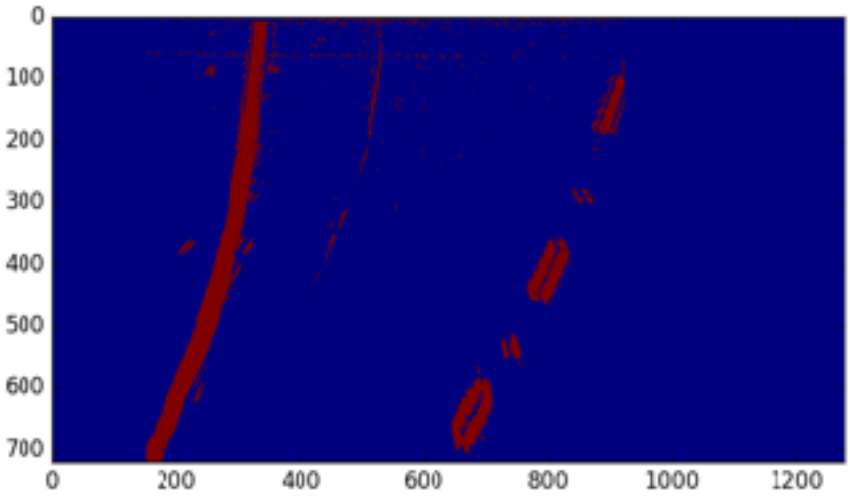
Step 3:
The image is then converted into HLS color formatting and the S channel is thresholded. The minimum value is 170 and the maximum is 255. Below is the result.



Step 4:
The image is then converted into grayscale formatting and a sobel is performed in the X direction. This is then thresholded as well. The minimum value for the first frame is 10 then for all other frames its is 20. The max value is 100. Below is the result.



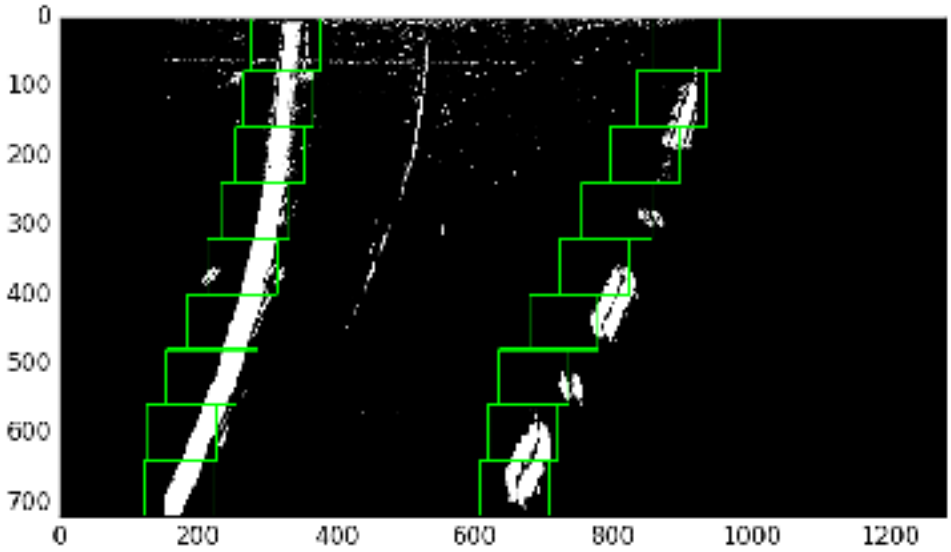
Step 5:
These two binary images are combined. This is the image utilized for finding the lanes. This image is also masked by using the “Region of interest” function to remove noise from the edges of the image. Below is the result.



Step 6:
Now to find the lanes. The first frame uses the sliding windows. All subsequent frames in the video use the past fit to determine where to search.

The concept with the sliding windows is simple. It uses a histogram along the bottom of the image to determine where the start of each lane is. Then it works up with 9 windows until the top of the image. The windows are re-centered if the minimum number of pixels is present. If the minimum is met then the window is centered about the mean X value of the pixels.

The past fit search concept is also straightforward. It uses the current running average polynomial coefficients as a basis for search. By using the line drawn on the last frame and a margin about that line it detects pixels and then generates the new polynomial. Below is an example of the sliding window finder.



Additional Details For Image Pipeline:
The above are the snapshots are for an individual frame. However this is not truly how the script functions. It uses a running average of the last 10 frames to plot the lane lines. This is done to dampen any changes or noise. Additionally if the frame being fed to the loop script has polynomial coefficients that have a percent error of greater than 100% of the current running average the data is discarded. The frame then gets the lane line based upon the running average.

Radius of Curvature Calculation:
The radius of curvature is calculation is based on the left lane line. This was done because the data is less noisy and much more constant. The radius of curvature calc first imports the current running average polynomial coefficients and then gets a first and second derivative. It then calculates the radius of curvature and uses the OpenCv function “put text” to place it on the image. Additionally if the R calculation is greater than 10% different from the current running average R then it is discarded. Otherwise it uses the last R that made it through the filter.

Distance From Center:
The distance from center calc simply measures the difference between the center of the lane and the center of the image. The center of the image has an X-value of 640. The center of the lane is half the lane width added to the left lane X-value. This is done inside the function “Draw_lines”. Again OpenCV “put text.”

Discussion:
I found that the biggest difficulty was simply noise. There is so much of it coming into the polynomial fit and you have to eliminate it before and after. I used a couple tools to eliminate the noise. First effective tuning of parameters is always the first line of defense. Second the

running average really helps, especially when it is more environmental than actual noise in the data (i.e. the car hitting a bump). Third I used a filter that relies on the percent difference. The new frames polynomial coefficients are compared to those of the running average. If they are greater than the set percent difference the coefficients are discounted and the lane is plotted on that frame of video with the un-changed running average.

As far as potential failure I'd say adverse lighting conditions are definitely the most likely to effect it. Additionally I would say un-familiar objects in the lane could have an effect (cars, trucks etc.). As far as environmental effects such as snow and rain I think the lighting associated with that sort of weather would first have an effect but since the snow has a similar color to the white lane line it would definitely cause an issue. The distance from center calculation could be thrown off if the camera was not mounted on the center of the dash board, due to the assumption that the center of the image is the center of the car.

All in all I feel really good about this project and feel it is a very robust pipeline that is able to track the line and make good estimates of radius and distance from center.