Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

Histogram of Oriented Gradients (HOG)

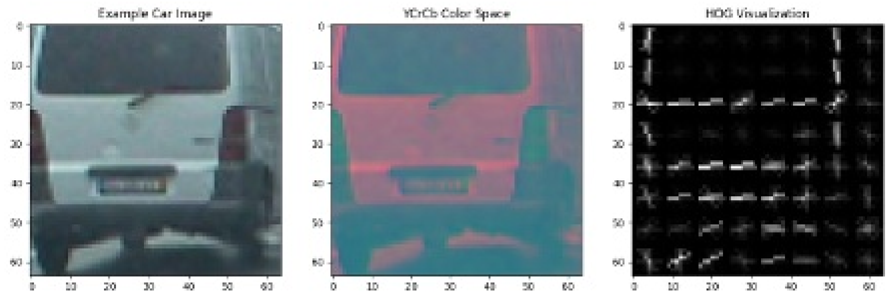
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

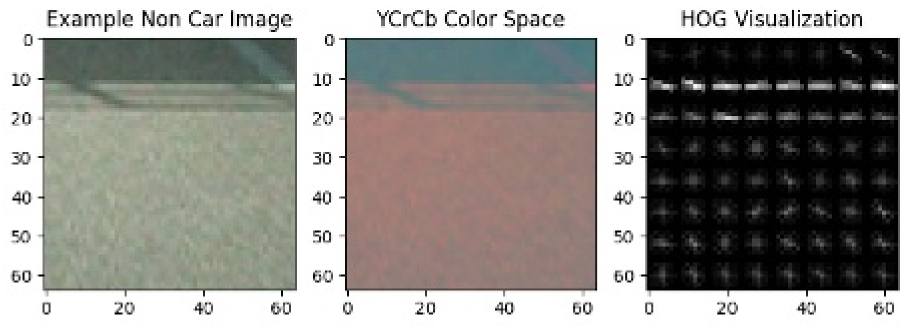
The code for this step is contained in lines 164 to 215 in function_lib.py.

I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=24`, `pixels_per_cell=(8, 8)` and `cells_per_block=(1, 1)`:





2. Explain how you settled on your final choice of HOG parameters.

Firstly I've tried change color space parameters and found out that 'YCrCb' and 'YUV' color space could achieve best results. I've tried different settings to test on samples of 500 car and non-car-images each. I found out that more orientation could be improve the results up to ca. 30. I've chosen 24 for further processes. I've also experimented with other parameters `pixels_per_cell` and `cells_per_block` as well and found out that the influence of these parameter are marginal. I've chosen `pixels_per_cell=(8, 8)` and `cells_per_block=(1, 1)` for further processing.

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

The code for this step is contained in lines 218 to 306 in function_lib.py.

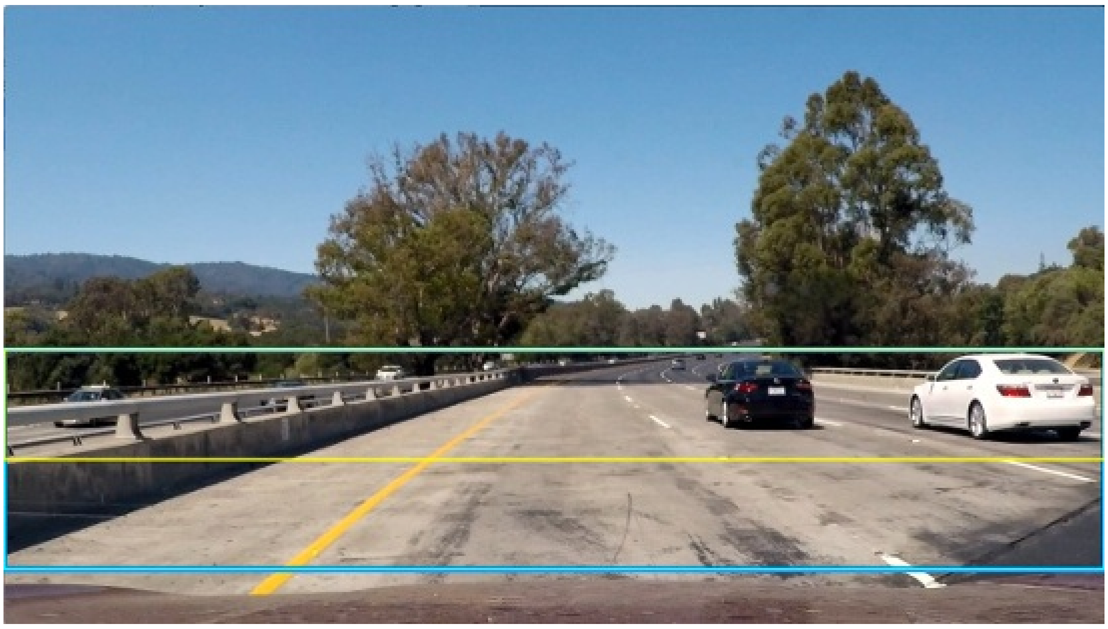
I trained a linear SVM using only HOG-features. I've tried other SVM algorithms such as 'rbf' or 'poly', they work also and could achieve slightly better recoginition results but are far more time-consuming. Also coding LinearSVC() is far more time-efficient than SVC(kernel='linear').

Firstly I've converted the image in YCrCb color space and extract HOG-features of all three channels. After all images are processed this way, I normalized the data using scikit StandardScaler over all data. And finally, I trained the normalized data using LinearSVC.

Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

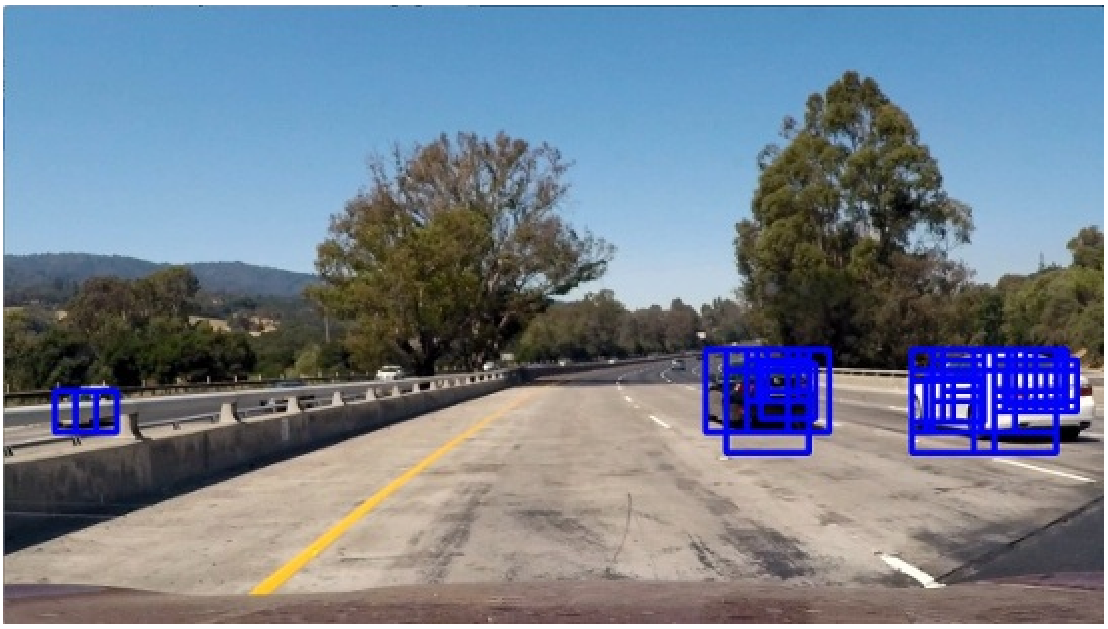
I've decided to search in three different scales: 96x96 in blue box, 64x64 in cyan box (same as blue box) and 48x48 in yellow box (only the upper half of blue box) :

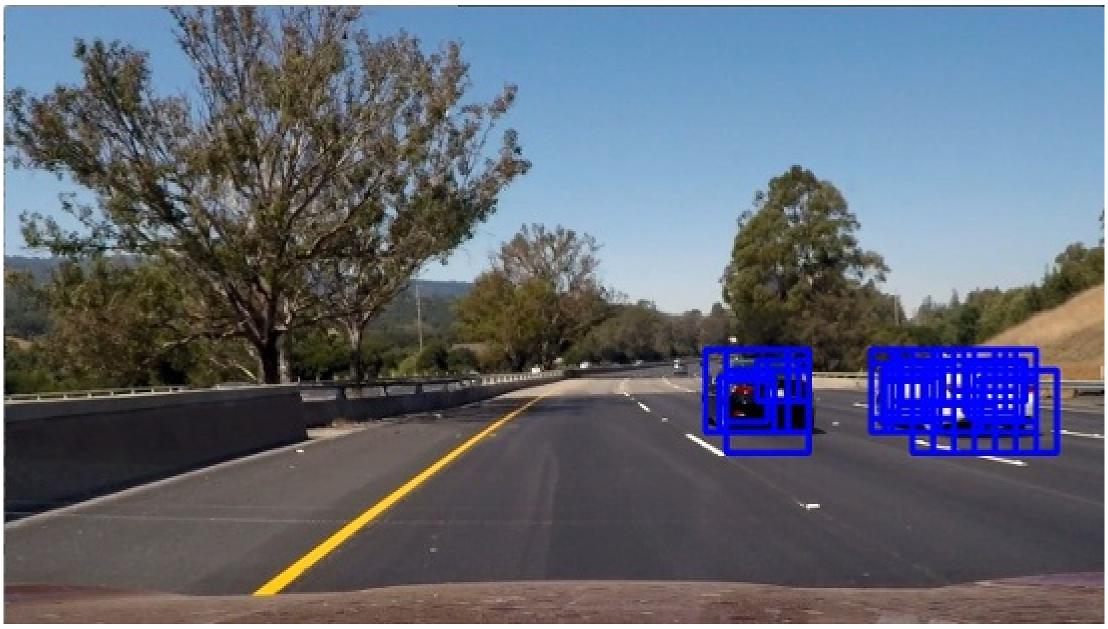


I've searched with smaller scale only in yellow box because the cars are far away and smaller in this area. Nearer cars can only appear large.

2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on three scales using YCrCb 3-channel HOG features only, which provided a nice result. Here are some example images:





Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

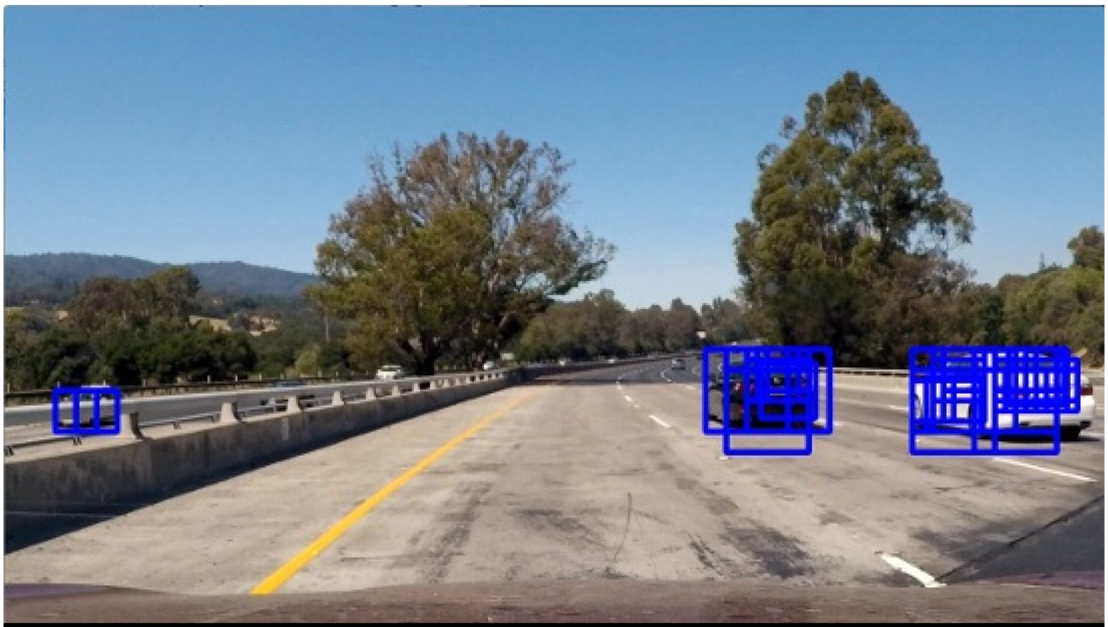
Here's a link to my video result

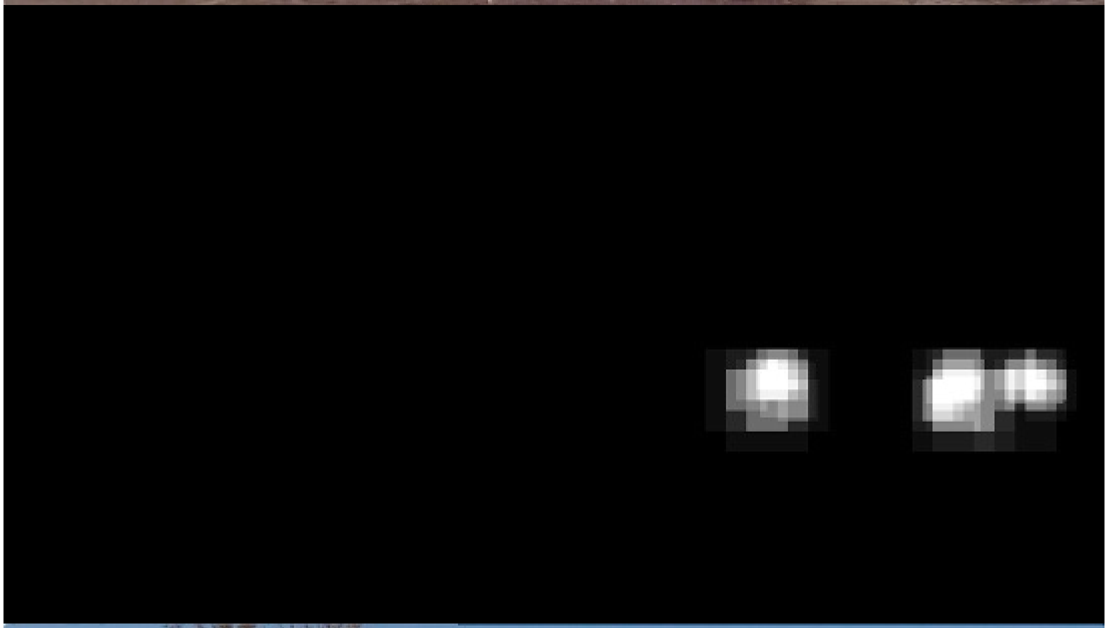
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

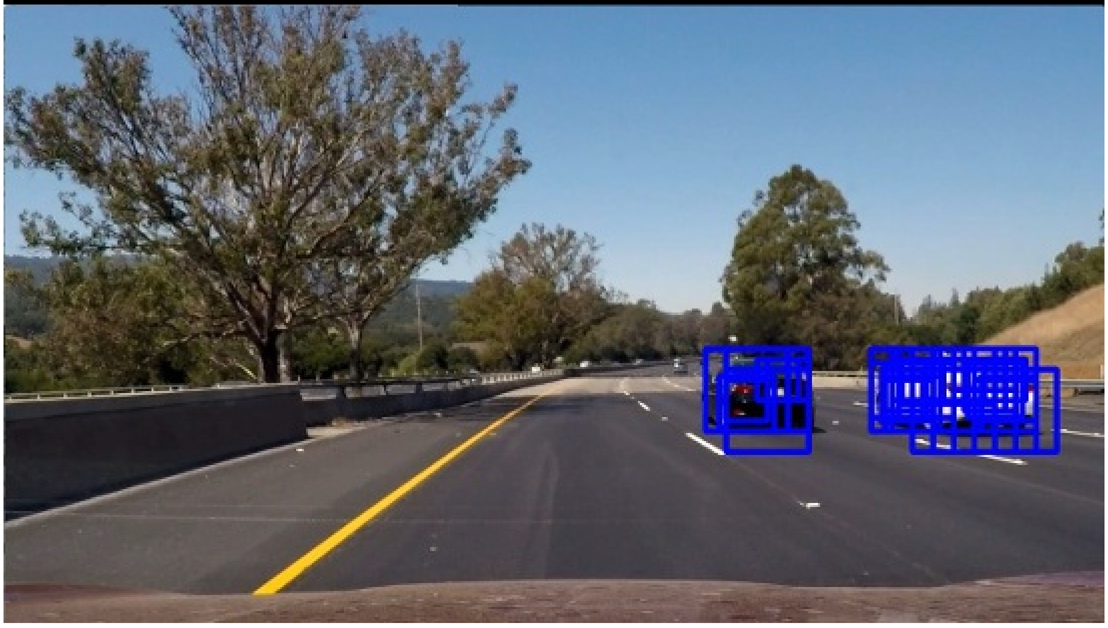
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

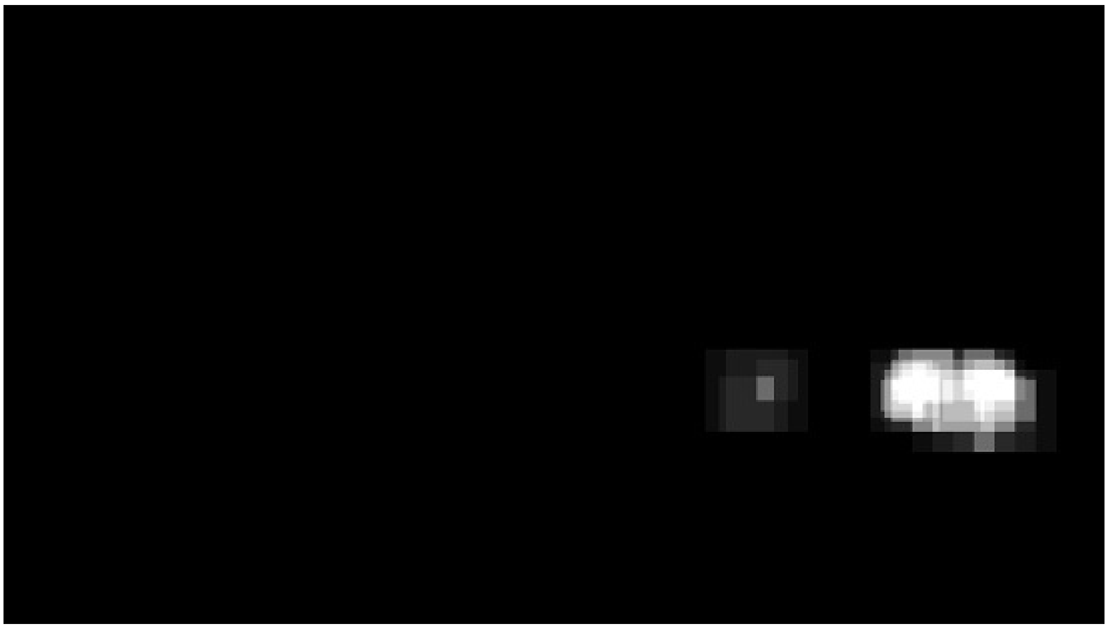
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

Here are 2 frames and their corresponding thresholded heatmaps:

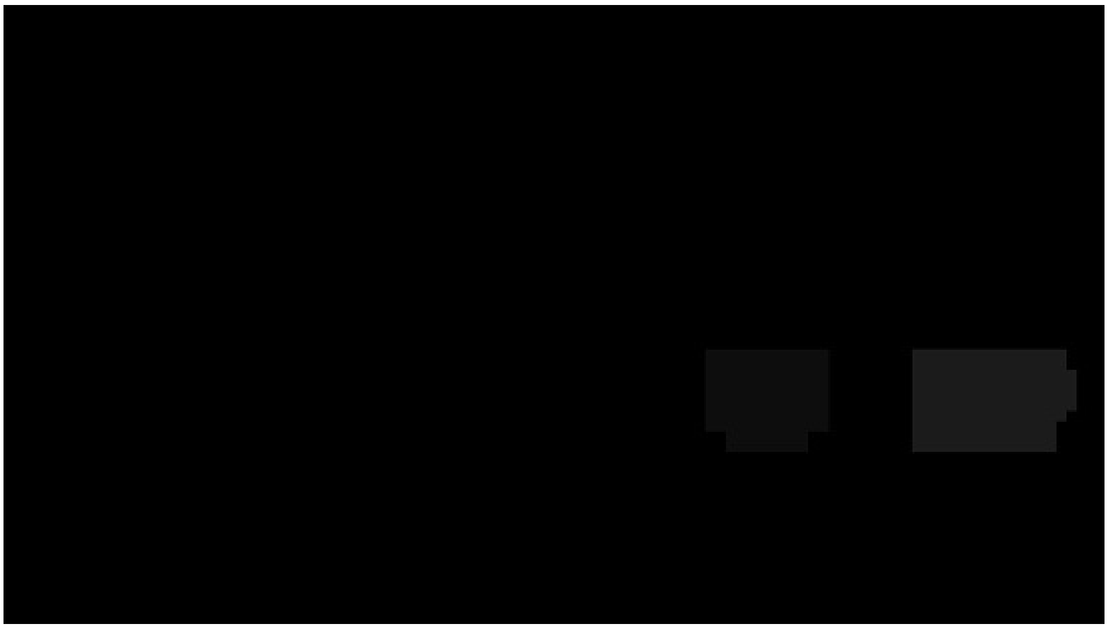




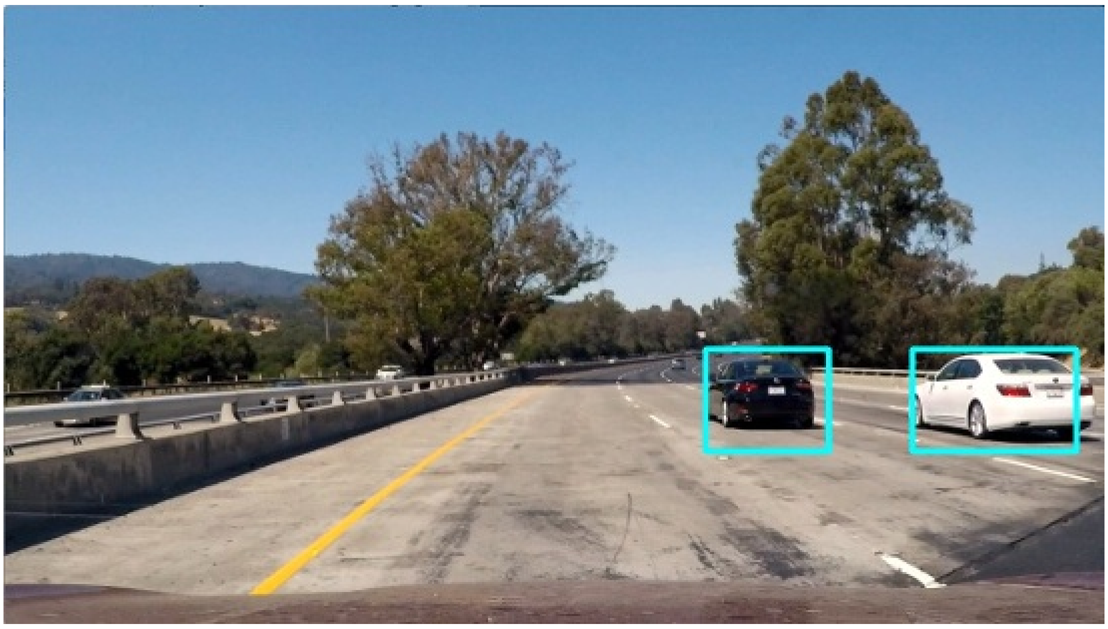


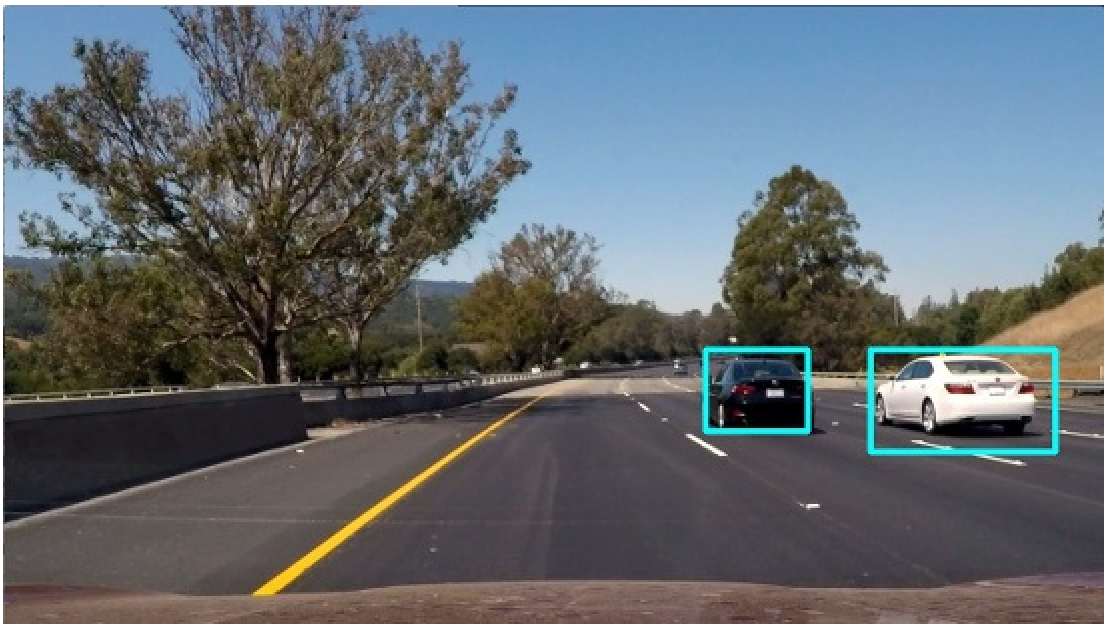


Here a the outputs of `scipy.ndimage.measurements.label()` on the integrated heatmap from 2 frames above:



Here the resulting bounding boxes are drawn onto the 2 frames above:





Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

The pipeline I've created does not contain any classes which iterate throughout the video processing that could provide information of frames before to identify false target. It only filters out false recognition via high threshold value. This could results in failed recognitions in one or another frame. Also I will combine the vehicle recognition with lane detection since this will narrow down the search areas. Currently the video processing is very time consuming and not real time.