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.

You're reading it!

Histogram of Oriented Gradients (HOG)

---

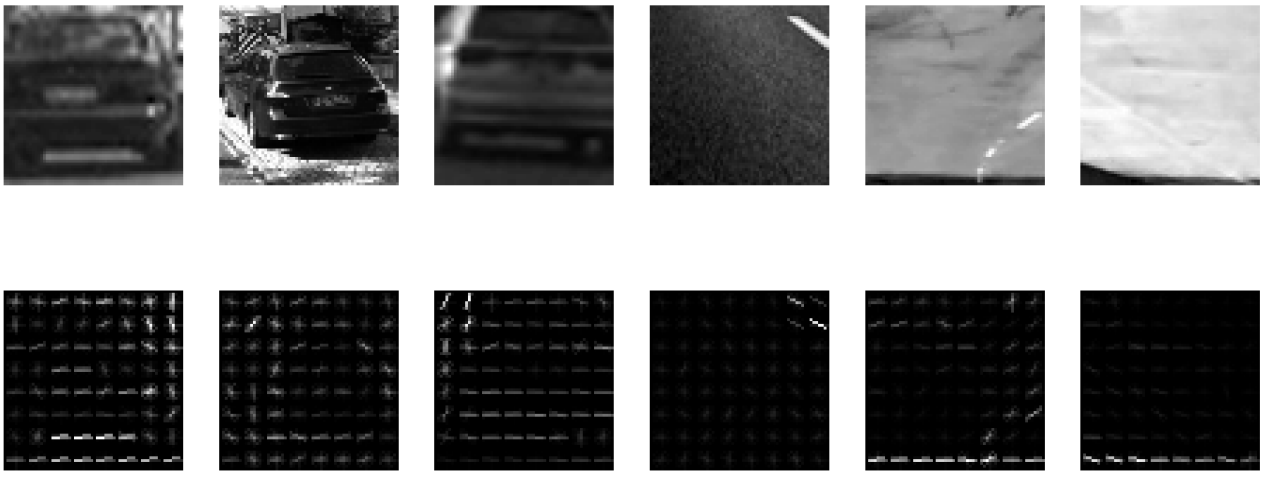
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in lines 95 through 127 of the file called pipeline.py. The lines 95 through 114 are only there for demonstrative and feature tuning purpose when lines 116 through 127 were used for training and prediction.

I started by reading in all the `vehicle` and `non-vehicle` images. I also extended the number of non-vehicles by flipping the images along x and y axes. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

I then explored different colour 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 `RGB` colour space only `G` channel and HOG parameters of `orientations=32`, `pixels_per_cell=(8,8)` and `cells_per_block=(2,2)`:



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

I tried various combinations of parameters and ended up with this configuration as the most optimal in the sense object detections:

```
orient = 32  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
```

In respect to the length of the feature vector and of therefore processing time 16 orientations and 16 pixels per cell work reasonably good but in some particular areas of the project video I couldn't get a satisfactory number of detection hits comparing to the fault positives. We don't have much of colour here (if any), so the HOG vector carries the most of the object detection.

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

I trained a linear SVM using the following set of features. The features were scaled using `StandardScaler()`, the training image data randomised and split into 80% of training set and 20% of test set.

```
# parameters of feature extraction
color_space = 'RGB' # Can be GRAY, RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 32  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 1 # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 32    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
```

The code can be found in the file pipeline.py lines 129 through 157.

Sliding Window Search

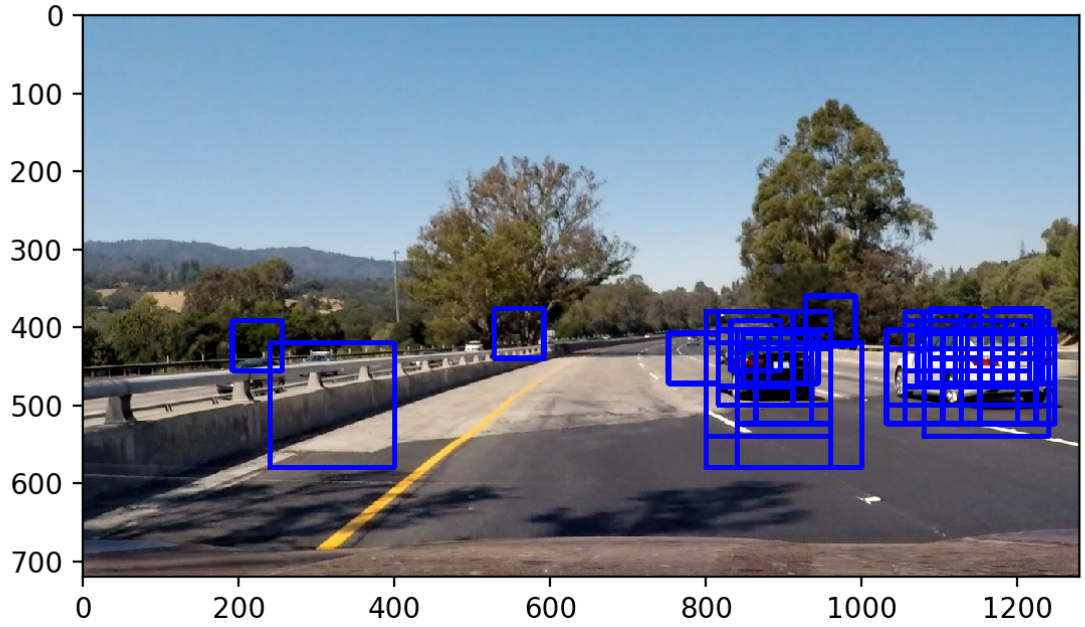
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

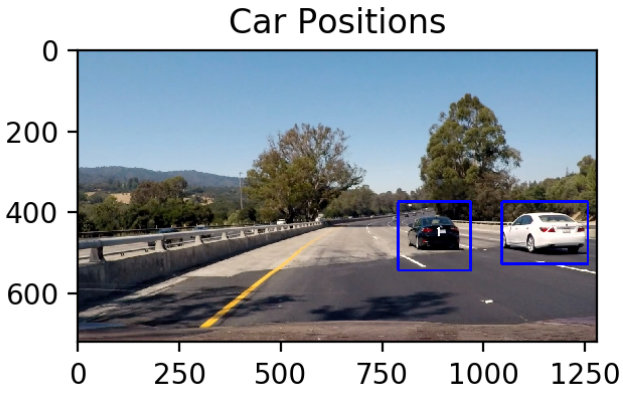
For this part I adapted the function from the Udacity class called `find_cars()` that also includes HOG sub-sampling. I extended the function to perform the sliding window search on multiple scales and in different regions. I selected the scales and ROIs to mythic the areas where cars can be seen and of which dimensions they can be expected there. The final set of parameters includes three regions of interest with three different scales respectively (line 162 in pipeline.py):

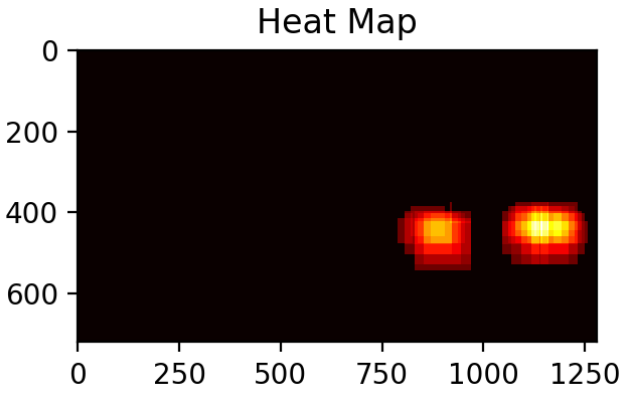
```
ystarts = [ 360, 380, 380]
ystops =  [ 500, 550, 670]
scales =  [ 1.0, 1.5, 2.5]
```

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

Ultimately I searched on three scales using YCrCb one-channel (Y) HOG features plus spatially binned colour and histograms of colour in the feature vector, 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](./project_video_processed.mp4)

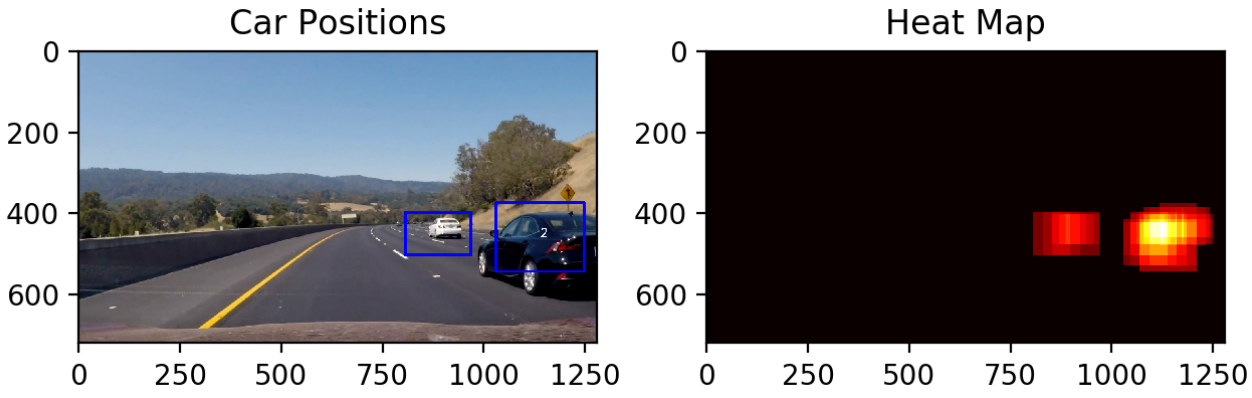
And here is the result with only one channel B&W image:

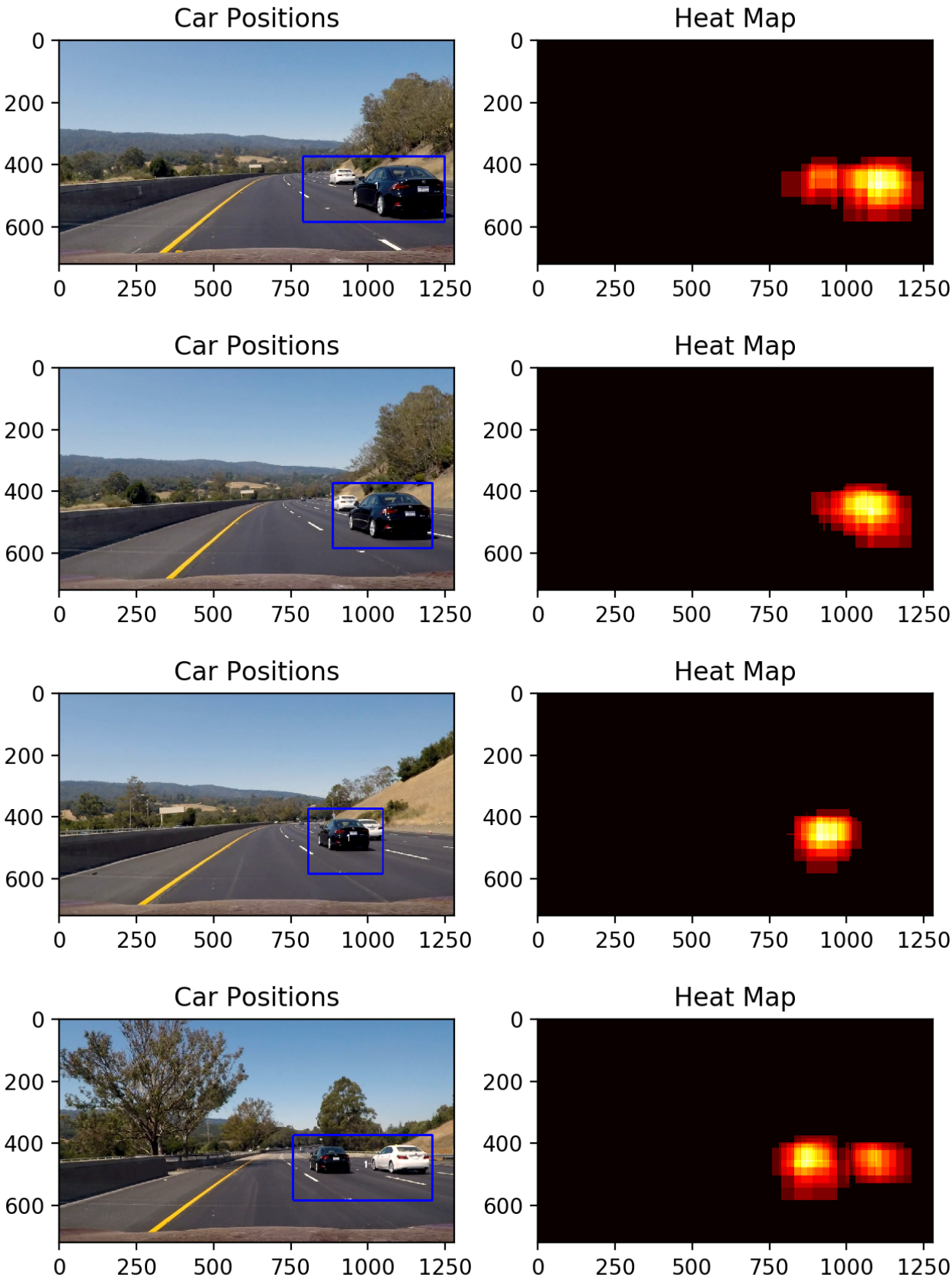
---

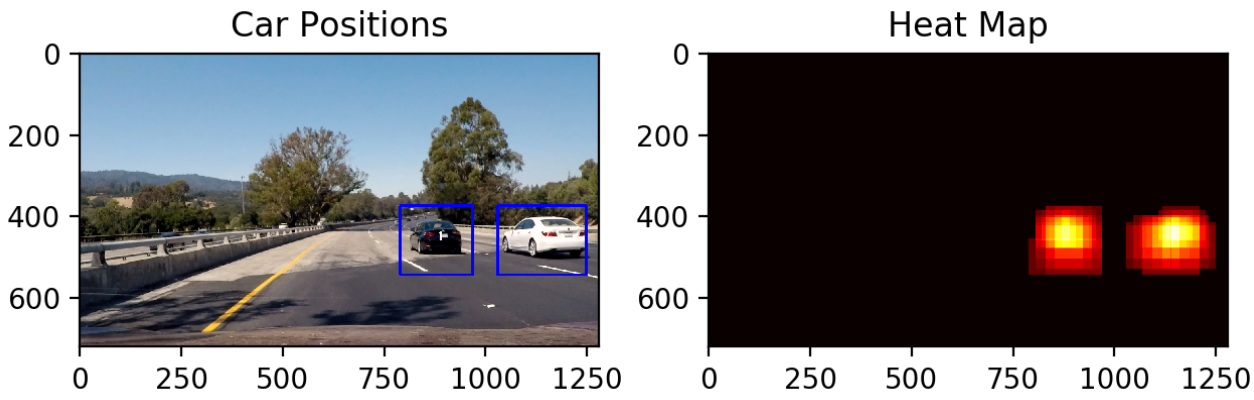
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap accumulated and averaged over 25 frames of the video. The accumulated heatmap was then thresholded 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. Additionally I implemented a filtering on the minimum size of the box inside the `draw_labeled_bboxes()` function.I constructed bounding boxes to cover the area of each blob detected.

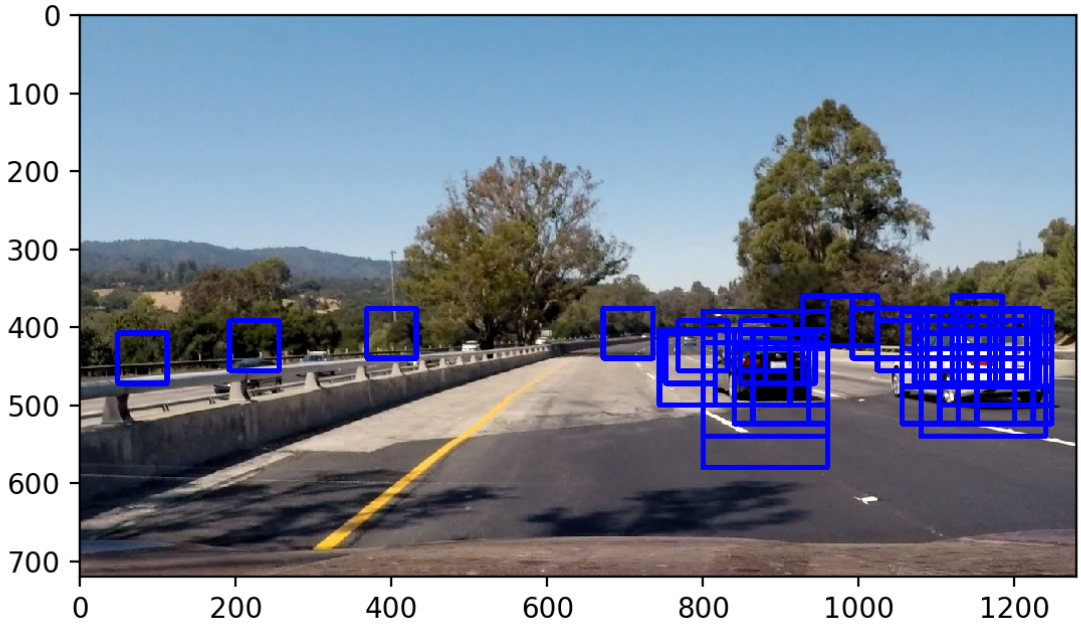
Here's an example result showing the heatmap from a series of frames of video (in this case not accumulated over multiple frames and then averaged but only thresholded), the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on each frame of the video:




Car Positions
Heat Map
Car Positions
Heat Map
Car Positions
Heat Map
Car Positions
Heat Map
0
200
400
600
0
250
500
750
1000
1250

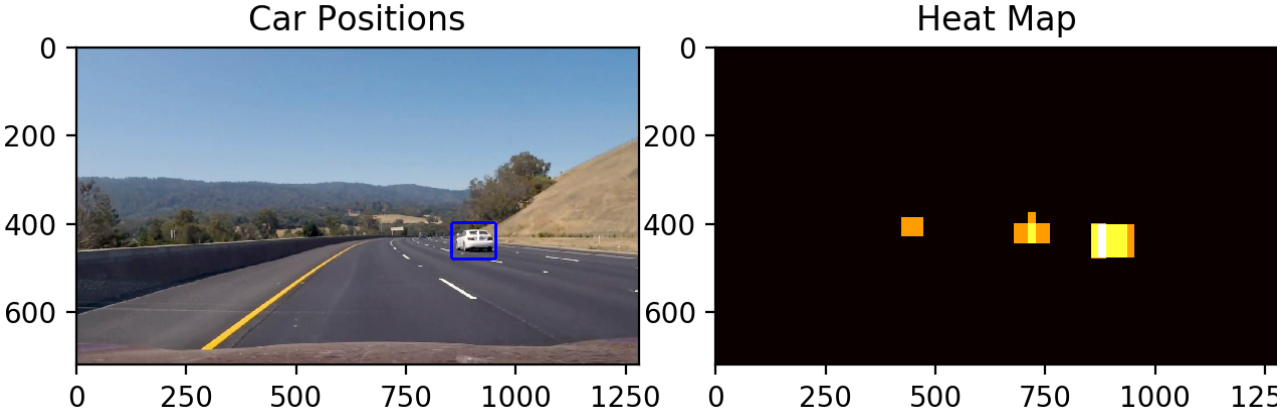


To give a better understanding what is going and how the filtering is working, here are our original hits (likely to be cars areas) from the last frame:



As you can see the area on the left because of being overlapped only once (one hit) was discarded by applying the threshold of 2 hits on the heatmap.

Bellow is another example where the minimum size restriction resulted in discarding some blobs even after applying the threshold:



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?

I tried exploring different colourspaces to add some more features and therefore more robustness to the algorithm, but I haven't found a significant improvement if the HOG features are done at least on one black-and-wide channel (it could be one of RGB channels or Y channel of YCrCb). Still having tow black-and-white cars on a black-and-white asphalt in some areas created a detection problem. Essentially if I reduce my filtering I would be able to detect the cars there but in other areas I would have then too many false positives. I wanted to avoid to do a sophisticated filtering and tracking of those specific cars which of course could be done. Is there a colourspace or a magic pill that works the best for such cases? I was thinking of adding more white cars into the dataset by inverting the colours but it didn't lead to the expected results.