

[Manage topics](#)

 24 commits

 1 branch

 0 releases

 1 contributor

Branch: master ▾














New pull request


Create new file


Upload files

Find File

Clone or download ▾

 ameter	updated readme to point to writeup	Latest commit 111380c on Dec 23, 2017
 <a href="#">examples</a>	initial commit	a year ago
 <a href="#">output_images</a>	finally got it tuned pretty well.	a year ago
 <a href="#">test_images</a>	initial commit	a year ago
 <a href="#">writeup_images</a>	added all writeup images	a year ago
 <a href="#">README.md</a>	updated readme to point to writeup	a year ago
 <a href="#">detect_vehicles.py</a>	added all writeup images	a year ago
 <a href="#">features.py</a>	added all writeup images	a year ago
 <a href="#">project_video.mp4</a>	initial commit	a year ago
 <a href="#">svc.p</a>	added suming within box. works well on test images... not so much on ...	a year ago
 <a href="#">test_video.mp4</a>	initial commit	a year ago
 <a href="#">train_model.py</a>	added all writeup images	a year ago
 <a href="#">writeup.md</a>	Update writeup.md	a year ago

 README.md



# 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.

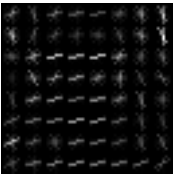
The code for this step is contained in lines 27 through 39 of the file called `./features.py` , and it's called on lines 39 and 43 of `./train_model.py` .

I started by reading in all the `vehicle` and `non-vehicle` images (lines 82-85 of `./train_model.py` ). 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=9` , `pixels_per_cell=8` and `cells_per_block=2` :



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

I settled on my final choice of HOG parameters through trial and error. I tried various combinations of parameters and color spaces, including RGB, HSV, HLS, and YCrCb color spaces, and the above combination produced the best results, acheiving approximately 98% prediction accuracy on the test set.

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).

I trained a linear SVM classifier (lines 112-118 of `./train_model.py` ) using the HOG features discussed above, as well as spatially binned color features (lines 42-46 of `./features.py` ) and color histogram features (lines 49-57 of `./features.py` ). I used GridSearchCV (lines 112-114 of `./train_model.py` to optimize the C value for the linear SVM classifier.

## Sliding Window Search

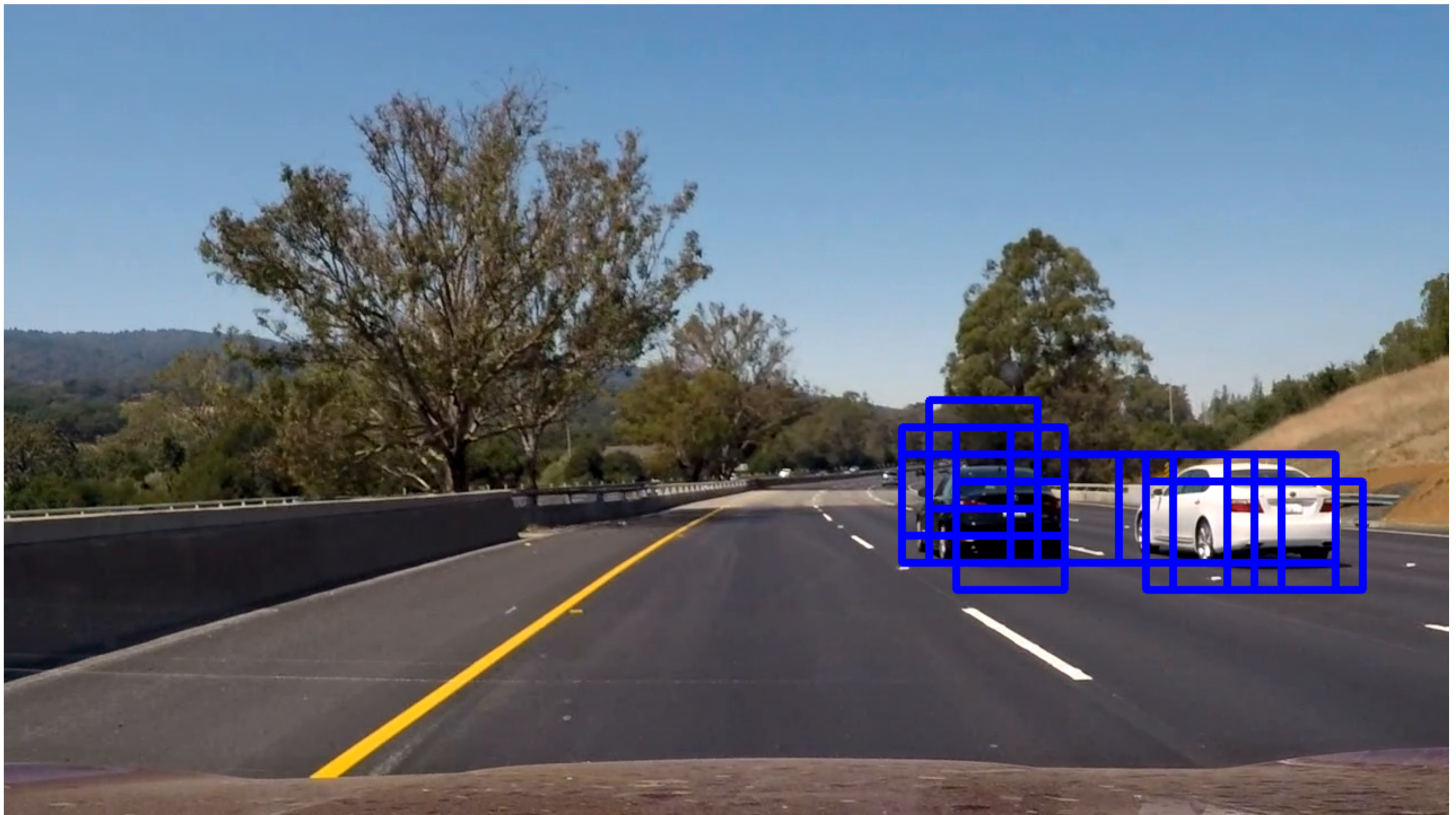
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide

what scales to search and how much to overlap windows?

I implemented a sliding window search of in input images (lines 70-146 of `./detect_vehicles.py` ). I searched the region of the image from just above the horizon down to slightly in front of the camera (ystart = 350, ystop = 656) and ignored the opposite side of the road (xstart = 459). I tested different scales, and settled on a scale of 1.5.

**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 a scale of 1.5 using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. I used GridSearchCV (lines 112-114 of `./train_model.py` ) to optimize the C value for the linear SVM classifier. I also normalized features using StandardScaler() (lines 94-98 of `train_model.py` ) Here is an example of the raw output of my sliding window search:



## 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.**



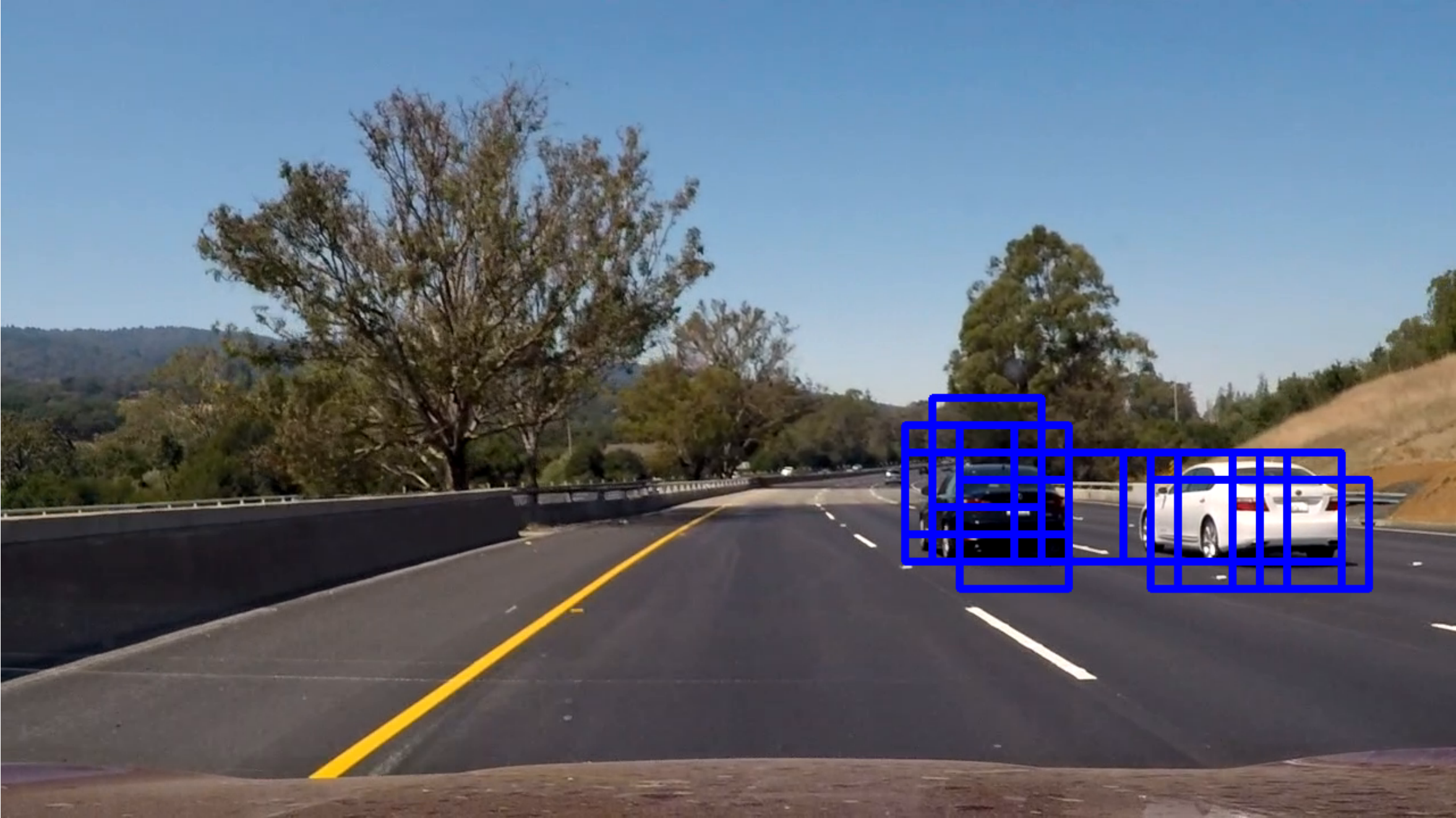
I recorded the positions of positive detections in each frame of the video (lines 137-143 of `detect_vehicles.py` ). From the positive detections I created a heatmap (line 144 of `detect_vehicles.py` ) and then thresholded that map to identify vehicle positions (line 164-171 of `detect_vehicles.py` ). I then used `scipy.ndimage.measurements.label()` and `scipy.ndimage.measurements.find_objects()` to identify individual blobs in the heatmap (lines 174-175 of `detect_vehicles.py` ). I then assumed each blob corresponded to a vehicle. I conducted a second level of false positive filtering based on the overall heat and position of the detections, determined whether they were for vehicles already being tracked, and then added the new positions to position arrays for either new or existing vehicles (lines 178-227 of `detect_vehicles.py` ). I created a `Vehicle` class to maintain data on each tracked vehicle (lines 27-65 of `detect_vehicles.py` ). Finally, I constructed bounding boxes to display the mean position of tracked vehicles smoothed over several frames (line 248 of `detect_vehicles.py` ).

Here's an example result showing the processing pipeline for a single video frame:

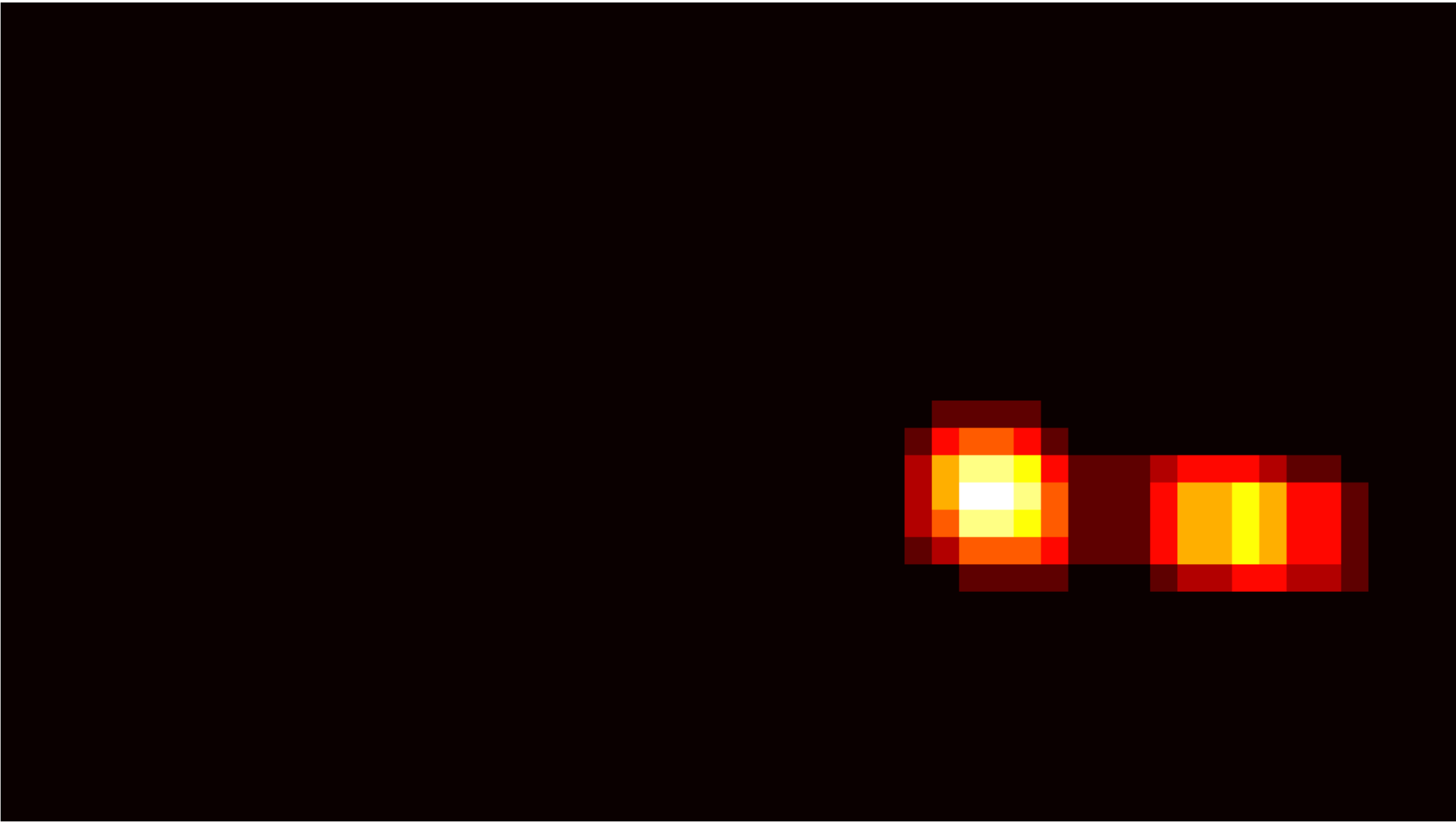
**Input video frame:**



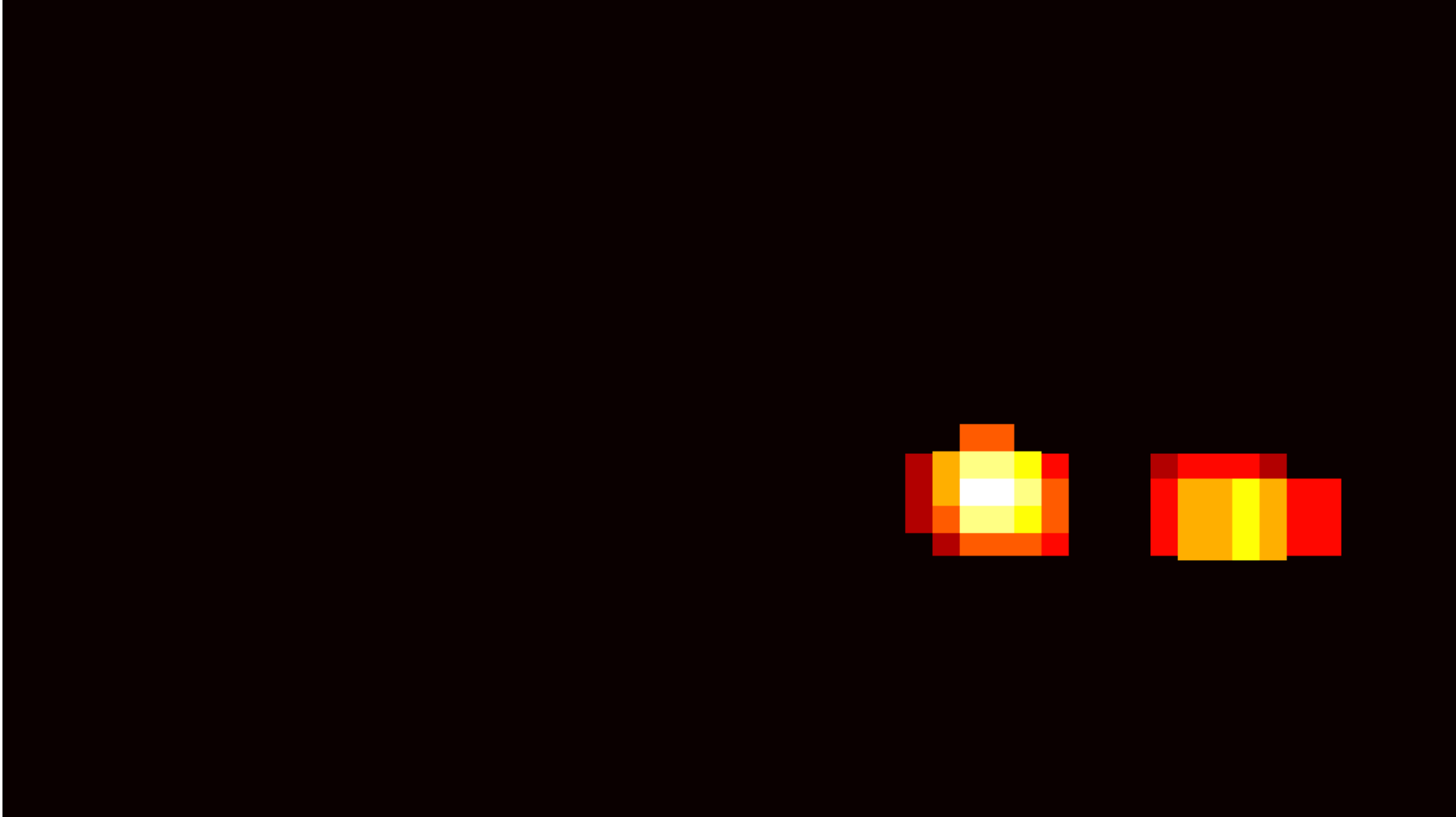
**Raw Detections:**



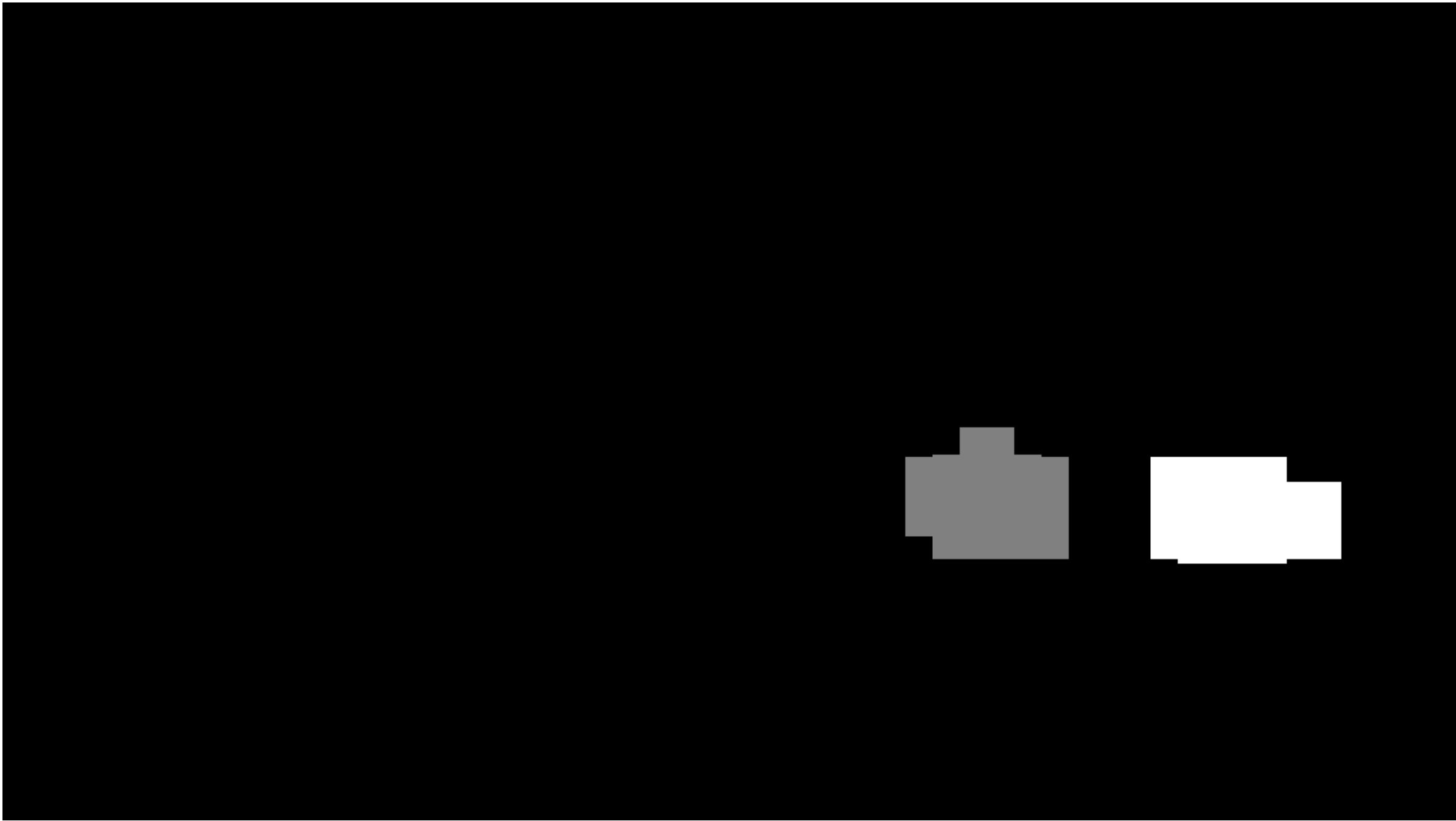
Raw Heatmap:



Thresholded Heatmap:

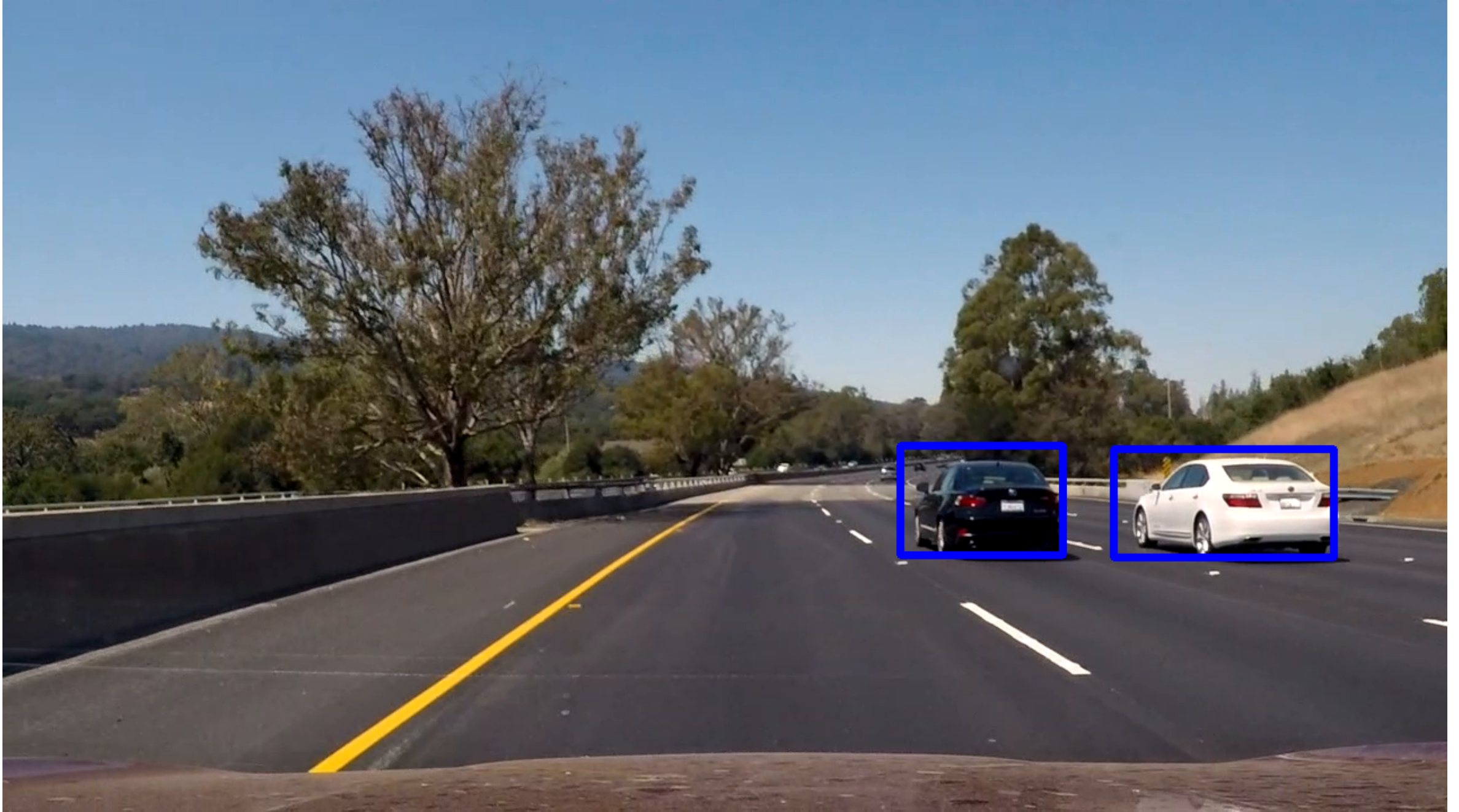


Labels:



Resulting bounding boxes:





## 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 had a lot of trouble tuning my processing and thresholding such that false positives were filtered out without filtering out without losing tracking of vehicles. I initially tried smoothing by storing several heatmaps and then combining them before passing the combined heatmap to `labels()`. However, I was unable to get consistently good, smooth results with this approach. Ultimately, I switched to tracking multiple positions for detected vehicles in a `Vehicle` class, and using that to smooth the results. This provided much better results. I also applied different heat thresholds to different regions of the image, so, for example, I applied a lower threshold to detections near the horizon than to detections closer to the camera (lines 165-172 of `./detect_vehicles.py`). Finally, in addition to thresholding heat values for individual pixels, I applied a second level filter based on the total heat within a detection and the position of the detection (line 188 of `./detect_vehicles.py`). These measures were helpful in achieving more consistent results.

Ultimately, my detection pipeline is tuned for the images and video included in this project. To create a more robust solution, more data, including different weather, road, lighting conditions, etc. would need to be included and tested.