



[< Back to Self-Driving Car Engineer](#)

Vehicle Detection and Tracking

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congrats on passing Term 1. You did an awesome job on this project and I hope that my suggestions are able to make your algorithm more robust!

Congratulations and I look forward to seeing you in Term 2!!

Writeup / README

The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

Hey the write up in the is great! I encourage that in the future you try to create it in the md format . This way you are able to publish work and projects on github for maximum visibility. If you don't have much experience with this, Udacity offers a course that I will link below.

<https://www.udacity.com/course/writing-readmes--ud777>

Histogram of Oriented Gradients (HOG)

Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.

Good job going with the YCrCb color channel! Research suggests it is more powerful!

Check it out: <https://pure.tue.nl/ws/files/3283178/Metis245392.pdf>

Also great job incorporating the use of spatial binning and color histograms!

The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.

Great job normalizing your data before training the classifier. Also the use of a Linear SVC was a great choice. Only advice here is to try manipulating the penalty parameter as a means to achieve better classification accuracy!

You can find more about penalty parameter here! ----> <http://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel>

Sliding Window Search

A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.

Great job with your sliding window search! Limiting your search area is a great way to speed up computation time! However, there should be a certain robustness to the algorithm. Finding the horizon and searching everything under it at two scales allows for the capture of a lot of data without computational strain.

Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)

Great job implementing a heat map and thresholding it! Some suggestions would be to use a scikit learns decision_function on your linear SVC to also help reduce even more false positives.

Video Implementation

The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

Your video overall looks great, perhaps average out the bounding boxes?

I will provide an example that I worked with, by no means is it the best but I hope it may be able to inspire future ideas :)

```
# Object Tracker

##### Globals needed #####
global cars
#####

class Object:
    def __init__(self, position):
        self.position = position
        self.new_postion = None
        self.count = 0
        self.frame = 1
        self.flag = False
        self.long_count = 0
        self.postion_average = []

    def update(self, temp_position):
        if abs(temp_position[2]-self.position[2]) < 100 and abs(temp_position[3]-self.
position[3]) < 100:
            if self.long_count > 2:
                self.postion_average.pop(0)
                self.postion_average.append(temp_position)
                self.new_postion = np.mean(np.array(self.postion_average), axis=0).ast
ype(int)

                self.position = self.new_postion
                self.frame = 1
                self.count += 1
```

```

        return False

    self.position = temp_position
    self.postion_average.append(temp_position)
    self.count+=1

    return False

else:
    return True

def get_position(self):
    self.frame+=1
    if self.count == 7 and self.long_count < 3 :
        self.new_postion = np.mean(np.array(self.postion_average), axis=0).astype
(int)

        self.count = 0
        self.frame = 1
        self.long_count += 1
        if self.long_count < 2:
            self.postion_average = []

    if self.frame > 10:
        self.flag = True

    return self.new_postion, self.flag

class Vehicle(Object):
    def __init__(self, position):
        Object.__init__(self, position)

class Person(Object):
    def __init__(self, position):
        Object.__init__(self, position)

class Sign(Object):
    def __init__(self, position):
        Object.__init__(self, position)
        self.label = None
        run_network()

    def run_network(self):
        # Crop out position and run through sign network
        # update self.label

```

```
# Takes in a list of calculated centroids calculated from current frame from your own
code (both good and bad)

# Deal with car tracking
for centroid in img_centroids:
    new = True
    for car in cars:
        new = car.update(centroid)
        if new == False:
            break
    if new == True:
        cars.append(Vehicle(centroid))

next_cars = []
positions = []

for car in cars:
    position, flag = car.get_position()
    if flag == False:
        next_cars.append(car)
        positions.append(position)

cars = next_cars

# Outputs current relevant positions.

try:
    for (x1, y1, x2, y2) in positions:
        cv2.rectangle(clone, (x1, y1), (x2, y2), (255, 0, 0), thickness=2)
except:
    pass
```

Discussion

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

[Rate this review](#)