By:

Abdullah Barnawi

Ryan Gadhi

Omar Almutirdi

Mohammed Alhaqbani

To:

Dr. Hamzah Luqman

Abstract

In this report we show our approach of implementing the car collision avoidance project based on a couple of chosen computer vision techniques. We use Histogram Of Oriented Gradients (HOG) for feature extraction, alongside with Support Vector Machines model (SVM) to detect the cars, and apply heatmap reduction technique during the detection process in order to get more accurate results and remove redundant detections. Subsequently, we use Probabilistic Hough Transform algorithm to detect the area of interest. Finally we build an algorithm that monitors the change of the area of nearby cars in order to predict a potential collision.

Introduction:

Car accidents are becoming more frequent every day, and the consequences of these accidents vary from no damage to humans and vehicles, up to a massive tragedy and the death of several people. Moreover, many of these accidents happen for pretty simple reasons, such as the driver getting busy with his or her phone and losing attention for a moment of the road. From here our project's idea came to raise. What if there is a way to inform the driver when a collision is about to occur and let him know before it was too late to recover his state and avoid the accident. This is what we attempt to implement during this project. In this report, we will describe the Car Collision Avoidance project and explain all the steps needed to implement it. We will discuss the methods used to identify and recognize cars, we will also address some issues related to the car recognition process and how we managed to solve it, then we will talk about what we did to detect a collision. An important note is that when we decided to work on this project, we agreed that we would implement everything by ourselves. This might reduce the quality of the result but will maximize the learning outcome, hence you might find some code that is not implemented in an optimized manner, or using normal loops instead of numpy's fast slicing technique.

# Literature review:

Object detection is one of the most studied computer vision problems. Therefore, there has been lots of techniques in literature attempting to solve it. The variation of these techniques fall mainly in the following two categories: the use of the classical machine learning models such as Logistic Regression with manually crafter features fed into it, or the use of Deep Learning techniques such as Recurrent Neural Networks with raw data. The features extracted by the computer vision techniques are fed into those machine learning models so that the patterns can be discovered automatically and later used for classification. In the later category however, there has been great success achieved by using raw data, that is, the pixels themselves. There have been, of course, other attempts that use pure computer vision methods and algorithms that have been proven to be good as well, but they , such as Eigenfaces, had limitations. In that, they have some requirements that sometimes make these methods not efficient to use. In Eigenfaces for example, K-Nearest-Neighbors (KNN) is used for finding the closest match in the training set, and calculating that takes a long time to give the prediction especially for large datasets.

One of the heavily used methods in object recognition is the sliding window technique. A common issue with that technique is the overlapping recognition areas. That is, recognizing the image multiple times, which leads to multiple recognition boxes being drawn. One method that has been attempted and that work relatively well is the use of heatmaps. The heat represent the number of overlapping recognition areas. That is, the higher the heat the more recognition boxes that has been drawn in that area, and consequently, the higher the probability of detecting a car over there. **Figure 1** presents the idea of the heatmap.

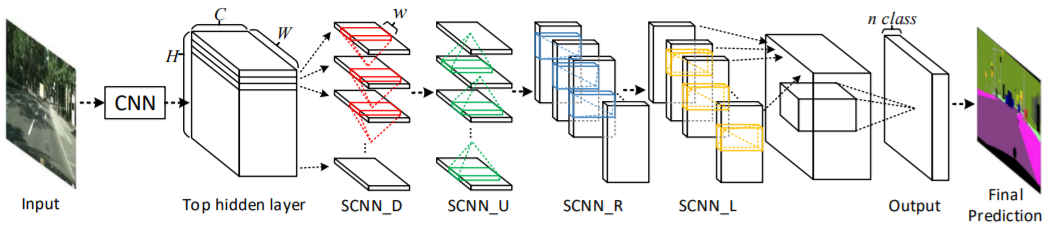
**Figure 1:** The following images show the idea of the heat map. Red area indicates the hottest area, and blue indicate the cold areas.

source : <https://www.youtube.com/watch?v=_9p7UVDk08Q>



Self-driving cars perform the collision prediction, which is a similar goal to our second part of the project. The first part of that is to know the region of the lane that the host car is driving in. In most real-word applications, self-driving cars depend on external sensors, all of which give their prediction system more information about the environment such as the distance from all objects on the street, or the GPS signal to expect and approximate the lane curvature. However, these applications also make great use of visual data, the images, but depend mostly on Deep Neural Networks methods. An example of that is the Spatial Recurrent Neural Networks, shown in **Figure 2.** That technique uses raw pixels, and the network architected is made in away such that it takes the job of extracting the patterns (i.e. the lane boundaries). This last method has been proven to work better than classical computer vision techniques such as the Hough transform, especially in cases that have lanes with high degree of lane curvature. However, Hough transform has the advantage of not needing to be trained with any sort of data. It discovers the lane boundaries (i.e. the lines) based off of the shape that Hough method is trying to match.

**Figure 2:** Spatial Recurrent Neural networks for lane detection. Reference: <https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132>



Collision detection can’t be achieved with the lane boundaries and car recognition only, there has to be some algorithm that can take both of these pieces of information and combine them in order to predict a potential collision. One simple method that can be used to achieve that is to calculate the rate of change of the area represented by the recognized cars with respect to time (i.e. with respect to the frame sequence). Most real-world applications use sensor data to know the exact rate of change of the distance between the host car and other cars on the street. After that rate is calculated, it is checked against a threshold. If the rate is too high, then it is possible that the car ahead is breaking, so the host car needs to break as well, or it could mean that the host car is accelerating, hence change of speed with respect to the time of the host car could be used as an input in the algorithm to minimize the false positives.

# System overview:

Our approach towards solving the collision avoidance problem was focused more on the computer vision techniques, since that was the topic studied in our course. We of course needed to use a machine learning model so that we learn the patterns of the feature descriptors (i.e. the ones extracted from our training data) of our images. Our system pipeline works by firstly detecting the cars in each frame. After that, going over all of the recognised portions of the frame and run our reduction algorithm, the one that utilizes the heatmaps in order to reduce the number of recognition boxes drown on that frame. Subsequently, and after we have exact clean boxes around the cars, we use them in the second phase, the collision detection phase.

Our collision detection phase considers the cars in the lane that the host car is driving in. The reasoning behind that is logical: If a car in another lane is breaking very fast (i.e. potentially causing an accident), that will not cause a danger to the host car driving in its separate lane. After we run our detection algorithm we display a warning if there is a potential car collision predicted. In the following sections we discuss our methods in more detail. The attached html includes the implementation details. In the following sections we will be presenting our approaches taken to achieve the collision avoidance, challenges faced, and methodologies utilized to resolve them.

## 

## 

## 

## 

## 

## 

## 

## 

## Car Recognition Phase

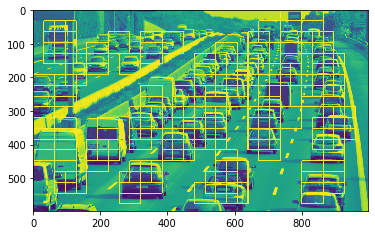
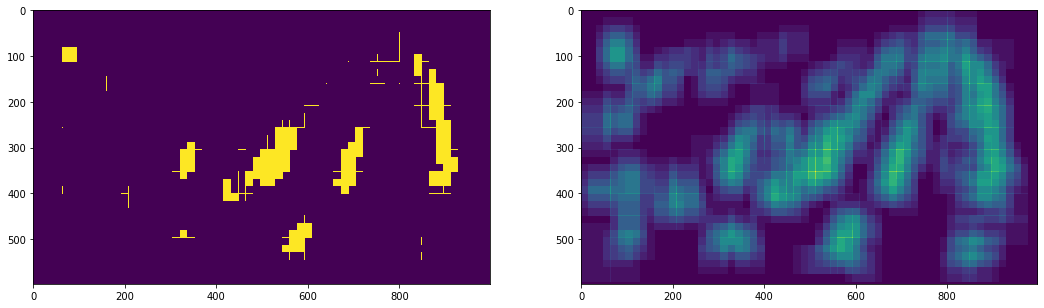
Since we are going to use a classical machine learning model (Since we want to extract the data features ourselves), we need to use a descriptor extraction algorithm. We have experimented with Scale-invariant Feature Transform (SIFT) algorithm, but we eventually chose to go with the free and probably more supported Histogram of Oriented Gradients (HOG) algorithm. After we gather all of the needed image descriptors, and we input them into a classical machine learning model. Here we have chosen to go with Support Vector Machines, since they have been proven to work well with image recognition in literature.

We had a dataset that included images of two classes, cars images and streets images with random noise. We had around eight thousand images from each class, and iteratively, we extracted their HOG features and trained an SVM model. Once we got the model read for use, we implemented the sliding window algorithm. Our version of it was using a window of the same size of the training images that is fixed. For detecting the close cars, however, we do pyramid representation of the frame. That is, we resize the frame to a smaller size so that the bigger cars and the close ones to the host car fit into our sliding window and therefore get recognized.

After implementing the sliding window, we had to deal with its common issue, recognizing the object (i.e. the car in our case) more than once. In fact, if we complete with this version of the sliding window, our collision prediction algorithm would run slower since it has to test against all redundant boxes, not to mention the incorrectly classified portions of the image (i.e. the false positives) would still be there. So we had to solve it using a reduction technique, a one that reduces the number of boxes, and what we used here is the heatmap technique.

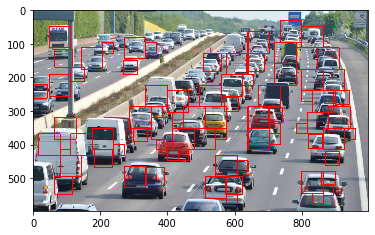
We did not follow an already implemented method for that. Instead, we made our own version of it. It works by gathering all the recognition boxes and calculate how they intersect. The more intersections there is, the hotter the area and the higher the probability is that there is a car in that overlapping portion. We achieved that by finding all possible intersections of all recognition boxes in the frame, after which adding them all together. Here, we had three dimensions. The normal horizontal and vertical dimensions that represent an overlapping area, and the depth dimension which includes all intersections in the original image. **Figure 3** shows our version of heatmap compared to the original image of recognition boxes.

Figure 3: the image on the left is the heatmap of the one on the right.



Other implementations of the heatmap do consider the time as an important part of the recognition. That is, if at a certain area there is a car detected then, it is likely that in the second frame the same car will be detected again with a little shift in the location. This is similar to the brightness consistency assumption of Lucas Kennedy but this one is for detected boxes. This method also has the benefit of reducing the number of false positives, since it is less likely that they will be consistently recognized in subsequent frames. However, the implementation of this three dimensional heatmap was more complex, and it would add a fourth dimension to our implementation, so we decided not to include the time variable. **Figure 4** shows the results of implementing this heatmap reduction algorithm, notice that the overlap of boxes is now significantly reduced. One thing worth mentioning here, is that we had a large number of nobs (i.e. hyperparameters) to change so that our algorithm would accommodate to all possible variations of video resolutions and frame rate. We have chosen appropriate values for the representation of results in this report.

**Figure 4:** The heatmap reduction method run over the previous image.



## 

## Collision Prediction Phase

We initially planned to track the cars and measure the change of their areas with respect to frames, however that approach is good but we also figured out that what if the car was on a different lane, and their area change increased dramatically , in our normal approach this behavior will detect a collision where there is no collision since the car is in a different lane. So , we found out that we only need to concentrate on cars that are in our lane. The following several sections well explain the procedure of detecting and tracking the lane

## 1- Detecting and tracking the lane

To detect the lane we do the following :

* Crop the image to a trapezoidal shape since the rest of the image will not be interesting to the line detection approach instead of adding more noise.
* We apply canny edge detection on the image to detect the edges
* We use hough transform on the cropped canny image to get a set of lines
* the lines that we obtain from the hough transform are too much and we only need two line , so we define a way ( or several ways ) to remove unnecessary lines

Note that some application could rely only on the trapezoidal shape as the area of danger and skip detecting the lane, since the objective of the lane is just to identify the danger area. In our application we apply lane detection to get more precision and to minimize the false positives.

1- **Cropping the image to a trapezoidal shape since the rest of the image will not be of interest to the line detection approach and applying edge detection on the cropped image :**

We can see in the image below (Figure 5) how we are able to apply canny only on the region that we are interested in, note that we applied several parameters for the region of interest. the one in the image gave us the best results.

**Figure 5:** Canny of the region of interest.



2- **Using hough transform on the cropped canny image to get a set of lines :**

As we see in **Figure 6**, applying hough transform on the canny image did not result in the expected output. The detection algorithm only needs 2 lines to represent our lane, more than 2 lines is considered extra noise hence, we had to remove all the extra lines

.

**Figure 6:** Result of hough transform on the right and the result of removing horizontal lines on the left..



From observing **Figure 6** we can immediately notice that all horizontal lines are not needed so we can directly delete them using their slopes. The result of removing them is also shown in the same figure. Before deleting them we had around 50 lines after deleting them we get around 20 lines. This is a great improvement since we removed 50% of noisy lines , however we only want 2 lines so we will need to remove 100% of the noises which we will see how shortly. To remove all lines except 2 we have several methods

* Taking the average slope of the right lines ( lines with negative slope) and the average slope of the left line (lines with positive slope)
* Taking the slopes with lowest value ( absolute value) for each right line and left line.
* Taking the slopes of lines with the highest absolute values for both right and left line

After experimenting with both 3 methods, we found the best results are generated using method #2. We can see the result of applying the method in (Fig 7).

**Figure 7:** Result of deleting all unneeded lines on the left and final line on the right.



After that, and since we have 2 points of each line, we can extend the lines to the vertices of the cropped image that we initially illustrated. The result is shown in the right image in **Figure 7**.

## 2- Predicting the collision

After We have succeeded in detecting the lane, we know need to monitor the cars that enter our lane, and check both the change in area and the change in distance. We tried to implement several methods to calculate the area of the box inside the lane. One of those methods was to calculate the intersection of the box with the lane, and use those points of intersection to calculate the shape’s area *(check the walkthrough for more details about this implementation).* However, this method has many complications so we used another method, which is to convert the lane into a black area, slice the box and calculate the number of 0 pixels in that box **Figure 8** shows two sliced boxed where the one on the left has 89% of its area inside the line, while the box on the right has 3% of its area inside the lane

**Figure 8:** Two sliced boxes intersected with the lane.



Now we can create our algorithm that will use the distance of the box and the area and will run every frame to predict the collision. The implementation details of the detection process can be found in the walkthrough.

# Experimental Results :

We can see the results of each step we implemented in the figures **Figures 1-8**. The final result is a video (named final\_ouput) which is attached with this report in the project folder. The result shows a video where a car drives and our system detects the cars nearby, while monitoring the lane for collisions. Note that in the video, there were no collisions, and hence we created our own test boxes to be able to illustrate what the system should do in case of a collision.

## Conclusion

To conclude, In this report have discussed our collision avoidance project. We described the methods and techniques used, and we presented the results that we have established through each phase of the project. As we can observe from the attached project walkthrough, we focused on using the techniques provided in the course to find and predict car collision. Moreover, we succeeded an achieved pretty good results. However, these results could be improved further and become much better by changing some of the techniques we used. for example, we could have used Lukas Kanede’s algorithm to track the cars, and reuse the car detection algorithm every 10-15 frames. We could also optimize the car detection code by developing our version of the heatmap and by using numpy’s arrays and its fast slicing.

## Our Project Evaluation and Opinion

This project was a very pleasant experience, we obtained more team management skills, and gained a big amount of new knowledge of an area that we were all un-knowledgeable of. Linear algebra made more meaning, since computer science aspect of it was understood and applied. We have become more interested in this subject and wish that the university keep offering such great courses.

## 

## References :

Haque, R., Islam, M., Alam, K., & Iqbal, H. (n.d.). *A Computer Vision based Lane Detection Approach*.

Mithi. (2019, June 19). Vehicle Detection with HOG and Linear SVM. Retrieved from https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a.

Parajuli, A. (2017, April 3). Smart car 102: Vehicle detection using HOG. Retrieved from https://medium.com/@trustabhishek/vehicle-detection-using-hog-fdc569e36baf.