CS/RBE 549 Computer Vision, Fall 2019

Project Report

Team - Traffic Signs

Member	Signature	Contribution (%)
Vrushabh Desai	[illegible]	25
Nagarjun Vinukonda	V.Nagarjun	25
Rishi Madduri	[illegible]	25
Kavit Shah	Kavit Shah	25

Grading:	Approach	_____/20
	Justification	_____/10
	Analysis	_____/15
	Testing & Examples	_____/15
	Documentation	_____/10
	Difficulty	_____/10
	Presentation	_____/20
	Total	_____/100

Table of Contents

LIST OF FIGURES

Traffic Signal Detection system

Abstract:

In day to day life traffic signal detection has become an important thing in order to avoid accidents. Our objective is to implement an object detecting algorithm that detects traffic signals accurately under different environmental conditions like illumination, motion, etc. In this paper, we used classical computer vision techniques like Hough, SIFT and Top hat filter to detect the traffic lights. We also used YOLO R-CNN for accurate detection and made a comparative analysis with other methods implemented. The results of this project are stated with different cases that defines both the prospects and challenges of our algorithm implementation.

SECTION-I:

Introduction:

Nowadays, traffic signals detection system has an immediate use to avoid accidents. From an online survey sourced by AAA foundation for traffic safety, deaths from drivers running red lights in America has reached more than 30% in 2017 from 2009 as shown in fig.1. [1]

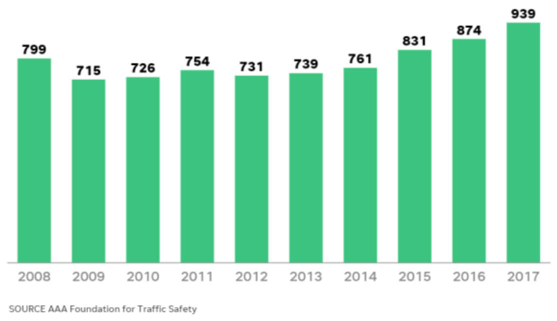


Fig. 1: *no. of killings due to red lights running Increased 31% from a low of 715 in 2009 to 939 in 2017.*

Our motivation is to develop a robust system that can detect traffic signals under different environmental conditions like night, more illumination, motion, rain, etc. that helps drivers prevent running over red lights. Poor detection of traffic lights due to these environmental conditions are one of the reasons for accidents.

This paper describes approaches implementing SIFT, Hough and Top hat filter methods to detect traffic signs. Further, for better detection neural networks (CNN) is used to classify traffic signs and detect accurately.

1.1 Data set:

As a part of our project we required a large dataset for testing different algorithms and approaches. We obtained the dataset of the Images from WPI's Embedded Computing Lab's website [2]. The images that we used were taken from a moving vehicle using a dashboard camera. Images in the dataset were different from one another in terms of the distance of the signal from the camera and the orientation of each image. We also used a video obtained from Youtube which was shot using a similar setup of dashboard camera. The video was used to demonstrate detection of Traffic Signals in real time.

This report is organized as follows. Section 2 describes different approaches used for detecting traffic signs. In Section 3, results and discussion is detailed. Section 4 describes future scope and followed by references.

SECTION-II:

Approach:

I. Scale Invariant Feature Transform (SIFT)

Scale Invariant Feature Transform (SIFT) is a technique used in computer vision to detect features in an image.

The way SIFT algorithm works is that it first creates a scale space of the image where the test image is progressively reduced in size in multiples of 2. Each of the resized image along with the original test image are progressively blurred in multiples of a constant (usually 1.2) and each set of blurred images of a given size form an octave as shown in fig.2. The Laplacian of Gaussian operation is performed on each octave, which is approximated using Difference of Gaussian (DOG) to speed up the computation process. The points of minima/ maxima are computed in each DOG images and checked if they lie on DOG images above and below the current image in each octave. If a point is found to be a minima/maximum in all three DOG images, it is declared as a feature point of the test image. The feature points with low contrast are eliminated and the remaining feature points are stored as feature vectors. Each feature vector is a vector of 128 numbers with stores overall magnitude and direction information of gradient at that keypoint, and thus can be referred as 'fingerprint' of a keypoint. [3]

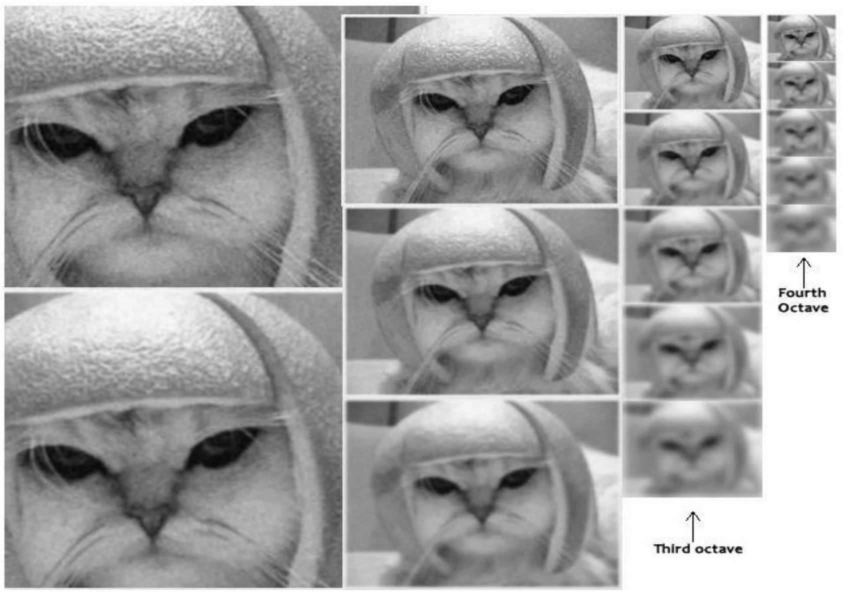


Fig 2: Scale space of image of a cat

In order to match keypoints in two images, the algorithm compares each feature vector from the new image with feature vectors of the test image and selects the 'best-match' based on minimum Euclidean distance between the two feature vectors [4].

For detection of traffic signal, this SIFT feature detection algorithm is applied on two images of traffic signals of same scenario taken from two different perspectives. As seen in fig.3 it was observed that SIFT was able to detect a few feature points on the signal for images which differ in perspective.

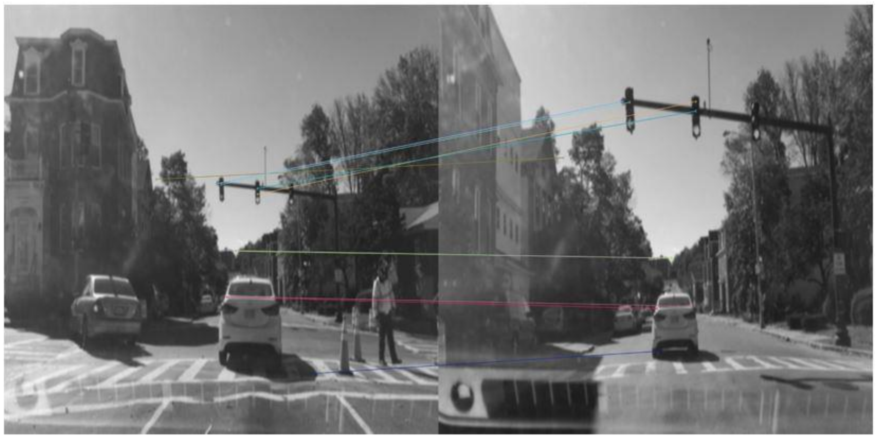


Fig 3: SIFT on two images of same scene but different perspective.

When applied on cropped of a traffic signal in the same scene and same perspective, SIFT was able to detect and match features with improved accuracy as shown in fig.4.

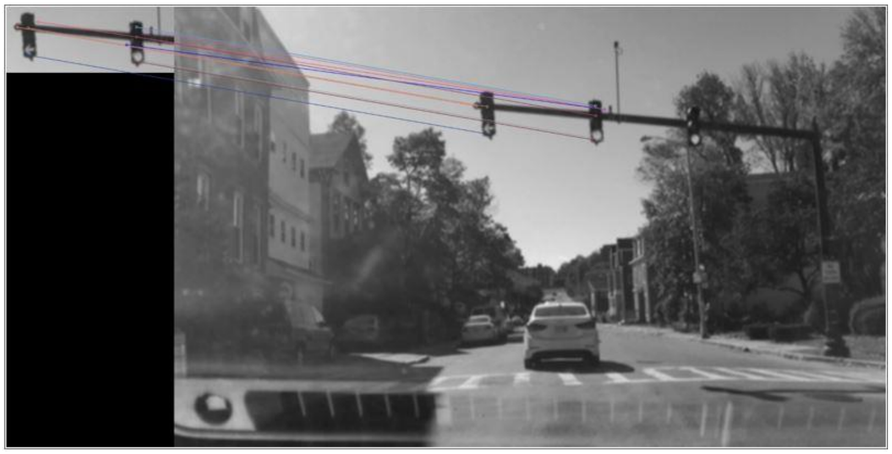


Fig 4. SIFT on images of traffic signals in same scene and perspective

The SIFT algorithm is invariant to scale but not to changes in perspective, thus when two images of same scene are compared from same perspective, the accuracy of the algorithm improved as compared to images from different perspective.

But when the scene is changed, the performance of SIFT algorithm becomes worse. As seen in fig.5, the matching accuracy SIFT algorithm deteriorates as it is matching any two points from the two scenes

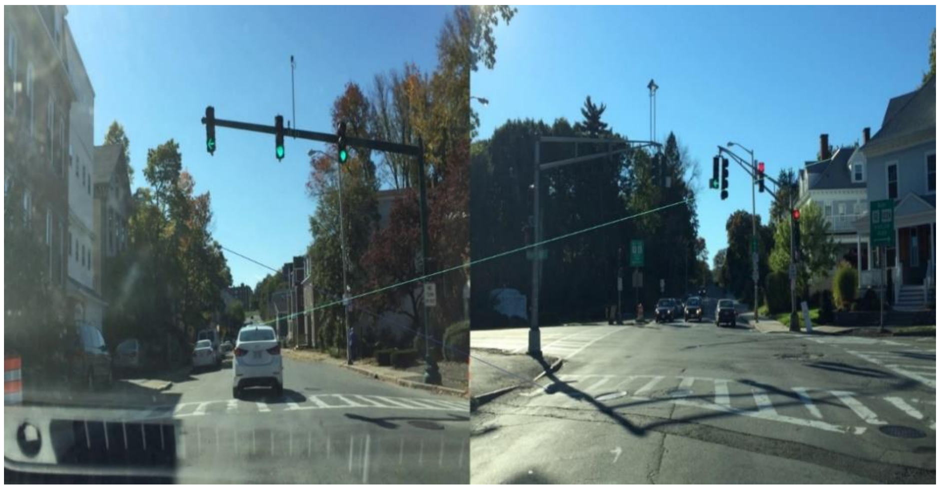


Fig 5. SIFT on images of traffic signals in two different scenes.

This accuracy deteriorates even further when two images from different scenes are matched where one of the images is too small (cropped image of traffic signal). This is evident in fig 6 where not even one feature point is matched between the two images.

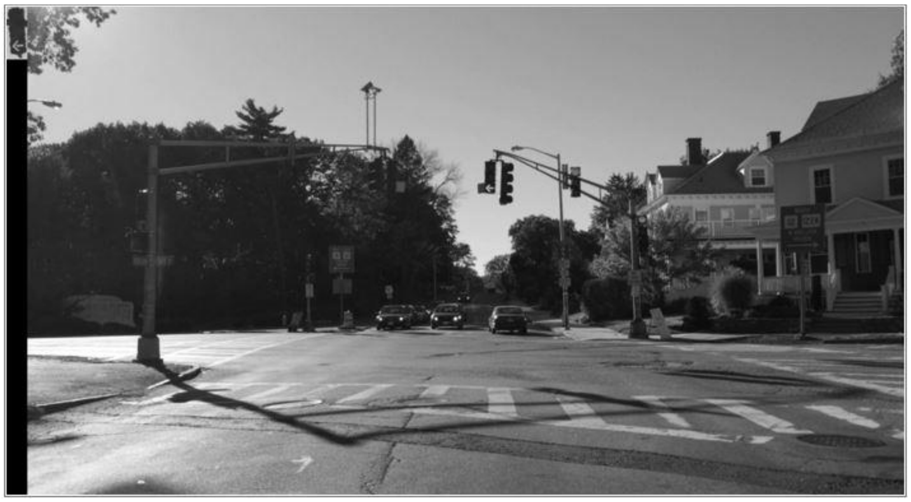


Fig6. SIFT on a cropped image with another image of signal from a different scene

The results of the analysis are similar to the properties of SIFT algorithm according to which SIFT can match features robustly in images from same scene and same perspective. But when the perspective is changed or even worse, the scene itself is changed, the algorithm fails to match features with accuracy and robustness.

II. Top Hat Filter with SURF feature detector:

Top hat filter is a mathematical morphological operation that extracts tiny elements and details from images.

It is used for Feature Extraction, Background Equalization, Image Enhancement and various other Image Processing operations. The size or width of the elements that are extracted using the top hat filter depends upon the choice of the structing element. The bigger Kernel we choose, the larger the element we get.

There are two types of Top Hat Filters:

- White Top Hat Transform: The White Top Hat transform is obtained by performing the Opening Operation on an image in grayscale and subtracting the result of this operation from the input image in grayscale

$Let\ A: E \rightarrow R\ be\ a\ grayscale\ image, mapping\ from\ Euclidan\ space\ to\ a\ real\ line$

$Let\ h(x) be\ the\ Structuring\ Element$

The White Top Hat filter is then given by:

$$F_w = A - (A\ o\ B)$$

Where $(A\ o\ B)$ is the opening operation performed by eroding an image using a structural element and then proceeding to dilate the result using the same structural element. It is mathematically represented by:

$$(A \ominus B) \oplus B$$

Where $\ominus$ & $\oplus$ $are\ erosion\ and\ dilation\ respectively.$

- Black Top Hat Filter: The black top hat filter is given by closing an Image first and then subtracting the input image from the results of the closing operation [5].

For the purpose of this project, we have used white top hat filter algorithm and the SURF detector.

SURF or Speeded Up Robust Features is a key point detector that is several times faster than SIFT. It works by applying an approximate of Laplacian of Gaussian mask to an image at multiple scales. SURF owes its computational speed to the use of integral images where the value of a pixel coordinate (x,y) is the sum of all the values in the rectangle defined by the origin and the pixel coordinates (x,y) . Value of the Pixel coordinates is calculated by: [6]

$$I_x(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j)$$

With the above equation, it only takes 4 sums or addition operations to calculate the pixel values or sum of intensities of a region. The Surf Descriptor is Scale and Rotationally invariant. Its feature descriptor depends on the sum of Harr wavelet transform around the point of interest.

SURF makes use of the Hessian Matrix for selecting the location and scale. The Hessian Matrix for a pixel is given by

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial^2 y} \end{bmatrix}$$

To adapt to a scale, the images are filtered by a Gaussian kernel. The Hessian Matrix $H(x,\sigma)$ at a scale σ is defined by

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix}$$

Where Lxx(x, σ) is the convolution of the Gaussian second order derivative with the image I in point x.

In order to be rotationally invariant, SURF identifies a reproducible orientation of the interest points. It does this by calculating the Haar-Wavelet response in the x and y direction in a circular neighbourhood around the key point. It then proceeds to calculate the sum of vertical and horizontal wavelet responses in the scanning area. SURF then changes the scanning orientation by 30° and proceeds to recalculates until the orientation with the largest sum value is found. The orientation is the main orientation of feature descriptor. [6]

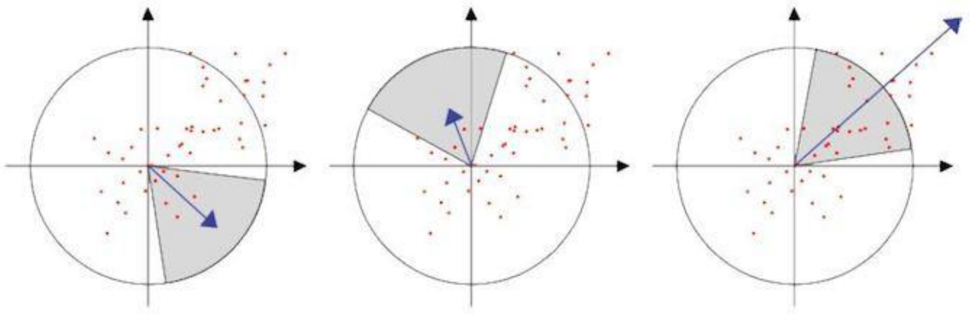


Fig 7: Rotational invariance property of SURF

The first step of the implementation of the Top Hat Filter algorithm was to import the RGB Image and convert it to grey scale.

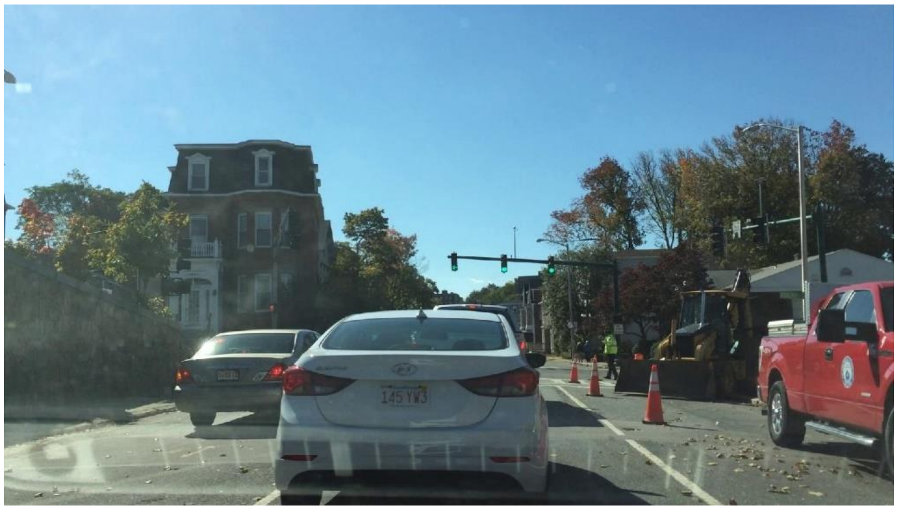


Fig 8: Test Case 1(a)-RGB

In the above image, there are three traffic lights present and all three of them are indicating the green light or the green signal. Our aim was to detect all three of these signals. Upon converting them to Grey scale and applying the White Top Hat Filter algorithm, the following results were obtained.

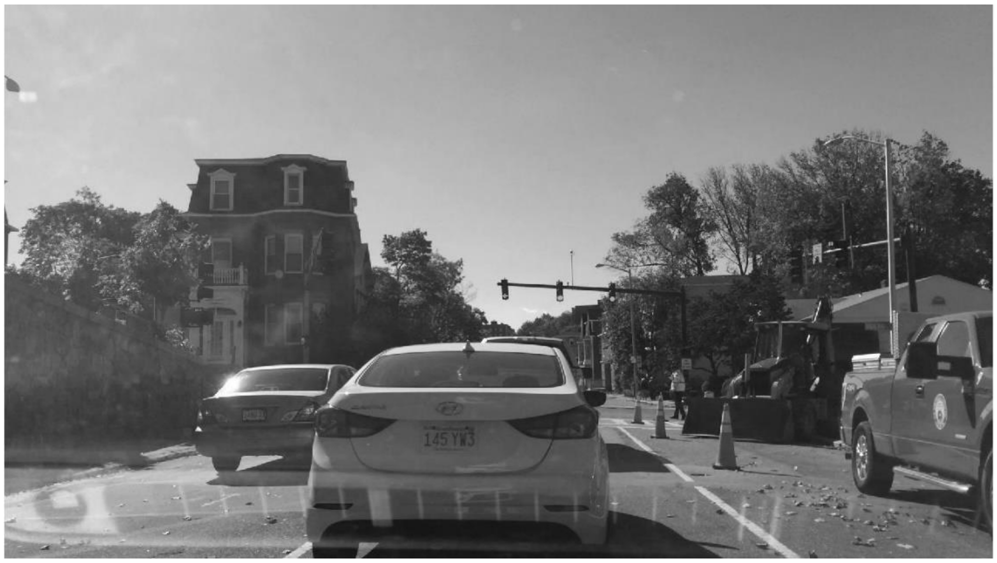


Fig 9: Test Case 1-b (Greyscale)

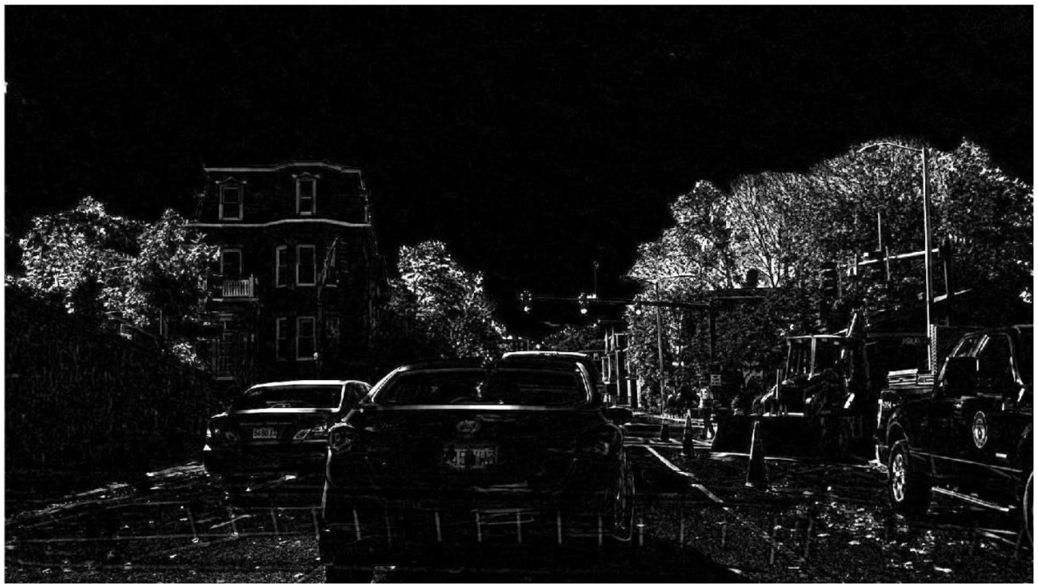


Fig 10: Test Case 1-c(Top Hat Filter)

The application of the Top Hat Filter results in the image shown above. As we can see, the regions with high intensity colours are now shown as bright spots on the screen. These bright spots can later be picked up by the SURF Feature Detector as key points.

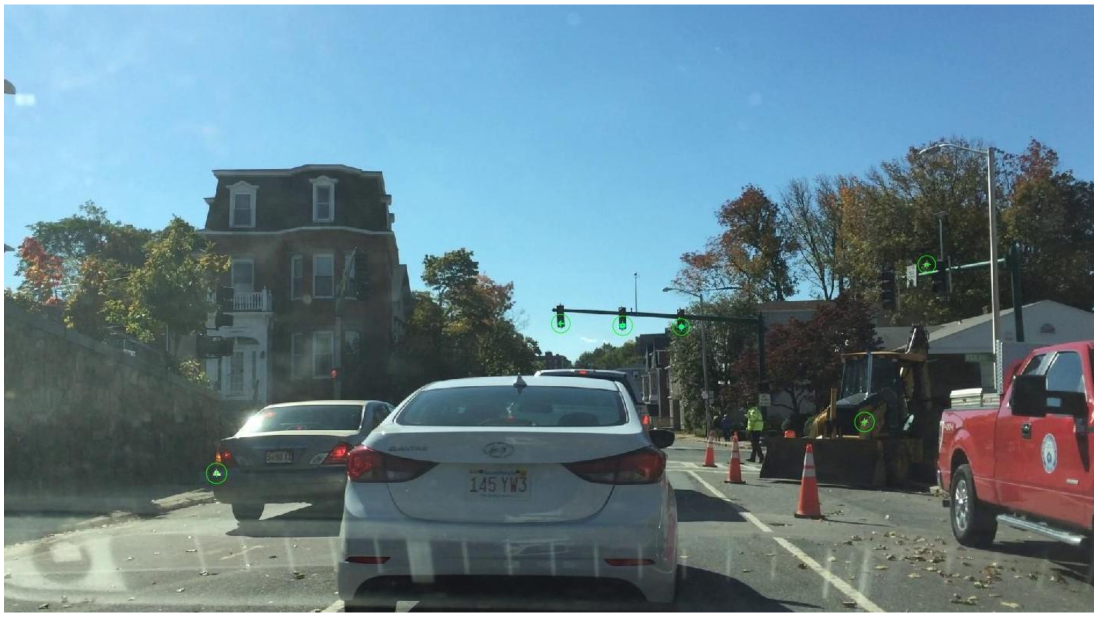


Fig 11: Test Case 1-d (Detection)

In the above image, the SURF feature detection algorithm detected the key points that were obtained after application of the top hat filter algorithm. However, along with the traffic signal, a few outliers were also detected.

In order to build a more robust detector, a few more test cases were run.

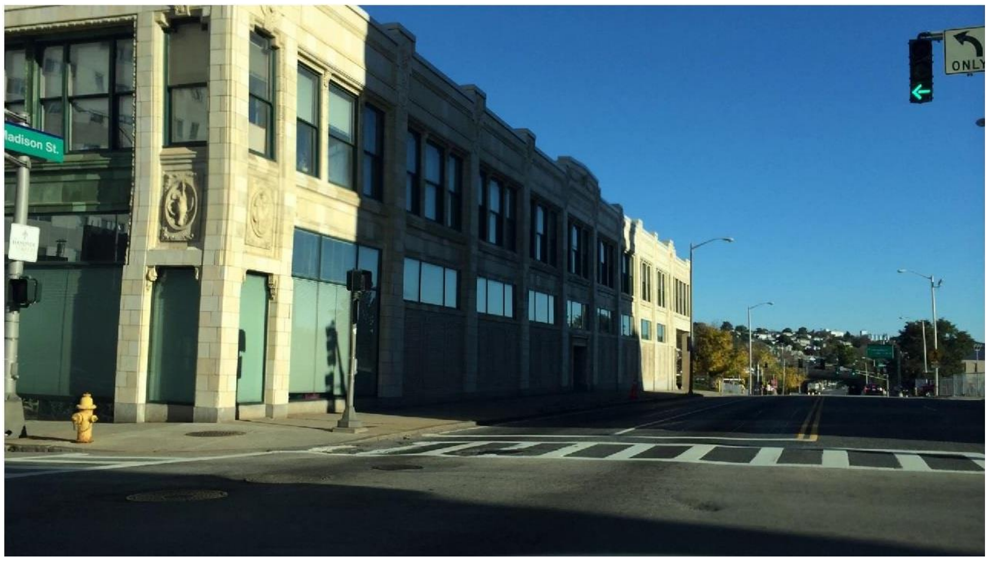


Fig 12: Test Case 2 (RGB)

The Euclidian distance between the dashboard camera and the traffic signal intended for detection is much less than the distance between them in the first test case. This is a slightly more complex image as the traffic signal is being overshadowed by an adjacent building, however, there is no shadow on the building on the left. The significance of these lighting conditions is reflected in the observation that, when the image is converted to grayscale and undergoes binary morphology operations, the building on the left and the glowing green signal will be of the same intensity.

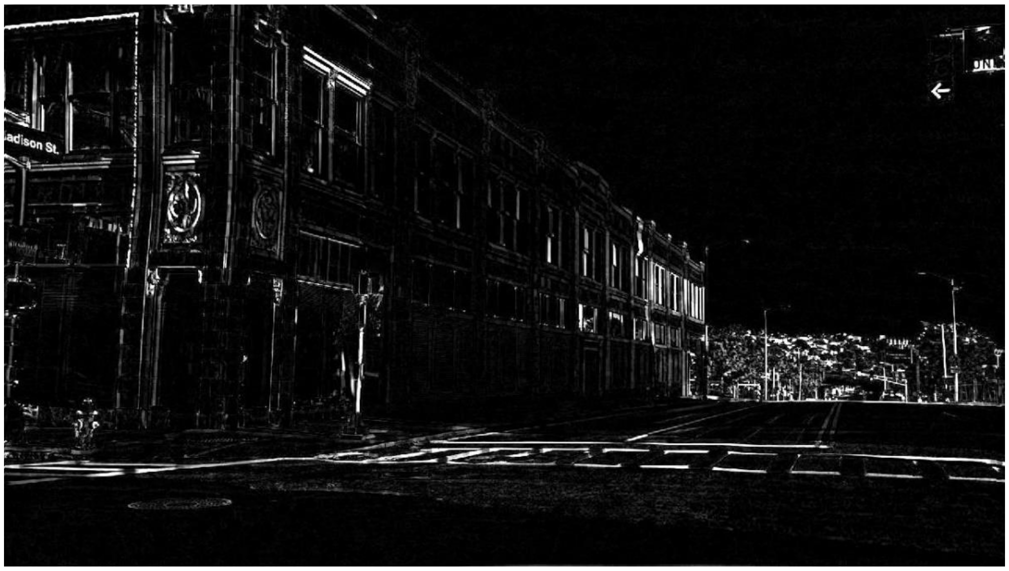


Fig 13: Test Case 2

In the above image, the traffic signal and the window frame on the left are glowing with same intensity.

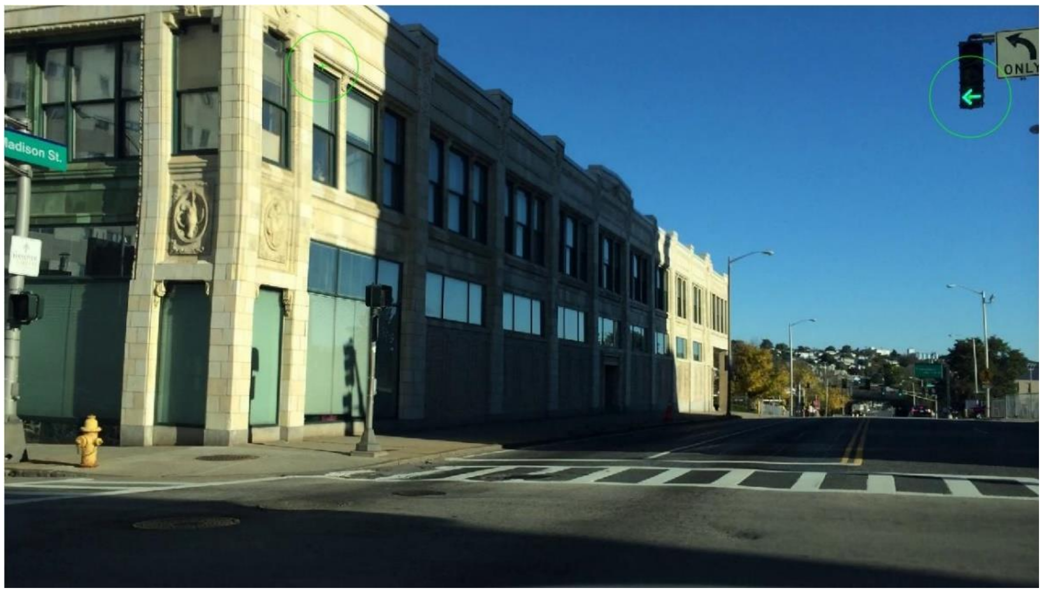


Fig 14: Test Case 2 SURF detection

The similar intensity causes the SURF feature detector to mark the outlier on the left.

For the third case, the following image was used.

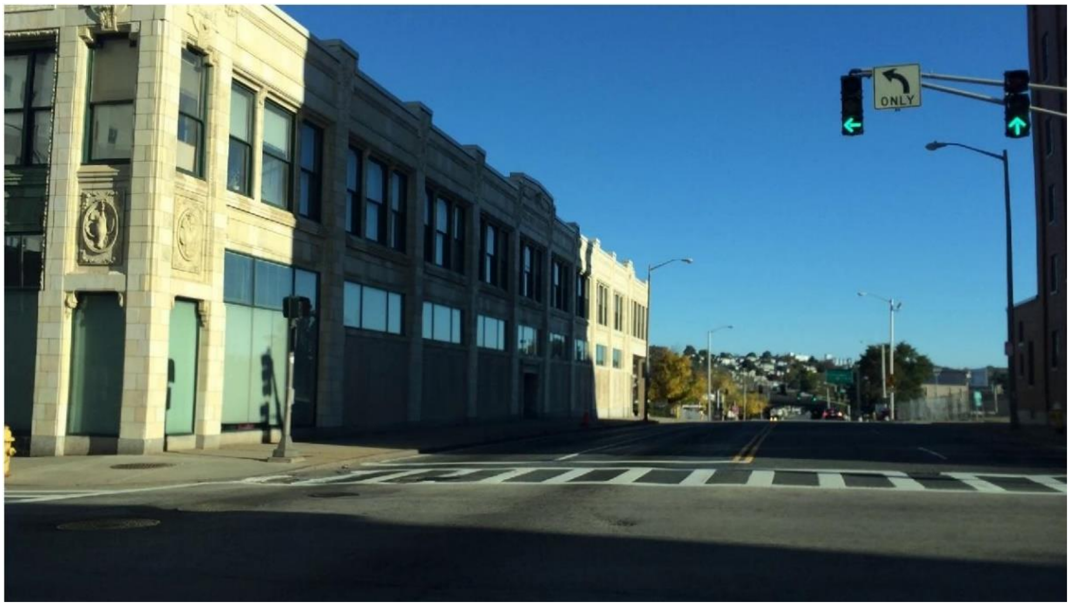


Fig 15: test 3

The third case is a slight modification of the second test image. In the third case, we attempted to detect two signals instead of one.

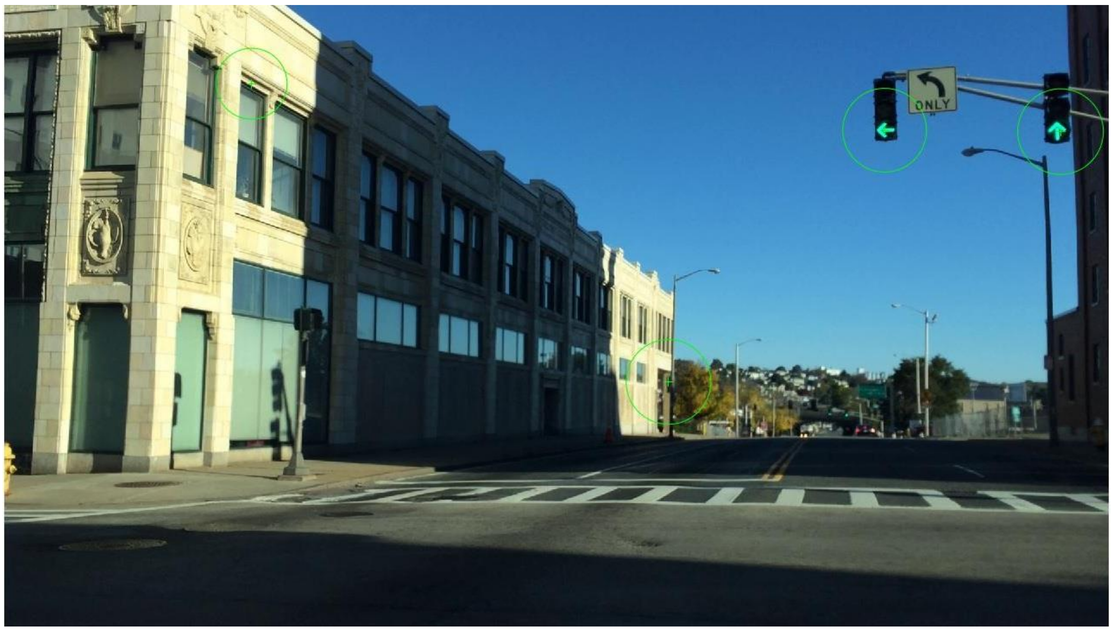


Fig 16: test case 4

While both the signals were successfully detected, the outlier on the left still persists.

In order to come up with a more accurate and a robust detector, we moved onto the Hough Circle implementation.

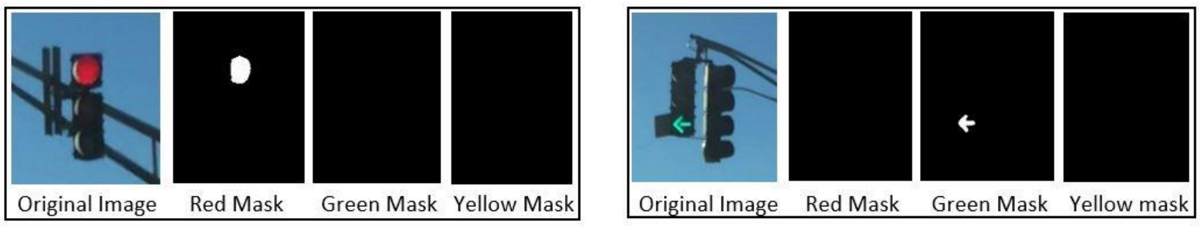
III. Hough Transform and Thresholding on HSV Colour space

Traffic signs usually have three main colors: Red, Green and Yellow. The area of interest can be narrowed down using this color information. We specified a range of threshold in RBG image space for red, green and yellow respectively. When applying this range of threshold in the given RBG image various outliers also lies in that specific range, since RBG colored range are susceptible to variation in illumination. So, to address this problem we used HSV (Hue, Saturation, and Variation) image space for thresholding. Using this HSV image our next goal is to define HSV range for red, green and yellow so we can segment our image based on three different colors. HSV range for three different colors used are as follows:

Color	**Lower Range (HSV)**	**Upper Range (HSV)**
Green	([50,100,100])	([90,255,255])
Yellow	([15,150,150])	([35,255,255])
Red	([160,100,100])	([180,255,255])

Table 1: HSV range for colors

Next step is to apply this color ranges on an HSV image and create a binary mask for red, yellow and green. Binary mask will have 0 assigned to all the pixels which are out of the specified range and 255 (1) elsewhere.



(a) (b)

Fig 17. Binary mask for Red, Green and Yellow: (a) with Red signal as Input Image, (b) with green signal as Input Image

Another object which is not traffic light can also lie in three HSV color range. So, to avoid this noisy outlier we used image morphological operations like opening followed by dilation. An opening is defined as an erosion followed by a dilation using the same structuring element for both operations.

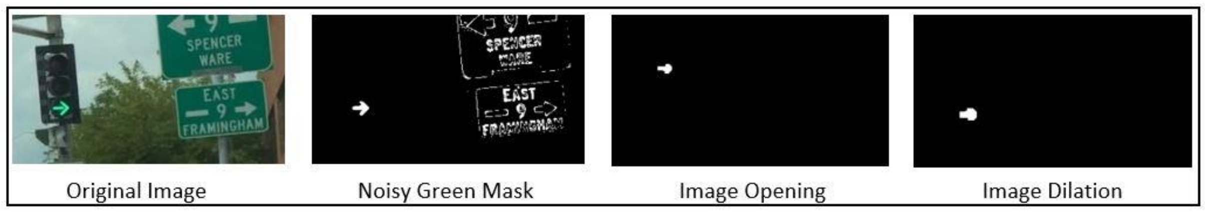


Fig .

Fig 18: Morphological operation on the image to remove outlier in binary mask

As now we know the area of our interest, we can find the exact pixel location where the signal was found using the dilated binary mask. Then compute Hough transform in that region to get exact outer boundary and centre of traffic signal. Now using this data, we drew Hough circles around traffic signal. To recognize it as a red, green or yellow we used dilated binary mask of all the three color and calculated total number of white pixels in each mask. Mask with maximum number of white pixels is declared as the color of the traffic light.

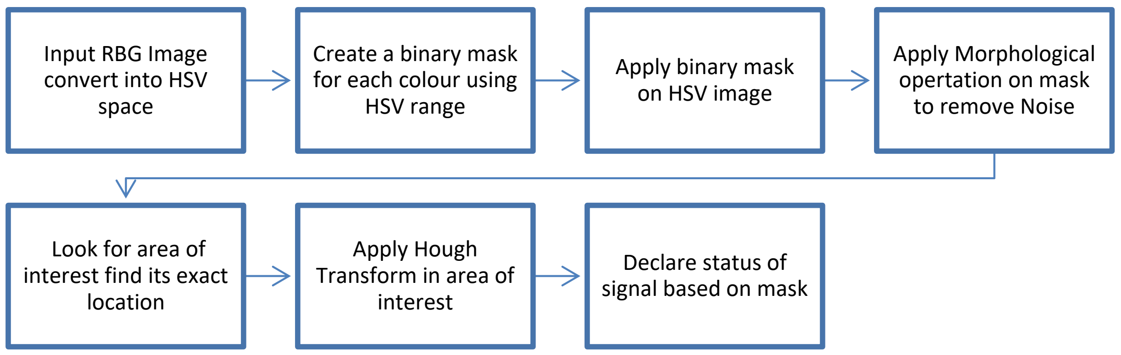

Input RBG Image convert into HSV space
Create a binary mask for each colour using HSV range
Apply binary mask on HSV image
Apply Morphological opertation on mask to remove Noise
Look for area of interest find its exact location
Apply Hough Transform in area of interest
Declare status of signal based on mask

Fig 19: Image Processing and Recognition (Hough Transform and Thresholding on HSV colour space)

This algorithm is invariant to illuminous and performs well in most of the case and was able to detect the traffic signal successfully. Following Image shows three different signal light and its detection using this algorithm.

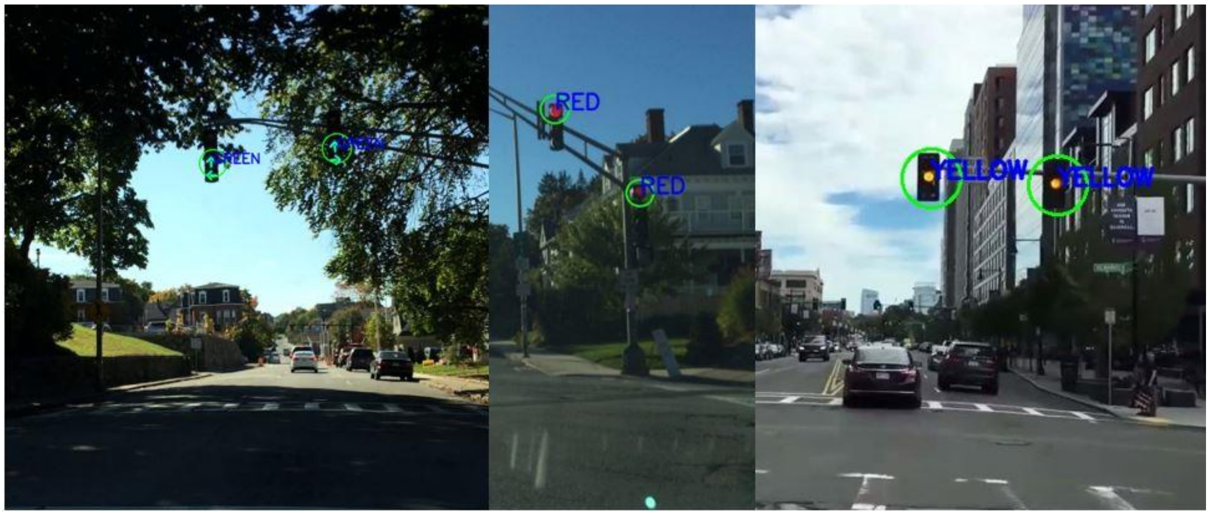


Fig 20: Final output of Hough Transform and Thresholding on HSV image algorithm

However, there are few cases where the noise in the binary mask is large and equivalent to traffic light. This algorithm performance poor in that cases. In the following image traffic signal in the extreme right has a green sign board which lies in the same HSV range as green light. This outlier is much stronger than the signal light below it. So, it is not completely errored by opening operation on mask. Hence the algorithm declares that traffic sign as traffic signal rather than actual signal below it.

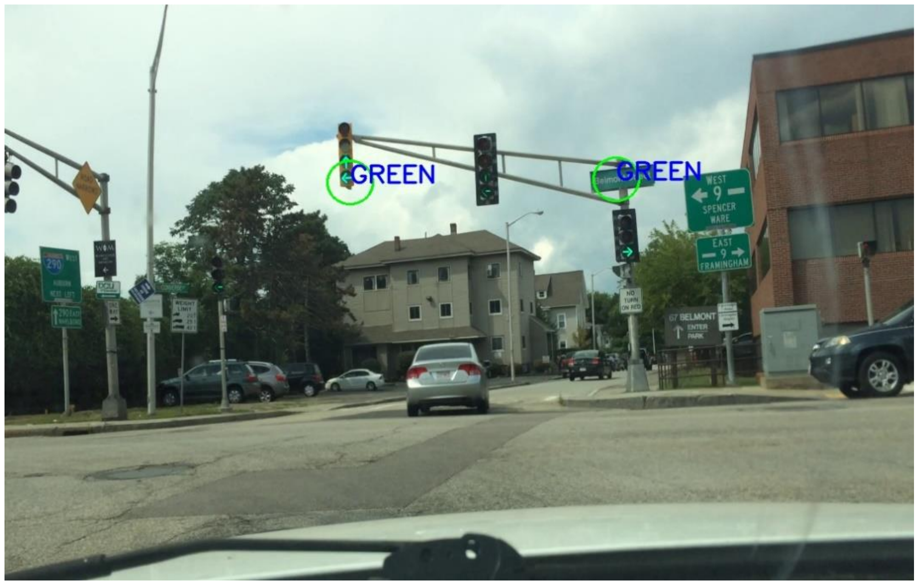
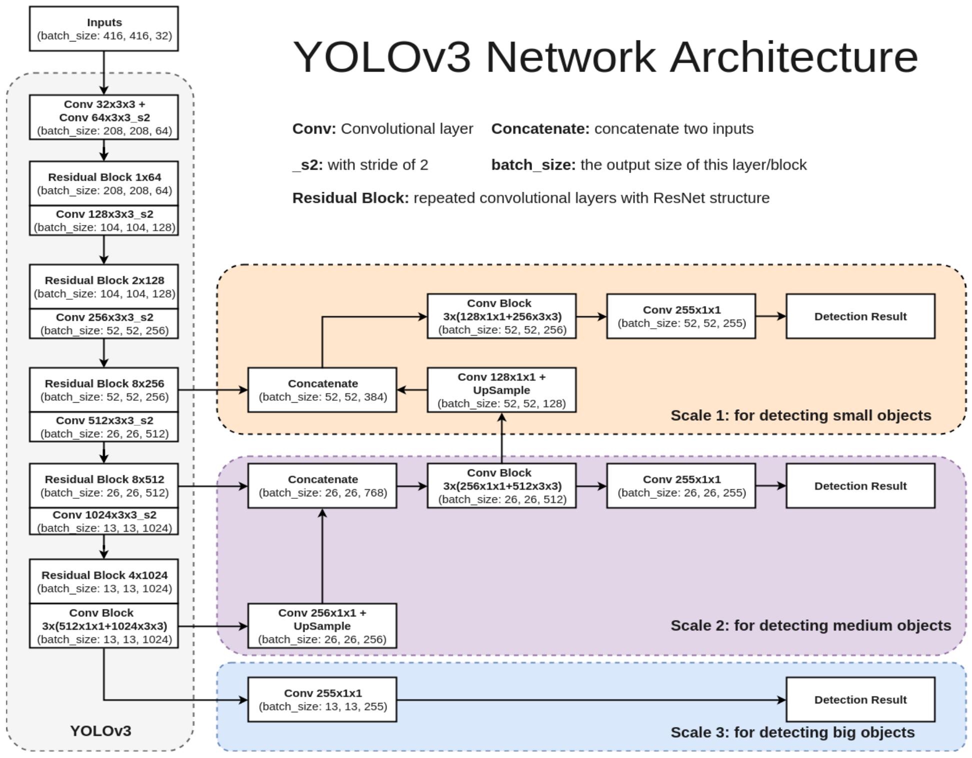


Fig 21: Case where Hough Transform and Thresholding on HSV image algorithm fails

IV. YOLOv3 (You Only Look Once)

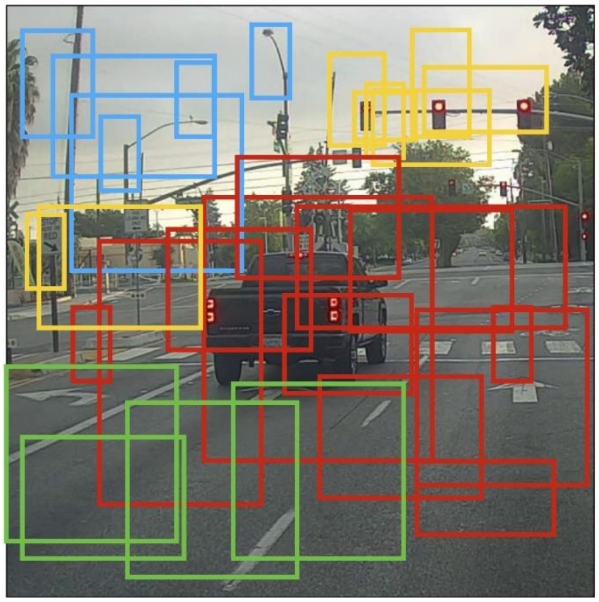
We used deep learning network architecture like YOLOv3 for real time object detection. It is a feature-learning based network that adopts 75 convolutional layers as its most powerful tool. In total there are 106 convolution layers for feature extraction. No fully connected layer is used. This structure makes it possible to deal with images with any sizes. No pooling layers are used, which saves us computing time and keep image features intact. Instead, a convolutional layer with stride two is used to down sample the feature map, passing size-invariant feature forwardly.[7]



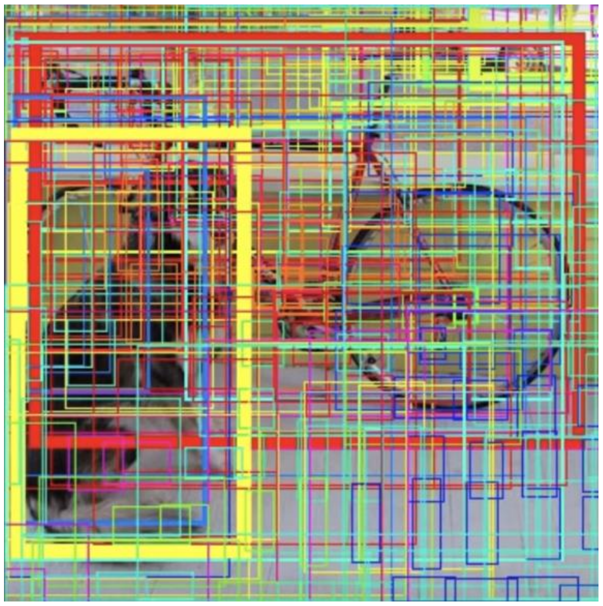
YOLOv3 Network Architecture
Conv: Convolutional layer
Concatenate: concatenate two inputs
_s2: with stride of 2
batch_size: the output size of this layer/block
Residual Block: repeated convolutional layers with ResNet structure
Inputs (batch_size: 416, 416, 32)
Conv 32x3x3 + Conv 64x3x3_s2 (batch_size: 208, 208, 64)
Residual Block 1x64 (batch_size: 208, 208, 64)
Conv 128x3x3_s2 (batch_size: 104, 104, 128)
Residual Block 2x128 (batch_size: 104, 104, 128)
Conv 256x3x3_s2 (batch_size: 52, 52, 256)
Residual Block 8x256 (batch_size: 52, 52, 256)
Conv 512x3x3_s2 (batch_size: 26, 26, 512)
Residual Block 8x512 (batch_size: 26, 26, 512)
Conv 1024x3x3_s2 (batch_size: 13, 13, 1024)
Residual Block 4x1024 (batch_size: 13, 13, 1024)
Conv Block 3x(512x1x1+1024x3x3) (batch_size: 13, 13, 1024)
YOLOv3
Conv Block 3x(128x1x1+256x3x3) (batch_size: 52, 52, 256)
Conv 255x1x1 (batch_size: 52, 52, 255)
Detection Result
Concatenate (batch_size: 52, 52, 384)
Conv 128x1x1 + UpSample (batch_size: 52, 52, 128)
Scale 1: for detecting small objects
Concatenate (batch_size: 26, 26, 768)
Conv Block 3x(256x1x1+512x3x3) (batch_size: 26, 26, 512)
Conv 255x1x1 (batch_size: 26, 26, 255)
Detection Result
Conv 256x1x1 + UpSample (batch_size: 26, 26, 256)
Scale 2: for detecting medium objects
Conv 255x1x1 (batch_size: 13, 13, 255)
Detection Result
Scale 3: for detecting big objects

Fig 22. YOLOv3 Network Architecture.

We tried to train this model using TensorFlow on the traffic light dataset that we used [8]. Data pre-processing involved in training this model was also one of the big challenges that we faced. So, we used a pre-trained YOLOv3 model available publicly on Darknet website [9]. Model weights and config file for the model was pre-trained. Pre-trained model can detect up to 80 different objects (e.g. Traffic signal, car, chair, etc.). We modified the model to just detect traffic signal from 80 different classes. Network creates a bounding box around the features that could probably be a traffic signal in each input image. This bounding box has its own probability which specifies that the object belongs to that class. The width of the bounding box defines how strong is the probability of that feature detected by the network corresponding to a pre-defined class. For example, in the fig. (a) network tries to find the features in the image which belong to traffic light. In fig (b) as the network see all the features of dog and cycle through all 106 CNN layers it calculates the probability and the thickness of bounding box grows.



(a)



(b)

Fig 23. Bounding boxes: (a) intermediate stage (b) After computing probability for each feature

Now the network knows which can be possible positions where the traffic signal in the input image can be. We set a threshold where we specify that bounding box with probability or confidence less than 75% is discarded. Thus, the network returns the center, width and height of the bounding box which has probability greater than 75%. [10] Using this window parameter (x, y, w, h) we create a rectangular window around each of the detecting traffic signals

Decision rule:

R= # of white pixels in red mask

G= # of white pixels in green mask

Y= # of white pixels in yellow mask

Condition	Signal Status
R > G & R > Y	STOP
Y > G & Y > R	GO SLOW
G > Y & G > R	GO

Table 2: signal conditions

Using that information, we create new variables for each of the detected bounding boxes and apply the algorithm discussed in Approach III. (Hough Transform and Thresholding on HSV image space) for detecting and displaying the status of lights of the detected signals as shown in the figure below.

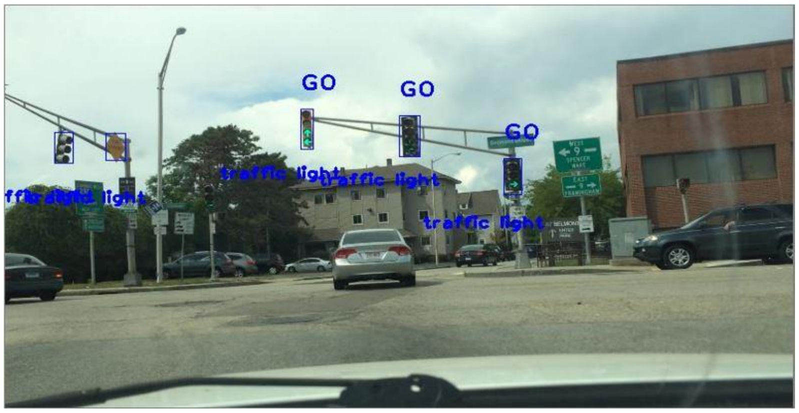


Fig 24: Applying YOLO and detecting the status of the signal

In order to display the status of the signal, we use the decision rule described in table 2. By applying the algorithm discussed in Approach III, we obtain binary mask images for each color of the signal (red, yellow and green). For each of the detected signals in each binary mask images (for R, Y & G) we compute the total number of white pixels which serve as an important variable in our decision rule.

This is evident in fig. 25 in which each of the detected traffic signal is used to apply the red, yellow and green masks and then declares the status of color of the image based upon the decision rule described in table 2.

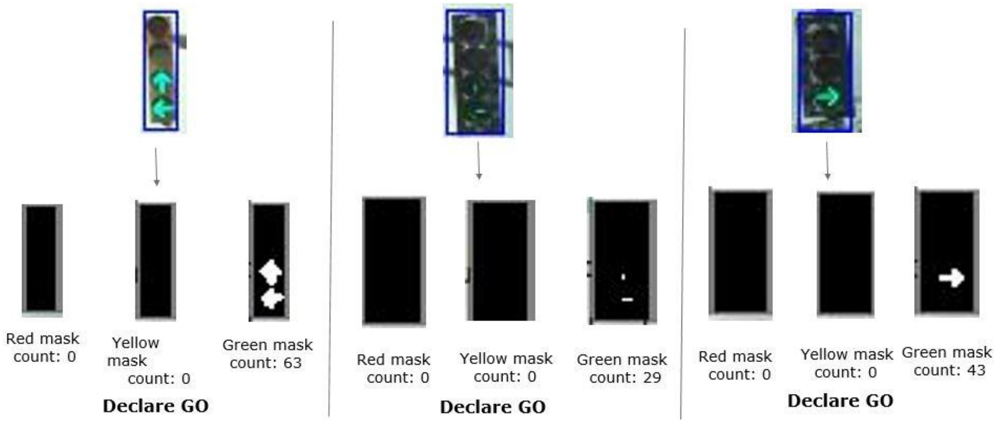


Fig 25: Applying Approach III on each of the detected traffic signals

The Yolov3 algorithm was even tested to run on real time by applying it on the video which is a travel vlog on the routes of Downton Boston [11]. The results of this performance can be observed in the video links added in Appendix 6 of the report.

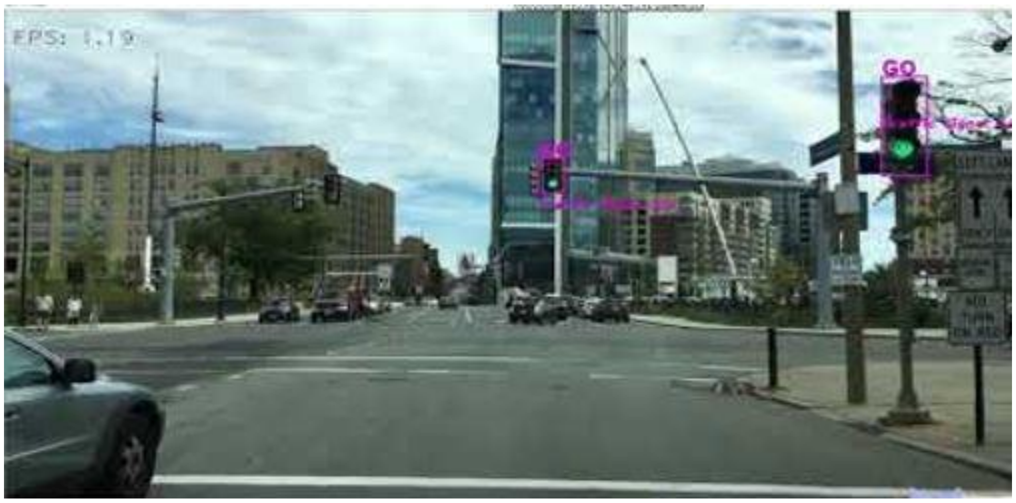


Fig 26. A still from the video of Appendix 6 where Yolov3 was applied on real-time video

The robustness of this algorithm is evident in Fig 27 where Yolov3 was able to detect and declare the object as a traffic signal and even its status under different lighting and illumination conditions. It was not only able to detect but also classified the status of the signal.

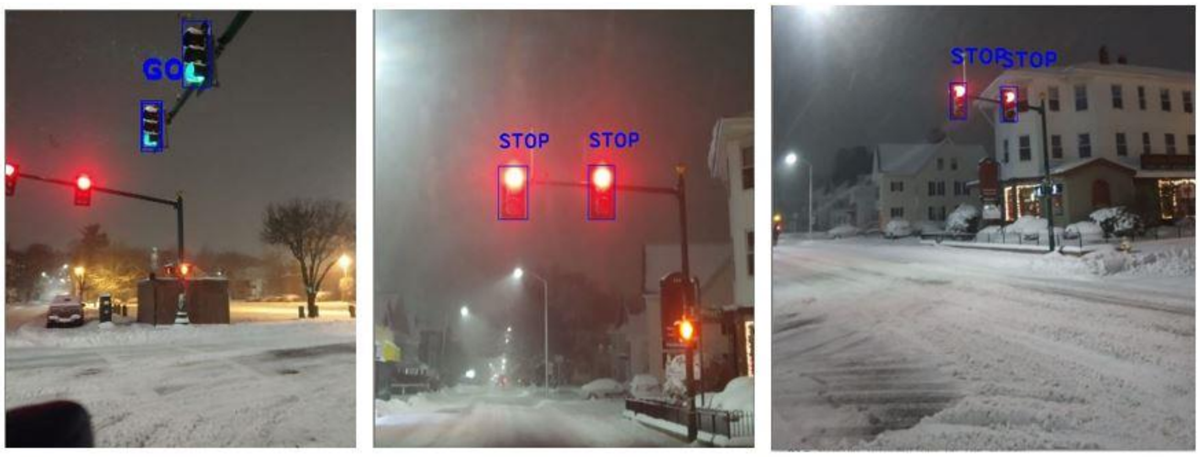


Fig 27: Applying Yolo in night conditions

SECTION III

3.1 Results:

1. SIFT: As a result of the analysis, it was found (Fig 3 and 4) that SIFT feature detection algorithm was able to detect features in two images when the two images are from scene and almost similar perspectives.

   But when there is a change in scenario or perspective (figure 5 and 6), the performance of the algorithm deteriorates, and it loses it accuracy.

2. Top Hat Filter with SURF:
   Upon applying the Top Filter along with the SURF Algorithm, while the traffic signals were being detected, there were multiple outliers. The methodology would detect the brightest spot on the image and would mark it as traffic signal. However, if another object of similar or greater intensity was present in the image, it would also be marked as a traffic signal which gave rise to inaccuracy in the technique. Moreover, with each image having different parameters such as distance of the signal from the camera, a single code can't detect and mark signals in all images as the kernel required for each image would be different and so the methodology is not robust and a different technique was needed for higher accuracy and robustness.

3. Hough Transform and thresholding on colour space:
   From the analysis, it was found that implementation of this algorithm was able to detect a traffic light and classify it into red, green or yellow properly.

   But this algorithm has certain inherent limitations like it can easily get confused if it detects a circular object in the image which has the same HSV color range as that of red, yellow or green traffic light.

4. YOLOv3:
   The YOLOv3 algorithm was able to detect the traffic signals with accuracy in robustness. In addition to this, it was able to detect signals even under different lighting conditions (day and night) and even in real time.

   The inherent drawback of this algorithm is the computation time. Due to its computational complexity, the computation drops to as low as 1.4 fps.

3.2 Conclusion:

As analyzed in the 'Results', each of the four methods implemented have their own advantages and disadvantages associated with them. Out of the four, YOLOv3 performed the best in terms of accuracy and robustness. It was even able to detect traffic signals under different lighting conditions and in real time. Once detected a traffic signal, it was even able to detect the color of the light of traffic light. Thus, in terms of performance, it can be sufficiently concluded that the YOLOv3 algorithm would be the best to be implemented, if need be, on an autonomous vehicle. Also, the current issues faced in implementing this algorithm can be further reduced by implementing it on sophisticated GPUs with current embedded computing boards such as NVIDIA Jetson.

SECTION IV:

Future Scope:

1. SIFT and SURF with SVM
2. (FUTURE SCOPE in TOP HAT FILTER)
3. (FUTURE SCOPE in Hough transform)
4. For this project, a pre-trained model was used for implementing Yolov3. A neural network can be trained to detect traffic signals on a new dataset.
5. The algorithm for Yolov3 can be further extended to detect green signals for left and right directions (Go-Left or Go-Right).

REFERENCES:

1. "Deaths from cars running red lights hit 10-year high, AAA study finds", usatoday.com,2019 [Online]. URL: https://www.usatoday.com/story/money/2019/__08/29/traffic-deaths-red-light-running-aaa-study. [Accessed: 12- Dec- 2019].
2. "Worcester Polytechnic Institute Embedded Computing Laboratory", *Computing.wpi.edu*, 2019. [Online]. Available: http://computing.wpi.edu/dataset.html. [Accessed: 12- Dec- 2019].
3. U. Sinha, "SIFT: Theory and Practice: Introduction - AI Shack", *Aishack.in*, 2019. [Online]. Available: http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/. [Accessed: 12- Dec- 2019].
4. "Top Hat Filter", Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Top-hat_transform [Accessed: 12- Dec- 2019].
5. "Speeded Up Robust Features" ETH Zurich [Online]. Available: https://www.vision.ee.ethz.ch/~surf/eccv06.pdf [Accessed: 12- Dec- 2019].
6. "SURF" Medium [Online]. Available: https://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e [Accessed: 12-Dec-2019].
7. "Scale-invariant feature transform", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform. [Accessed: 12- Dec- 2019].
8. *Cyberailab.com*, 2019. [Online]. Available: https://www.cyberailab.com/home/a-closer-look-at-yolov3. [Accessed: 12- Dec- 2019].
9. J. Redmon, "YOLO: Real-Time Object Detection", *Pjreddie.com*, 2019. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 12- Dec- 2019].
10. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *arXiv.org*, 2019. [Online]. Available: https://arxiv.org/abs/1506.02640. [Accessed: 12- Dec- 2019].
11. "YouTube",Youtube.com,2019.[Online].Available:https://www.youtube.com/watch?v=XLh4 Yt10wb4. [Accessed: 12- Dec- 2019].

Appendix

[1] SIFT Code-Python:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv


from mpl_toolkits import mplot3d
'''Once this submodule is imported, a three-dimensional axes can be created by passing the
keyword projection='3d' to any of the normal axes creation routines'''

def _3Dplot(img1,r,col):

ax = plt.axes(projection='3d')
x = np.linspace(0, col, col)      #define the axis X
y = np.linspace(0, r, r)          #define the axis Y
X, Y = np.meshgrid(x, y)

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, img1, rstride=1, cstride=1,cmap='viridis', edgecolor='none')
ax.set_title('surface');
############# used of rotation of the plot #####################
for angle in range(0, 360):
ax.view_init(90, angle)
#       plt.draw()
#       plt.pause(.001)

##########################################################################################
#######################
def threshold(img,r,col):
for i in range(r):
for j in range(col):
if img[i][j] > 70:
img[i][j] = 255
img_g = cv.GaussianBlur(img,(5,5),4)
return img_g
##########################################################################################
#########################

def row_col(img):
r = img.shape[0]              # stores height of the image
col = img.shape[1]             # stores widhth of the image
return r,col

##########################################################################################
#######################
```

```
def Shift(img,minHessian):

surf = cv.xfeatures2d.SURF_create(minHessian)
surf.setExtended(True)

keypoints, des = surf.detectAndCompute(img,None)
print("number of key points:",np.size(keypoints))

img_keypoints = np.empty((img.shape[0], img.shape[1], 3), dtype=np.uint8)
img_key = cv.drawKeypoints(img, keypoints, img_keypoints)

#img_key = cv.drawKeypoints(img,keypoints,None,(255,0,0),4)
return keypoints,des,img_key


#############################################################################################
########################
def match(img1,img2,des1,des2,kp1,kp2):
# BFMatcher with default params
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Apply ratio test
good = []
for m,n in matches:
if m.distance < 0.75*n.distance:
good.append([m])

# cv2.drawMatchesKnn expects list of lists as matches.
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,outImg = None,flags=2)
return img3

#############################################################################################
########
def main():
img1 = cv.imread(r'C:\Users\vrush\Jupyter Noteboks\CV\12.jpg')
img2 = cv.imread(r'C:\Users\vrush\Jupyter Noteboks\CV\t1.jpg')

r1,col1 = row_col(img1)
r2,col2 = row_col(img2)

#_3Dplot(img1,r,col)

#img1 = threshold(img1,r1,col1)
#img2 = threshold(img2,r2,col2)

kp1,des1,img_s1 = Shift(img1,minHessian = 200)
```

```
kp2,des2,img_s2 = Shift(img2,minHessian = 200)



imgf1 = match(img1,img2,des1,des2,kp1,kp2)
imgm = cv.resize(imgf1,(1080,720))


cv.imshow('match', imgm)
cv.waitKey(0)
cv.destroyWindow('match')
##########################################################################################
#########
if __name__=="__main__":
main()
```

---

[2] Top Hat Filter-MATLAB:

```
clear all;
close all;
clc;

%Reading the image and converting to Gray Scale

IMG1=imread('IMG5.jpg');
figure(); %fig 1
imshow(IMG1)
IMG2=IMG1;
IMG1=rgb2gray(IMG1);
figure(); %fig 2
imshow(IMG1)

%Creating a Window of size 5x5 to apply Top Hat Filter
se = strel('disk',5);
f1 = imadjust(imtophat(IMG1,se));
figure(); %fig 3
imshow(f1)

%Running a for loop to binarize the Image
for i=1:1080
for j=1:1920
if f1(i,j)<90
f1(i,j)=1;
end
end
end

%Applying Gaussin to reduce noise
f1 = imgaussfilt(f1,16);
```

```
figure();

imshow(f1)
%Applying SURF to detect features
points = detectSURFFeatures(f1);

%Displaying Image
imshow(IMG2); hold on;

%Plotting 5 Strongest points
plot(points.selectStrongest(5));
```

[3] Hough Transform and Thresholding on HSV-Python:

```
import numpy as np
import cv2
import os



def HSV_Values(b,g,r):
    c = np.uint8([[[b,g,r]]])
    print(c)
    hsvg = cv2.cvtColor(c,cv2.COLOR_BGR2HSV)
    print (hsvg)
    lower = hsvg[0][0][0] - 10,100,100
    upper = hsvg[0][0][0] + 10,255,255
    print(lower)
    print(upper)
################################################################################
##############
def Track(frame):
    #Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of blue color in HSV
    lower_green = np.array([50, 100, 100])
    upper_green = np.array([90, 255, 255])

    lower_red1 = np.array([0,100,100])
    upper_red1 = np.array([10,255,255])
    lower_red2 = np.array([160,100,100])
    upper_red2 = np.array([180,255,255])

    lower_yellow = np.array([15,150,150])
    upper_yellow = np.array([35,255,255])

    # Threshold the HSV image to get only blue colors
    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
    maskg = cv2.inRange(hsv, lower_green, upper_green)
    masky = cv2.inRange(hsv, lower_yellow, upper_yellow)
    maskr = cv2.add(mask1, mask2)

    '''Here it will take the HSV img value and if the value lies in the range
     it will store that particular index value as 1 (255) else 0(0). Thus we get our mask'''

    # Bitwise-AND mask and original image
    #res = cv2.bitwise_and(frame,frame, mask= maskg)
    #cv2.imshow('res',res)
    '''Here it will take the mask and see which index has 1(255) and it will allow
   that BGR value to display in the image rest (0 0 0)'''
    return maskg,masky,maskr
################################################################################
##############
def Open(mask):
    kernal = np.ones((5,5),np.uint8)
    close = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernal)
    return close

################################################################################
##############
def detect(maskg,masky,maskr):

    r_circles = cv2.HoughCircles(maskr, cv2.HOUGH_GRADIENT, 1, 80,
                               param1=50, param2=10, minRadius=0, maxRadius=30)

    g_circles = cv2.HoughCircles(maskg, cv2.HOUGH_GRADIENT, 1, 60,
                                 param1=50, param2=10, minRadius=0, maxRadius=30)

    y_circles = cv2.HoughCircles(masky, cv2.HOUGH_GRADIENT, 1, 30,
                                 param1=50, param2=5, minRadius=0, maxRadius=30)

    font = cv2.FONT_HERSHEY_SIMPLEX
    size = cimg.shape
    r = 5
    bound = 4.0 / 10


    if r_circles is not None:
        r_circles = np.uint16(np.around(r_circles))

        for i in r_circles[0, :]:
            if i[0] > size[1] or i[1] > size[0]or i[1] > size[0]*bound:
                continue

            h, s = 0.0, 0.0
            for m in range(-r, r):
```

```
            for n in range(-r, r):

                if (i[1]+m) >= size[0] or (i[0]+n) >= size[1]:
                    continue
                h += maskr[i[1]+m, i[0]+n]
                s += 1
        if h / s > 50:
            cv2.circle(cimg, (i[0], i[1]), i[2]+10, (0, 255, 0), 2)
            cv2.circle(maskr, (i[0], i[1]), i[2]+20, (255, 255, 255), 2)
            cv2.putText(cimg,'RED',(i[0], i[1]), font, 1,(255,0,0),2,cv2.LINE_AA)

if g_circles is not None:
    g_circles = np.uint16(np.around(g_circles))

    for i in g_circles[0, :]:
        if i[0] > size[1] or i[1] > size[0] or i[1] > size[0]*bound:
            continue

        h, s = 0.0, 0.0
        for m in range(-r, r):
            for n in range(-r, r):

                if (i[1]+m) >= size[0] or (i[0]+n) >= size[1]:
                    continue
                h += maskg[i[1]+m, i[0]+n]
                s += 1
        if h / s > 100:
            cv2.circle(cimg, (i[0], i[1]), i[2]+20, (0, 255, 0), 2)
            cv2.circle(maskg, (i[0], i[1]), i[2]+10, (255, 255, 255), 2)
            cv2.putText(cimg,'GREEN',(i[0], i[1]), font, 1,(255,0,0),2,cv2.LINE_AA)

if y_circles is not None:
    y_circles = np.uint16(np.around(y_circles))

    for i in y_circles[0, :]:
        if i[0] > size[1] or i[1] > size[0] or i[1] > size[0]*bound:
            continue

        h, s = 0.0, 0.0
        for m in range(-r, r):
            for n in range(-r, r):

                if (i[1]+m) >= size[0] or (i[0]+n) >= size[1]:
                    continue
                h += masky[i[1]+m, i[0]+n]
                s += 1
        if h / s > 50:
            cv2.circle(cimg, (i[0], i[1]), i[2]+20, (0, 255, 0), 2)
            cv2.circle(masky, (i[0], i[1]), i[2]+10, (255, 255, 255), 2)
            cv2.putText(cimg,'YELLOW',(i[0], i[1]), font, 0.7,(255,0,0),2,cv2.LINE_AA)

    return cimg
##########################################################################################
##############
def main():

    cimg1 = cv2.imread(r'H:\Masters Study\Computer Vision\Project\g11.jpg')
    cimg = cv2.resize(cimg1,(1080,720))


    #check HSV ramge Values for a particular colour
    HSV_Values(0,255,0)


    #get the masked images
    maskg,masky,maskr = Track(cimg)



    kernel = np.ones((2,2),np.uint8)
    opening_g = cv2.morphologyEx(maskg,cv2.MORPH_OPEN,kernel, iterations = 4)
    opening_y = cv2.morphologyEx(masky,cv2.MORPH_OPEN,kernel, iterations = 1)
    opening_r = cv2.morphologyEx(maskr,cv2.MORPH_OPEN,kernel, iterations = 1)

    # sure background area
    sure_bg_g = cv2.dilate(opening_g,kernel,iterations=3)
    sure_bg_y = cv2.dilate(opening_y,kernel,iterations=3)
    sure_bg_r = cv2.dilate(opening_r,kernel,iterations=3)

    cimg = detect(sure_bg_g,sure_bg_y,sure_bg_r)

    cv2.imshow('maskg', opening_g)
    cv2.imshow('maskr', sure_bg_g)
    cv2.imshow('masky', maskg)

    cv2.imshow('detected results', cimg)
    #cv2.imwrite(os.path.join('H:\Masters Study\Computer Vision\Project', 'green.jpg'),cimg)


    cv2.waitKey(0)
    cv2.destroyAllWindows()

##########################################################################################
##############
if __name__=="__main__":
    main()
```

[4] YOLO Object Detection-Python:

```
# YOLOv3 on images

import cv2
import numpy as np

# Function of code: Thresholding in HSV Color space to detect the status of the signal as 'Red', 'Green'
or 'Yellow'

def Status(X1):
   Hsv= cv2.cvtColor(X1, cv2.COLOR_BGR2HSV)
   hsv.append(Hsv)
   #  variable to store total # of white pixels in binary mask images of each color
   countr=0
   countg=0
   county=0
   # creating masks for each color of the signal
   MASK1= cv2.inRange(Hsv,L_red1,u_red1)
   mask1.append(MASK1)
   MASK2= cv2.inRange(Hsv,L_red2,u_red2)
   mask2.append(MASK2)
   MASKG= cv2.inRange(Hsv, l_green,u_green)
   maskg.append(MASKG)
   MASKy= cv2.inRange(Hsv, l_yellow, u_yellow)
   masky.append(MASKy)
   MASKr= cv2.add(MASK1, MASK2)
   maskr.append(MASKr)
   dime1=np.shape(MASKr)
   dime.append(dime1)
   (H1,W1)= dime1

   # computing the total number of white pixels in binary mask images of each color
   for j in range(H1):
      for k in range(W1):
         if MASKr[j][k]==255:
            countr+=1
         if MASKG[j][k]==255:
            countg+=1
         if MASKy[j][k]==255:
            county+=1

   L= [0, countr, county, countg]
   final_color= L.index(max(L))
   # declaring the status of the signal on the condition of max(R, G, Y)
   if final_color==1:
        status="STOP"
    elif final_color==2:
        status="SLOW DOWN"
    elif final_color==3:
        status="GO"
    else:
        status=""
    return status



# Load Yolo
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes =[]
txt= "image"
wind_name=[]



# load classes
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# apply the NN layers
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]


# create a new random color for each new box to be constructed
colors = np.random.uniform(0, 255, size=(len(classes), 3))
color=(255,0,0)

# Loading image
img = cv2.imread("C:\\Users\\Kavit\\OneDrive\\Desktop\\CV Project]\\YOLO\\YOLO img\\Wpi Signal
4.jpeg")
img = cv2.resize(img, None, fx=0.5, fy=0.5)
height, width, channels = img.shape

# Detecting objects

# Arguments of cv2.dnn.blobFromImage= (imgae, scale_factor, (size), (mean subtraction), invert blue
with red {True/False}, crop yes or no)
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)


net.setInput(blob) # giving blob as input to the net
outs = net.forward(output_layers)
```

```
# Showing informations on the screen
class_ids = []
confidences = []
boxes = []

# computing the window parameters of each object of given classes detected
for out in outs:
  for detection in out:

      scores = detection[5:]

      class_id = np.argmax(scores)

      confidence = scores[class_id]
      if confidence > 0.5 and class_id ==9:

        center_x = int(detection[0] * width)
        center_y = int(detection[1] * height)
        w = int(detection[2] * width)
        h = int(detection[3] * height)
        #cv2.circle(img, (center_x, center_y), 10, (0, 255, 0), 2)

        # Rectangle coordinates
        x = int(center_x - w / 2)
        y = int(center_y - h / 2)

        boxes.append([x, y, w, h])
        confidences.append(float(confidence))

        # appending the class_ids for each iteration of every detected class elements
        class_ids.append(class_id)

# applying non-maximal suppression
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
# non-maximal supression removes double identifications for same objects

# the limits for the HSV color spaces for three colors
L_red1= np.array([0,100,100])
u_red1=np.array([10,255,255])
L_red2=np.array([160,100,100])
u_red2=np.array([180,255,255])
l_green=np.array([50,100,100])
u_green=np.array([90,255,255])
l_yellow=np.array([15,150,150])
u_yellow=np.array([35,255,255])
mask1=[]
mask2=[]
maskr=[]
```

```
masky=[]
maskg=[]
dime=[]
l_red2=np.array
X=[]
Y=[]
X1=[]
blur=[]
hsv=[]
font = cv2.FONT_HERSHEY_COMPLEX
img_new= img.copy()
#print(indexes)
font = cv2.FONT_HERSHEY_PLAIN

# this loop adds the name of the detected class and also the status of color of the signal
for i in range(len(boxes)):
    if i < len(indexes):
    #if str(classes[class_ids[i]]) =='traffic signal'
        #print(i)
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        print (label)
        X.append(x)
        Y.append(y)

        X1.append(img_new[Y[i]: Y[i]+h, X[i]:X[i] + w])


        #color = colors[1]

        name= "signal" + str(i+1)
        wind_name.append(name)

        cv2.rectangle(img, (x, y), (x + w, y + h), color, 1)

        status= Status(X1[i])
        print(i)
        print(status)
        cv2.putText(img, status, (x, y -50 + 30), font, 1.6, color, 2)

# display the final computed image
cv2.imshow("Image", img)



cv2.waitKey(0)
cv2.destroyAllWindows()
```

[5] YOLO Real Time Detection-Python:

```
import cv2
import numpy as np
import time

# Load Yolo
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
   classes = [line.strip() for line in f.readlines()]
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Loading image
cap = cv2.VideoCapture('C:\\Users\\Kavit\\OneDrive\\Desktop\\Detectors\\Test1_Trim.mp4')

font = cv2.FONT_HERSHEY_PLAIN
starting_time = time.time()
frame_id = 0
L_red1= np.array([0,100,100])
u_red1=np.array([10,255,255])
L_red2=np.array([160,100,100])
u_red2=np.array([180,255,255])
l_green=np.array([50,100,100])
u_green=np.array([90,255,255])
l_yellow=np.array([15,150,150])
u_yellow=np.array([35,255,255])
while True:
   _, frame = cap.read()
   frame_id += 1

   height, width, channels = frame.shape

   # Detecting objects
   blob = cv2.dnn.blobFromImage(frame, 0.00392, (416,416), (0, 0, 0), True, crop=False)

   net.setInput(blob)
   outs = net.forward(output_layers)

   # Showing informations on the screen
   class_ids = []
   confidences = []
   boxes = []
```

```
X1=[]
X=[]
Y=[]
hsv=[]
mask1=[]
mask2=[]
maskr=[]
masky=[]
maskg=[]
dime=[]
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.25 and class_id==9:
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)


indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.8, 0.6)
frame_new= frame.copy()
print(len(boxes))
for i in range(len(boxes)):
    #print(boxes[i])
    if i in indexes:
        print ('i= ', i)
    #if str(classes[class_ids[i]]) == 'traffic signal':
        x, y, w, h = boxes[i]
        #label = str(classes[class_ids[i]])
        label= 'Traffic Signal'
        countr=0
        countg=0
        county=0
        confidence = confidences[i]
        #print(label)
        X.append(x)
        Y.append(y)
        color = colors[class_ids[i]]
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        cv2.putText(frame, label + " " + str(round(confidence, 2)), (x, y + 50+25), font, 2, color, 2)

        X1.append(frame_new[Y[i]:Y[i] + h, X[i]:X[i] + w])

        Hsv= cv2.cvtColor(X1[i], cv2.COLOR_BGR2HSV)
        hsv.append(Hsv)
        MASK1= cv2.inRange(Hsv,L_red1,u_red1)
        mask1.append(MASK1)
        MASK2= cv2.inRange(Hsv,L_red2,u_red1)
        mask2.append(MASK2)
        MASKG= cv2.inRange(Hsv, l_green,u_green)
        maskg.append(MASKG)
        MASKy= cv2.inRange(Hsv, l_yellow, u_yellow)
        masky.append(MASKy)
        MASKr= cv2.add(MASK1, MASK2)
        maskr.append(MASKr)
        dime1=np.shape(maskr[i])
        dime.append(dime1)
        (H1,W1)= dime1
        for j in range(H1):
           for k in range(W1):
              if MASKr[j][k]==255:
                 countr+=1
              if MASKG[j][k]==255:
                 countg+=1
              if MASKy[j][k]==255:
                 county+=1

        L= [0, countr, county, countg]
        final_color= L.index(max(L))

        if final_color==1:
           status="STOP"
        elif final_color==2:
           status="SLOW DOWN"
        elif final_color==3:
           status="GO"
        else:
           status=""
        cv2.putText(frame, status + " " , (x, y -27 +25), font, 2, color, 3)
        #for j in range (H1):
           #print (j)
    elapsed_time = time.time() - starting_time
    fps = frame_id / elapsed_time
    cv2.putText(frame, "FPS: " + str(round(fps, 2)), (10, 50), font, 2, (0, 0, 0), 1)
    cv2.imshow("Image", frame)
    #cv2.imshow("red", maskr[1])
    #cv2.imshow("yellow", masky[1])
    #cv2.imshow("green", maskg[1])
    #cv2.imshow("n img", y1)
    key = cv2.waitKey(1)
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

[6] "YouTube", Youtube.com, 2019. [Online]. Available:
https://www.youtube.com/watch?v=6thHwfyPkGg. [Accessed: 12- Dec- 2019].

[7] "YouTube", Youtube.com, 2019. [Online]. Available:
https://www.youtube.com/watch?v=PsYKZ-kU1rA [Accessed: 12- Dec- 2019].