1. Introduction & Motivation

In the modern world, as conventional automobile industries inch towards self-driven cars, the need for computer vision engineers is rising rapidly. There is a constant need to make a car completely autonomous. Thus, being Computer Science students, we thought it would be a good idea to take few steps into the world of Computer Vision and its applications in the modern world. We couldn't think of a better project than one that included a self-driving car. Hence, we thought of applying concepts that we learned from our Computer Vision course to this project.

Our aim is to detect lanes on the road and speed limit signs from the view a camera sensor of an autonomous driving car.

2. Methods

2.1.1 Color Extraction and Transformation

From our original color image, we need to extract only white and yellow colors. The reason we do this is to only detect lanes which are yellow or white in color (depends on the geographical region of the road). For the separation of colors, we have implemented a function which only selects *yellow* and *white* channels from our color image.

2.1.2 Edge detection using Sobel operator

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:

The x-coordinate is defined here as increasing in the "right"-direction, and the y-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{G_x^2 + G_y^2}$$

Using this information, we can also calculate the gradient's direction:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

where, for example, Θ is 0 for a vertical edge which is lighter on the right side.

2.1.3 Region of Interest Selection

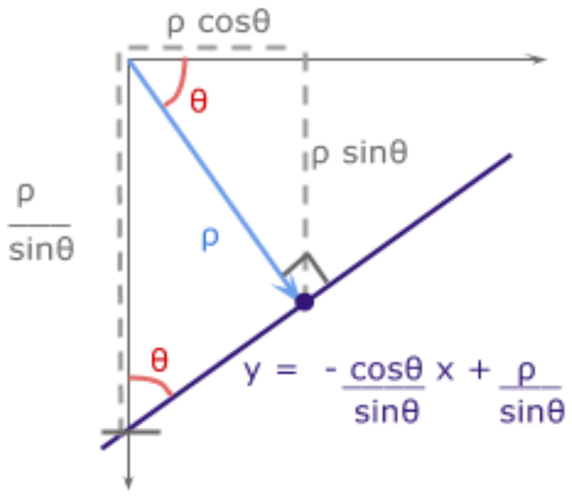
A Region of Interest is a region of the image on which we want to work. Generally, this is selected by manipulating the number of rows and columns of the original image.

2.1.4 Hough Transform for line detection

Algorithm:

1) **Edge detection** (E.g. using canny, sobel, adaptive thresholding): The resultant binary/grey image will have 0s indicating non-edges and 1s or above indicating edges. This is our input image.
2) **Rho range and Theta range creation**: ρ ranges from -max_dist to max_dist where max_dist is the diagonal length of the input image. θ ranges from −90∘ to 90∘. You can have more or less bins in the ranges to tradeoff accuracy, space and speed.
3) **Hough accumulator** of θ vs ρ. It is a 2D array with the number of rows equal to the number of ρ values and the number of columns equal to the number of θ values.
4) **Voting in the accumulator**: For each edge point and for each θ value, find the nearest ρ value and increment that index in the accumulator. Each element tells how many points/pixels contributed "votes" for potential line candidates with parameters (ρ, θ).
5) **Peak finding**: Local maxima in the accumulator indicates the parameters of the most prominent lines in the input image. Peaks can be found most easily by applying a threshold or a relative threshold (values equal to or greater than some fixed percentage of the global maximum value).

Deriving rho: ρ = x cos θ + y sin θ



With basic trigonometry, we know that for right-angled triangles,

$$sin\theta = \frac{opposite}{hypotenuse} \text{ and } cos\theta = \frac{adjacent}{hypotenuse}$$

We want to convert the cartesian form y = mx + b with parameters (m, b) to polar form with parameters (ρ, θ).

The line from the origin with distance ρ has a gradient of $\frac{sin\theta}{cos\theta}$. The line of interest, which is perpendicular to it, will have a negative reciprocal gradient value of $-\frac{cos\theta}{sin\theta}$. For b, the y-intercept of the line of interest, $sin\theta = \frac{\rho}{b}$.

With $m = \frac{-cos\theta}{sin\theta}$ and $b = \frac{\rho}{sin\theta}$, we get $\rho = xcos\theta + ysin\theta$.

2.1.5 Template Matching

Template matching is a technique for finding areas of an image that match (are similar) to a template image (patch).

We need two primary components:

a. **Source image (I):** The image in which we expect to find a match to the template image.

b. **Template image (T):** The patch image which will be searched in the source image.

Our goal is to detect the highest matching area; The brightest locations indicate the highest matches. To identify the matching area, we have to *compare* the template image against the source image by sliding it.

Initially we select a region of interest in both the source image and template image. The region of interest in the source image is the right side of the image as speed signs are usually on the right side of the image. And the region of selection in the template image is the area where the word 'limit' is present.

We then only work on the region selected that is by cropping the image to only the region that is selected. This is done to reduce the computations by working only on the region selected.

We then apply convolution by using the cropped template image as the kernel by sliding the template over the image. By **sliding**, we mean moving the patch one pixel at a time (left to right, up to down).

After we get the matched image, we scale the intensity values to be in the range between 0 and 1. Then, we apply thresholding so as to remove any additional noise and get only the point where the template has matched properly.

We then map this matched pixel by calculating its co-ordinates in the cropped image and translating it back so that we get the co-ordinates of the matched point in the original image.

We draw a circle around this point to show the detected speed sign on the original image.

3. Implementation details

The two main tasks that were to be performed were: detection of lanes and speed limit signs on the road.

Pipeline (image processing steps) for performing lane detection:

i) Color space transformation:

Extracting *white* and *yellow* channels from the color image. This was done because the lanes on the road are generally of two colors: white & yellow. The white lane is a series of alternating dots and yellow lines are solid. Images are loaded in RGB space from which we can easily extract *white* and *yellow* lines from the image. However, whenever there is a shadow on the road, these lanes are not clearly visible. To overcome this problem, we chose to use the HLS (Hue, Lightness, Saturation) space transformation.

ii) Edge Detection:

The next step is to find the edges in the image. Thus, we have used Sobel operator for detecting edges from the image. We calculated the x-gradient, y-gradient, tan-1 and the thresholded edge map.

iii) Region of Interest Selection:

To just detect the lanes and not the remaining environment, we decided to define a region of interest. This region of interest looks like a trapezium where we expect to find lanes.

iv) Hough transform:

On the given region of interest, we calculated the Hough transform to define the lanes in the image.

Pipeline (image processing steps) for performing detection of speed limit signs:

i) Region of Interest Selection:

As in the above case, we defined a region of interest in the source image where we will search the given template. In this case, the template is pre-loaded image of a speed limit sign found on the United States roads.

ii) Template Matching:

On the selected region of interest, we search for the template by using template matching algorithm explained above. This will give us the correlation image where on all the possible matched locations we will see bright spots.

iii) Thresholding of the bright spot:

To find the spot we are looking for, we performed thresholding so that the resulting image has only one bright spot.

iv) Mapping to image space:

Since, all the above steps were done on the selected region of interest, we need to find the location of the bright spot in the original image and highlight it.

For performing both the operations on the same image, we first passed our image thorough lane detection system and then passed that lane detected image through template matching system to detect speed limit signs.

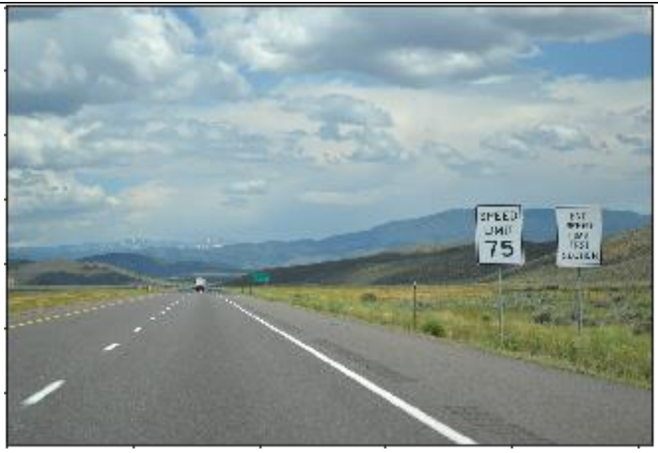
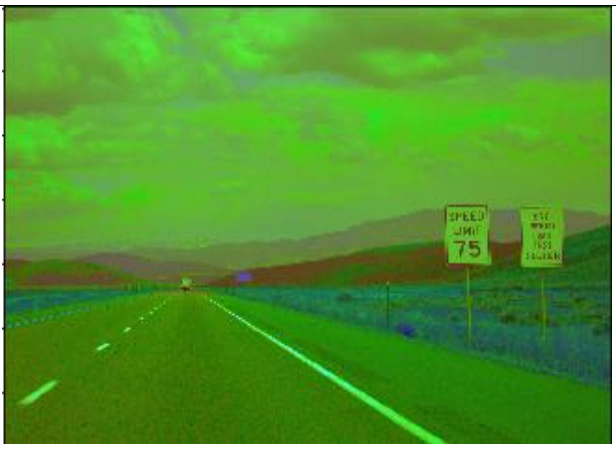
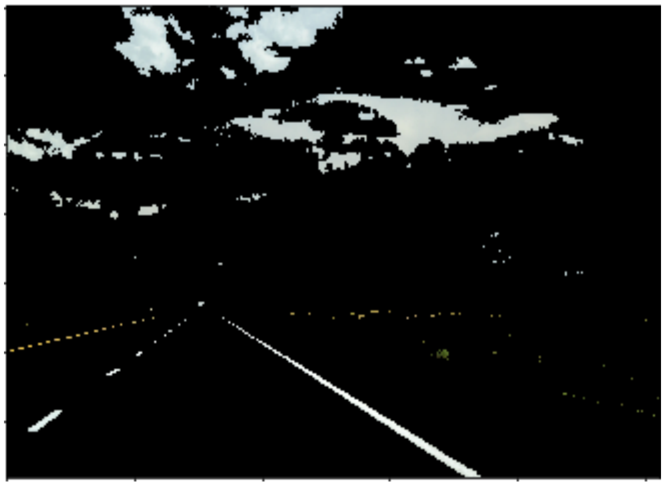
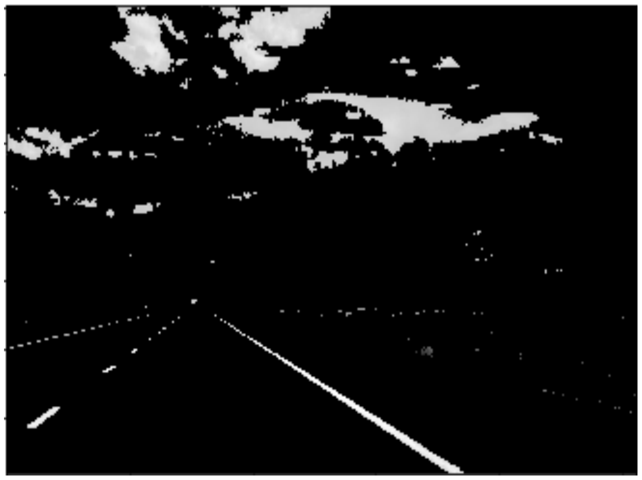
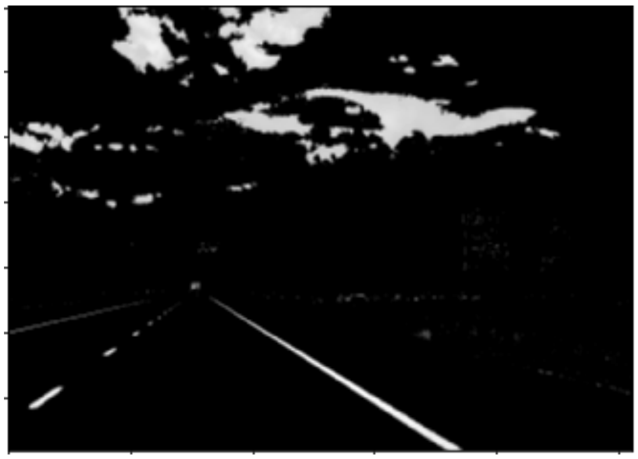
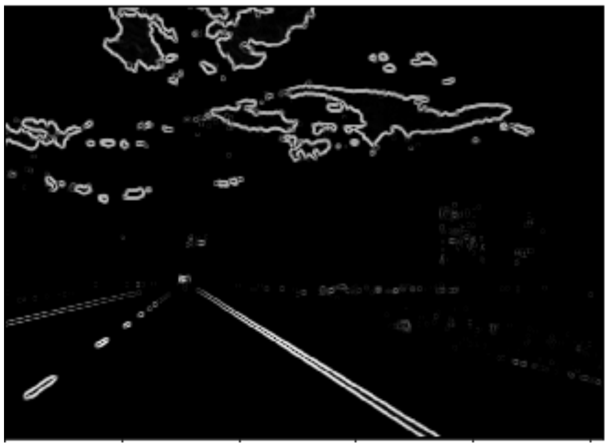
4. Challenges Faced

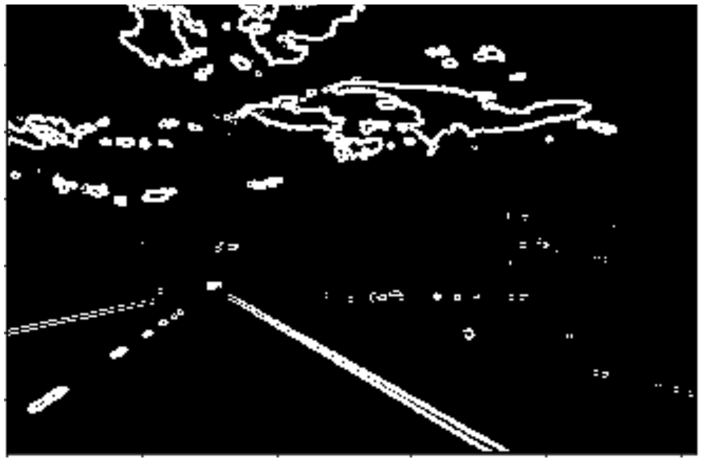
We faced several challenges while implementing this:

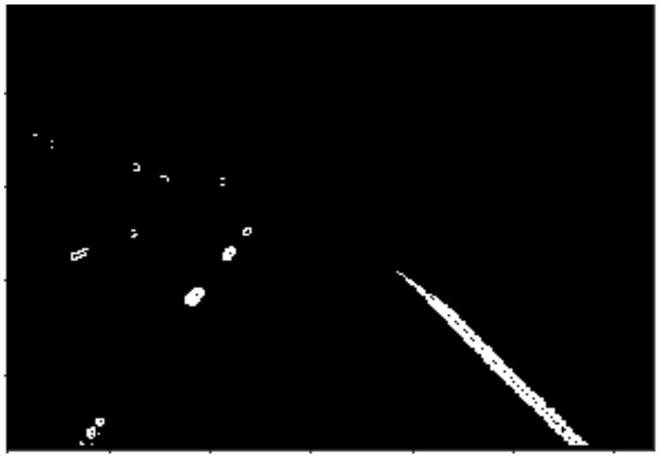
i) First and foremost, the computation time for template matching is pretty high. To overcome that we defined a region of interest.
ii) Finding the appropriate test images and videos was also a challenge. Since we are not using any OpenCV functions, we had to make several assumptions to make the scene simple. So, finding test dataset was little bit difficult.
iii) Since each image was in different scene setting and performing object detection via thresholding requires hit and trial methods. So, to find that optimal setting which can work on most of the images, we limited ourselves to only working on bright day scenes.
iv) For lane detection, whenever there was a tree on the side of road which casted a shadow on the lanes, our initial system couldn't detect the lanes. So, we had to perform color space transformation to get a better image and then converted that color spaced transformed image to gray scale to work further.
v) For detecting road signs, there was the issue of orientation mismatch of templates.
vi) Computation time again came in factor when we applied this system on videos. It took really long time to generate results for few seconds of video.

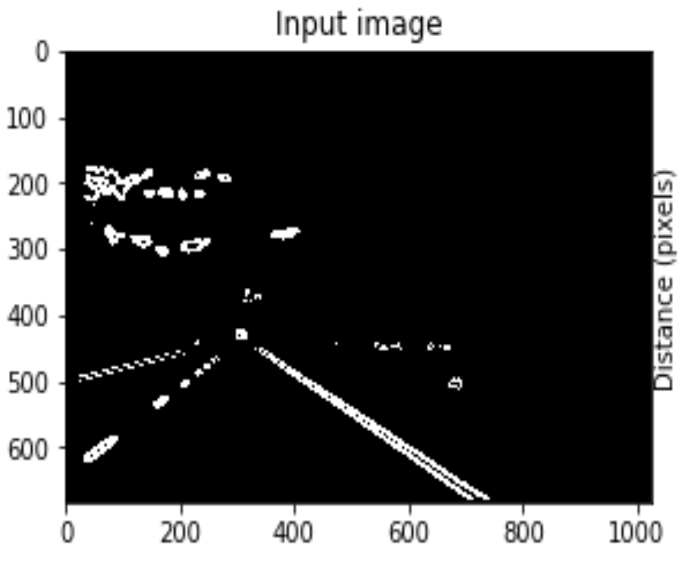
5. Critical Discussion & Results

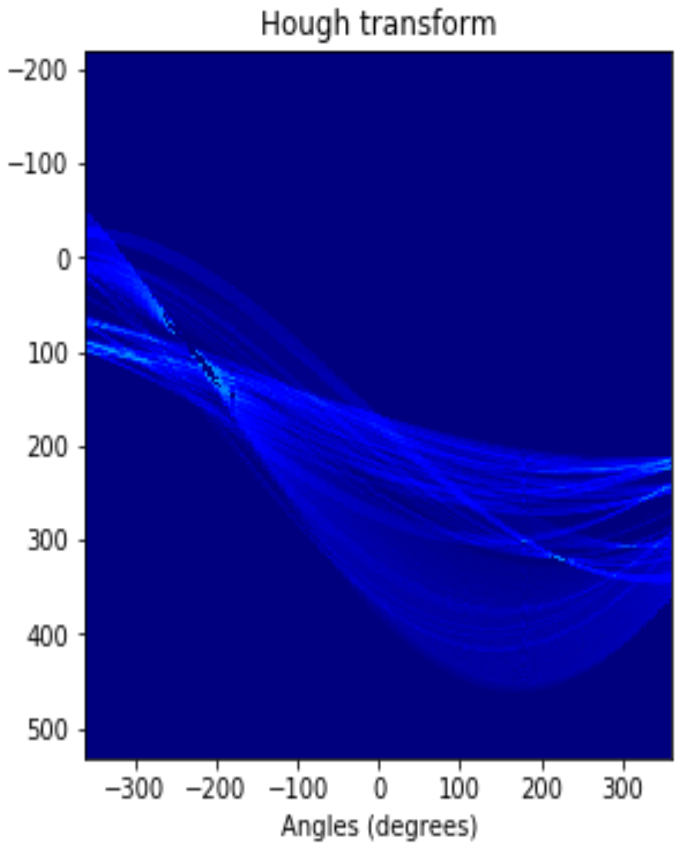
Below are the step by step results obtained on an image.

 **Original Image**	 **Image in HLS space**
 **White Yellow Selected image**	 **Gray Scale image**
 **Smoothed Image**	 **Edge Map**

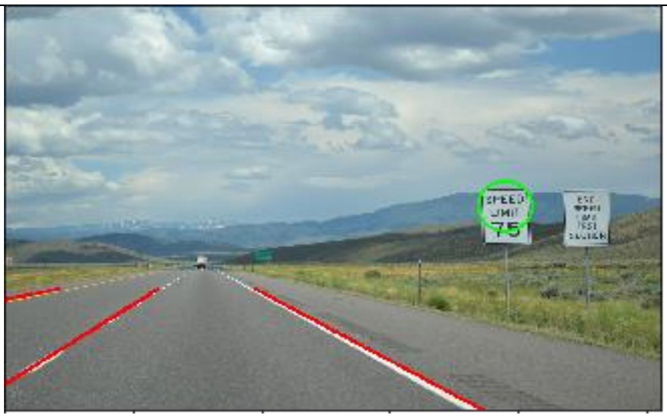

Thresholded edge map


Region of Interest Selection





Original Image



Final lane detected, and sign detected image

The template which we searched for:

SPEED
LIMIT

<u>Design Decisions:</u>

In the above steps, first step was to perform color space transformation. We tried two HLS (Hue, Light and Saturation) and HSV (Hue, Saturation and Value) transformations. While both gave pretty good pictures but for the region of images where there was some shade on the road, HLS performed better. So, we chose that.

We smooth every image with a 7*7 linear filter before applying edge detection so that extra noise can be removed from the image. Also, we went with the Sobel operator for edge detection. One possibility was to use Canny edge detection. But the results with the Sobel operator were quite good. So, we chose to stick with Sobel operator.

Another decision was to select the region of interest. So, for this we made following assumptions:

For lane detection, we assumed driver's view range is limited and drew a trapezium around his frontal view. We also intuitively thought that since car is moving and for the range of trapezium lanes will always fall in straight line. This way we compensated for curves on the road. And we can see from the video results that this assumption works well.

Shortcoming: - In case, there is a steep curve on the road, then this assumption will fail and we will detect the wrong lanes.

Results of the run of this system on some other images: