ROAD LANE DETECTION AND OBSTACLE TRACKING

CREATIVE AND INNOVATIVE PROJECT

TEAM 4 -

MAHJABEEN A

POOJA S

BHARATH M

OBJECTIVE

- To develop a vision-based real-time lane detection and tracking using PPHT, where different lighting conditions, and different road types, i.e., straight and curved, are considered.
- Using real world dataset to evaluate the lane boundary detection rate and the time complexity.
- Using YOLO algorithm to detect obstacles on the road and provide a robust object tracking system along with road lane detection.

INTRODUCTION

- Lane detection and tracking is an important component of the Advanced Driver Assistance System.
- Lane line detection and identification has become a basic and necessary functional module in the field of vehicle safety and intelligent vehicle navigation.
- The performance of various lane detection methods drops in the case of different lighting conditions
- Effective lane detection is challenging due to different types of roads as well as occlusion caused by various obstacles.
- Obstacle detection also plays a major role in avoiding accidents.

INTRODUCTION

CREATIVE AND INNOVATIVE PROJECT

The method proposed is as follows:

- In the pre-processing stage, images are transformed to grayscale image and smoothed. Edge detection is applied using Sobel filter. Finally, Otsu's thresholding is applied.
- A simple Adaptive Region of Interest (AROI) is used to reduce the computational complexity.
- PPHT is used as it reduces false positive rate.
- Kalman filter is used to track both borders of each lane markings. This additional lane-tracking step increases the probability to detect lane markings in poor conditions and improves efficiency.
- YOLO algorithm is then used to detect obstacles like vehicles, pedestrians, animals etc.

LITERATURE SURVEY

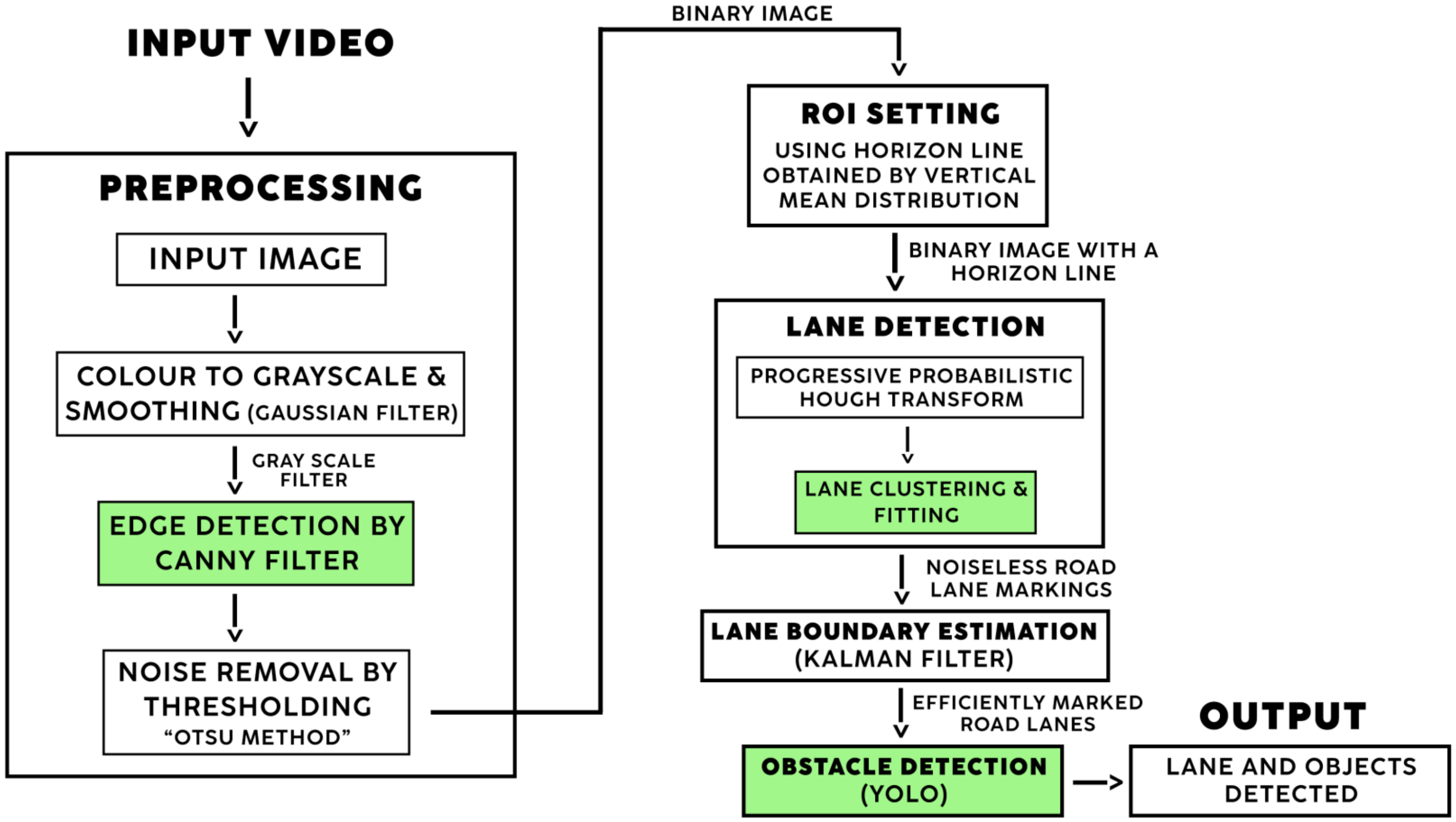
	AUTHOR	METHODOLOGY	ADVANTAGES	DISADVANTAGES
1	Abdelh mid Mamme, Guangqian Lu et al	Uses PPHT and MSER for lane detection.	PPHT returns two end-points of the detected line markings.	MSER is more computationally expensive.
2	Yassin Kortli, Mehrez, J. Subash Chandra Bose et al	Uses RGB to grayscale image conversion, and a Gaussian filter.	Performs well in bad weather conditions.	Faces issue in case of blur lane marks and bent road surfaces.
3	Mohamed Aly	Inverse Perspective Mapping (IPM) is used as the algorithm.	Works at high rates of 50 Hz.	Presence of obstacles on road reduces accuracy.
4	Ziqiang Sun	Yellow & white lane lines are processed separately in HSV space and then combined.	Robust and adaptive.	High no. of false positives.

LITERATURE SURVEY

5	Shahanaz Syed, Rudra Hota et al	Weighted regression to fit a curve is used here.	Better estimation of lane markings.	Does not work well under heavy traffic conditions and when the lane changes are fast.
6	Hiroyuki Komori, Kazunori Onoguchi	Detects various types of lane markings and road boundaries.	Detects blurred lane markings and shoulders with complicated shapes.	Cannot be evaluated using public datasets.
7	Ping Wei, Linhai Xu, Nanning Zheng	Incorporates prior spatial-temporal knowledge with lane appearance features.	Performs well under various challenging weather conditions.	May produce false results in some special traffic conditions.
8	Maurício Braga de Paula, Cláudio Rosito Jung	Lane boundaries are detected using a linear–parabolic lane model.	Consistent results for video sequences acquired with different devices.	Produces classification errors due to deviations of the lane tracker from the actual lane boundaries

CREATIVE AND INNOVATIVE PROJECT

ARCHITECTURAL DIAGRAM



CREATIVE AND INNOVATIVE PROJECT

DETAILED MODULE DESIGN

MODULE 1 - PRE-PROCESSING

1. Converting video to image sequences.
2. Converting an RGB image to grayscale images and smoothing. Smoothing is used to reduce noise or to produce a less pixelated image.
3. Edge detection is then performed using Sobel filter. It works by calculating the gradient of image intensity at each pixel within the image. It finds the direction of the largest increase from light to dark and the rate of change in that direction.
4. Image thresholding is done to remove noises using "Otsu method".

INPUT : Road Lane videos
OUTPUT : Binary Image

CREATIVE AND INNOVATIVE PROJECT

DETAILED MODULE DESIGN

MODULE 1 - PRE-PROCESSING

ALGORITHM -

Conversion from RGB to Grayscale :

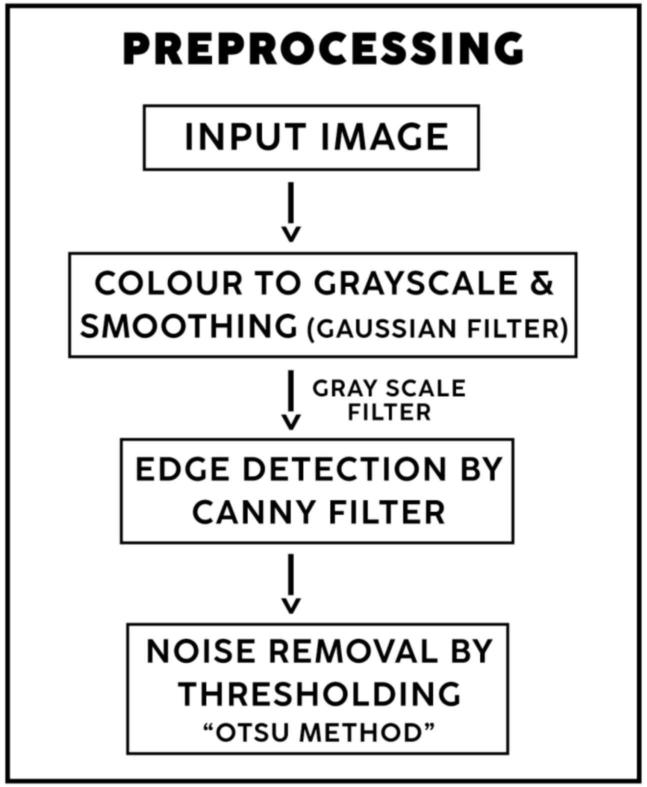
```
gray_img = cv.cvtColor(img,
cv.COLOR_RGB2GRAY)
blur = cv.GaussianBlur(gray_img,
(5, 5), 0)
```

Edge detection :

```
edge_det = cv.Canny(gray_img, 50,
100, apertureSize=3)
```

Otsu thresholding :

```
ret, thresh1  = cv.threshold(edge, 120, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
```



CREATIVE AND INNOVATIVE PROJECT

DETAILED MODULE DESIGN

CREATIVE AND INNOVATIVE PROJECT

MODULE 2 - ROI SETTING

1. An AROI is established using a horizon line that effectively removes noisy line segments and reduces the computational complexity.
2. The horizon line is used to divide the scene into road region and sky region.

INPUT : Binary Image
OUTPUT : Binary Image with a horizon line

ROI SETTING

USING HORIZON LINE OBTAINED BY VERTICAL MEAN DISTRIBUTION

ALGORITHM -

Setting the region of interest in our image :

```
region_of_interest_vertices = [(700, height), (width/2, height/1.37),   (width-400, height)]
vertices = np.array([region_of_interest_vertices], np.int32)
mask = np.zeros_like(edge)
match_mask_color = (255)
cv.fillPoly(mask, vertices, match_mask_color)
masked_image = cv.bitwise_and(edge, mask)
```

DETAILED MODULE DESIGN

MODULE 3 - LANE DETECTION

1. Progressive Probabilistic Hough Transform (PPHT) is used to extract the straightest lines in the AROI.
2. The noisy lines (with unqualified angles) detected by PPHT can be removed using K-means clustering technique.

INPUT : Binary Image with a horizon line

OUTPUT : Noiseless road lane markings

ALGORITHM -

PPHT algorithm :

PHT :

```
lines = cv.HoughLinesP(cropped_image, rho=2, theta=np.pi/180,
threshold=50, lines=np.array([]), minLineLength=10, maxLineGap=30)
```

CREATIVE AND INNOVATIVE PROJECT

DETAILED MODULE DESIGN

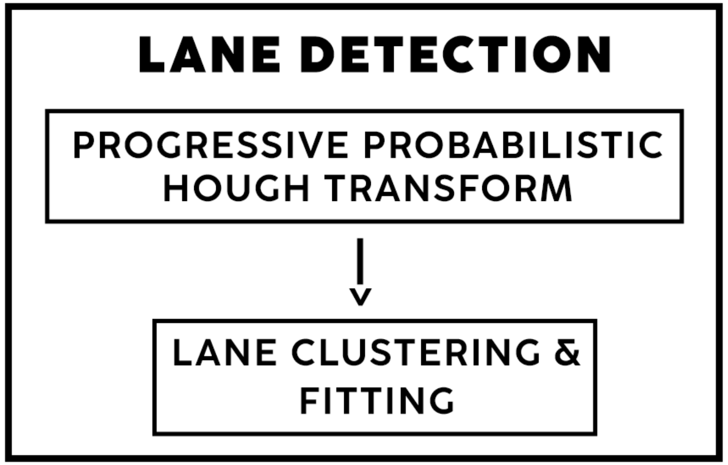
CREATIVE AND INNOVATIVE PROJECT

MODULE 3 - LANE DETECTION

ALGORITHM -

K-Means clustering :

```
n_clusters=4
kmeans = KMeans(n_clusters=n_clusters, n_init = 4)
kmeans.fit(lines)
centroids = kmeans.cluster_centers_
lines = scaler.inverse_transform(lines)
centroids = scaler.inverse_transform(centroids)
```



DETAILED MODULE DESIGN

MODULE 4 - LANE BOUNDARY ESTIMATION

1. In order to increase the fidelity and the efficiency of lane detection system, Kalman Filter is used to track both limits of each lane markings extracted by PPHT.

INPUT : Extracted road lane markings
OUTPUT : Efficiently marked road lanes

ALGORITHM -

LANE BOUNDARY ESTIMATION (KALMAN FILTER)

Kalman filter algorithm :

```
cv.KalmanFilter(self.state_size, self.meas_size, self.contr_size)
self.kf.transitionMatrix = np.eye(self.state_size, dtype=np.float32)
self.kf.measurementMatrix = np.zeros((self.meas_size,
self.state_size), np.float32)
lt = LaneTracker(2, 0.1, 15)
```

CREATIVE AND INNOVATIVE PROJECT

DETAILED MODULE DESIGN

MODULE 5 - OBSTACLE DETECTION

1. Detecting obstacles like vehicles, pedestrian etc. using the YOLO algorithm and providing a warning.

INPUT : Efficiently marked road lanes

OUTPUT : Obstacles Detected

OBSTACLE DETECTION (YOLO)

ALGORITHM -

YOLO algorithm :

Divide the input images in various grids

Perform image classification

Predict the class probability of each vehicle present in the image

Detecting objects in video :

!./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show /content/drive/MyDrive/video_2.mp4 -i 0 -out_filename /content/drive/MyDrive/video_out2.avi

CREATIVE AND INNOVATIVE PROJECT

IMPLEMENTATION DETAILS

CREATIVE AND INNOVATIVE PROJECT

MODULE 1 – PREPROCESSING

```
[ ] def get_vertices(image):
        rows, cols = image.shape[:2]
        bottom_left  = [cols*0.15, rows]
        top_left     = [cols*0.45, rows*0.6]
        bottom_right = [cols*0.95, rows]
        top_right    = [cols*0.55, rows*0.6]

        ver = np.array([[bottom_left, top_left, top_right, bottom_right]], dtype=np.int32)
        return ver

    def grayscale(img):
        gray_img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
        blur = cv.GaussianBlur(gray_img, (5, 5), 0)
        print("GrayScaled Image")
        cv2_imshow(blur)
        return blur

    def edgeDetection(gray_img):
        edge_det = cv.Canny(gray_img, 50, 100, apertureSize=3)
        print("Edge Detection using Canny")
        cv2_imshow(edge_det)
        return edge_det

    def noiseRemoval(edge):
        ret, thresh1  = cv.threshold(edge, 120, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
        print("OTSU thresholding")
        cv2_imshow(thresh1)
        return thresh1

    def preprocessing(img):
        gray_img = grayscale(img)
        edge = edgeDetection(gray_img)
        edge = noiseRemoval(edge)
        #print("After pre-processing")
        #cv2_imshow(edge)
        return edge
```

IMPLEMENTATION DETAILS

OUTPUT

INPUT FRAME

GRAYSCALE IMAGE

CANNY FILTER

OTSU THRESHOLDING

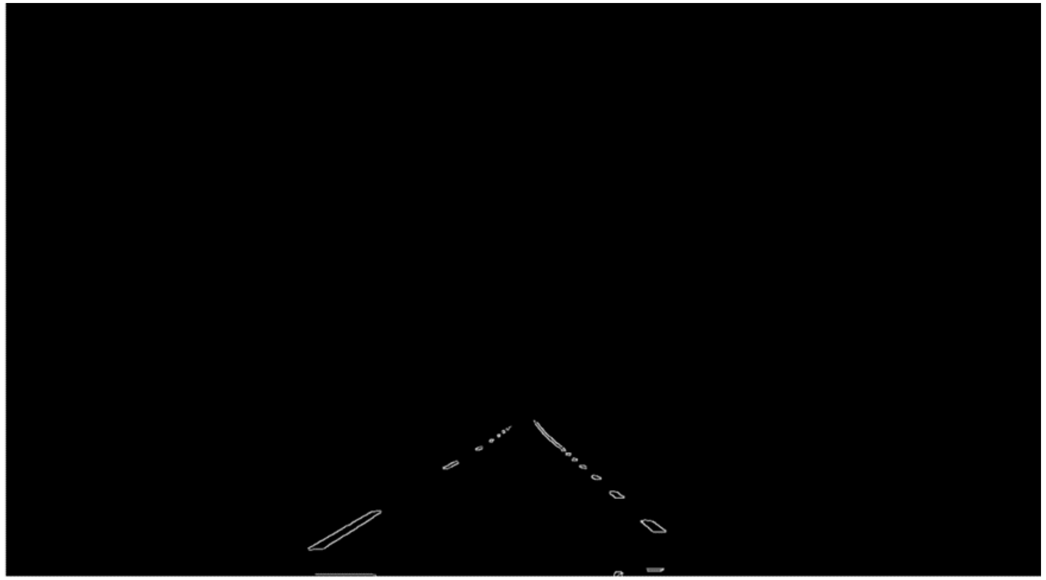
CREATIVE AND INNOVATIVE PROJECT

IMPLEMENTATION DETAILS

MODULE 2 – ROI SETTING

```
[ ]  def regionOfInterest(img, edge):
         height = img.shape[0]
         width = img.shape[1]
         #region_of_interest_vertices = [(700, height), (width/2, height/1.37), (width-400, height)]
         region_of_interest_vertices = [(300, height), (width/2, height/1.37), (width-300, height)]
         vertices = np.array([region_of_interest_vertices], np.int32)
         mask = np.zeros_like(edge)
         match_mask_color = (255)
         cv.fillPoly(mask, vertices, match_mask_color)
         masked_image = cv.bitwise_and(edge, mask)
         print("ROI")
         cv2_imshow(masked_image)
         return masked_image
```

OUTPUT



CREATIVE AND INNOVATIVE PROJECT

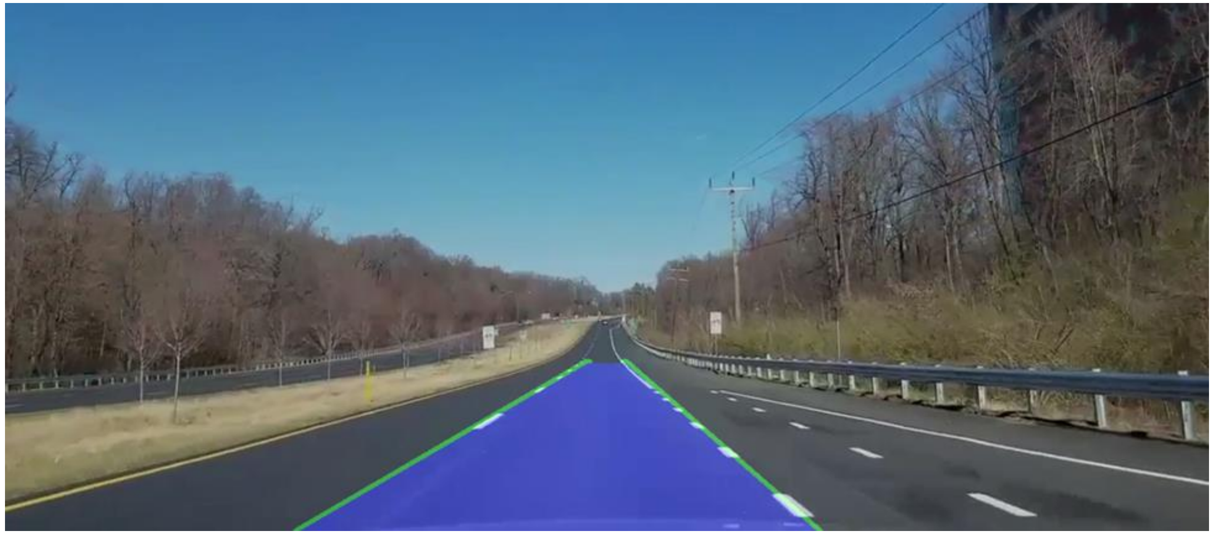
IMPLEMENTATION DETAILS

CREATIVE AND INNOVATIVE PROJECT

MODULE 3 – LANE DETECTION

```python
roi_theta = 0.4
road_horizon = 10
def draw_lines(img, lines):
    blank_image = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)

    for line in lines:
        for x1, y1, x2, y2 in line:
            cv.line(blank_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

    return cv.addWeighted(img, 0.8, blank_image, 1, 0.0)

def laneDetectionPPHT(img, cropped_image):
    lines = cv.HoughLinesP(cropped_image, rho=2, theta=np.pi/180,
                           threshold=50, lines=np.array([]), minLineLength=10, maxLineGap=30)
    print("In ppht")
    print(lines)
    image_with_lines = draw_lines(img, lines)
    return lines, image_with_lines
```

OUTPUT

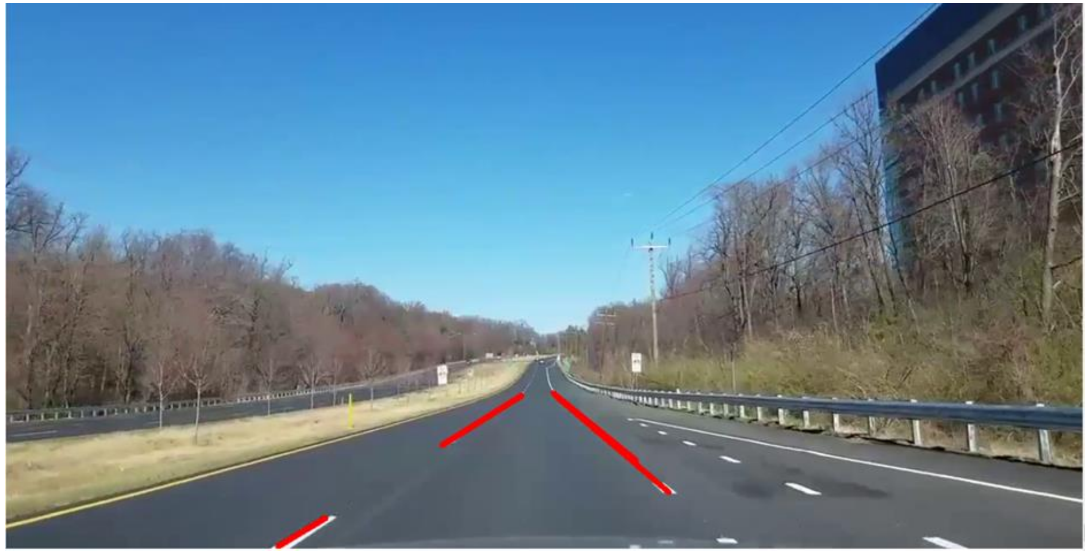


IMPLEMENTATION DETAILS

MODULE 3 – LANE DETECTION

```
def kmeans_clustering(lines):
    print("in kmeans")
    print(lines)
    #preprocessing features to be in (0-1) range
    scaler = MinMaxScaler()
    lines = scaler.fit_transform(lines)

    #checking K-Means Clustering Algorithm performance
    n_clusters=4
    kmeans = KMeans(n_clusters=n_clusters, n_init = 4)
    kmeans.fit(lines)
    centroids = kmeans.cluster_centers_

    lines = scaler.inverse_transform(lines) #getting back our original values
    centroids = scaler.inverse_transform(centroids) #scaling centroids to be simmilar to our original lines

    plt.scatter(lines[:,0],lines[:,1])
    plt.scatter(centroids[:,0],centroids[:,1])
    plt.show()

    print(centroids)
    return centroids
```

OUTPUT



CREATIVE AND INNOVATIVE PROJECT

IMPLEMENTATION DETAILS

MODULE 4 – LANE BOUNDARY ESTIMATION

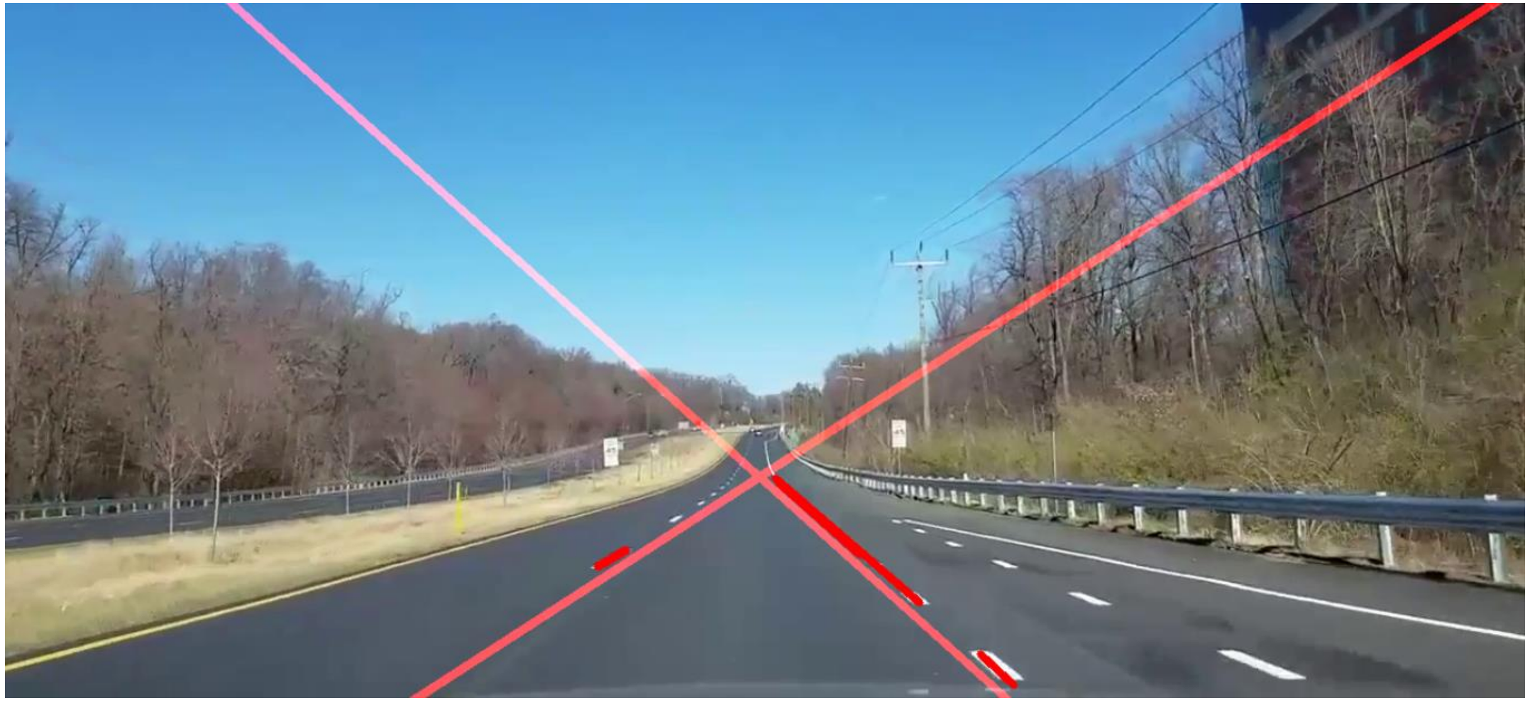
```
def _update_dt(self, dt):
    for i in range(0, self.state_size, 2):
        self.kf.transitionMatrix[i, i+1] = dt

def _first_detect(self, lanes):
    for l, i in zip(lanes, range(0, self.state_size, 8)):
        self.state[i:i+8:2, 0] = l
    self.kf.statePost = self.state
    self.first_detected = True

def update(self, lanes):
    if self.first_detected:
        for l, i in zip(lanes, range(0, self.meas_size, 4)):
            if l is not None:
                self.meas[i:i+4, 0] = l
        self.kf.correct(self.meas)
    else:
        lanes = lanes
        if lanes.count(None) == 0:
            self._first_detect(lanes)

def predict(self, dt):
    if self.first_detected:
        self._update_dt(dt)
        state = self.kf.predict()
        lanes = []
        for i in range(0, len(state), 8):
            lanes.append((state[i], state[i+2], state[i+4], state[i+6]))
        return lanes
    else:
        return None
```

CREATIVE AND INNOVATIVE PROJECT

IMPLEMENTATION DETAILS

CREATIVE AND INNOVATIVE PROJECT

OUTPUT



IMPLEMENTATION DETAILS

MODULE 5 – OBSTACLE DETECTION

```
[ ] #Download pre-trained YOLOv4 weights
    !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
```

```
--2022-06-03 10:39:40--  https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weight
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/ba4b6380-889c-11ea-9
--2022-06-03 10:39:40--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/ba4b6
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.110.133, 185.199.108.133, 185.199
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 257717640 (246M) [application/octet-stream]
Saving to: 'yolov4.weights'

yolov4.weights      100%[===================>] 245.78M   238MB/s    in 1.0s

2022-06-03 10:39:41 (238 MB/s) - 'yolov4.weights' saved [257717640/257717640]
```

```
[ ] !./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show /content/drive/MyDrive/ovideo.mp4 -i 0 -out_filename /content/drive/MyDrive/video_yolo_out.mp4
```

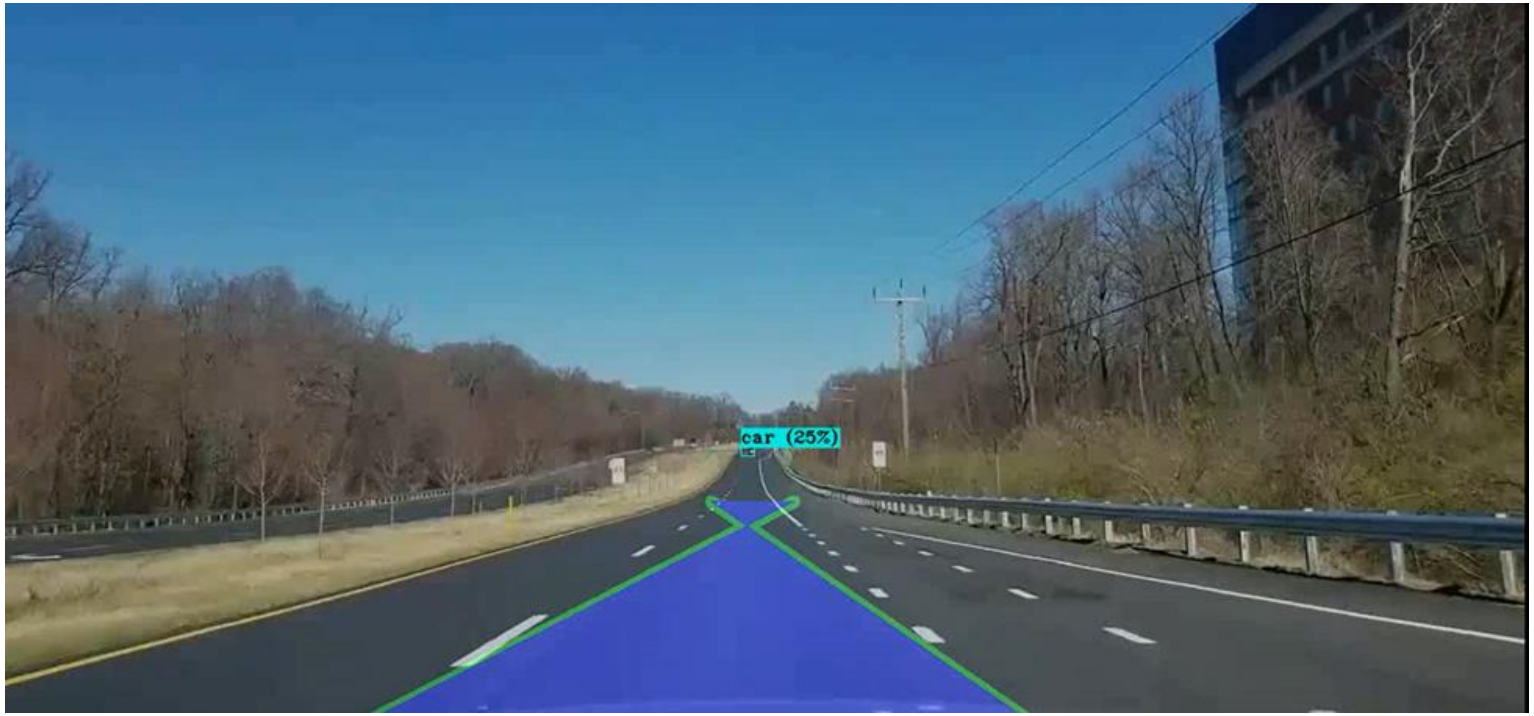
```
 CUDA-version: 11010 (11020), cuDNN: 7.6.5, CUDNN_HALF=1, GPU count: 1
 CUDNN_HALF=1
 OpenCV version: 3.2.0
Demo
 0 : compute_capability = 750, cudnn_half = 1, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    608 x 608 x   3 ->  608 x 608 x  32 0.639 BF
   1 conv     64       3 x 3/ 2    608 x 608 x  32 ->  304 x 304 x  64 3.407 BF
   2 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   3 route  1                                    ->  304 x 304 x  64
   4 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   5 conv     32       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  32 0.379 BF
   6 conv     64       3 x 3/ 1    304 x 304 x  32 ->  304 x 304 x  64 3.407 BF
   7 Shortcut Layer: 4,  wt = 0, wn = 0, outputs: 304 x 304 x  64 0.006 BF
   8 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x  64 0.757 BF
   9 route  8 2                                  ->  304 x 304 x 128
  10 conv     64       1 x 1/ 1    304 x 304 x 128 ->  304 x 304 x  64 1.514 BF
  11 conv    128       3 x 3/ 2    304 x 304 x  64 ->  152 x 152 x 128 3.407 BF
  12 conv     64       1 x 1/ 1    152 x 152 x 128 ->  152 x 152 x  64 0.379 BF
```

CREATIVE AND INNOVATIVE PROJECT

IMPLEMENTATION DETAILS

CREATIVE AND INNOVATIVE PROJECT

OUTPUT



EVALUATION METRICS

The criteria that are used in the evaluation of lane detection performance:

ACCURACY

Accuracy indicates the fraction of predictions that the model made were correct.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

PRECISION

Precision is defined as the ratio of correctly classified positive (True Positives) samples to a total number of classified positive samples (True Positives and False Positive).

$$Precision = \frac{TP}{TP + FP}$$

TP = True positive

TN = True negative

RECALL

The recall is calculated as the ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples

$$Recall = \frac{TP}{TP + FN}$$

FP = False positive

FN = False negative

F1-SCORE

F1 Score is the weighted average of Precision and Recall.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

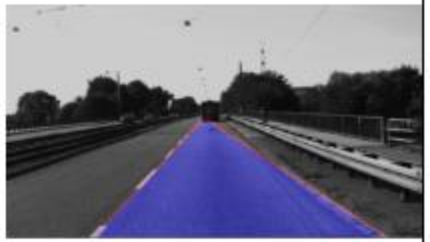
CREATIVE AND INNOVATIVE PROJECT

TEST CASES AND VALIDATION

Frame No.	Input Frame	Lane detected Frame	Binary output	Metrics
1				Precision : 0.806 Recall : 0.806 Accuracy : 0.963 F1-score : 0.806
2				Precision : 0.736 Recall : 0.736 Accuracy : 0.911 F1-score : 0.736

CREATIVE AND INNOVATIVE PROJECT

TEST CASES AND VALIDATION

3				Precision : 0.776 Recall : 0.776 Accuracy : 0.916 F1-score : 0.776
4				Precision : 0.757 Recall : 0.757 Accuracy : 0.912 F1-score : 0.757

CREATIVE AND INNOVATIVE PROJECT

TEST CASES AND VALIDATION

OVERALL METRICS

ACCURACY – 0.925

PRECISION – 0.769

RECALL – 0.769

F1-SCORE – 0.769

REFERENCES

1. https://ieeexplore.ieee.org/abstract/document/9085354
2. https://ieeexplore.ieee.org/document/9213624
3. https://ieeexplore.ieee.org/document/9412400
4. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7905284
5. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4621152
6. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.176.57&rep=rep1&type=pdf
7. https://mdpi-res.com/d_attachment/sensors/sensors-16-01276/article_deploy/sensors-16-01276.pdf
8. https://ieeexplore.ieee.org/document/4580573
9. https://ieeexplore.ieee.org/document/7128388

CREATIVE AND INNOVATIVE PROJECT