**Synopsis for BRAINWAVE ’16 – D.I.P.**

**Team ID: 120.IEEE DTU Team Name: rain**

Team Members: Abhyudai (3rd Year CS), Anupam Singh (3rd Year IT), Ved Vasu Sharma (3rd Year EC)

College: JSS Academy of Technical Education, C-20/1, Sector 62, Noida, Uttar Pradesh 201301

**Round 1, PHASE 1:**

**Problem Statement: Detection of Number Plate:**

* Plates of private car and motorised two-wheeler (Black text on white)
* Plates of commercial vehicles (Black text on yellow)
* Plates of commercial vehicles available for rent (Yellow text on black)
* Vehicles of foreign consultants (White text on blue)
* Official Cars of President of India and state governors (Gold embossed on red)

**Solution:**

**Software Used:**

Python, OpenCV (Computer Vision Library)

**Step 1: Threshing the image to differentiate the text area from the rest of the image. (Separating the number plate from rest of the image)**

a) Sample Image

- Thresholding an image is identification of objects based on different colour pixels

- It is mainly segmenting an image based on different colour value of the pixel.

- The given image is tested for multiple parameters and most generalised case is selected for threshing the image for number plate.

- The following Python command from OpenCV library are used for threshing the image as shown:

- **gray =cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**  #grayscale conversion **ret, thresh = cv2.threshold(gray,80,255,cv2.THRESH\_BINARY\_INV)**  #Binary Inverse Threshing

b) Threshed Image

****

**Step2: Contour Detecting and Contour Filtering**

* The threshed image shows that the number plates are separated from rest of the image. Therefore, if the contours are drawn on the image then a full contour will be formed around the number plate with many other unwanted contours as shown.

Python Command:

**contours, hierarchy = cv2.findContours(thresh7,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)**

**cv2.drawContours(img,contours,-1,(0,255,0),1)**

c) Image with all the contours

* As large number of contours are detected, for getting the contours of the number plate various filters are applied such as differentiation on the basis of area, perimeter and length: breadth ratio.
* On applying the filters the contour on the plates and few other similar contours are detected as shown in the figure.

d) Image with contours after filtering.

Few unwanted contours are also detected which are removed if no number/text is found in the further procedure.

* Now the bounding rectangle is identified and the plates are cropped into separate images for applying Recognition Algorithm.

**contour\_rect = cv2.boundingRect (contour)**

**[X, Y, Width, Height] = contour\_rect**

**print X , Y , Height , Width**

**cv2.rectangle(img,(X, Y),(X + Width, Y + Height),(0,0,255),1)**

**imgRoi = img [Y : Y + Height , X : X + Width , : ]**

**D:\ieee\imgRoi0.jpg**

D:\ieee\imgRoi1.jpg

D:\ieee\imgRoi2.jpg

D:\ieee\imgRoi3.jpg

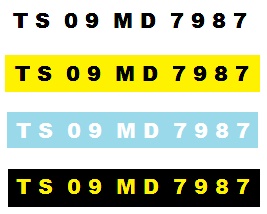
e) Final plates:

**Step 3: Detection of Number plate format. (Type of vehicle):**

* Once the number plate is detected in RGB format, number plate image is threshed to the nearest high or low value of pixel. This gives us the specific RGB value at each pixel (either 0 or 255).

**ret, thresh1 = cv2.threshold (img,200,255,cv2.THRESH\_BINARY)**

Original Template Image Threshed Template Image



* Now value of each pixel is checked for colours (Only two colour exists: Background and text)

1. **Yellow = (R =255 ,G=255 ,B=0 )**
2. **Blue = (R =0 ,G=255 ,B=255 )**
3. **Black = (R =0 ,G=0 ,B=0 )**
4. **White = (R =255 ,G=255 ,B=255 )**
5. **Red = (R =255 ,G=0 ,B=0 )**
6. **Gold (Yellow for thresh) = (R =255 ,G=255 ,B=0 )**

* The colour having the maximum number of pixels is the background.
* As the background is unique for each type of number plate so type can be identified by the background pixel value.

**Round 1, Phase 2: Detection of the state or Union Territory to which the vehicle is registered**

* After successfully, detecting the number plate and the category of the vehicle we move to the most interesting part of the task, identifying the registration no. of the vehicle to determine its state or Union Territory.

**Step 1: Preparing the training set**

* We apply OCR using the **k-nearest neighbour** method of openCV to identify the registration no. of the vehicle. The k- value used in this process is 1.

**Step 2: Pre-processing**

* In this process we first take images of alphabets and numbers under certain fonts and pre-process their images.
* During pre-processing, we have used grayscale, threshing and resizing the images to a certain size and then identifying the contour which is of use for us.
* The below code snippet is used for pre-processing:

**imgGray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**imgThresh = cv2.adaptiveThreshold(imgBlurred,255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY\_INV,11,2)**

**imgROIResized = cv2.resize(imgROI, (RESIZED\_IMAGE\_WIDTH, RESIZED\_IMAGE\_HEIGHT))**

* We have used grayscale because the contour detection method works on grayscale images only.
* To differentiate the text area from the rest of the image, threshing is required.
* Resizing the image helps in forming a normalized version of the training set.

**Step 3: Storing the normalized data set**

* We then store the pixel values of the now obtained images of the various samples in the file flattened.txt

**np.savetxt("saved\_data/flattened\_images.txt", npaFlattenedImages)**

* The file which is used to identify the designated pixel value to the target character is saved with the name classification.txt
* The above two files contain the two training sets.

**Step 4: Prediction**

* When we are to identify the character on screen, we call the prediction method of k-nearest via its object.

**kNearest = cv2.KNearest()**

**kNearest.train(npaFlattenedImages, npaClassifications)**

* This method calculates the Euclidean distance b/w the pixel value of the saved images and the given image and return the most nearest value depending upon the value of k- used.

In the Euclidean plane, ifp = (*p*1, *p*2) and q = (*q*1, *q*2) then the distance is given by

\mathrm{d}(\mathbf{p},\mathbf{q})=\sqrt{(q_1-p_1)^2 + (q_2-p_2)^2}.

* Now based upon the first two characters returned via this method, we identify the state or union territory of the vehicle.
* The last digit of the registration no. will be used in the next round to calculate the violation of the odd- even fine.

**Round 2, PHASE 1:**

**Problem Statement: Detection of the crossing stop Line or violation of “Stopping at Pedestrian Crossing” rule.**

For detecting the pedestrian crossing we are considering roads as the region of interest which is located at the bottom part of the image.

* The similar processes of image processing (threshing, blurring, contour detection) are applied to separate the white background from the black colour road in the region of interest.

****

* The region with the crossing is identified with the maximum and minimum pixel of contours on the y axis.
* The contour on this region of the pedestrian crossing is checked for at the upper edge. If the contour is not continuous (the contour is distorted in shape) then the car is ahead of the crossing. The nearest plate is the plate of the car which has crossed the region.

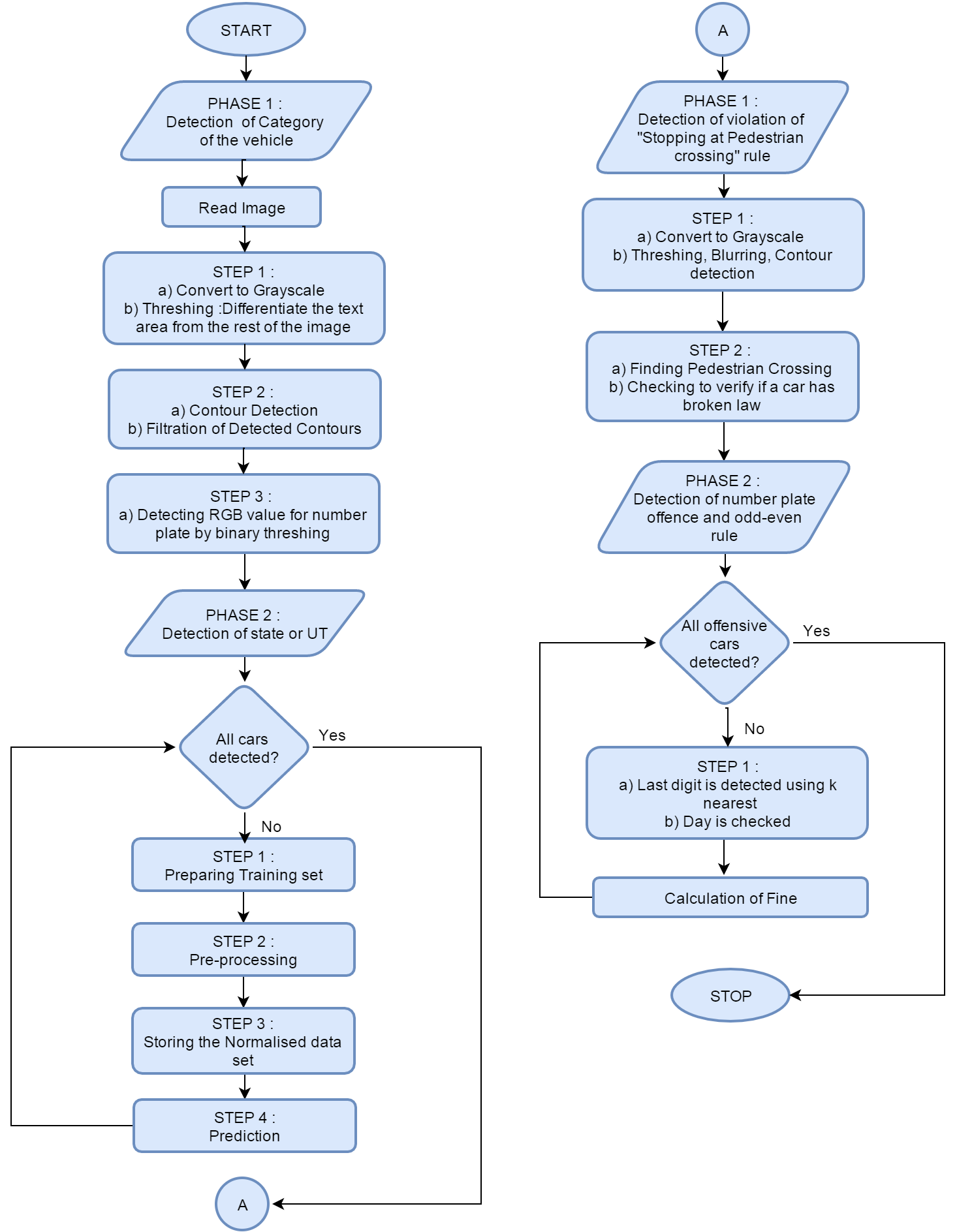
**PHASE 2: Detection of Number Plate Offence, which is violation of Odd-Even rule.**

* The last digit of the text on the plate as detected by the method (K- nearest Neighbour) in Phase 2 of round 1 is recognised.
* Input is taken at the run time for the day and checked with the conditions of odd-even

**Round 3: Calculation of the total fine pertaining to the number of rules being broken.**

* The information of the rules broken is stored with all the number plates and fine is calculated accordingly.

**The workflow of the whole process is depicted below:**



Contact :

Ved Vasu Sharma – om.vedvasu@gmail.com