Optimizing Automatic Number Plate Recognition for Indian number plates

Raghavan Naresh Sarangapani,

Final Project Proposal,

ECE 595 – Introduction to 2D/3D image processing.

Project Plan:

1. Problem statement:

The Indian number plate system is very complex, as there is no standard fonts for the number plates [1]. Further, other problems like usage of multiple lines, stickers, noise in images, and using multiple colors to identify different type of vehicles, request the need for a customized ANPR algorithm.

2. Purpose of project:

The main purpose of this project is as follows:

1. Implement a method to identify the number plates and segment the individual characters in the image.
2. Using a k-means clustering algorithm to identify the individual characters by building a test dataset [3].
3. Identifying the color of the plate, by converting the RoI of the identified plate from the source image to HSV color space and identifying the mean color [5].
4. Improving the performance of k-means by training the dataset.

3. Work statement:

Initial work will start with getting the sample images to train the algorithm, this will be done with the help of google images, as there is no dataset for Indian number plates. To download the multiple images a chrome plugin Fatkun batch download is used [7].

Initially the image is converted to grayscale image, after noise removal using a Gaussian filter the image is then segmented and morphological filters will be used to identify the area of the contour for the number plate. The constraints of the contours are made with specific dimensions to properly identify the contour with the number plate.

The character set segmented is used for matching with a k-means database for each alphabet and number, based on each addition of character map for each segment that is added to the dataset. The k-means clustering chooses an image that matches the character with a majority obtained from multiple k levels [3]. The dataset will be stored using a xml file, which will be updated after each image.

To protect the identity of the owner of the vehicle, the implementation of this project concentrates more on training the dataset of the algorithm. So, for the training set of images the program will learn individual numbers/alphabets in the image instead of learning the actual sequence as in the number plate. This will protect the privacy of the image owners, as the images are obtained from google search.

Based on the comparison the user will manually train the algorithm, awarding it for success and penalizing it for failures. The following conclusions can be reached.

1. k-means is correct, in this case the k-means dataset is updated for the corresponding character.
2. k-means is wrong, the user should enter the result of the actual character. This is later updated to the dataset of k-means cluster.

Finally, a few handpicked images which are identified as complex to classify are given as inputs to the algorithm and the performance is evaluated.

As a future implementation, the performance of the algorithm will be compared with the Tensorflow dataset.

4. Project schedule:

TASK	DIFFICULTY (1-5)	DATE	ACTUAL TIME (Hours)	RESOURCES
Number plate location in source image (Noise removal, segmentation, morphological filtering and contour identification)	3	4/16/2017	6	Microsoft Visual Studio 2013 , OpenCV 2.4.10
Implementing k-means clustering for character recognition	3	4/18/2017	4	Microsoft Visual Studio 2013 , OpenCV 2.4.10 , [3]
Setting up methods to setup user input based learning for the algorithm	2	4/21/2017	3	Microsoft Visual Studio 2013 , OpenCV 2.4.10, XML file to store dataset
Using HSV method to identify color of the number plate	3	4/22/2017	-	Microsoft Visual Studio 2013 , OpenCV 2.4.10, XML file to store dataset, [5]
Training the dataset, with multiple images, tweaking performance of k-means	2	4/25/2017	8	Microsoft Visual Studio 2013 , OpenCV 2.4.10, XML file to store dataset
Final report and powerpoint presentation	3	4/28/2017	5	Microsoft Powerpoint, Microsoft Word

5. Literature review:

There are many implementations for ANPR across various countries which have a standard font for number plates, but for Indian numbers, there are a few papers which show significant contribution.

The authors in [8] have customized the ANPR to work for Indian number plates. They use a DSP based module in which the input images are segmented and contours are used to identify the location of the number plate. Next the template matching is used with a fixed template to match the characters and do the classification. The shortcomings of this method is that Indian license plates have different fonts, and [8] does not use any learning algorithm. The proposed method in this project will add support for additional fonts to be added to the template with the help of the learning method.

The authors in [9], also use a similar method for number plate identification as used in [8], but they differ in the usage of a support vector machine. But in this case the authors do not enhance the performance by using Tessaract and also do not account to the identification of the color of the number plate. In the proposed approach of this project, the system will learn from user input and also will identify the color of the number plate.

The authors in [4] compare the performance of various edge detectors like Sobel, Canny, Gabor and Log-Gabor edge detectors. In this analysis it is concluded that using number plates from Pakistan (which are somewhat similar to the Indian number plates), better performance and quicker time was obtained by using Log-Gabor edge detector.

6. Work accomplished:

At first a basic probing with all the related work on Automatic Number Plate Detection was done using opencv forums. The overview of the project is given in the following steps:

1. Preprocessing the Input image to get the thresholded image.
2. Finding contours in the image, drawing rectangles around contours and using this to look for license plates.
3. Eliminating contours based on conditions for identifying the license plates, and finally fixing a contour which bounds the license plates.
4. Rotating, resizing the identified plate image.
5. Next the contour function is used to identify contours from the licence plate interior.
6. The remaining co-ordinates are sorted to get the characters from the oesecodomyjem

7. A KNN means algorithm is started to receive the inputs and predict the characters in the plate.
8. If the output values is similar to the license plate, then the user can either quit or continue to next images,
9. If a few letter are not the same, then the users can train using the KNN algorithm.
10. The training data is stored in separate files and used for the next time the program executes.

The following section contains all the steps explained in detail.

a) Preprocessing:

- In this step, many preprocessing techniques are followed to obtain the threshold image.
- At the first the image is converted to the HSV channel and the luminance channel is separated from image. This is done to get a better number image.
- Next a morphological filter is applied to get the top and bottom hat images. This gives 4 sets of images which can be used to enhance the output threshold image.
- Finally, the adaptive thresholding algorithm is run on the image to get the thresholded output image.

b) FindContours:

- In this step, we find the contours in the image using the Findcontour API.
- Once the contours are found, the code identifies contours that can have closer relations to a rectangular license plate.
- To select contours closely matching to number plates, a minimum recatangle is drawn around the contour, and the distance of all the points from the center is measured. The contour with largest distance will possibly have its contour closer to the sides of the rectangle.

c) Extracting the plates:

- Based on the obtained RoI, the number plate image can be extracted.
- At first a rotation matrix is derived from the center of the number plate.
- If there are any skews then the image can rotated based on the rotation matrix.
- Finally, a section of this rotated image is cropped and this is used as the image of the plate.

d) Finding characters:

- To find characters we can again preprocess the image and find contours in it.

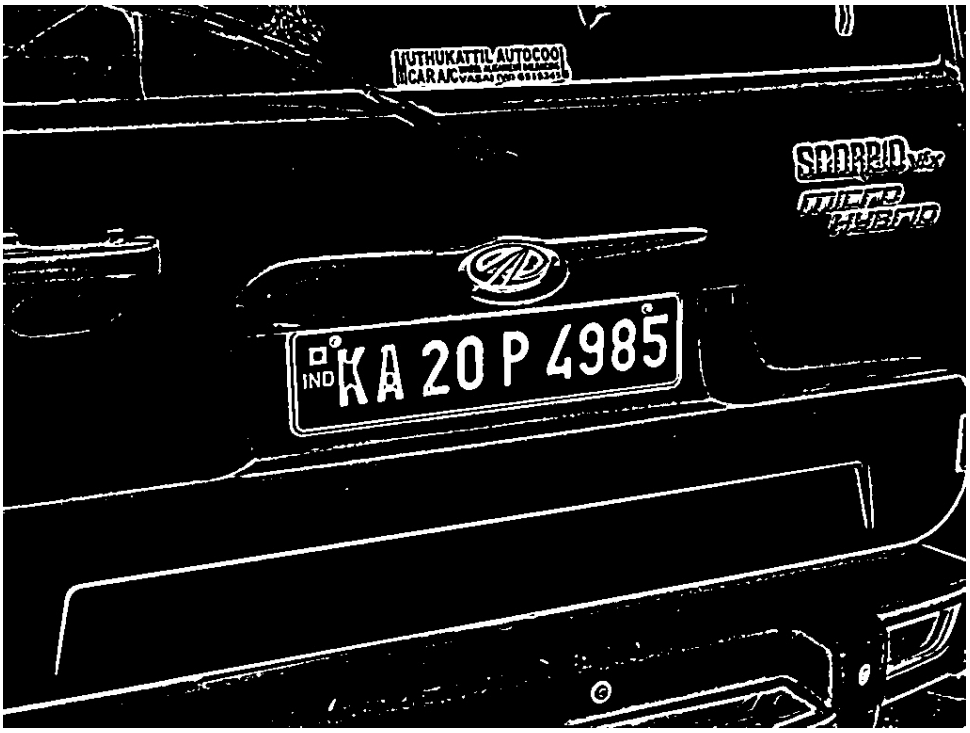
- To avoid smaller objects, we sample take the area of the biggest rectangle around a contour, that has the biggest contour. Contours that are very less that this area are neglected.
- Once these procedures are done, the obtained contour is sorted based on the X input of the bounding rectangle. This is done to get the characters in the license plate from the left to right.

e) K-means clustering:
- In this step the obtained character contours are loaded onto a KMeans algorithm for identification.
- The two files associated with this is for classification and another for image storage. Both of these files are written back with latest details when new information.

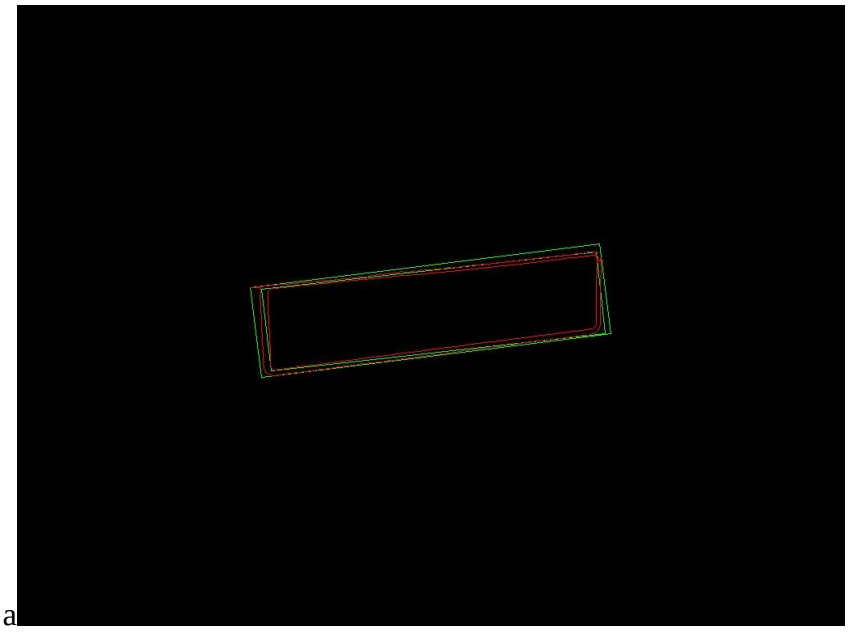
7. Results:

1. The following input image is loaded to the program.

2. Next the image is sent for preprocessing, to get the Threshold min:



3. After finding contours in the image, it can be use to identify the plate location, as shown in the following image.

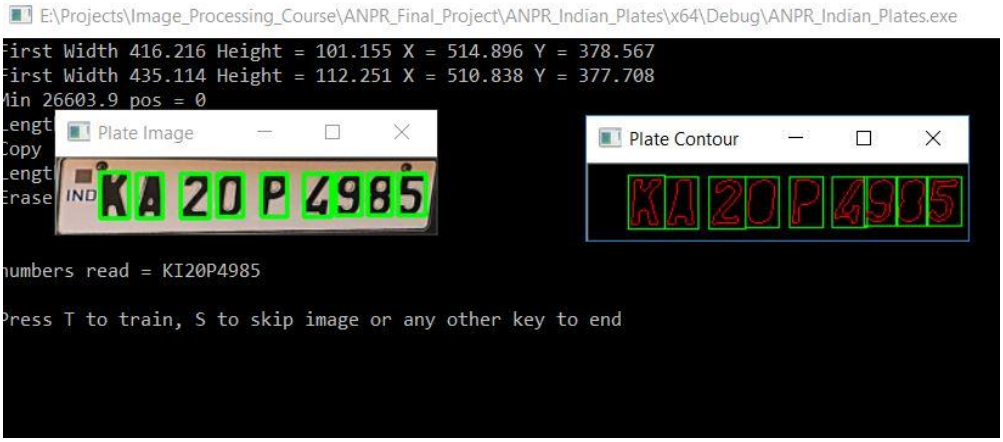


a

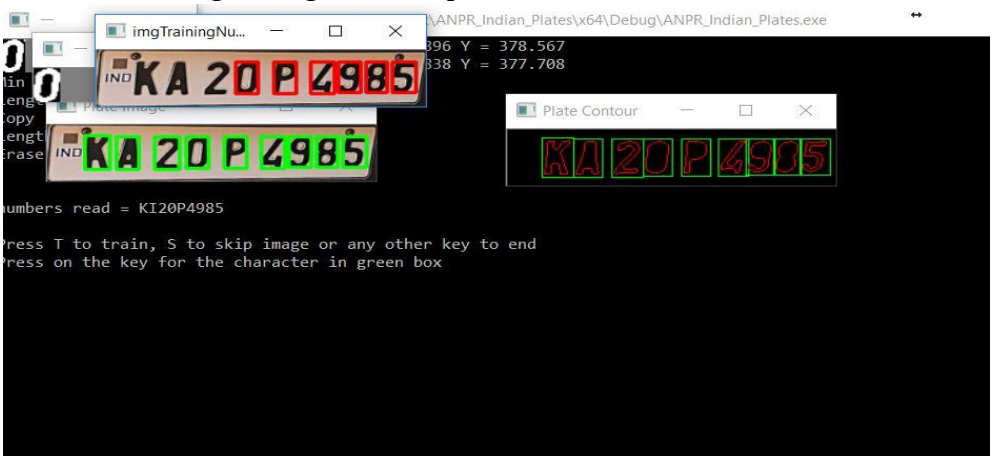
4. After this a rotation matrix is found and the image is rotated and resized to fixed aspect ratio.

5. Based on the above image, we now try to find the characters within the plate. The character counted are marked with a green line.

6. Initially the algorithm with some basic training identifies most of the characters.



7. One user training the algorithm improves:



8. Future Enhancements:

In the current project there are some issue with the identification of the number plate, as shown in the following cases.

- Currently the algorithm does not support multiple license plates.

- Need for other robust methods to identify plates as in this case the front engine guard confuses the algorithm as a plate contour.

- In this case the number has a lighter font on the top and darker in the bottom, so in this case during morphological filtering the top characters erase off. So the algorithm should support multiple parameters for multiple locations in the plate.

- Due to the precence of light in some images the rectangle contour is not obtained and hence alternative methods have to be explored for this case.

REFERENCES:

1. http://www.drivespark.com/off-beat/different-car-registration-number-plates-in-india-explained-014558.html
2. http://docs.opencv.org/trunk/d7/ddc/classcv_1_1text_1_1OCRTesseract.html
3. https://github.com/MicrocontrollersAndMore/OpenCV_3_KNN_Character_Recognition_Cpp
4. Lubna, M. F. Khan and N. Mufti, "Comparison of various edge detection filters for ANPR," 2016 Sixth International Conference on Innovative Computing Technology (INTECH), Dublin, 2016, pp. 306-309. doi: 10.1109/INTECH.2016.7845061
   URL: http://ieeexplore-ieee-org.proxy.ulib.uits.iu.edu/stamp/stamp.jsp?tp=&arnumber=7845061&isnumber=7845006
5. http://opencv-srf.blogspot.in/2010/09/object-detection-using-color-seperation.html
6. https://www.youtube.com/watch?v=dVT7cFcSg0s
7. https://chrome.google.com/webstore/detail/fatkun-batch-download-ima/nnjjahlikiabnchcpehcpkdeckfgnohf
8. P. Kulkarni, A. Khatri, P. Banga and K. Shah, "Automatic Number Plate Recognition (ANPR) system for Indian conditions," 2009 19th International Conference Radioelektronika, Bratislava, 2009, pp. 111-114.
   doi: 10.1109/RADIOELEK.2009.5158763
   URL: http://ieeexplore-ieee-org.proxy.ulib.uits.iu.edu/stamp/stamp.jsp?tp=&arnumber=5158763&isnumber=5158702
9. A. K. Singh and S. Roy, "ANPR Indian system using surveillance cameras," 2015 Eighth International Conference on Contemporary Computing (IC3), Noida, 2015, pp. 291-294.
   doi: 10.1109/IC3.2015.7346695
   URL: http://ieeexplore-ieee-org.proxy.ulib.uits.iu.edu/stamp/stamp.jsp?tp=&arnumber=7346695&isnumber=7346637

APPENDIX CODE:

```
#include "Main.h"

class Contour_Class {

public:

	std::vector<cv::Point> ptContour;

	cv::Rect boundingRect;

	static bool Sort_Contour(const Contour_Class& cwdLeft, const
Contour_Class& cwdRight)

	{

		return(cwdLeft.boundingRect.x < cwdRight.boundingRect.x);

	}

};

const int MIN_CONTOUR_AREA = 100;

const int RESIZED_IMAGE_WIDTH = 20;

const int RESIZED_IMAGE_HEIGHT = 30;

const int RESIZED_PLATE_WIDTH = 250;

const int RESIZED_PLATE_HEIGHT = 50;

const int FILE_NAME_NUM_START = 19;

int main(int argc, char ** argv)

{

	Mat KNN_Char_Class_Mat, KNN_Char_Image_Mat;

	Ptr<KNearest>  KNN_Algo(KNearest::create());

	char Input_FIlename[100];

	int Input_File_Num = 19;

	vector< vector< Point> > Input_contours;

	vector< vector< Point> > Plate_contours;

	vector<Contour_Class> Final_contour;

	vector<vector<Point> >::iterator Itr_Contour;

	vector<RotatedRect> Input_Img_Rect;

	int Loop_int_1 = 0, Loop_int_2 = 0, Loop_int_3 = 0;

	Point2f Point_2f_1, Point_2f_2;

	Init_Train_KNN(KNN_Char_Class_Mat, KNN_Char_Image_Mat);
```

```
loop_back:

	Mat Input_Image, Thresh_Image, Thresh_Image_Copy, Plate_Image,
Final_Image, Plate_Thresh_Clone,Plate_Thresh;

	sprintf(Input_FIlename, "Images/image%d.jpg", Input_File_Num);

	Input_Image = imread(Input_FIlename, CV_LOAD_IMAGE_COLOR);

	if (!Input_Image.data)

	{

		cout<< "Error: Couldn't open Input Image, press any key to
quit\n"<<endl;

		waitKey(0);

		return 1;

	}

	KNN_Algo->train(KNN_Char_Image_Mat, ROW_SAMPLE, KNN_Char_Class_Mat);

	Preprocess_Image(Input_Image, Thresh_Image);

	Thresh_Image_Copy = Thresh_Image.clone();

	findContours(Thresh_Image_Copy, Input_contours, CV_RETR_LIST,
CV_CHAIN_APPROX_SIMPLE);//CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

	Mat Contour_Image(Thresh_Image.size(), CV_8UC3, Scalar(0.0, 0.0, 0.0));

	Mat Rect_Contour_Image(Thresh_Image.size(), CV_8UC3, Scalar(0.0, 0.0,
0.0));

	Loop_int_1 = 0;

	Itr_Contour = Input_contours.begin();

	for(;(Itr_Contour != Input_contours.end());)

	{

		RotatedRect Contour_Min_Rect = minAreaRect(Mat(*Itr_Contour));

		if (!Limit_Plate_Size(Contour_Min_Rect, 3.2))

		{

			Itr_Contour = Input_contours.erase(Itr_Contour);

		}

		else

		{

			++Itr_Contour;
```

```
                cout << "First Width " << Contour_Min_Rect.size.width << "
Height = " << Contour_Min_Rect.size.height
                        << " X = " << Contour_Min_Rect.center.x << " Y = " <<
Contour_Min_Rect.center.y << endl;


                Input_Img_Rect.push_back(Contour_Min_Rect);


                Point2f rect_points[4];
                Input_Img_Rect[Loop_int_1].points(rect_points);
                for (int Loop_int_2 = 0; Loop_int_2 < 4; Loop_int_2++)
                      line(Rect_Contour_Image, rect_points[Loop_int_2],
rect_points[(Loop_int_2 + 1) % 4], Scalar(0.0, 255.0, 0.0), 1, 8);


                Loop_int_1++;
            }
      }


      cv::drawContours(Rect_Contour_Image, Input_contours, -1, Scalar(0.0,
0.0, 255.0));


      Loop_int_2 = 0;
      Loop_int_3 = 0;


      if (Input_contours.size() > 0)
      {


            double Contour_Dist[100];
            double Contour_Min_Dist = 999999;
            int Contour_Min_Pos = 0;


            Itr_Contour = Input_contours.begin();


            for (Loop_int_2 = 0; Loop_int_2<Loop_int_1; Loop_int_2++)
            {
```

```
                vector<Point> Indv_Contour = Input_contours.at(Loop_int_2);

                Point_2f_1 = Input_Img_Rect[Loop_int_2].center;

                Contour_Dist[Loop_int_2] = 0;

                for (Loop_int_3 = 0; Loop_int_3 < Indv_Contour.size();
Loop_int_3++)
                {
                      Point_2f_2 = Indv_Contour.at(Loop_int_3);
                      Contour_Dist[Loop_int_2] += norm(Point_2f_2 -
Point_2f_1);
                }

                if (Contour_Dist[Loop_int_2] <= Contour_Min_Dist)
                {
                      Contour_Min_Dist = Contour_Dist[Loop_int_2];
                      Contour_Min_Pos = Loop_int_2;
                }
          }

          cout << "Min " << Contour_Min_Dist << " pos = " <<
Contour_Min_Pos << endl;

          Itr_Contour = Input_contours.begin();

          for (Loop_int_2 = 0; Loop_int_2 < Loop_int_1; Loop_int_2++)
          {
                cout << "Length " << Loop_int_2 << " is = " <<
Contour_Dist[Loop_int_2] << endl;
                if (Loop_int_2 != Contour_Min_Pos)
                {

                      Itr_Contour = Input_contours.erase(Itr_Contour);
```

```
                        cout << "Erase " << endl;
                }
                else
                {
                        cout << "Copy " << endl;
                        Itr_Contour++;
                }
        }

        Extract_Plate(Input_Image, Input_contours, Plate_Image);

        Final_Image = Plate_Image.clone();

        Mat Plate_Gray, Plate_Blurred;

        cvtColor(Plate_Image, Plate_Gray, CV_BGR2GRAY);

        GaussianBlur(Plate_Gray, Plate_Blurred, Size(5, 5), 0);

        adaptiveThreshold(Plate_Blurred, Plate_Thresh, 255.0,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV, 11, 2);

        Plate_Thresh_Clone = Plate_Thresh.clone();

        findContours(Plate_Thresh_Clone, Plate_contours, CV_RETR_LIST,
CV_CHAIN_APPROX_SIMPLE);//CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);

        Mat Plate_Contour_Image(Plate_Thresh.size(), CV_8UC3, Scalar(0.0,
0.0, 0.0));

        Itr_Contour = Plate_contours.begin();

        Loop_int_1 = 0;
```

```
            int Plate_Contour_Area[100];
            double Contour_Max_Dist = 0;
            int Contour_Max_Pos = 0;

            for (; (Itr_Contour != Plate_contours.end());)
            {

                  Rect Plate_Rec = boundingRect(Mat(*Itr_Contour));

                  float Aspect_Contour = (float)Plate_Rec.width /
(float)Plate_Rec.height;

                  if (Aspect_Contour < 1)
                  {
                        ++Itr_Contour;

                        Plate_Contour_Area[Loop_int_1] = Plate_Rec.width *
Plate_Rec.height;

                        if (Plate_Contour_Area[Loop_int_1] >
Contour_Max_Dist)
                        {
                              Contour_Max_Dist =
Plate_Contour_Area[Loop_int_1];
                              Contour_Max_Pos = Loop_int_1;
                        }

                        Loop_int_1++;
                  }
                  else
                  {
                        Itr_Contour = Plate_contours.erase(Itr_Contour);
                  }
```

```
            }

            Plate_Contour_Area[Loop_int_1] = -1;

            Itr_Contour = Plate_contours.begin();

            Loop_int_1 = 0;

            for (; (Itr_Contour != Plate_contours.end());)
            {
                  if (Plate_Contour_Area[Loop_int_1] < (Contour_Max_Dist -
200))
                  {
                        Itr_Contour = Plate_contours.erase(Itr_Contour);
                  }
                  else
                  {
                        Rect Plate_Rec = boundingRect(Mat(*Itr_Contour));

                        Point2f rect_points[4];
                        rect_points[0] = Point2f(Plate_Rec.x-2, Plate_Rec.y-
2);
                        rect_points[1] = Point2f(Plate_Rec.x +
Plate_Rec.width+2, Plate_Rec.y-2);
                        rect_points[2] = Point2f(Plate_Rec.x +
Plate_Rec.width+2, Plate_Rec.y + Plate_Rec.height+2);
                        rect_points[3] = Point2f(Plate_Rec.x-2, Plate_Rec.y +
Plate_Rec.height+2);

                        for (int Loop_int_2 = 0; Loop_int_2 < 4;
Loop_int_2++)
                        {
```

```
                              line(Plate_Contour_Image,
rect_points[Loop_int_2], rect_points[(Loop_int_2 + 1) % 4], Scalar(0.0,
255.0, 0.0), 1, 8);

                              //line(Final_Image, rect_points[Loop_int_2],
rect_points[(Loop_int_2 + 1) % 4], Scalar(0.0, 255.0, 0.0), 1, 8);

                        }

                        ++Itr_Contour;

                  }


                  Loop_int_1++;

            }


            drawContours(Plate_Contour_Image, Plate_contours, -1, Scalar(0.0,
0.0, 255.0));


            //Insertion_Sort(Plate_contours, Plate_Sorted_contours);


            //vector< vector< Point> >
Plate_Sorted_contours(Plate_contours.size());


            for (int i = 0; i < Plate_contours.size(); i++)

            {

                  Contour_Class contourWithData;

                  contourWithData.ptContour = Plate_contours[i];

                  contourWithData.boundingRect =
cv::boundingRect(contourWithData.ptContour);

                  Final_contour.push_back(contourWithData);

            }


            std::sort(Final_contour.begin(), Final_contour.end(),
Contour_Class::Sort_Contour);



            string Plate_String;
```

```
            for (int Loop_int_1 = 0; Loop_int_1 < Final_contour.size();
Loop_int_1++)
            {
                  Mat Plate_ROI, Plate_ROI_Resize, Plate_ROI_Float,
Plate_ROI_Flat_Float;
                  Mat Plate_Curr_Char(0, 0, CV_32F);


                  Rect Char_Rect = Final_contour[Loop_int_1].boundingRect;


                  rectangle(Plate_Image,Char_Rect,Scalar(0, 255, 0),2);


                  Plate_ROI = Plate_Thresh(Char_Rect);


                  resize(Plate_ROI, Plate_ROI_Resize,
Size(RESIZED_IMAGE_WIDTH, RESIZED_IMAGE_HEIGHT));


                  Plate_ROI_Resize.convertTo(Plate_ROI_Float, CV_32FC1);


                  Plate_ROI_Flat_Float = Plate_ROI_Float.reshape(1, 1);


                  KNN_Algo->findNearest(Plate_ROI_Flat_Float, 1,
Plate_Curr_Char);


                  float fltCurrentChar = (float)Plate_Curr_Char.at<float>(0,
0);


                  Plate_String = Plate_String + char(int(fltCurrentChar));


                  Plate_ROI.release();
                  Plate_ROI_Resize.release();
                  Plate_ROI_Float.release();
                  Plate_ROI_Flat_Float.release();
                  Plate_Curr_Char.release();
            }
```

```
            cout << "\n\n" << "License plate read is = " << Plate_String <<
"\n\n";

            namedWindow("Plate Contour", CV_WINDOW_AUTOSIZE);
            imshow("Plate Contour", Plate_Contour_Image);

            namedWindow("Plate Image", CV_WINDOW_AUTOSIZE);
            imshow("Plate Image", Final_Image);

            Plate_String.clear();

            Plate_Gray.release();
            Plate_Blurred.release();
            Plate_Contour_Image.release();

      }
      else
      {
            cout << "No Contours found, skip image" << endl;
      }

      cout << "Press T to train, S to skip image or any other key to end" <<
endl;

      int Char_In = waitKey(0);
      if ((Char_In == 84) || (Char_In == 116))
      {
            cout << "Press on the key for the character in red box" << endl;

            for (int Loop_int_1 = 0; Loop_int_1 < Plate_contours.size();
Loop_int_1++)
            {
```

```
                Mat Plate_ROI, Plate_ROI_Resize, Plate_ROI_Float,
Plate_ROI_Flat_Float;

                Rect Char_Rect = boundingRect(Plate_contours[Loop_int_1]);

                rectangle(Final_Image, Char_Rect, Scalar(0, 0, 255), 2);

                Plate_ROI = Plate_Thresh(Char_Rect);

                resize(Plate_ROI, Plate_ROI_Resize,
Size(RESIZED_IMAGE_WIDTH, RESIZED_IMAGE_HEIGHT));

                imshow("Training Char", Plate_ROI);
                imshow("Training Char Resize", Plate_ROI_Resize);
                imshow("Training Plate", Final_Image);

                Char_In = waitKey(0);

                if (find(intValidChars.begin(), intValidChars.end(),
Char_In) != intValidChars.end())
                {

                      KNN_Char_Class_Mat.push_back(Char_In);

                      Plate_ROI_Resize.convertTo(Plate_ROI_Float,
CV_32FC1);

                      Plate_ROI_Flat_Float = Plate_ROI_Float.reshape(1, 1);

                      KNN_Char_Image_Mat.push_back(Plate_ROI_Flat_Float);
                }

                Plate_ROI.release();
                Plate_ROI_Resize.release();
```

```
            Plate_ROI_Float.release();
            Plate_ROI_Flat_Float.release();

        }

        //Update_Train_KNN(KNN_Char_Class_Mat, KNN_Char_Image_Mat);

        Input_Image.release();
        Thresh_Image.release();
        Thresh_Image_Copy.release();
        Plate_Image.release();
        Final_Image.release();
        Plate_Thresh_Clone.release();
        Plate_Thresh.release();
        Contour_Image.release();
        Rect_Contour_Image.release();

        //Input_File_Num++;
        goto loop_back;

}
else if ((Char_In == 83) || (Char_In == 115))
{
        Input_Image.release();
        Thresh_Image.release();
        Thresh_Image_Copy.release();
        Plate_Image.release();
        Final_Image.release();
        Plate_Thresh_Clone.release();
        Plate_Thresh.release();
        Contour_Image.release();
        Rect_Contour_Image.release();
```

```
            Input_File_Num++;
            goto loop_back;
      }

      Update_Train_KNN(KNN_Char_Class_Mat, KNN_Char_Image_Mat);

      KNN_Char_Class_Mat.release();

      KNN_Char_Image_Mat.release();
}

int Init_Train_KNN(Mat& Char_Classify, Mat& Char_Image)
{

      FileStorage Char_Class_File("KNN_Classify_Char.xml", 0);
      if (Char_Class_File.isOpened() == false)
      {
            cout << "Error opening KNN_Classify_Char.xml , press any key to
exit" << endl;
            waitKey(0);
            return(0);
      }

      Char_Class_File["Char_classify"] >> Char_Classify;
      Char_Class_File.release();

      FileStorage Char_Image_File("KNN_Char_Images.xml", 0);
      if (Char_Image_File.isOpened() == false)
      {
            cout << "Error opening KNN_Char_Images.xml , press any key to
exit" << endl;
            waitKey(0);
```

```
            return(0);
      }

      Char_Image_File["Char_Images"] >> Char_Image;
      Char_Image_File.release();

      return(1);

}

int Update_Train_KNN(Mat& Char_Classify, Mat& Char_Image)
{

      FileStorage Char_Class_File("KNN_Classify_Char.xml",
FileStorage::WRITE);
      if (Char_Class_File.isOpened() == false)
      {
            cout << "Error writing KNN_Classify_Char.xml , press any key to
exit" << endl;
            waitKey(0);
            return(0);
      }

      Char_Class_File << "Char_classify" << Char_Classify;


      FileStorage Char_Image_File("KNN_Classify_Image.xml",
FileStorage::WRITE);
      if (Char_Image_File.isOpened() == false)
      {
            cout << "Error writing KNN_Char_Images.xml , press any key to
exit" << endl;
            waitKey(0);
            return(0);
```

```
	}
	Char_Image_File << "Char_Images" << Char_Image;
	Char_Class_File.release();
	Char_Image_File.release();

	return(1);
}

int Preprocess_Image(Mat& Input_Image, Mat& Threshold_Image)
{
	Mat HSV_Image, Top_Hat, Black_Hat, GS_Top_Hat, GS_Top_Neg_Black_Hat,
Blur_Image, Morph_Image, Gray_Image;

	vector<Mat> HSV_Channels;

	Mat Hat_Struct = getStructuringElement(CV_SHAPE_RECT, Size(3, 3));

	cvtColor(Input_Image, HSV_Image, CV_BGR2HSV);

	split(HSV_Image, HSV_Channels);

	Gray_Image = HSV_Channels[2];

	morphologyEx(Gray_Image, Top_Hat, CV_MOP_TOPHAT, Hat_Struct);
	cv::morphologyEx(Gray_Image, Black_Hat, CV_MOP_BLACKHAT, Hat_Struct);

	GS_Top_Hat = Gray_Image + Top_Hat;
	GS_Top_Neg_Black_Hat = GS_Top_Hat - Black_Hat;

	Morph_Image = GS_Top_Neg_Black_Hat.clone();

	GaussianBlur(Gray_Image, Blur_Image, Size(5, 5), 0);
```

```
	adaptiveThreshold(Blur_Image, Threshold_Image, 255.0,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV, 19, 9);


	HSV_Image.release();

	Top_Hat.release();

	Black_Hat.release();

	GS_Top_Hat.release();

	GS_Top_Neg_Black_Hat.release();

	Blur_Image.release();

	Morph_Image.release();

	Gray_Image.release();


	return(1);

}


int Extract_Plate(Mat& Input_Image, vector< vector< Point> >& Input_contours,
Mat& Output_Plate_Image)

{

	vector<vector<Point> >::iterator Itr_Contour;

	RotatedRect Contour_Min_Rect;

	float Contour_Aspect, Contour_Angle;

	Mat Plate_Rot_Matrix, Rot_Image, Crop_Image;

	Size Plate_size;


	Itr_Contour = Input_contours.begin();


	Contour_Min_Rect = minAreaRect(Mat(*Itr_Contour));


	Contour_Aspect = (float)Contour_Min_Rect.size.width /
(float)Contour_Min_Rect.size.height;

	Contour_Angle = Contour_Min_Rect.angle;
```

```
	if (Contour_Aspect < 1)
		Contour_Angle = 90 + Contour_Angle;

	Plate_Rot_Matrix = getRotationMatrix2D(Contour_Min_Rect.center,
Contour_Angle, 1);

	warpAffine(Input_Image, Rot_Image, Plate_Rot_Matrix,
Input_Image.size(), CV_INTER_CUBIC);

	Plate_size = Contour_Min_Rect.size;

	if (Contour_Aspect < 1)
		swap(Plate_size.width, Plate_size.height);

	getRectSubPix(Rot_Image, Plate_size, Contour_Min_Rect.center,
Crop_Image);

	Output_Plate_Image.create(RESIZED_PLATE_HEIGHT, RESIZED_PLATE_WIDTH,
CV_8UC3);
	resize(Crop_Image, Output_Plate_Image, Output_Plate_Image.size(), 0, 0,
INTER_CUBIC);

	return(1);
}



bool Limit_Plate_Size(RotatedRect Plate_Rect, float Plate_Aspect)
{
	float Limit_err, Limit_rmin, Limit_rmax, Limit_Aspect;
	int Limit_min, Limit_max, Limit_area;

	Limit_err = 0.4;
```

```
	Limit_min = 2500 * Plate_Aspect;
	Limit_max = 250000 * Plate_Aspect;


	Limit_rmin = (Plate_Aspect - (Plate_Aspect * Limit_err));
	Limit_rmax = (Plate_Aspect + (Plate_Aspect * Limit_err));


	Limit_area = (Plate_Rect.size.height * Plate_Rect.size.width);
	Limit_Aspect = (float)Plate_Rect.size.width /
(float)Plate_Rect.size.height;


	if ((Limit_area < Limit_min || Limit_area > Limit_max) || (Limit_Aspect
< Limit_rmin || Limit_Aspect > Limit_rmax))
	{
		return false;
	}
	else
	{
		return true;
	}


}
```