Number Plate Recognition

Our approach is divided into two stages:
1. Extraction of the number plate
2. Character Recognition

EXTRACTION OF NUMBER PLATE

Our objective is to perform operations which will help us extract the number plate.
Number plate has black on white background.

1. To avoid the effect of illumination on the input image, an illumination correction is applied to the input. HSV color space is used to find the value, which influences the illumination. The pixels with high V (value) in the HSV color space, are given lower V values.
2. The image is converted to a grayscale image.
3. Otsu thresholding is used to find a value of threshold so that image can be divided into foreground and background.
4. A closing operation, is performed on the image, which enlarges the boundary of the foreground objects, that is , numbers on the number plate.
5. A median blur is performed on the image to remove salt and pepper noise from the image. This image has the numbers on the plate clearly visible, along with other things.
6. The candidate regions need to be filtered out. The blobs in the dilated version of above image are detected, and have been approximated as rectangles.
7. Qualification for candidate region has been divided into two steps.
    - 7.1. In the first step, the aspect ratio (between 1 to 4), and ratio of height of rectangle and image (atleast more than 1/32) have been used.
    - 7.2. The rectangles which qualify this, are analyzed for top left y coordinate and height.
    - 7.3. Information about y coordinate and height is stored with some error bound.
    - 7.4. Considering all rectangles around numbers to be having same y coordinate and height. The times of occurrence of each height and y coordinate is stored. The mode for these two parameters is calculated.
    - 7.5. In the second step, the value of mode for height of rectangle and y coordinate along with aspect ratio, is used to find out if the given region qualifies or not.
    - 7.6. Once it meets the requirements, we use the binary image stored previously, as it is very clear and run character recognition on the known region of the binary image.

CHARACTER RECOGNITION

We modified the Tesseract library to suit our demands.

NOTE: It is impossible to say that the code will always run under 25ms, it entirely depends on:

- **The computer running the program**
- **Amount of RAM available**
- **Resolution of the image**

Dependencies:
Tesseract

How to install tesseract?

sudo apt-get install autoconf automake libtool
sudo apt-get install libpng12-dev
sudo apt-get install libjpeg62-dev
sudo apt-get install libtiff4-dev
sudo apt-get install zlib1g-dev
sudo apt-get install libicu-dev
sudo apt-get install libpango1.0-dev
sudo apt-get install libcairo2-dev
sudo apt-get install libleptonica-dev

Unpack the tesseract.tar.gz folder using tar xvzf
Execute the following commands

cd #To the folder you extracted to
make clean
./autogen.sh
./configure
make
sudo make install
sudo ldconfig
cd training
make clean
cd ..
make training
sudo make training-install
export TESSDATA_PREFIX= The place where you extracted the file

OpenCV- Install using this script https://help.ubuntu.com/community/OpenCV

Running the code
To run the code type ./num /path/to/image/Imagename_here
The code will save the extracted letters in the directory where the code is present.

Compiling the code
Compile using this command
g++ -O2 `pkg-config --cflags opencv` -o `basename num.cpp .cpp` num.cpp `pkg-config --libs opencv tesseract`

Sample:
Given the input image as below.

After illumination correction we get:

Otsu thresholding gives:

possible candidate regions are shown below:

With detected rectangles:

These are sent for character recognition.