Searching for a Number Plate

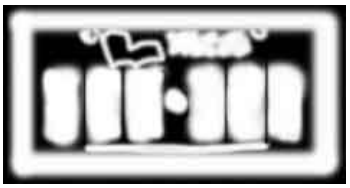
Attempted Methods:

- Cross Correlation with exact crops of the car's
- Cross Correlation with generic images based on number plates
- High Density Edge detection

Cross-correlation when using a crop from the same image is a guaranteed win-case, which is why my first solution, matching 'edged' images against a library of known edged numberplates, was put aside in favour of finding an alternate, less controlled solution.

To the right you can see the perfect location of the numberplate in the image, before the segmentation stage. The green dot indicates the location.

A tutor suggested that it was possible to get a good correlation on most pictures with simple representation of a numberplate. Intending to use the template in a correlation with an edged image I outlined all the likely parts of the image to be detected and attempted to find a correlation. This could very well have been to how specific the template I made was, although 4 of 6 cars all have similar numberplate design and features. I believe this might have had more potential if I reduced the amount of white in the template, making it less likely to have a higher correlation and changed some of the zero values into negatives of values less than -1. The blank space inside the numberplate's boarder would be a good area for this to be done as there should be little to no edges that aren't text inside the numberplate.



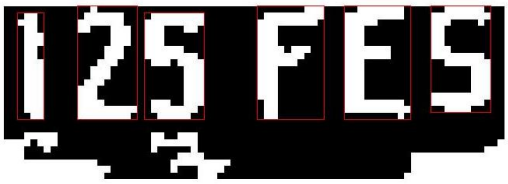
The last major attempted method for this assignment used the idea that numberplates represent a relatively dense area of edges. Provided that the majority of the background can have its edges reduced, this method stands a good chance at finding a numberplate. Unfortunately this method was unable to be tuned well enough to work successfully with more than 2 of the supplied images.

The most important part of this method is finding the edges of the image. I found edges using 3 different calls to edge() and combined them together to give a good covering through the image. Canny, Canny on a blurred image and Sobel methods were used to create an image. A 50pixel wide border was turned zeroed around the outside of the picture in order to remove any high density edging that might have been happening to the edge of frame. the function bwareaopen() is then used to remove all white spots that are less than 10 pixels in size which catches all the tiny lines and dots made by edging process.

All horizontal and vertical lines longer than 100pixels are removed. This thresh is arbitary but could be changed to a ratio of the image's size. Horizontal and Vertical Lines are identified by their bounding box's width to height ratio, if the one is more than 15 times the other, then it's considered thin enough to be a line.

The intensity density of an image can be found by using a larger Gaussian blur on the image. Pulling the position of the highest value from the blurred image provides us with the location of the numberplate, provided that the pre-processing was adequate. So far, there has not been a case of *multiple* highest densities, although this is a foreseeable problem. Shown below are the localisation results for each of the 6 provided images, with the point of highest density being the highlighted dot in the image.

The spot of highest density clearly finds the numberplate and so the next step of the process is segmentation so that we end up with a final product like that shown below.



125TES

The Sobel method edged most numberplates quite well, but didn't properly close them all off. To get around this 2 calls of imclose() were used on the edge image with horizontal and vertical, 3 pixel long lines as structural elements. This helps close any gaps in a numberplate's outline filling it in white once imfill() is called. The image is also eroded and then the remaining blobs labelled. Using the point found during the localisation step the appropriate blob is selected and then marked on the original image.

To recognise that characters on the plate a library of templates is needed. These were found online and are not original. Each template is only the size of the letter, with no extraneous black space. To match the letters on the numberplate the cropped image (the bit that was "marked" in the previous paragraph) is thresholded and before using regionprops() to find the bounding box of each blob in the image. Letters are assumed to be the regions that have a larger height than width.

For each 'letter' found in the numberplate I then use normxcorr2() with the letter as the template and a resized character from the afore mentioned library as the image. The highest value of the result is compared with the last known highest value. If the new one is higher it becomes the new value to beat and the character that caused it is stored. This ensures that I find the character with the highest correlation to the 'letter.'

As you can see in the pictures above, this has a 5/6 success on the bluecar.jpg numberplate. A happy side effect of how my code is written allows numberplate text to be written regardless of whether or not the numberplate was successfully localised. For example, the car with the numberplate "TPY 32P" doesn't have a successful localisation, yet when the code is run we see that it's identified "ffTPY3ZPF." All the 'f/F's in that string are from the poor cropping that an unlocalised picture has. If we ignore those letters we find that this numberplate also has a 5/6 score, mistaking a 2 for a Z.

7TDRTQ47XA76HI

The character recognition also has some success with the other image that manages to localise. However in this case it grabbed many more 'letters' than were meant. The TDR77H is there in the string but so is the T and X of TEXAS while the 7, Q and I are all from items outside of the numberplate.

This script runs quickly, with less than 5 seconds of processing for each image.

CODE:

```
clc, clear, close all
%% Load ALL the images!
img1 = imread('20980008.jpg');
img2 = imread('20980035.jpg');
img3 = imread('20980060.jpg');
img4 = imread('20980065.jpg');
blue = imread('bluecar.jpg');
kings = imread('kingswood.jpg');

i = blue;
ig = i;

% get the size of the picture.
rs = size(ig, 1);
cs = size(ig, 2);

if size(i,3) == 3 % make sure it's gray
    ig = rgb2gray(i);
end

ig = histeq(ig);

% Normal edging
is = edge(ig, 'sobel');
ic = edge(ig, 'canny');

% Gauss then canny, also sobel
G = fspecial('gaussian',[5 5],2);
ib = imfilter(ig,G,'same');
ibc = edge(ib, 'canny');
ibs = edge(ib, 'sobel');

% Add them all together
itot = ibc + ic + is;

% pre-proc for edge density
itot([1:50 end-50:end],:) = 0;
itot(:,[1:50 end-50:end]) = 0;
insl = bwareaopen(itot, 10);
L = bwlabel(insl, 8);
STATS = regionprops(logical(L),'BoundingBox');
thresh = 0;

% remove long & thin bb's probs straight lines
% imshow(L)
for p = 1:size(STATS),
    bb = STATS(p).BoundingBox;
    if bb(3) > thresh && bb(4) < bb(3)/10
        L(L==p) = 0;
%         rectangle('Position', bb, 'EdgeColor', 'green');
    elseif bb(4) > thresh && bb(3) < bb(4)/10
        L(L==p) = 0;
%         rectangle('Position', bb, 'EdgeColor', 'blue');
    else
%         rectangle('Position', bb, 'EdgeColor', 'red');
    end
end

% we also use edge density check
H = fspecial('gaussian',[round(rs/5) round(cs/5)],5);
tm = imfilter(double(logical(L)), H,'same'); % blur it a tonne
maxV = max(max(tm)); % get the highest value
allMaxValsMask = tm==maxV;
[R,C] = ind2sub(size(allMaxValsMask), find(allMaxValsMask));

```

```
SE = strel('disk',5,4);
dx = imdilate(allMaxValsMask, SE);

% Finding the entire number plate
% Segmentation
sx = strel('line', 3, 0);
sy = strel('line', 3, 90);
ts = imclose(is, sx); % attempt to complete unfinished/broken lines
ts = imclose(ts, sy);
holes = imfill(ts, 'holes'); % find cut off areas
s1 = strel('disk', 5, 0);
t1 = imerode(holes, s1);
F = bwlabel(t1);
np = F(R,C);
Mask = F==np;
% imshow(Mask);

% We want a nice rectangle
stats = regionprops(Mask, 'BoundingBox');
cropper = i;
cropper = imcrop(cropper, stats(1).BoundingBox);
subplot(2,2,2), imshow(cropper);
subplot(2,2,1), imshow(i), rectangle('Position', stats(1).BoundingBox, 'EdgeColor', 'r');

% Now to try OCR?
% load all the alpha numeric character we have.
imgPath = 'Character Images/';
imgType = '*.bmp'; % change based on image type
images  = dir([imgPath imgType]);
for idx = 1:length(images)
    Seq{idx} = imread([imgPath images(idx).name]);
end

txt = imcomplement(im2bw(cropper, 0.6));
      txt = bwareaopen(txt, 20);
      statsofletters = regionprops(txt, 'BoundingBox');
      subplot(2,2,3),imshow(txt);
      lets = {};
      for idx = 1:length(statsofletters),
          tmp = statsofletters(idx).BoundingBox;
          if tmp(3) < tmp(4)
              rectangle('Position', tmp, 'EdgeColor', 'r');
              bestLetter = Seq{1}; bestVal = 0;
              for template = 1:length(Seq),
                  cross = normxcorr2(imcrop(txt, tmp),imresize(Seq{template},[tmp(4)+1 tmp(3)
  +1]));
                  if max(max(cross)) > bestVal
                      bestLetter = Seq{template};
                      bestVal = max(max(cross));
                      templateNum = template;
                  end
              end
              lets = [lets; templateNum];
          end
      end

      String = '';
      for idx = 1:length(lets),
          hemp = images(lets{idx}).name(1);
          String = [String hemp];
      end

      String
      subplot(2,2,4), text(0,0.5,String,'fontsize',30,'color','k')
      axis off
      % close all
      % imshow(imfuse(dx,imresize(ig,size(dx))))
```