Licence Plate Recognition

Design of Visual Systems Final Project

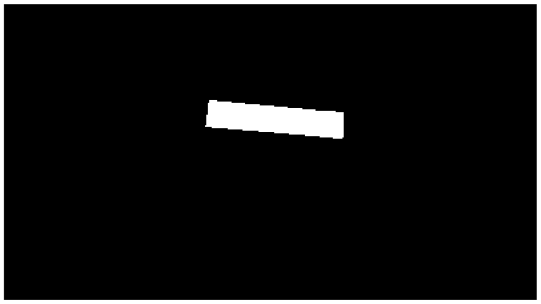
Clemence Sulmont & Ciara Bates

Section 1 : Open Image

```
% Clear workspace
clear; clc;

% Load and show the image of the front of a car
f = imread("car_img/cars34.png");
imshow(f)
title('Initial Image');
```


```
% The image must be greyscale for all processing, thsi also makes the code run
% faster as each pixel has only 1 intensity value rather than colour information.
img = im2double(rgb2gray(f));

% The image must be rotated slightly for the corner recognition function
% less likely to find corners that are in the exact same row and/or column
img = imrotate(img, -5, 'bilinear', 'crop');

% Removing (cropping) the top third of the image (windshield)
[height,width, numChannels] = size(img);
startrow = floor(height/3) + 1;
img = img(startrow:end, :, :);
imshow(img)
```


```
output = getCornersLicense(img);
```

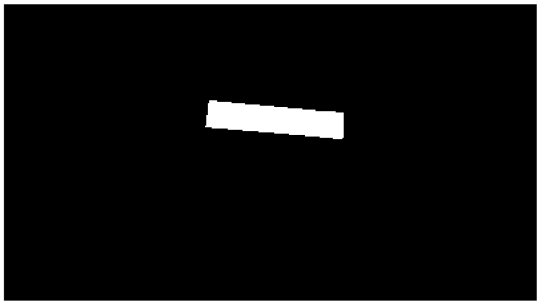


```
fourcorners = 4×2 single matrix
  155.6535   73.9197
  255.0000   82.0000
  253.0963  100.2300
  153.0000   90.0000
```

```
min_x = floor(min(output(:, 1)));
max_x = floor(max(output(:, 1)));
min_y = floor(min(output(:, 2)));
max_y = floor(max(output(:, 2)));

padding = 30;

% Construct the rectangle vector for cropping an image
RECT_original = [min_x-padding, min_y - padding, max_x - min_x + padding*2, max_y -
min_y + padding*2];

cropped_1 = imcrop(img,RECT_original);
imshow(cropped_1)
```

Section 2 : Homography Transform

```
% Get 4 input points for the homography transform
output = getCornersLicense(img);
```



```
fourcorners = 4×2 single matrix
  155.6535   73.9197
  255.0000   82.0000
  253.0963  100.2300
  153.0000   90.0000
```

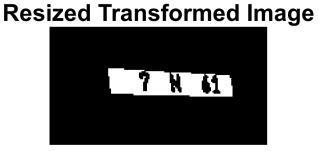
```
xi = output(:,1);
yi = output(:,2);

% Define the base points for the transformation
base = [1 1; 1 size(cropped_1, 1); size(cropped_1, 2) 1; size(cropped_1, 2)
size(cropped_1, 1);];


% Compute the homography transform
tf = fitgeotrans([xi, yi], base, 'projective');
H = tf.T;

% Perform the homography transformation
[xf, xf_ref] = imwarp(cropped_1, tf);
```

```
% Specify the desired width and height of the output image (in pixels)
desired_width = size(cropped_1, 2);
desired_height = size(cropped_1, 1);

% Resize the output image while maintaining its aspect ratio
xf_resized = imresize(xf, [desired_height, desired_width]);

% Display the resized transformed image
figure;
imshow(xf_resized);
hold on;
title('Resized Transformed Image');
```


```

```

Section 3 : Cropping the image (again)

```
corners = getCornersLicense(xf_resized);
```




```
fourcorners = 4×2 single matrix
    45    32
   137    52
   136    52
    45    32
```

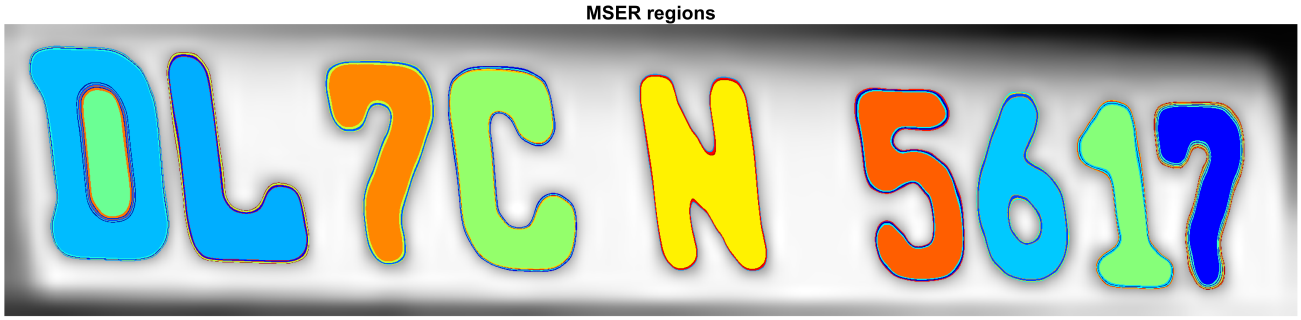
```
min_x = floor(min(corners(:, 1)));
max_x = floor(max(corners(:, 1)));
min_y = floor(min(corners(:, 2)));
max_y = floor(max(corners(:, 2)));

% Construct the rectangle vector for cropping an image
RECT = [min_x, min_y, max_x - min_x, max_y - min_y];
```

```
cropped = imcrop(xf_resized,RECT);

figure
imshow(cropped)
title('cropped image')
```

Section 4 : Segmenting Letters using MSER regions

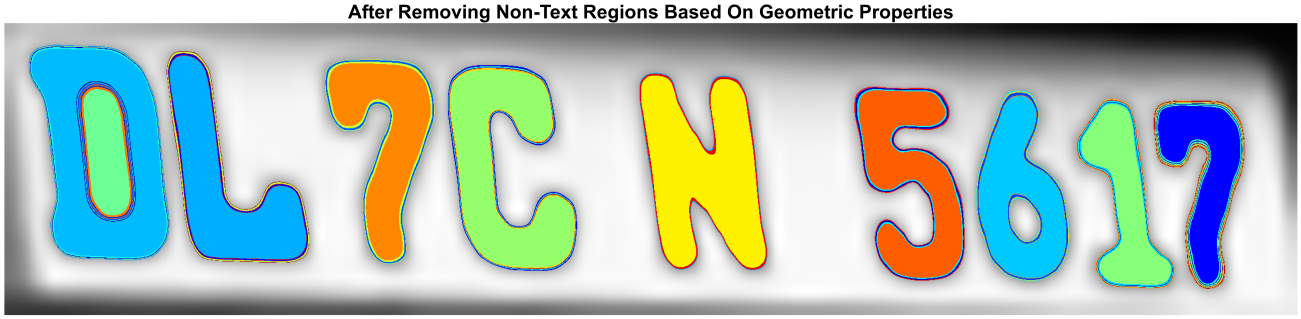
```
I = imresize(cropped,[500 NaN]); %resize so the number of rows of pixels is 500.
%The number of columns is automatically calculated based on aspect ratio



imshow(I)
```


```
% Detect MSER regions.
[mserRegions, mserConnComp] = detectMSERFeatures(I, ...
    "RegionAreaRange",[10000 60000],"ThresholdDelta",5);

figure
imshow(I)
hold on
plot(mserRegions, "showPixelList", true,"showEllipses",false)
title("MSER regions")
hold off
```

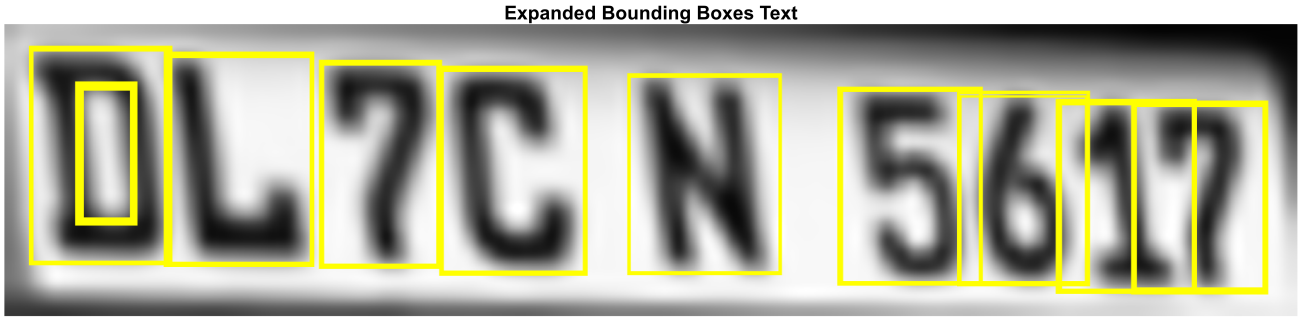



```
% Use regionprops (region properties) to measure MSER properties
mserStats = regionprops(mserConnComp, "BoundingBox", "Eccentricity", ...
    "Solidity", "Extent", "Euler", "Image");

% Compute the aspect ratio using bounding box data.
bbox = vertcat(mserStats.BoundingBox);
w = bbox(:,3);
h = bbox(:,4);
aspectRatio = w./h;

% Threshold the data to determine which regions to remove. These thresholds
% may need to be tuned for other images.
filterIdx = aspectRatio' > 3;
filterIdx = filterIdx | [mserStats.Eccentricity] > .995 ;
filterIdx = filterIdx | [mserStats.Solidity] < .3;
filterIdx = filterIdx | [mserStats.Extent] < 0.2 | [mserStats.Extent] > 0.9;
filterIdx = filterIdx | [mserStats.EulerNumber] < -4;

% Remove regions
mserStats(filterIdx) = [];
mserRegions(filterIdx) = [];

% Show remaining regions
figure
imshow(I)
hold on
plot(mserRegions, "showPixelList", true,"showEllipses",false)
title("After Removing Non-Text Regions Based On Geometric Properties")
hold off
```

After Removing Non-Text Regions Based On Geometric Properties

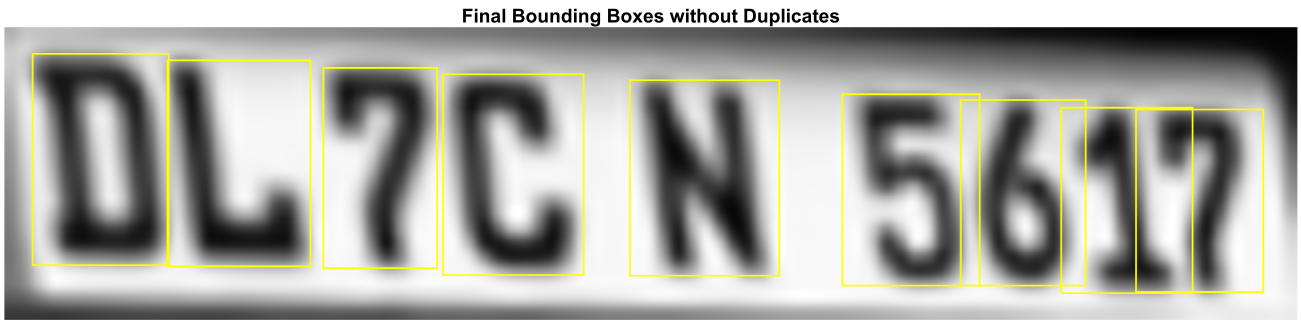



```
%----------------

% Get bounding boxes for all the regions
bboxes = vertcat(mserStats.BoundingBox);

% Convert from the [x y width height] bounding box format to the [xmin ymin
% xmax ymax] format for convenience.
xmin = bboxes(:,1);
ymin = bboxes(:,2);
xmax = xmin + bboxes(:,3) - 1;
ymax = ymin + bboxes(:,4) - 1;

% Expand the bounding boxes by a small amount.
expansionAmount = 0.02;
xmin = (1-expansionAmount) * xmin;
ymin = (1-expansionAmount) * ymin;
xmax = (1+expansionAmount) * xmax;
ymax = (1+expansionAmount) * ymax;

% Clip the bounding boxes to be within the image bounds
xmin = max(xmin, 1);
ymin = max(ymin, 1);
xmax = min(xmax, size(I,2));
ymax = min(ymax, size(I,1));

% Show the expanded bounding boxes
expandedBBoxes = [xmin ymin xmax-xmin+1 ymax-ymin+1];
IExpandedBBoxes = insertShape(I,"rectangle",expandedBBoxes,"LineWidth",3);

figure
imshow(IExpandedBBoxes)
title("Expanded Bounding Boxes Text")
```




```
%-------------

% Initialize an empty array to store indices of unique bounding boxes
uniqueIndices = [];

% Calculate the area of each bounding box
areas = (xmax - xmin + 1) .* (ymax - ymin + 1);

% Iterate over each bounding box
for i = 1:size(expandedBBoxes, 1)
    % Flag to indicate if current bounding box is unique
    isUnique = true;

    % Iterate over already identified unique bounding boxes
    for j = 1:length(uniqueIndices)
        % Calculate intersection area
        x_overlap = max(0, min(xmax(i), xmax(uniqueIndices(j))) - max(xmin(i),
xmin(uniqueIndices(j))));
        y_overlap = max(0, min(ymax(i), ymax(uniqueIndices(j))) - max(ymin(i),
ymin(uniqueIndices(j))));
        intersection_area = x_overlap * y_overlap;

        % Calculate area ratio (intersection over min area)
        area_ratio = intersection_area / min(areas(i), areas(uniqueIndices(j)));

        % If area ratio is above a threshold, consider the bounding box duplicate
        if area_ratio > 0.5 % Adjust threshold as needed
            isUnique = false;
            break;
        end
    end

    % If current bounding box is unique, add its index to uniqueIndices
    if isUnique
        uniqueIndices = [uniqueIndices, i];
    end
end
```

```
% Extract unique bounding boxes
uniqueBBoxes = expandedBBoxes(uniqueIndices, :);

% Show the final bounding boxes after removing duplicates
IFinalBBoxes = insertShape(I, "rectangle", uniqueBBoxes, "LineWidth", 3);

figure
imshow(IFinalBBoxes)
title("Final Bounding Boxes without Duplicates")
```




```
%---------------------

% Initialize cell array to store individual letter ROIs and their corresponding
bounding boxes
letterROIs = cell(size(uniqueBBoxes, 1), 1);
orderedBBoxes = uniqueBBoxes; % Copy bounding boxes for sorting

% Sort bounding boxes by their x-coordinates (xmin) This is so they are
% read and displayed in the correct order
[~, order] = sort(orderedBBoxes(:, 1));
orderedBBoxes = orderedBBoxes(order, :);

% Iterate over each bounding box
for i = 1:size(orderedBBoxes, 1)
    % Extract ROI using bounding box coordinates
    xmin = orderedBBoxes(i, 1);
    ymin = orderedBBoxes(i, 2);
    width = orderedBBoxes(i, 3);
    height = orderedBBoxes(i, 4);
    roi = imcrop(I, [xmin, ymin, width, height]);

    roi = imresize(roi,[100,NaN]); % resize the image to height 400px
    roi = imbinarize(roi); % Binarize the image


    % Define the desired dimensions for the resized image
```

```
    desired_height = 80;
    desired_width = 60;

    % Calculate the padding amounts for each side
    pad_height = max(0, desired_height - size(roi, 1));
    pad_width = max(0, desired_width - size(roi, 2));
    pad_top = floor(pad_height / 2);
    pad_bottom = ceil(pad_height / 2);
    pad_left = floor(pad_width / 2);
    pad_right = ceil(pad_width / 2);

    % Pad the image evenly around the whole image with ones
    padded_image = padarray(roi, [pad_top, pad_left], 1, 'pre');
    padded_image = padarray(padded_image, [pad_bottom, pad_right], 1, 'post');

    %padded_image = imresize(padded_image,[200,150]); %ensuring the image size is
correct

    clean_letter = imcomplement(padded_image);
    CC = bwconncomp(clean_letter);
    numpixels = cellfun(@numel, CC.PixelIdxList);
    [biggest, idx] = max(numpixels);

    for k = 1:length(numpixels)
        if k~= idx
            clean_letter(CC.PixelIdxList{k}) = 0;
        end
    end

    % Store preprocessed ROI
    letterROIs{i} = imcomplement(clean_letter);

end

figure;

% Display each ROI in a subplot, in order of when they appear in the image
% (left to right)
for i = 1:length(letterROIs)

    % Create subplot with adjusted aspect ratio
    subplot(1, length(letterROIs), i);

    % Display ROI
    imshow(letterROIs{i})

    % Set title
    title(['ROI ' num2str(order(i))]); % Display original order
end
```

Section 5 : Apply MATLAB's OCR function

```
% Apply OCR
licence_plate_ocr = "";

for i = 1:length(letterROIs)

    letter = letterROIs{i};

    figure
    imshow(letter)

    results = ocr(letter, 'TextLayout',
'Character','CharacterSet','ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890l#');
    disp(results)
    licence_plate_ocr = licence_plate_ocr + results.Text

end
```

```
  ocrText with properties:

                      Text: ''
    CharacterBoundingBoxes: [0×4 double]
      CharacterConfidences: [0×1 single]
                     Words: {0×1 cell}
        WordBoundingBoxes: [0×4 double]
          WordConfidences: [0×1 single]
                TextLines: {0×1 cell}
     TextLineBoundingBoxes: [0×4 double]
       TextLineConfidences: [0×1 single]
licence_plate_ocr =
""
```

```
  ocrText with properties:

                      Text: 'L'
    CharacterBoundingBoxes: [3 1 65 98]
      CharacterConfidences: 0.7267
                     Words: {'L'}
        WordBoundingBoxes: [3 1 65 98]
          WordConfidences: 0
                TextLines: {'L'}
     TextLineBoundingBoxes: [3 1 65 98]
       TextLineConfidences: 0
licence_plate_ocr =
"L"
```

```
  ocrText with properties:

                      Text: '1'
    CharacterBoundingBoxes: [4 1 52 99]
      CharacterConfidences: 0.9911
```

```
                    Words: {'1'}
          WordBoundingBoxes: [4 1 52 99]
            WordConfidences: 0.9378
                  TextLines: {'1'}
      TextLineBoundingBoxes: [4 1 52 99]
        TextLineConfidences: 0.9378
licence_plate_ocr =
"L1"
```

```
  ocrText with properties:

                      Text: 'C'
    CharacterBoundingBoxes: [5 1 61 98]
      CharacterConfidences: 0.9449
                    Words: {'C'}
         WordBoundingBoxes: [5 1 61 98]
           WordConfidences: 0.6144
                 TextLines: {'C'}
     TextLineBoundingBoxes: [5 1 61 98]
       TextLineConfidences: 0.6144
licence_plate_ocr =
"L1C"
```

```
  ocrText with properties:

                      Text: 'N'
    CharacterBoundingBoxes: [8 2 61 96]
      CharacterConfidences: 0.9493
                    Words: {'N'}
         WordBoundingBoxes: [8 2 61 96]
           WordConfidences: 0.6454
                 TextLines: {'N'}
     TextLineBoundingBoxes: [8 2 61 96]
       TextLineConfidences: 0.6454
licence_plate_ocr =
"L1CN"
```

```
  ocrText with properties:

                      Text: ''
    CharacterBoundingBoxes: [0×4 double]
      CharacterConfidences: [0×1 single]
                     Words: {0×1 cell}
        WordBoundingBoxes: [0×4 double]
          WordConfidences: [0×1 single]
                TextLines: {0×1 cell}
     TextLineBoundingBoxes: [0×4 double]
       TextLineConfidences: [0×1 single]
licence_plate_ocr =
"L1CN"
```

```
  ocrText with properties:

                      Text: ''
    CharacterBoundingBoxes: [0×4 double]
      CharacterConfidences: [0×1 single]
                     Words: {0×1 cell}
        WordBoundingBoxes: [0×4 double]
          WordConfidences: [0×1 single]
                TextLines: {0×1 cell}
     TextLineBoundingBoxes: [0×4 double]
       TextLineConfidences: [0×1 single]
licence_plate_ocr =
"L1CN"
```

```
  ocrText with properties:

                      Text: '1'
    CharacterBoundingBoxes: [11 1 50 98]
      CharacterConfidences: 0.9911
```

```
                    Words: {'1'}
          WordBoundingBoxes: [11 1 50 98]
            WordConfidences: 0.9375
                  TextLines: {'1'}
      TextLineBoundingBoxes: [11 1 50 98]
        TextLineConfidences: 0.9375
licence_plate_ocr =
"L1CN1"
```

```
  ocrText with properties:

                      Text: '1'
    CharacterBoundingBoxes: [12 1 47 99]
      CharacterConfidences: 0.9923
                     Words: {'1'}
         WordBoundingBoxes: [12 1 47 99]
           WordConfidences: 0.9459
                 TextLines: {'1'}
     TextLineBoundingBoxes: [12 1 47 99]
       TextLineConfidences: 0.9459
licence_plate_ocr =
"L1CN11"
```

```
disp('License number = ')
```

```
License number =
```

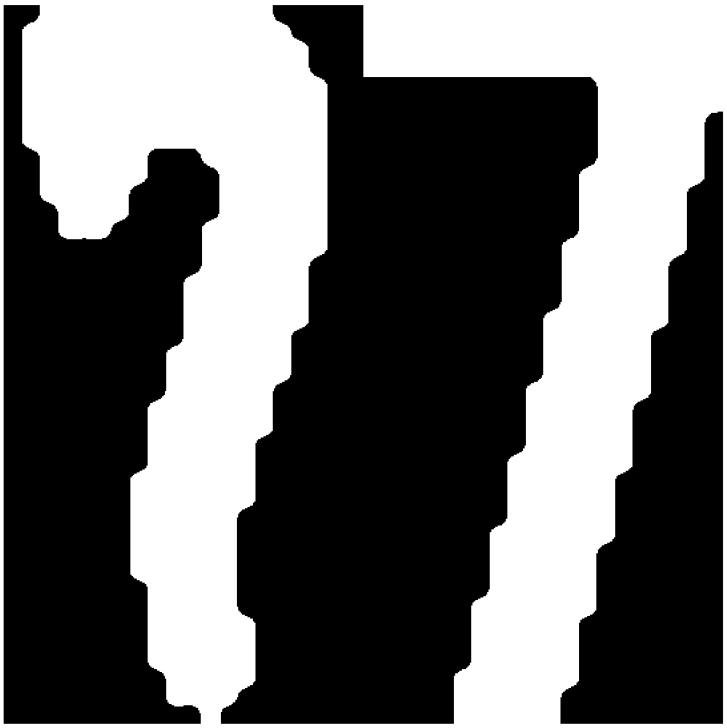
```
disp(licence_plate_ocr)
```

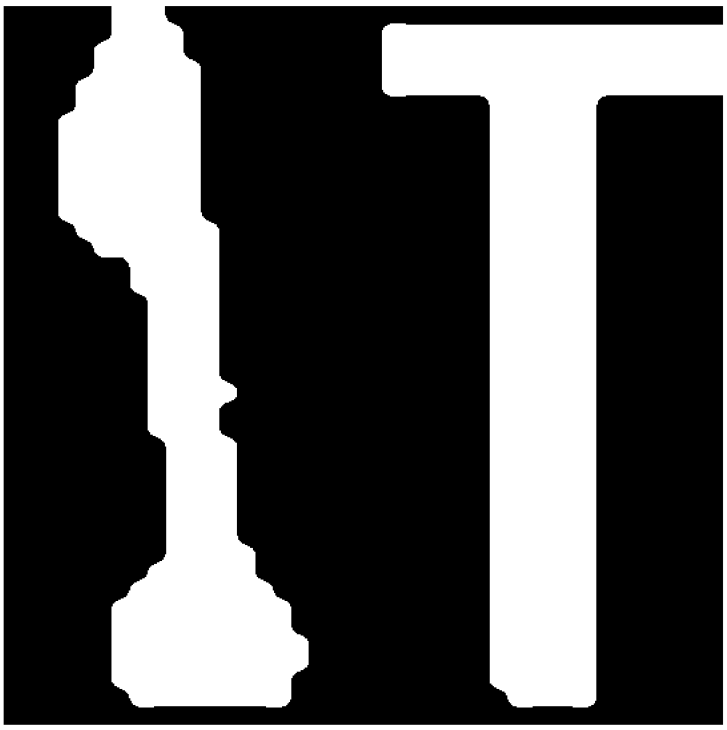
```
L1CN11
```

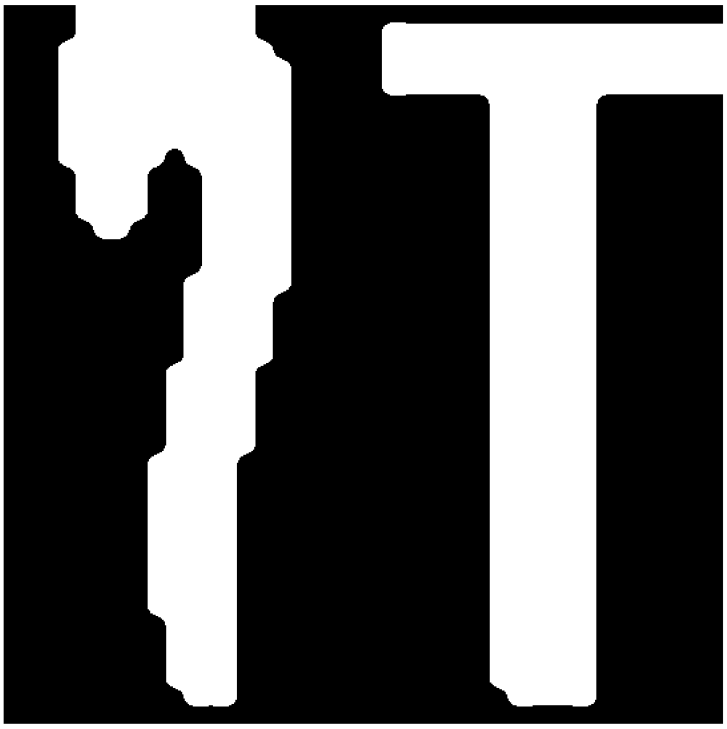
Section 6 : Detecting letters using similarity

```
% Define the folder path
folder_path = 'templates';

% Get a list of all PNG files in the folder
templates = dir(fullfile(folder_path, '*.png'));

candidateImage = cell(length(templates),2);

for p=1:length(templates)
    [~,fileName] = fileparts(templates(p).name);
    candidateImage{p,1} = fileName;
    candidateImage{p,2} = imread(fullfile(templates(p).folder,templates(p).name));
```

```
end

licenseNumber = '';

for p=1:length(letterROIs)
    letterImage = imcomplement(letterROIs{p});

        % Compare to templates
        distance = zeros(1,length(templates));
        for t=1:length(templates)
            letterImage = imresize(letterImage,size(candidateImage{t,2}));
            distance(t) = abs(sum((letterImage-
double(candidateImage{t,2})).^2,"all"));
        end

        [d,idx] = min(distance);
        letter = candidateImage{idx,1};

        if strcmp(letter, 'map')
            letter = '~';
        end

        figure
        montage({letterImage,candidateImage{idx,2}})

        licenseNumber(end+1) = letter;
end
```







```
disp('License number = ')
```

```
License number =
```

```
disp(licenseNumber)
```

```
DL7CXS4TT
```

```
function sortedcorners = getCornersLicense(grayImg)
    %Binarize image and isolate the license plate
    f3 = imadjust(grayImg, [0.5 1], [0 1]);
    f3 = imbinarize(f3);
    f3 = imclearborder(f3);
    CC = bwconncomp(f3);

    numPixels = cellfun(@numel, CC.PixelIdxList);
    [biggest, idx] = max(numPixels);
    for k = 1:length(numPixels)
        if k ~= idx
            f3(CC.PixelIdxList{k}) = 0;
        end
```

```
    end

    %Hide the letters to get better corner detection
    f4 = imfill(f3, "holes");

    imshow(f4)

    %Detect the 4 corners of the license plate
    points = detectBRISKFeatures(f4);
    mainpoints = points.selectStrongest(30).Location;

    [toppoint, topidx] = min(mainpoints(:,2));
    [bottompoint, bottomidx] = max(mainpoints(:,2));
    [leftpoint, leftidx] = min(mainpoints(:,1));
    [rightpoint, rightidx] = max(mainpoints(:,1));

    fourcorners = [mainpoints(topidx,:); mainpoints(rightidx,:);
mainpoints(bottomidx,:); mainpoints(leftidx,:)]

    rowSums = sum(fourcorners,2);
    % Sort the sums, obtaining the sorted indices
    [~, sortedIndices] = sort(rowSums);
    % Use the indices to reorder the matrix rows
    sortedcorners = fourcorners(sortedIndices, :);
end
```