Signals and Systems

Project Report

Parsa KafshduziBukani 810102501

CA2

Section 1

Part 1:

```
% part 1
[filename, pathname] = uigetfile({'*.jpg;*.png;*.jpeg;*.bmp', 'Image Files (*.jpg, *.png, *.jpeg, *.bmp)'}, ...
    'Select the license plate image');

img_path = fullfile(pathname, filename);
plate_img = imread(img_path);
```

Part 2:

```
% part 2
resized_img = imresize(plate_img, [300 500]);
```

Part 3:

```
% part 3
function gray_img = mygrayfun(image)
    R = double(image(:,:,1));
    G = double(image(:,:,2));
    B = double(image(:,:,3));

    gray_img = 0.299 * R + 0.578 * G + 0.114 * B;
    gray_img = gray_img / 255;
end

gray_img = mygrayfun(resized_img);
```

Part 4:

```
% part 4
function binary_img = mybinaryfun(gray_img, threshold)
    binary_img = gray_img > threshold;
end

threshold = 0.5;  % you can adjust this manually (e.g., 0.4–0.6)
binary_img = mybinaryfun(gray_img, threshold);

figure;
imshow(binary_img);
title(['Binary Image (threshold = ', num2str(threshold), ')']);
```

Part 5:

After thresholding, the binary image may contain small noisy regions that do not correspond to actual characters. We implemented a custom function, **myremovecom**, which uses Depth-First Search (**DFS**) to find all connected pixels in a component (4-connectivity) and removes any component smaller than a specified size. This cleans the image while preserving the main characters on the license plate.

```
% part 5
function cleaned = myremovecom(binary_img, n)
    [rows, cols] = size(binary_img);
    visited = false(rows, cols);
    cleaned = binary_img;

    for i = 1:rows
        for j = 1:cols
            % Find one connected component
            if binary_img(i, j) == 0 && ~visited(i, j)
                [component_pixels, visited] = dfs(binary_img, i, j, visited);

                % Remove it if smaller than threshold
                if size(component_pixels, 1) < n
                    for k = 1:size(component_pixels, 1)
                        cleaned(component_pixels(k,1), component_pixels(k,2)) = 1;
                    end
                end
            end
        end
    end
end
```

```
function [pixels, visited] = dfs(binary_img, x, y, visited)
    [rows, cols] = size(binary_img);
    stack = [x, y];
    pixels = [];

    while ~isempty(stack)
        [cx, cy] = deal(stack(end,1), stack(end,2));
        stack(end,:) = []; % pop

        if cx<1 || cx>rows || cy<1 || cy>cols
            continue;
        end
        if visited(cx,cy) || binary_img(cx,cy) == 1
            continue;
        end

        visited(cx,cy) = true;
        pixels = [pixels; cx, cy];

        % 4-connected neighbors
        stack = [stack; cx-1, cy];
        stack = [stack; cx+1, cy];
        stack = [stack; cx, cy-1];
        stack = [stack; cx, cy+1];
    end
end

min_size = 250;
clean_img = myremovecom(binary_img, min_size);
```

Part 6:

In this step, we created a custom function **mysegmentation** to label connected components in the binary image without using **bwlabel**. Using our previously defined **DFS** routine, each unvisited black pixel (0) starts a search that marks all connected pixels with the same label. The result, visualized via label2rgb, shows each segmented region in a different color — effectively separating the plate's individual characters.

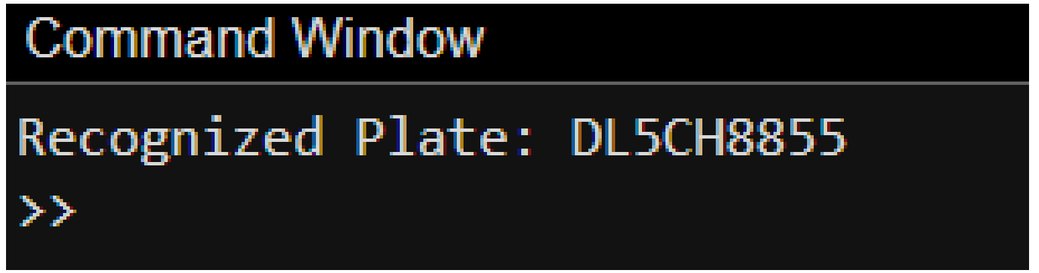
```
% part 6
function [labeled, num] = mysegmentation(binary_img)
    [rows, cols] = size(binary_img);
    visited = false(rows, cols);
    labeled = zeros(rows, cols);
    label = 0;

    for i = 1:rows
        for j = 1:cols
            % Found a new black component (pixel == 0)
            if binary_img(i, j) == 0 && ~visited(i, j)
                label = label + 1;
                [component_pixels, visited] = dfs(binary_img, i, j, visited);

                % Assign the same label to all connected pixels
                for k = 1:size(component_pixels, 1)
                    labeled(component_pixels(k,1), component_pixels(k,2)) = label;
                end
            end
        end
    end

    num = label; % number of segments
end

[L, num] = mysegmentation(clean_img);

colored_labels = label2rgb(L, 'jet', 'k', 'shuffle');
figure; imshow(colored_labels);
title(['Segmented Components: ', num2str(num)]);
```

Part 7 & 8:

In these steps, each segmented region from the labeled image was compared with stored character templates using correlation matching. For every segment, the one with the highest correlation value above a defined threshold was selected as the recognized symbol. All recognized characters were then concatenated in left-to-right order to form the complete license-plate text, which was displayed and saved to a .txt file for verification.

```
% part 7
function recognized_text = recognize_plate(segmented_img, num_segments, template_dir, threshold)
    recognized_text = '';

    % Find bounding boxes of each segment
    boxes = zeros(num_segments, 3); % [index, min_col, mean_row]
    for k = 1:num_segments
        [r, c] = find(segmented_img == k);
        if isempty(r), continue; end
        boxes(k, :) = [k, min(c), mean(r)];
    end

    % Sort by x-position (left to right)
    boxes = sortrows(boxes, 2);
    valid_indices = boxes(:, 1);

    % Character recognition
    for idx = valid_indices'
        mask = (segmented_img == idx);
        [r, c] = find(mask);
        if isempty(r), continue; end
        char_img = mask(min(r):max(r), min(c):max(c));
        char_img = imresize(char_img, [50 30]);

        best_corr = 0;
        best_char = '';
        templates = dir(fullfile(template_dir, '*.png'));
        templates = [templates; dir(fullfile(template_dir, '*.bmp'))];

        for t = 1:length(templates)
            temp = imread(fullfile(template_dir, templates(t).name));
            temp = imresize(temp, [50 30]);
            corr_val = corr2(double(char_img), double(temp));

            if corr_val > best_corr
                best_corr = corr_val;
                [~, name, ~] = fileparts(templates(t).name);
                best_char = upper(name);
            end
        end

        % Accept only if above threshold
        if best_corr >= threshold
            recognized_text = [recognized_text best_char];
        end
    end
end
```

```
detection_threshold = 0.4;
template_dir = 'p1/Map Set';
recognized_text = recognize_plate(L, num, template_dir, detection_threshold);

% part 8
fid = fopen('recognized_plate.txt', 'w');
fprintf(fid, '%s\n', recognized_text);
fclose(fid);

disp(['Recognized Plate: ', recognized_text]);
```



Section 2

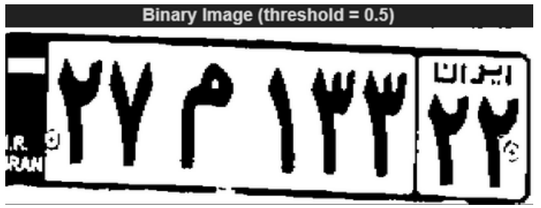
In this section, the same image-processing pipeline was extended to recognize Persian (Iranian) license plates.
Unlike the previous part, no database was provided, so a custom template set was created containing Persian letters (ج، س، ص، ط، ق، ل، م، ن، و، ه، ی) and Persian digits (۰–۹).
The overall procedure remained the same — preprocessing, segmentation, correlation-based matching, and text reconstruction — but with three key adjustments:

1. Image resizing:
The input plate image was resized to *200 × 600* pixels for better normalization of Persian plate proportions.

2. Noise removal threshold:
The minimum connected-component size was increased to *400 pixels (min_size = 400)* to remove small specks and isolate clear character regions.

3. Output formatting:
Because mixed Persian–Latin text caused right-to-left display issues, a conversion function was added to map all recognized Persian digits and letters back to their English equivalents.

```
% part 8
recognized_text_en = fa_plate_to_en(recognized_text);

fid = fopen('recognized_Persian_plate_EN.txt','w');
fprintf(fid, '%s\n', recognized_text_en);
fclose(fid);

disp(['Recognized IR Plate (in EN): ' recognized_text_en]);

function s_en = fa_plate_to_en(s)
    pairs = {
        '۰','0'; '۱','1'; '۲','2'; '۳','3'; '۴','4'; '۵','5'; '۶','6'; '۷','7'; '۸','8'; '۹','9';
        'ج','J'; 'س','S'; 'ص','S'; 'ط','T'; 'ق','Q'; 'ل','L'; 'م','M'; 'ن','N'; 'و','V'; 'ه','H'; 'ی','Y'
    };
    s_en = s;
    for k = 1:size(pairs,1)
        s_en = strrep(s_en, pairs{k,1}, pairs{k,2});
    end
end
```




```
Command Window
Recognized IR Plate (in EN): 27M13322
>>
```

Section 3

This method detects the license plate in a car's front image using RGB normalized cross-correlation (NCC) with a small plate template (origin.jpg). Each color channel (R, G, B) is correlated separately with the image, and the results are averaged to form a single correlation map. The maximum correlation point marks where the template best matches the image.
From this location, the code computes a red rectangle (exact match) and expands it to a larger green box to fully include the plate area. The box size is defined as width = 7 × tH and height = 2 × tH, tuned for larger plates in this dataset.
The cropped region inside the green box is saved as the plate image, which is then passed to the next processing stages for character recognition. This RGB-NCC approach is fast, color-aware, and effective for detecting large plates in frontal car images.

```
% Part 1 – Plate Detection

[filename, pathname] = uigetfile({'*.jpg;*.png;*.jpeg;*.bmp', ...
    'Image Files (*.jpg, *.png, *.jpeg, *.bmp)'}, ...
    'Select the license plate image');

img_path = fullfile(pathname, filename);
car_img = imread(img_path);

template = imread('origin.jpg');

car_img = im2double(imresize(car_img, [200, 300]));
template = im2double(imresize(template, [20, 10]));

corrR = normxcorr2(template(:,:,1), car_img(:,:,1));
corrG = normxcorr2(template(:,:,2), car_img(:,:,2));
corrB = normxcorr2(template(:,:,3), car_img(:,:,3));

corrMap = (corrR + corrG + corrB) / 3;

% Find the location of the best match
[maxVal, maxIdx] = max(corrMap(:));
[maxRow, maxCol] = ind2sub(size(corrMap), maxIdx);
```

```
% Compute matched region coordinates
tH = size(template, 1);
tW = size(template, 2);
row = maxRow - tH + 1;
col = maxCol - tW + 1;

% Define green rectangle region (plate estimation)
plate_box = [col-10, row-10, 7*tH, 2*tH];
plate_img = imcrop(car_img, plate_box);

figure('Position', [0, 0, 800, 400]);
imshow(car_img); hold on;
rectangle('Position', [col, row, tW, tH], 'EdgeColor', 'r', 'LineWidth', 2);
rectangle('Position', plate_box, 'EdgeColor', 'g', 'LineWidth', 2);
hold off;
title('Detected Plate Region');
```