“Counting Money" Based on MATLAB

Table of the content

Project Description

In this project, a template matching method based on correlation comparison was developed to solve the problem of banknote denomination recognition using MATLAB. This method applies denoising, edge extraction, contour closure, and rotation correction to the target image, followed by correlation matching with labelled templates to achieve project goal. A very precise recognition was shown by the method, allowing great localization of banknotes and recognition of denomination. Below is the testing process and sample images

1. **Grayscale and Denoising**
2. **Binarization Masking**
3. **Rotation Correction**
4. **Scale Correction**
5. **Correlation Comparison**

Figure 1: Ideal Template Image

Figure 2: Test Images

How to build

To build the packages, follow these steps:

1. Install MATLAB (In this project, we use R2021b as example)
2. Install Image Processing Toolbox
3. Download this project
4. Open the file named **project**

5. run this code

Documentation

Grayscale and Denoising

This section is mainly used to remove the pretzel noise from the image. Random pepper and salt noise on image is removed using 'midfil2' method in MATLAB. This method uses median filter to filter the extreme pixel points in the image, which is often used to remove this type of noise, and performs more efficiently than traditional convolutional methods.

```
%% Read image
%This section is mainly used to read and grayscale the image.
img = imread('search_noise.png');
img1 = rgb2gray(img); % change the image from rgb to gray
figure(1); imshow(img1); title('gray image');% show figure


%% Image Denoising
%This part is mainly used for image denoising
grayimg = medfilt2(img1, [5,5]); % Processing of images using median filtering
figure(2); imshow(grayimg); title('The denoised image');% show figure
```

imread

Read image from graphics file

`A = imread(filename)` reads the image from the file specified by `filename`, inferring the format of the file from its contents. If `filename` is a multi-image file, then `imread` reads the first image in the file.

rgb2gray

Convert RGB image or colormap to grayscale

`I = rgb2gray(RGB)` converts the truecolor image `RGB` to the grayscale image `I`. The `rgb2gray` function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance. If you have Parallel Computing Toolbox™ installed, `rgb2gray` can perform this conversion on a GPU.

example

```
img1 = rgb2gray(img); % change the image from rgb to gray
figure(1); imshow(img1); title('gray image');
```

medfilt2

Two-dimensional median filter

`J = medfilt2(I,[m n])` Perform median filtering, where each output pixel contains the median value in an **m×n** neighborhood around the corresponding pixel in the input image.

example

```
grayimg = medfilt2(img1, [5,5]); % Processing of images using median filtering
figure(2); imshow(grayimg); title('The denoised image');% show figure
```

Binarization Masking

The purpose of binarization is to lock out the location of the banknotes in the target image, this can be done by first extracting the edge of the image and then do the fill and finally select the largest of several connected images to get the mask, and then the use of the mask in the original image can frame the location of the target banknotes.
Where edge detection is done using MATLAB's 'edge' function using Canny method, which gets the edge of the image by finding the local maximum of gradient, which is resistant to interference as compared to other methods .
As for closing and filling the connected area, MATALB's **'imclose'** method and **'imfill'** method are used respectively, both methods use morphological methods to fill the area surrounded by the edges to get the connected area .

```
%% Binarization Masking
% Detecting image edges
BW = edge(grayimg, 'canny');
figure(3); imshow(BW); title('Canny edge detect result');

% Morphological processing
BW1 = imclose(BW, strel('disk', 25)); %closed the region
BW2 = bwareaopen(BW1, 100);% delete the noise from the enviroment
BW3 = imfill(BW2, 'holes');% fill the hole in the closed region
figure(4); imshow(BW3); title('morphological processing result');

% Connected domain analysis
[L, num] = bwlabel(BW3);
stats = regionprops(L, 'Area', 'PixelIdxList', 'BoundingBox', 'Orientation',
'Centroid');
[~, index] = sort([stats.Area], 'descend');
% Select the top three regions in terms of number of pixels (if they exist)
numRegions = min(3, length(stats));
```

---

edge

Finding edges in a 2D grayscale image

`BW = edge(I,method)` detects edges in image `I` using the edge-detection algorithm specified by `method`.

example

```
BW = edge(grayimg, 'canny'); %Find edges using the Canny method.
figure(3); imshow(BW); title('Canny edge detect result');
```

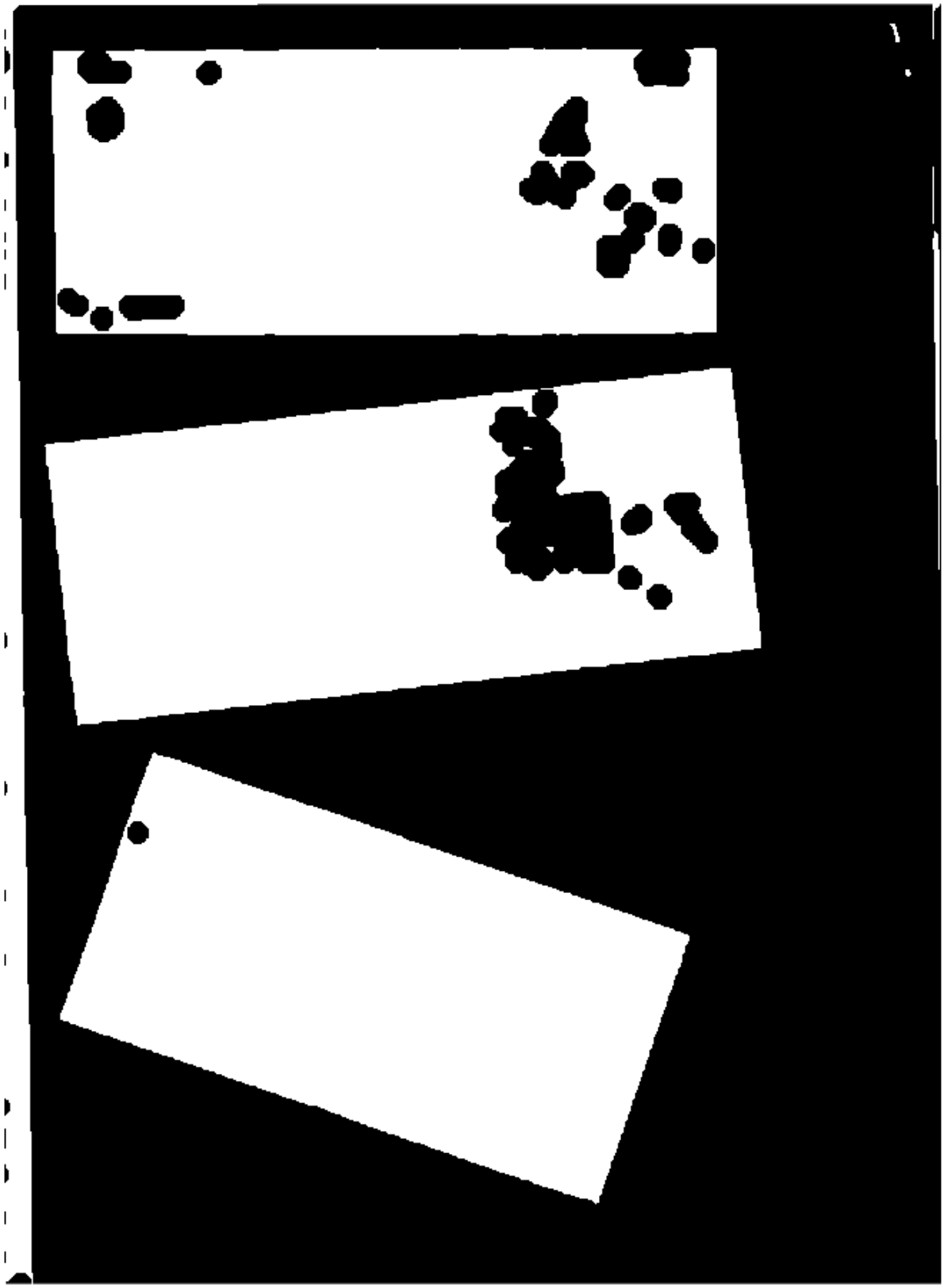
imclose

Morphologically close image

`J = imclose(I,SE)` performs morphological closing on the grayscale or binary image `I`, using the structuring element `SE`. The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations.

example

```
BW1 = imclose(BW, strel('disk', 25)); %closed the region
```



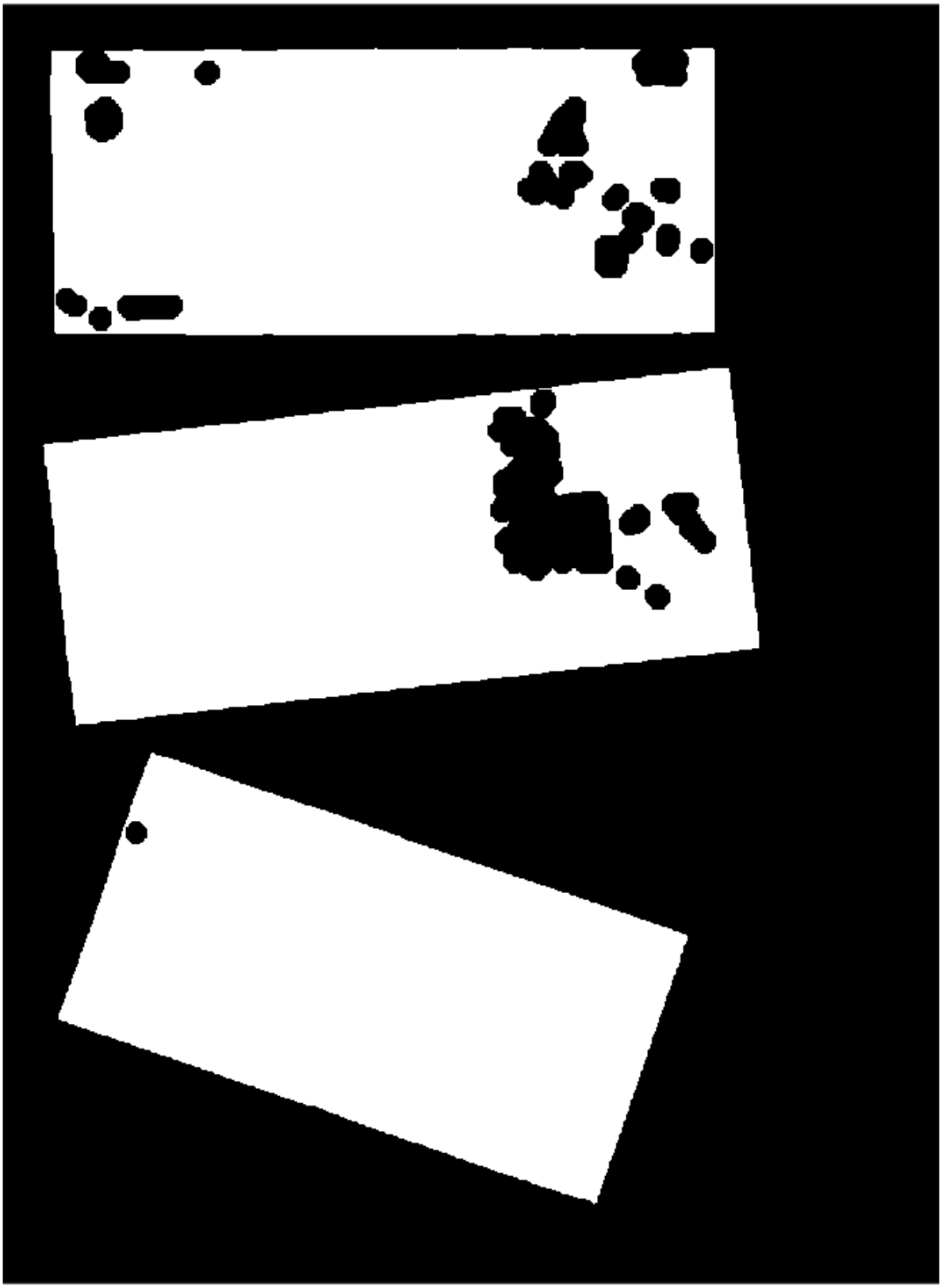
bwareaopen

Remove small objects from binary image

`BW2 = bwareaopen(BW,P)` removes all connected components (objects) that have fewer than `P` pixels from the binary image `BW`, producing another binary image, `BW2`. This operation is known as an *area opening*.

example

```
BW2 = bwareaopen(BW1, 100);% delete the noise from the enviroment
```



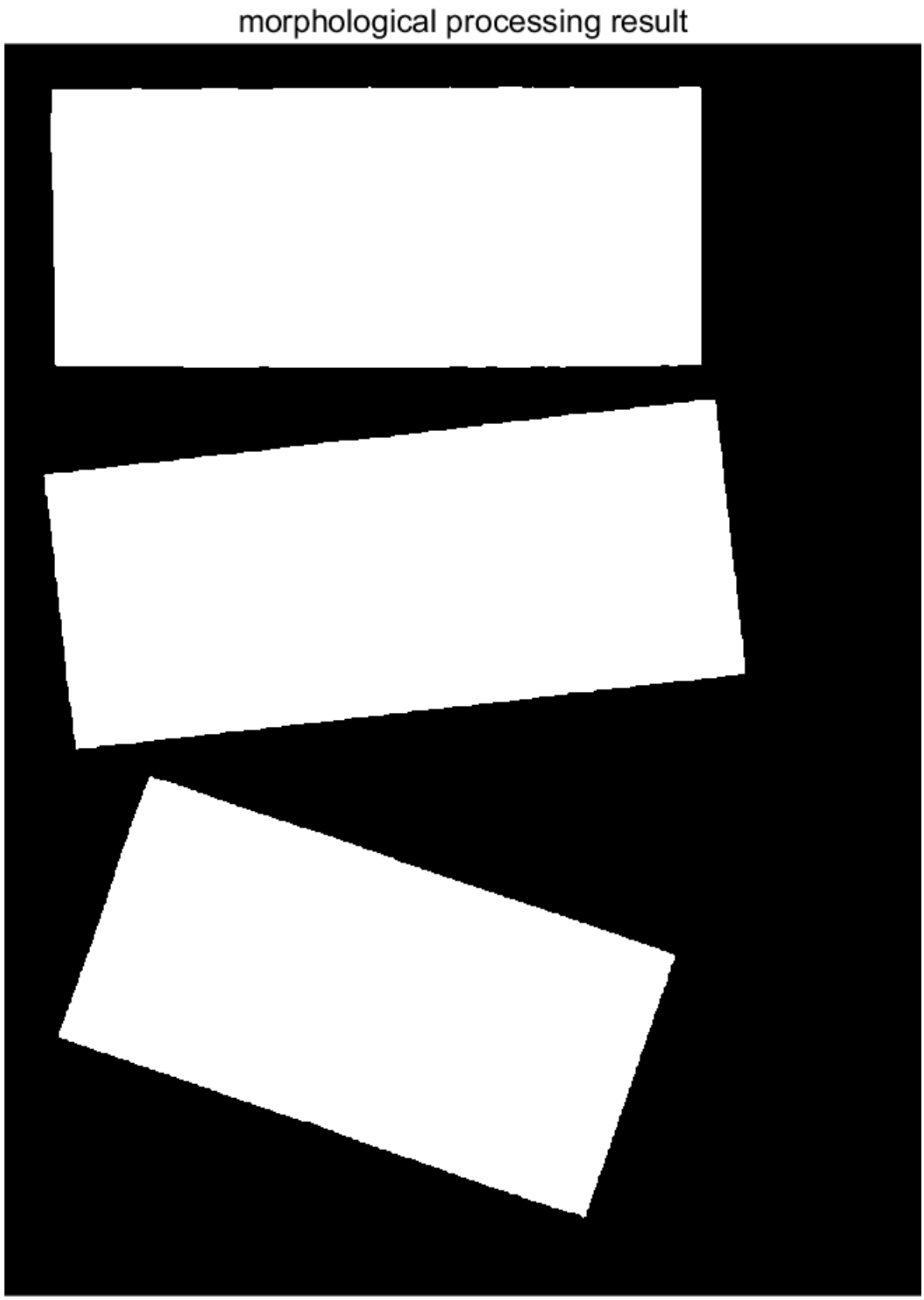
imfill

Fill image regions and holes

`BW2 = imfill(BW,"holes")` fills holes in the input binary image `BW`. In this syntax, a hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

example

```
BW3 = imfill(BW2, 'holes');% fill the hole in the closed region
```



bwlabel

Label connected components in 2-D binary image

`[L,n] = bwlabel(___)` returns a label matrix, where `conn` specifies the connectivity. also returns `n`, the number of connected objects found in `BW`.

example

```
[L, num] = bwlabel(BW3);
```

regionprops

The `regionprops` function measures properties such as area, centroid, and bounding box, for each object (connected component) in an image. `regionprops` supports both contiguous regions and discontiguous regions

example

```
stats = regionprops(L, 'Area', 'PixelIdxList', 'BoundingBox', 'Orientation',
'Centroid');
stats
```

```
stats =

  包含以下字段的 3×1 struct 数组:

    Area
    Centroid
    BoundingBox
    Orientation
    PixelIdxList
```

sort

Sort array elements

`B = sort(___,direction)` returns sorted elements of `A` in the order specified by `direction` using any of the previous syntaxes. `'ascend'` indicates ascending order (the default) and `'descend'` indicates descending order.

`[B,I] = sort(___)` also returns a collection of index vectors for any of the previous syntaxes. `I` is the same size as `A` and describes the arrangement of the elements of `A` into `B` along the sorted dimension. For example, if `A` is a vector, then `B = A(I)`.

example

```
[~, index] = sort([stats.Area], 'descend');
```

```
>> stats.Area

ans =

      975885


ans =

      944021


ans =

      809004
```

```
numRegions = min(3, length(stats));
```

The number of the region is depend on the min(3, length(stats)).

Rotation Correction

For each extracted target banknote region, the rotation angle of the banknote is obtained by combining horizontal orientation and direction of the region. The image angle of the target region is rotated to match the image in the standard template using MATLAB's 'imrotate' method .

```
%% Rotation Correction
figure;
for i = 1:numRegions
    % Get current area
    currentRegion = false(size(BW3));
    currentRegion(stats(index(i)).PixelIdxList) = true;

    % Get the direction of the area
    angle = stats(index(i)).Orientation;

    % rotation correction
    rotatedRegion = imrotate(currentRegion, -angle, 'bilinear', 'crop');
    rotatedImg = imrotate(img, -angle, 'bilinear', 'crop');

    % Find the exact boundary of the rotated area
    [rows, cols] = find(rotatedRegion);
    minRow = min(rows);
    maxRow = max(rows);
```

```
    minCol = min(cols);
    maxCol = max(cols);

    % Keying out the current area
    croppedImg = rotatedImg(minRow:maxRow, minCol:maxCol, :);

    % Resize to match template
    resizedImg = imresize(croppedImg, size(template_gray));

    % Storage area information
    regions{i}.image = croppedImg;
    regions{i}.resized = resizedImg;
    regions{i}.centroid = stats(index(i)).Centroid;
    regions{i}.bbox = stats(index(i)).BoundingBox;

    % Getting Boundaries
    regions{i}.boundary = bwboundaries(currentRegion);

    % Framing the area of interest on the rotationally corrected image
    subplot(2, numRegions, i);
    imshow(rotatedImg);
    hold on;
    rectangle('Position', [minCol, minRow, maxCol-minCol+1, maxRow-minRow+1],
...
              'EdgeColor', 'r', 'LineWidth', 2);
    text(minCol, minRow-10, sprintf('area %d', i), 'Color', 'red', ...
         'FontSize', 12, 'BackgroundColor', 'white');
    title(sprintf('ROI %d (Rotated)', i));
    hold off;

    % Show keyed area
    subplot(2, numRegions, i + numRegions);
    imshow(croppedImg);
    title(sprintf('Keying out the area %d', i));
end
```

imrotate

Rotate image

`J = imrotate(I,angle,method,bbox)` also uses the `bbox` argument to define the size of the output image. You can crop the output to the same size as the input image or return the entire rotated image.

example

```
rotatedRegion = imrotate(currentRegion, -angle, 'bilinear', 'crop');
rotatedImg = imrotate(img, -angle, 'bilinear', 'crop');
```

Scale Correction

Extracting the templates for standard banknotes from the template atlas is consistent with the methods in 2.1 and 2.2 and will not be discussed much here. The last thing that needs to be taken care of before performing the correlation comparison is to make sure that the resolution of the template is the same as the resolution of the target image, which can be done here using MATLAB's 'imresize' method .

```
for i = 1:length(denominations)
    % Get the bounding box of the current bill
    bbox = stats_template(index_template(i)).BoundingBox;

    % Crop out the current bill area
    template_regions{i} = imcrop(template_gray, bbox);

    % Resize to ensure all templates have the same dimensions
    template_regions{i} = imresize(template_regions{i}, [200, 400]);  % Can be
resized as needed

    subplot(1, length(denominations), i);
    imshow(template_regions{i});
    title(sprintf('$%d Template', denominations(i)));
end
```

imresize

Resize image

`B = imresize(A,scale)` returns image `B` that is `scale` times the size of image `A`. The input image `A` can be a grayscale, RGB, binary, or categorical image.

If `A` has more than two dimensions, then `imresize` only resizes the first two dimensions. If `scale` is between 0 and 1, then `B` is smaller than `A`. If `scale` is greater than 1, then `B` is larger than `A`. By default, `imresize` uses bicubic interpolation.

`B = imresize(A,[numrows numcols])` returns image `B` that has the number of rows and columns specified by the two-element vector `[numrows numcols]`.

example

```
resizedImg = imresize(croppedImg, size(template_gray));
```

imcrop

Crop image

`Icropped = imcrop(I,rect)` crops the image `I` according to the position and dimensions specified in the crop rectangle `rect`. The cropped image includes all pixels in the input image that are completely *or partially* enclosed by the rectangle.

The actual size of the output image does not always correspond exactly with the width and height specified by `rect`. For example, suppose `rect` is `[20 20 40 30]`, using the default spatial coordinate system. The upper left corner of the specified rectangle is the center of the pixel with spatial (*x*,*y*) coordinates (20,20). The lower right corner of the rectangle is the center of the pixel

with spatial (*x*,*y*) coordinates (60,50). The resulting output image has size 31-by-41 pixels, not 30-by-40 pixels.

example

```
template_regions{i} = imcrop(template_gray, bbox);
```

Correlation Comparison

The correlation was calculated using the 'corr2' method in MATLAB, which calculates the correlation between pixel matrices based on the following equation

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}}$$

```
figure;
best_match = zeros(1, numRegions);
for i = 1:numRegions
    subplot(1, numRegions, i);
    region_gray = rgb2gray(regions{i}.resized);
    imshow(region_gray);
    hold on;

    max_correlation = -inf;

    for j = 1:length(denominations)
        % Ensure that the area image and the template have the same dimensions
        resized_region = imresize(region_gray, size(template_regions{j}));

        % Calculating Correlation
        correlation = corr2(resized_region, template_regions{j});

        if correlation > max_correlation
            max_correlation = correlation;
            best_match(i) = j;
        end
    end
```

```
    title(sprintf('Area %d: $%d', i, denominations(best_match(i))));
    hold off;
end
```

corr2

2-D correlation coefficient

`R = corr2(A,B)` returns the 2-D correlation coefficient `R` between arrays `A` and `B`.

example

```
correlation = corr2(resized_region, template_regions{j});
```

This function is used to calculate the correlation between two figure. Through it we can find the match matrix.