The British University in Egypt (BUE)

Computer Engineering DY4

Computer Vision

(23COMP13H)

To Dr. Maryam Alberry and Eng. Marwa Raafat

Egyptian Currency Detector

Abdelrahman Mostafa 206395

Pierre Tamer 198987

Ali Taha 200212

Abdelrahman Almonzer 198194

Abdelrahman Ramadan 2000896

Introduction

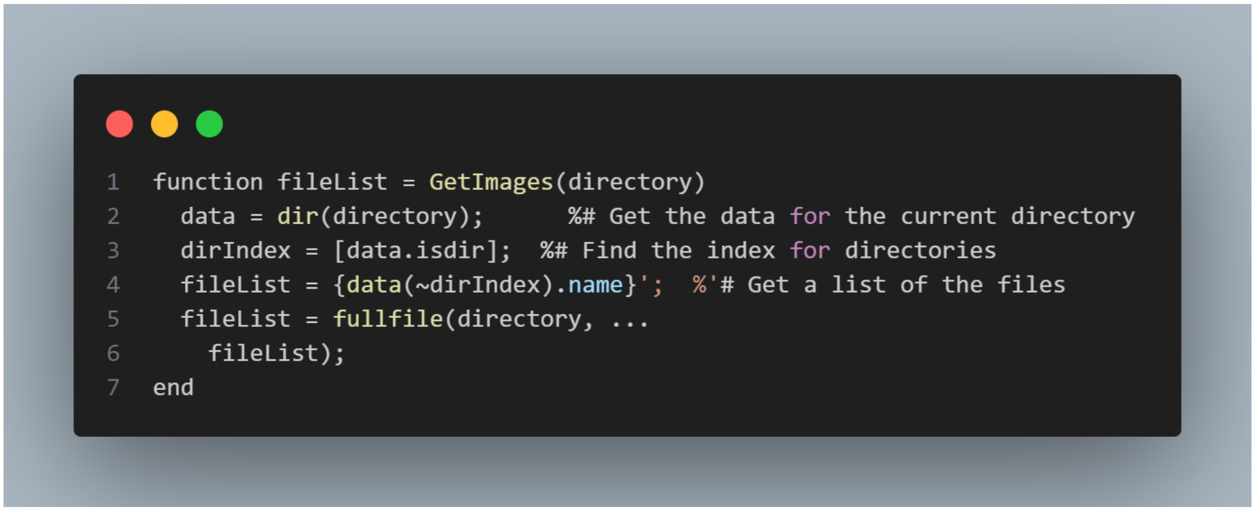
Our project focuses on implementing a computer vision module for the automated detection of Egyptian currency. In a world where technology continually evolves, the need for efficient and secure currency authentication is paramount. This project addresses this need by utilizing advanced computer vision techniques to recognize and verify various denominations of Egyptian banknotes.

For the implementation of our automated Egyptian currency detection project, we utilized MATLAB as the primary platform. MATLAB's image processing and computer vision toolbox provided a convenient and effective environment for developing recognition algorithms. Leveraging the capabilities of MATLAB, we were able to create a system that accurately identifies and verifies various denominations of Egyptian banknotes.

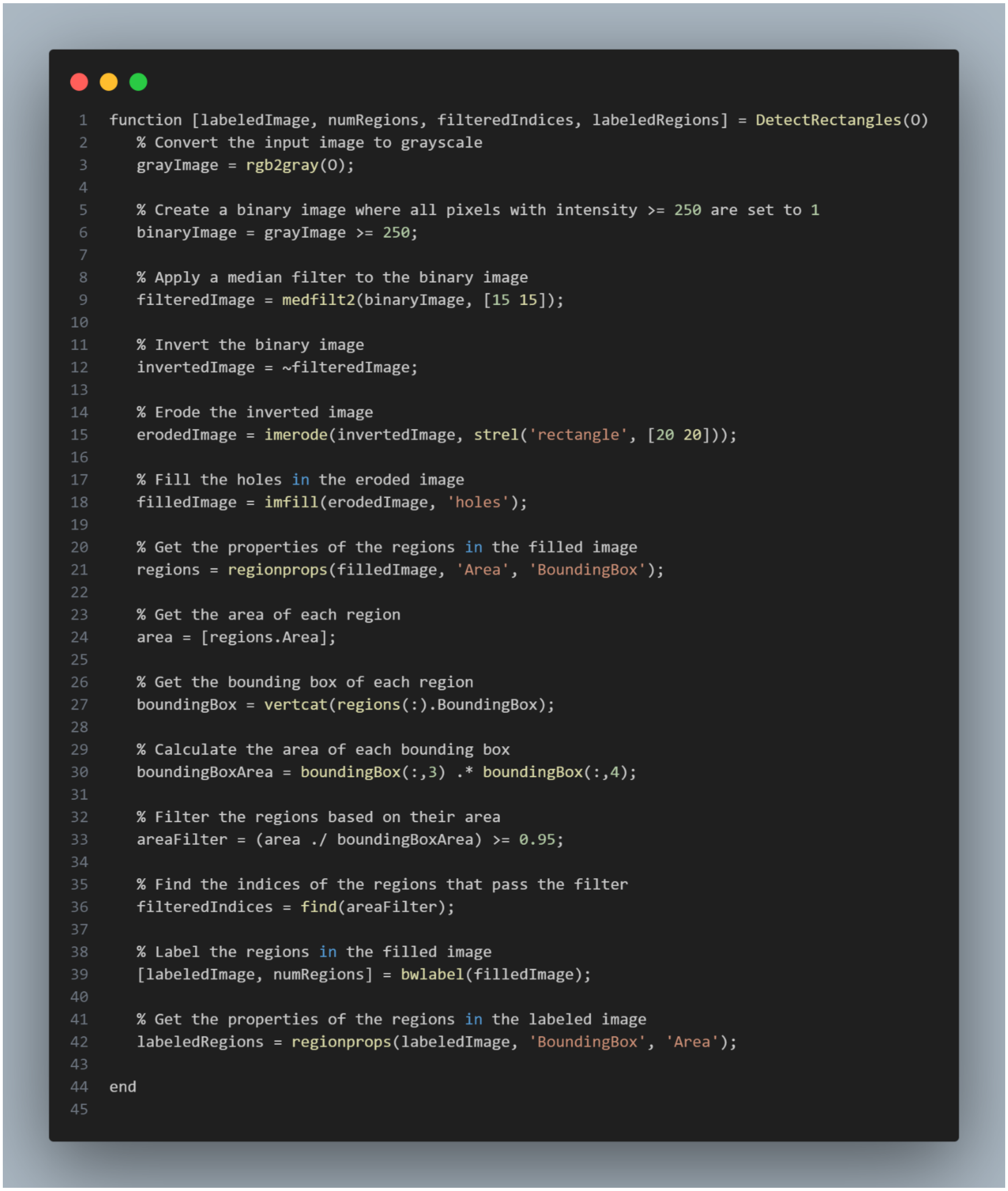
Results

Below are screenshots of the code implementation and images results.

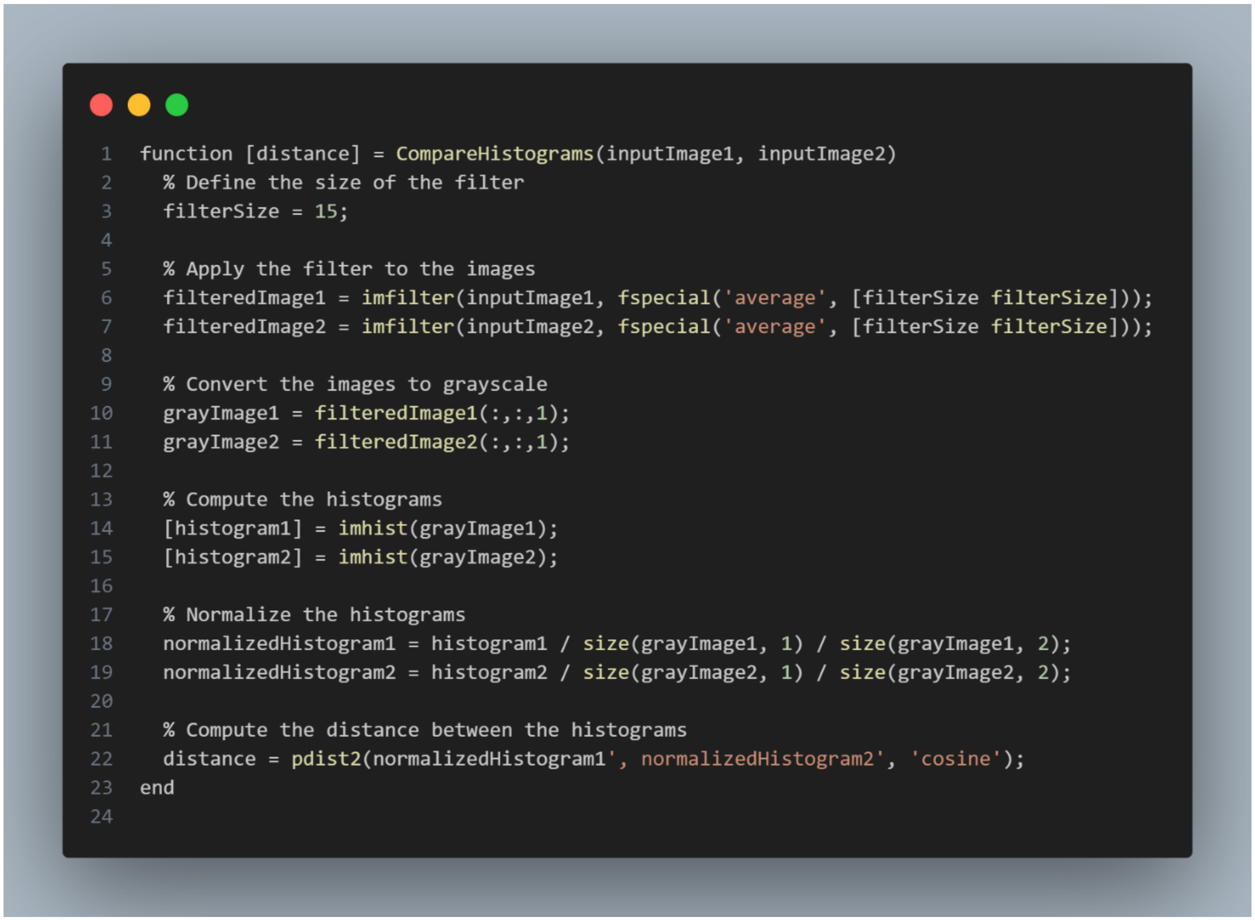
Code



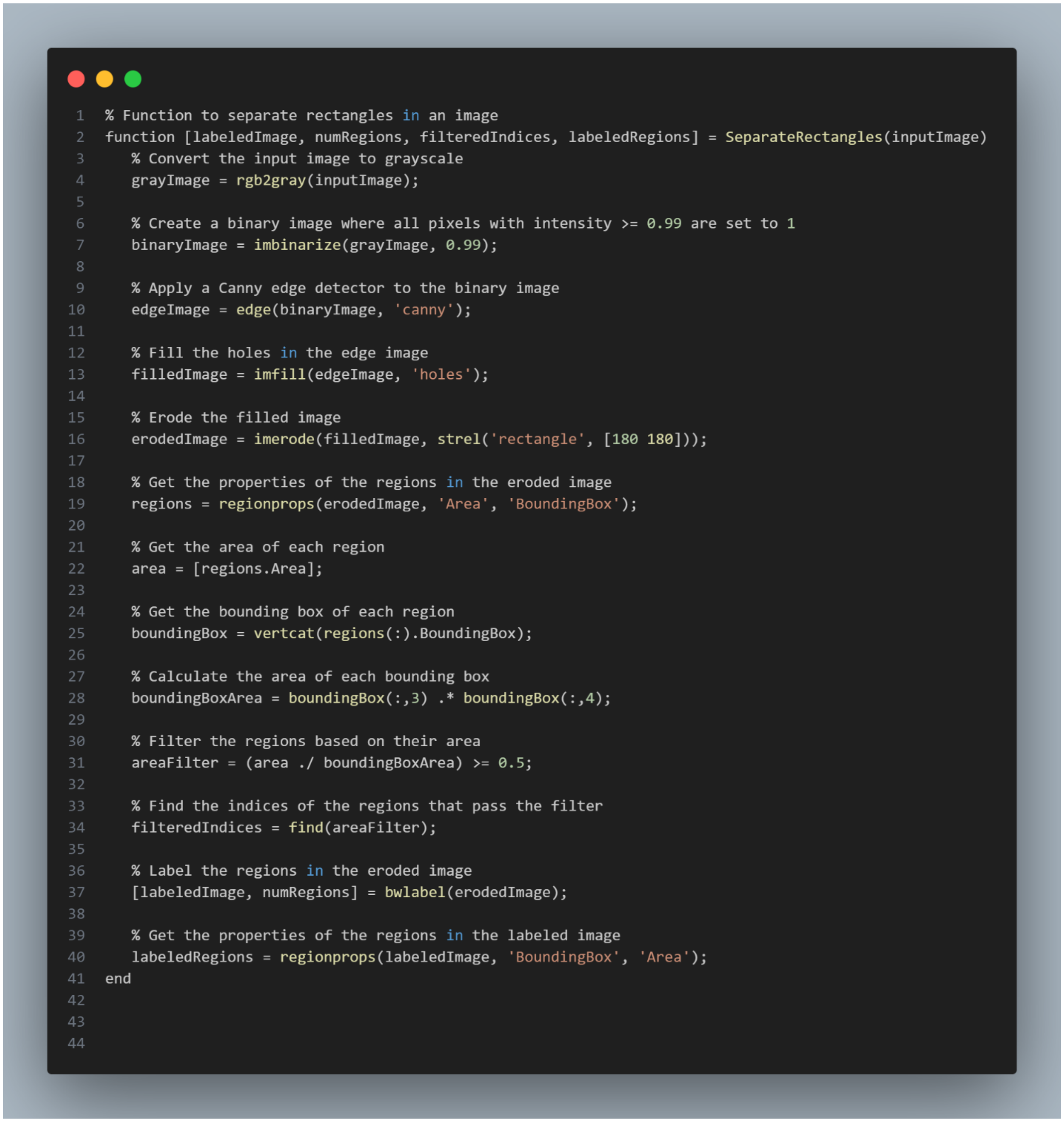
```
function fileList = GetImages(directory)
  data = dir(directory);      %# Get the data for the current directory
  dirIndex = [data.isdir];  %# Find the index for directories
  fileList = {data(~dirIndex).name}';  %'# Get a list of the files
  fileList = fullfile(directory, ...
    fileList);
end
```



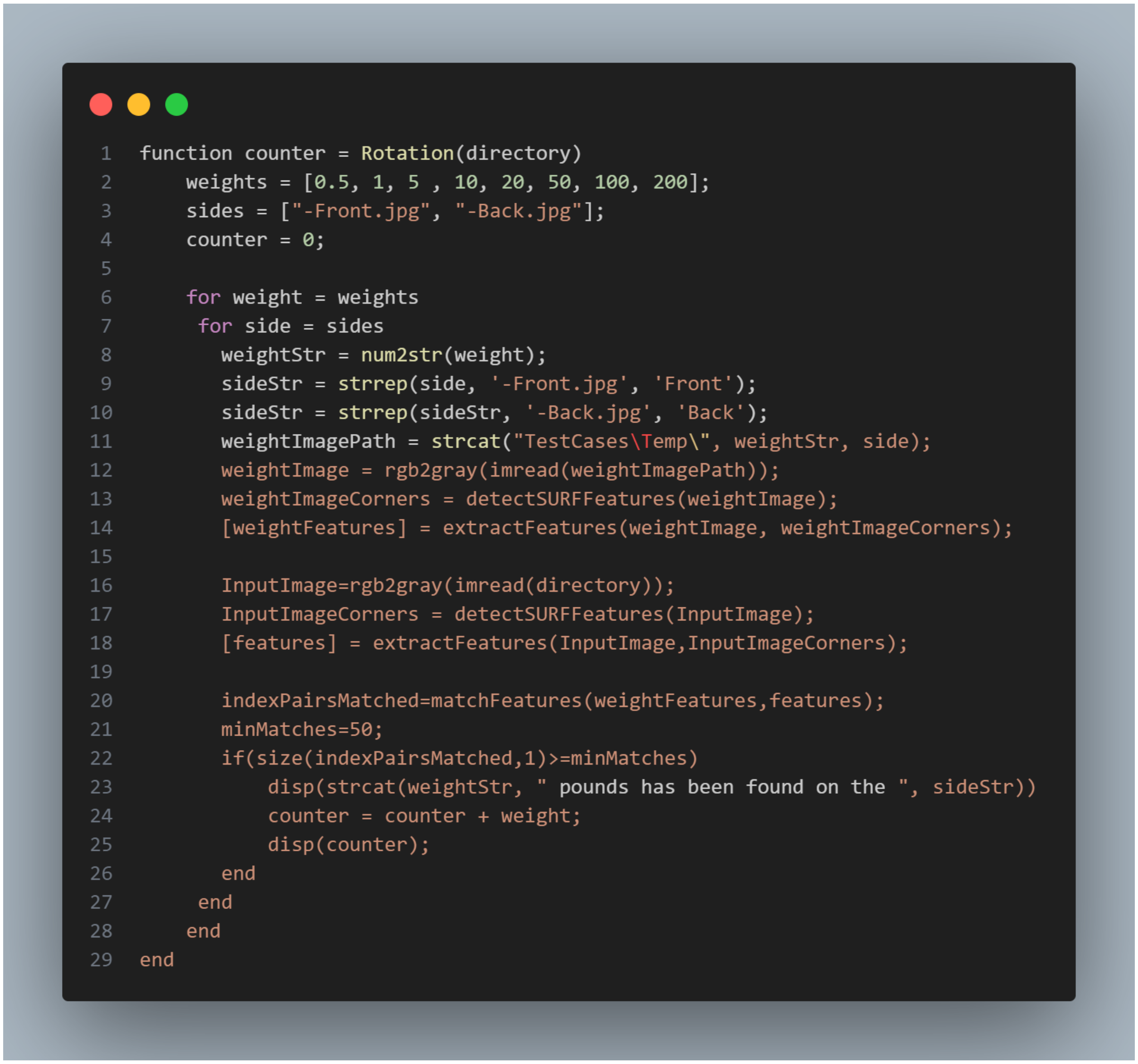
```
function [labeledImage, numRegions, filteredIndices, labeledRegions] = DetectRectangles(O)
   % Convert the input image to grayscale
   grayImage = rgb2gray(O);

   % Create a binary image where all pixels with intensity >= 250 are set to 1
   binaryImage = grayImage >= 250;

   % Apply a median filter to the binary image
   filteredImage = medfilt2(binaryImage, [15 15]);

   % Invert the binary image
   invertedImage = ~filteredImage;

   % Erode the inverted image
   erodedImage = imerode(invertedImage, strel('rectangle', [20 20]));

   % Fill the holes in the eroded image
   filledImage = imfill(erodedImage, 'holes');

   % Get the properties of the regions in the filled image
   regions = regionprops(filledImage, 'Area', 'BoundingBox');

   % Get the area of each region
   area = [regions.Area];

   % Get the bounding box of each region
   boundingBox = vertcat(regions(:).BoundingBox);

   % Calculate the area of each bounding box
   boundingBoxArea = boundingBox(:,3) .* boundingBox(:,4);

   % Filter the regions based on their area
   areaFilter = (area ./ boundingBoxArea) >= 0.95;

   % Find the indices of the regions that pass the filter
   filteredIndices = find(areaFilter);

   % Label the regions in the filled image
   [labeledImage, numRegions] = bwlabel(filledImage);

   % Get the properties of the regions in the labeled image
   labeledRegions = regionprops(labeledImage, 'BoundingBox', 'Area');

end
```



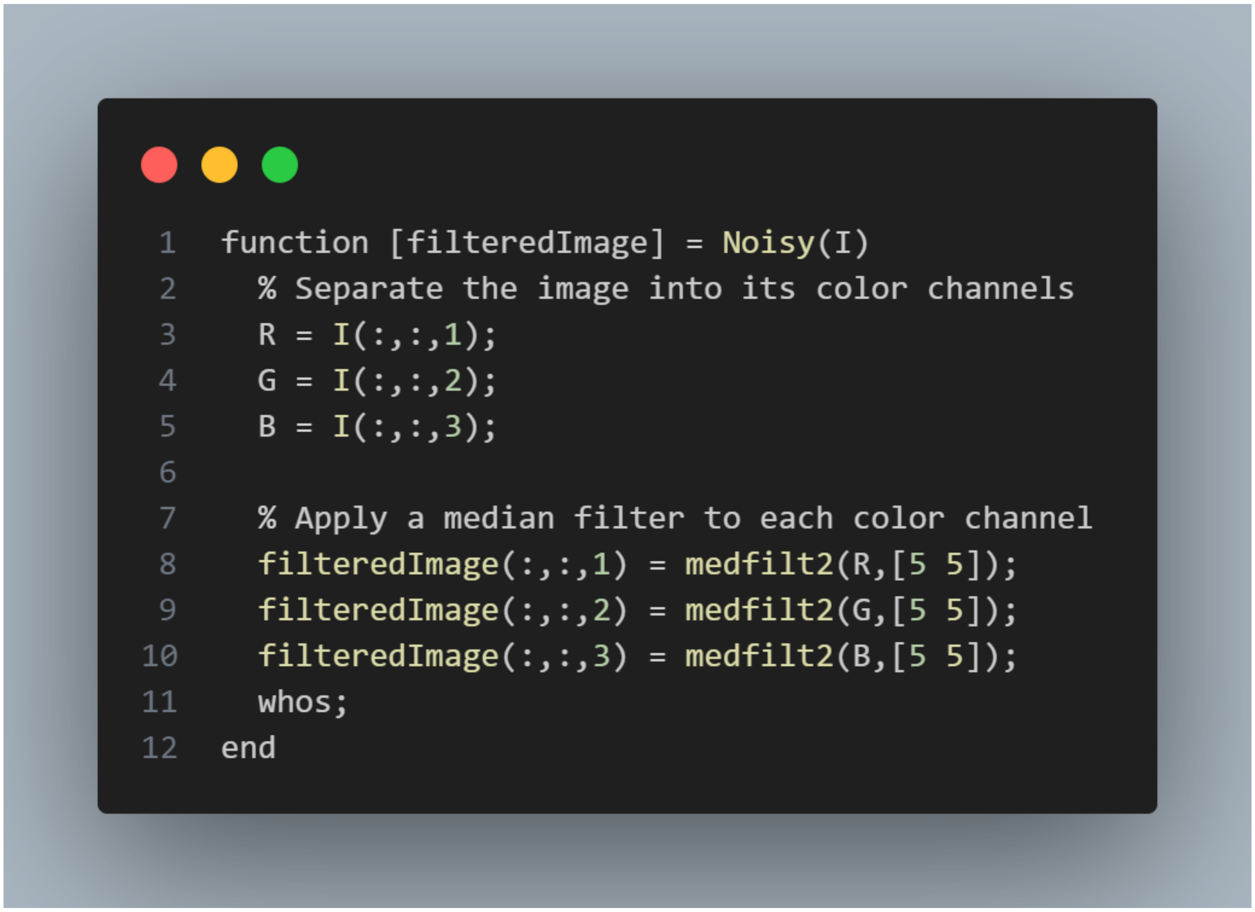
```
function [distance] = CompareHistograms(inputImage1, inputImage2)
  % Define the size of the filter
  filterSize = 15;

  % Apply the filter to the images
  filteredImage1 = imfilter(inputImage1, fspecial('average', [filterSize filterSize]));
  filteredImage2 = imfilter(inputImage2, fspecial('average', [filterSize filterSize]));

  % Convert the images to grayscale
  grayImage1 = filteredImage1(:,:,1);
  grayImage2 = filteredImage2(:,:,1);

  % Compute the histograms
  [histogram1] = imhist(grayImage1);
  [histogram2] = imhist(grayImage2);

  % Normalize the histograms
  normalizedHistogram1 = histogram1 / size(grayImage1, 1) / size(grayImage1, 2);
  normalizedHistogram2 = histogram2 / size(grayImage2, 1) / size(grayImage2, 2);

  % Compute the distance between the histograms
  distance = pdist2(normalizedHistogram1', normalizedHistogram2', 'cosine');
end
```



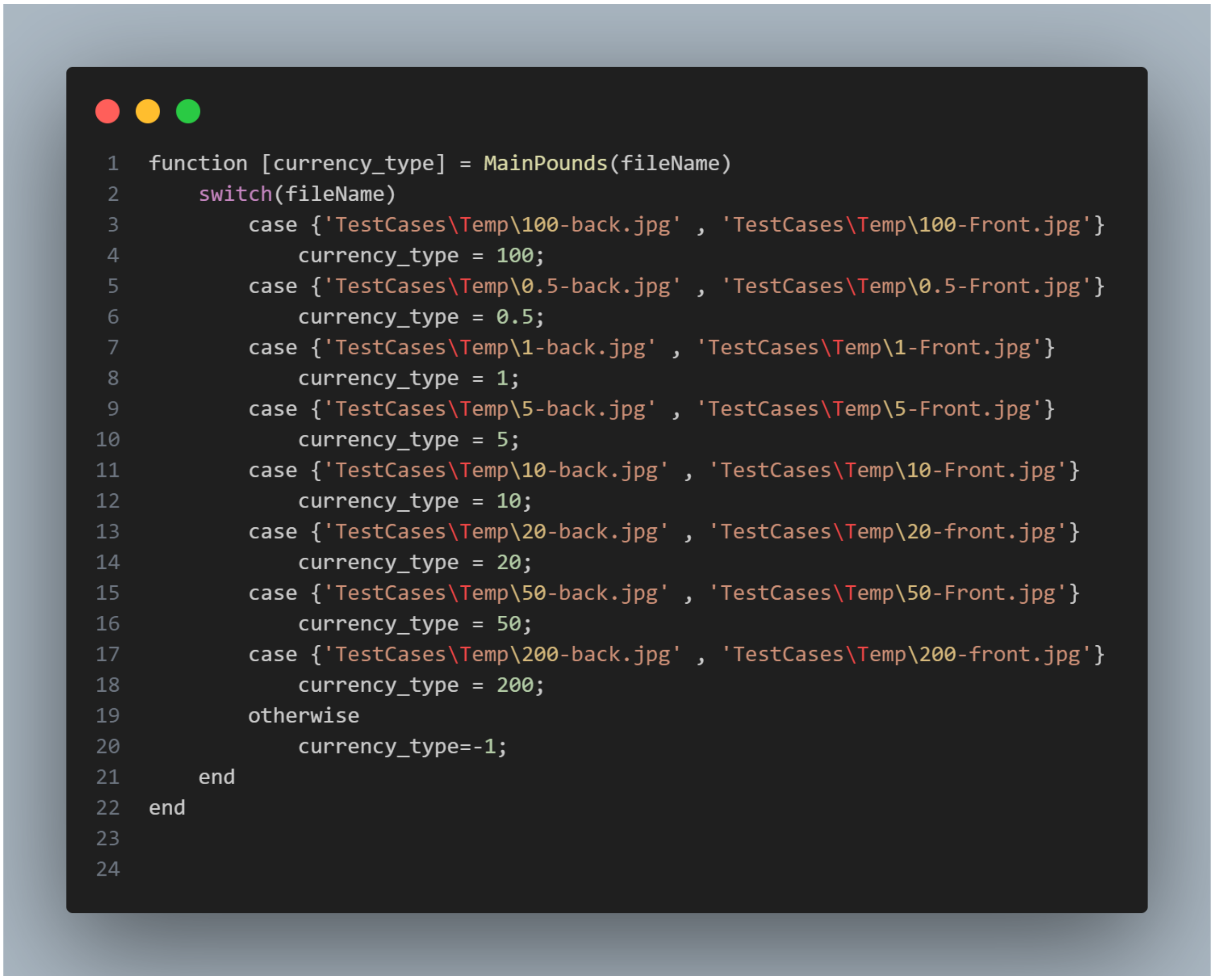
```
% Function to separate rectangles in an image
function [labeledImage, numRegions, filteredIndices, labeledRegions] = SeparateRectangles(inputImage)
   % Convert the input image to grayscale
   grayImage = rgb2gray(inputImage);

   % Create a binary image where all pixels with intensity >= 0.99 are set to 1
   binaryImage = imbinarize(grayImage, 0.99);

   % Apply a Canny edge detector to the binary image
   edgeImage = edge(binaryImage, 'canny');

   % Fill the holes in the edge image
   filledImage = imfill(edgeImage, 'holes');

   % Erode the filled image
   erodedImage = imerode(filledImage, strel('rectangle', [180 180]));

   % Get the properties of the regions in the eroded image
   regions = regionprops(erodedImage, 'Area', 'BoundingBox');

   % Get the area of each region
   area = [regions.Area];

   % Get the bounding box of each region
   boundingBox = vertcat(regions(:).BoundingBox);

   % Calculate the area of each bounding box
   boundingBoxArea = boundingBox(:,3) .* boundingBox(:,4);

   % Filter the regions based on their area
   areaFilter = (area ./ boundingBoxArea) >= 0.5;

   % Find the indices of the regions that pass the filter
   filteredIndices = find(areaFilter);

   % Label the regions in the eroded image
   [labeledImage, numRegions] = bwlabel(erodedImage);

   % Get the properties of the regions in the labeled image
   labeledRegions = regionprops(labeledImage, 'BoundingBox', 'Area');
end
```



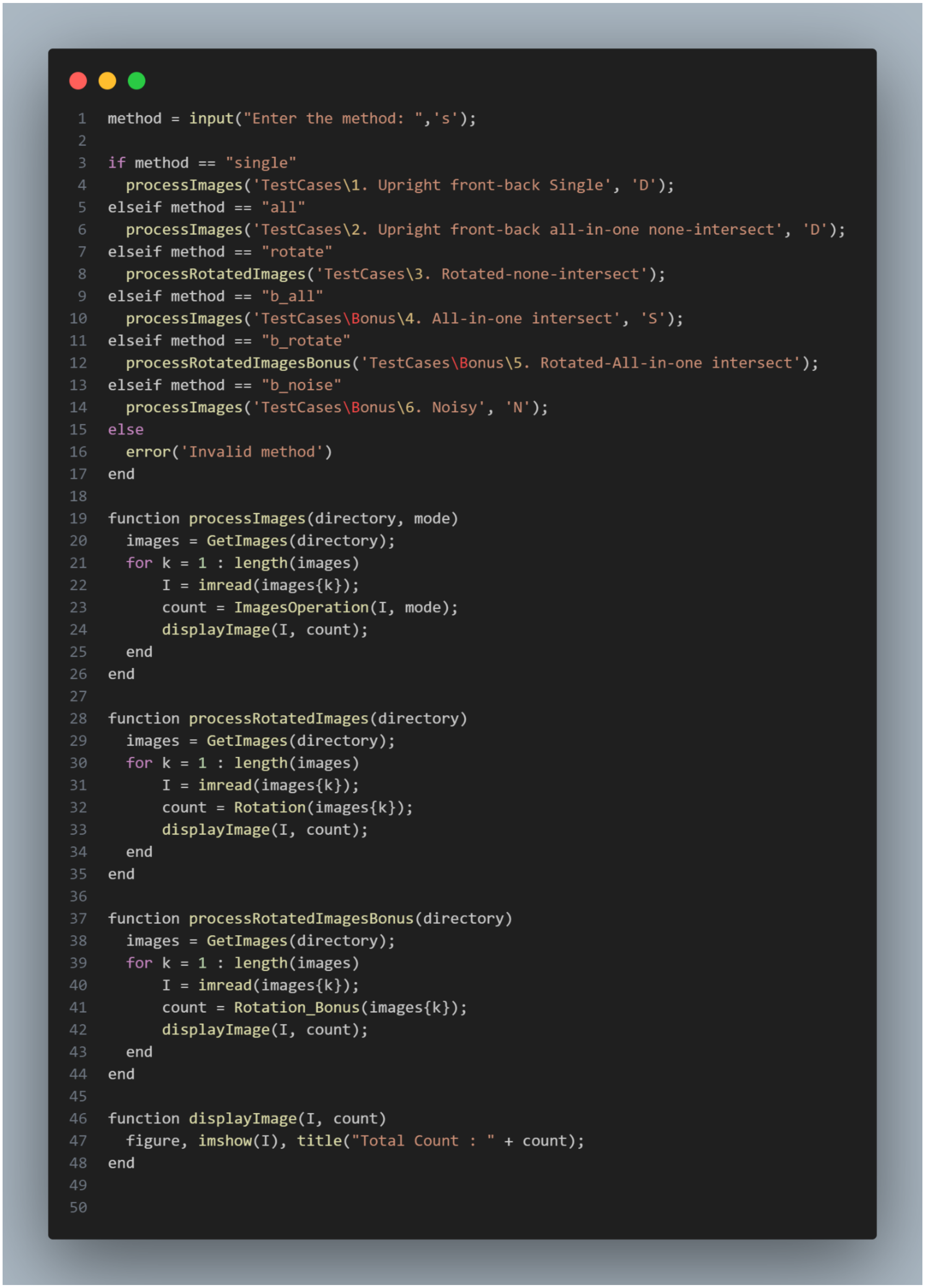
```
function counter = Rotation(directory)
    weights = [0.5, 1, 5 , 10, 20, 50, 100, 200];
    sides = ["-Front.jpg", "-Back.jpg"];
    counter = 0;

    for weight = weights
     for side = sides
       weightStr = num2str(weight);
       sideStr = strrep(side, '-Front.jpg', 'Front');
       sideStr = strrep(sideStr, '-Back.jpg', 'Back');
       weightImagePath = strcat("TestCases\Temp\", weightStr, side);
       weightImage = rgb2gray(imread(weightImagePath));
       weightImageCorners = detectSURFFeatures(weightImage);
       [weightFeatures] = extractFeatures(weightImage, weightImageCorners);

       InputImage=rgb2gray(imread(directory));
       InputImageCorners = detectSURFFeatures(InputImage);
       [features] = extractFeatures(InputImage,InputImageCorners);

       indexPairsMatched=matchFeatures(weightFeatures,features);
       minMatches=50;
       if(size(indexPairsMatched,1)>=minMatches)
           disp(strcat(weightStr, " pounds has been found on the ", sideStr))
           counter = counter + weight;
           disp(counter);
       end
     end
    end
end
```



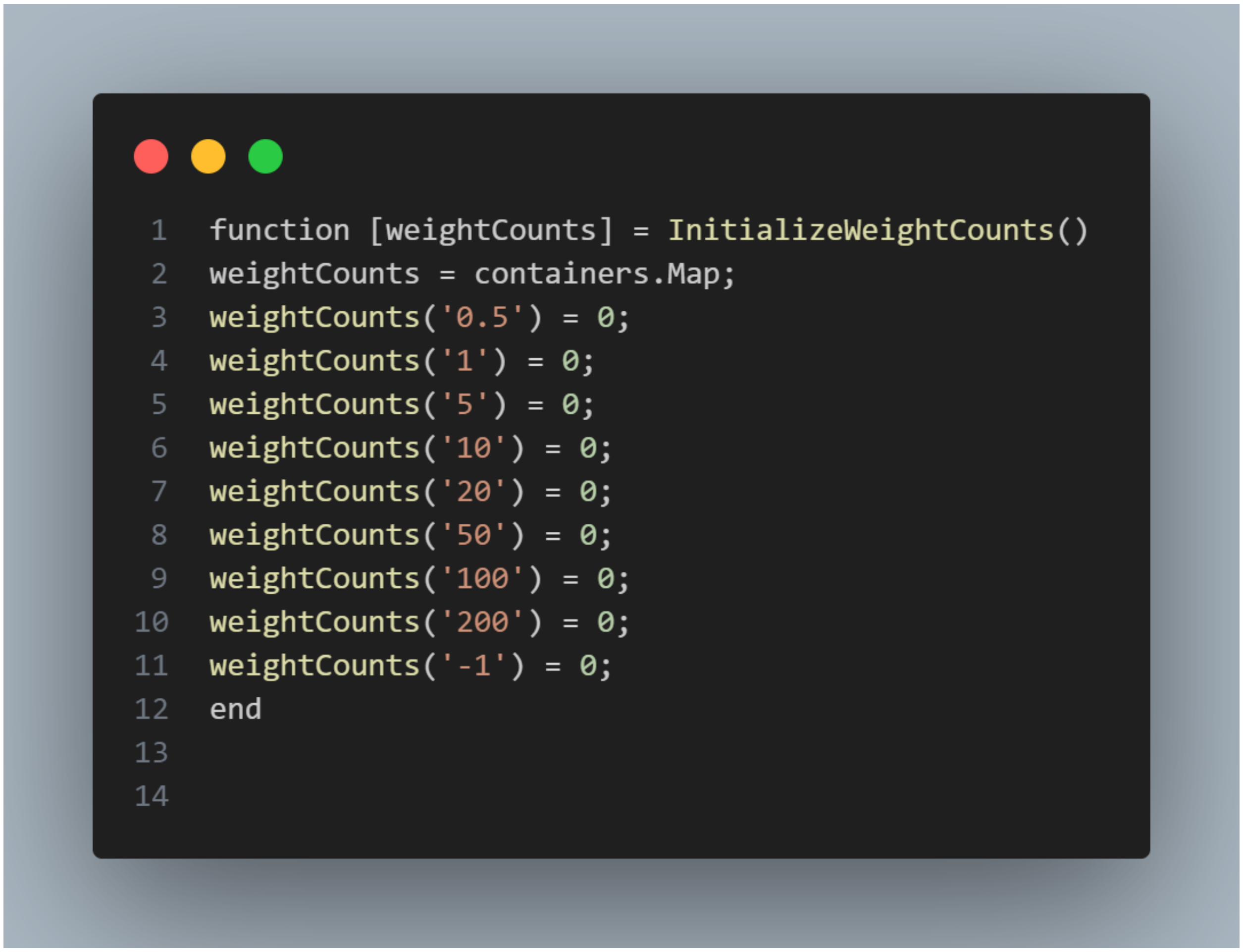
```
function [filteredImage] = Noisy(I)
  % Separate the image into its color channels
  R = I(:,:,1);
  G = I(:,:,2);
  B = I(:,:,3);

  % Apply a median filter to each color channel
  filteredImage(:,:,1) = medfilt2(R,[5 5]);
  filteredImage(:,:,2) = medfilt2(G,[5 5]);
  filteredImage(:,:,3) = medfilt2(B,[5 5]);
  whos;
end
```



```
function [currency_type] = MainPounds(fileName)
    switch(fileName)
        case {'TestCases\Temp\100-back.jpg' , 'TestCases\Temp\100-Front.jpg'}
            currency_type = 100;
        case {'TestCases\Temp\0.5-back.jpg' , 'TestCases\Temp\0.5-Front.jpg'}
            currency_type = 0.5;
        case {'TestCases\Temp\1-back.jpg' , 'TestCases\Temp\1-Front.jpg'}
            currency_type = 1;
        case {'TestCases\Temp\5-back.jpg' , 'TestCases\Temp\5-Front.jpg'}
            currency_type = 5;
        case {'TestCases\Temp\10-back.jpg' , 'TestCases\Temp\10-Front.jpg'}
            currency_type = 10;
        case {'TestCases\Temp\20-back.jpg' , 'TestCases\Temp\20-front.jpg'}
            currency_type = 20;
        case {'TestCases\Temp\50-back.jpg' , 'TestCases\Temp\50-Front.jpg'}
            currency_type = 50;
        case {'TestCases\Temp\200-back.jpg' , 'TestCases\Temp\200-front.jpg'}
            currency_type = 200;
        otherwise
            currency_type=-1;
    end
end
```

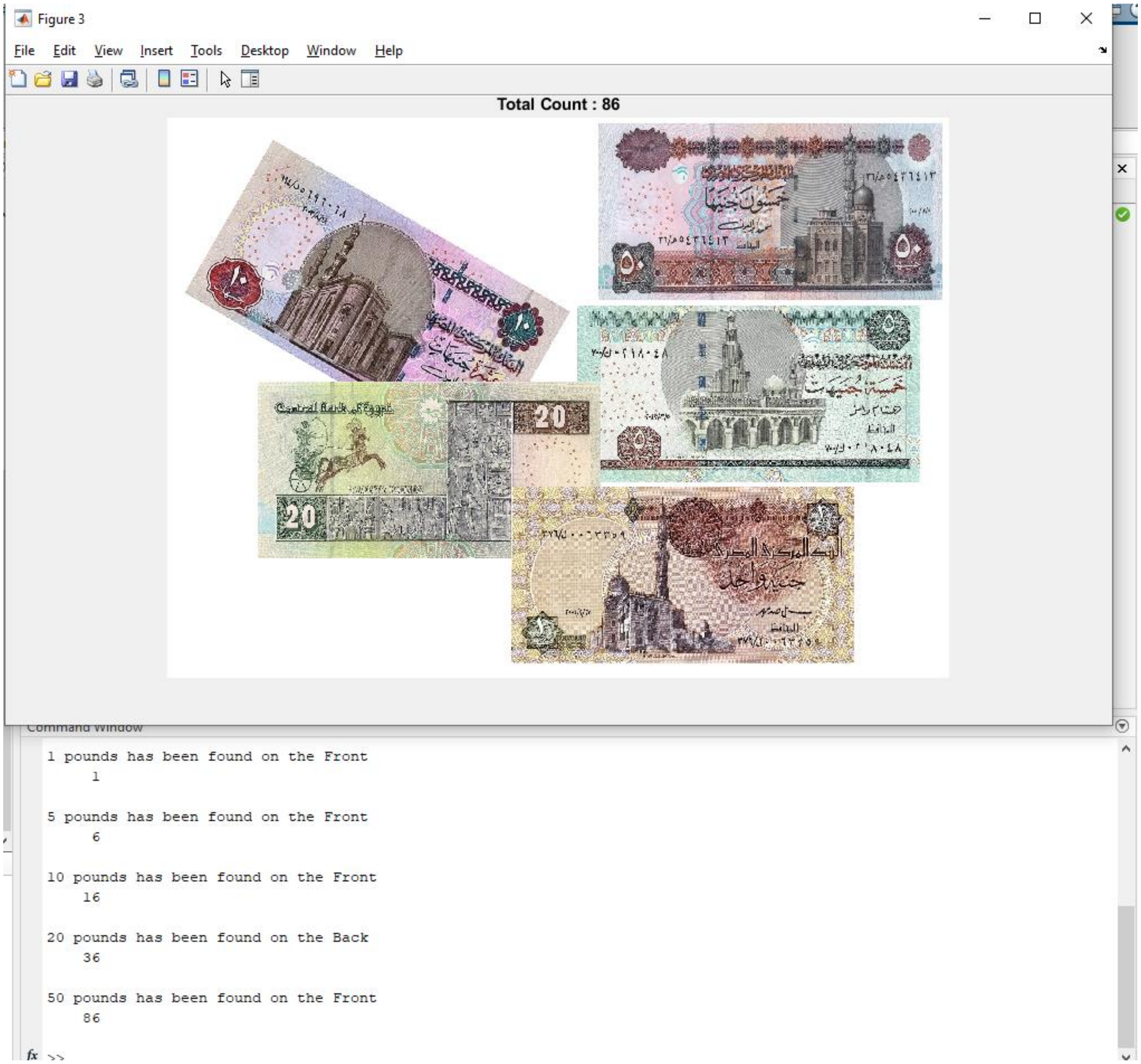


```
method = input("Enter the method: ",'s');

if method == "single"
  processImages('TestCases\1. Upright front-back Single', 'D');
elseif method == "all"
  processImages('TestCases\2. Upright front-back all-in-one none-intersect', 'D');
elseif method == "rotate"
  processRotatedImages('TestCases\3. Rotated-none-intersect');
elseif method == "b_all"
  processImages('TestCases\Bonus\4. All-in-one intersect', 'S');
elseif method == "b_rotate"
  processRotatedImagesBonus('TestCases\Bonus\5. Rotated-All-in-one intersect');
elseif method == "b_noise"
  processImages('TestCases\Bonus\6. Noisy', 'N');
else
  error('Invalid method')
end

function processImages(directory, mode)
  images = GetImages(directory);
  for k = 1 : length(images)
      I = imread(images{k});
      count = ImagesOperation(I, mode);
      displayImage(I, count);
  end
end

function processRotatedImages(directory)
  images = GetImages(directory);
  for k = 1 : length(images)
      I = imread(images{k});
      count = Rotation(images{k});
      displayImage(I, count);
  end
end

function processRotatedImagesBonus(directory)
  images = GetImages(directory);
  for k = 1 : length(images)
      I = imread(images{k});
      count = Rotation_Bonus(images{k});
      displayImage(I, count);
  end
end

function displayImage(I, count)
  figure, imshow(I), title("Total Count : " + count);
end
```

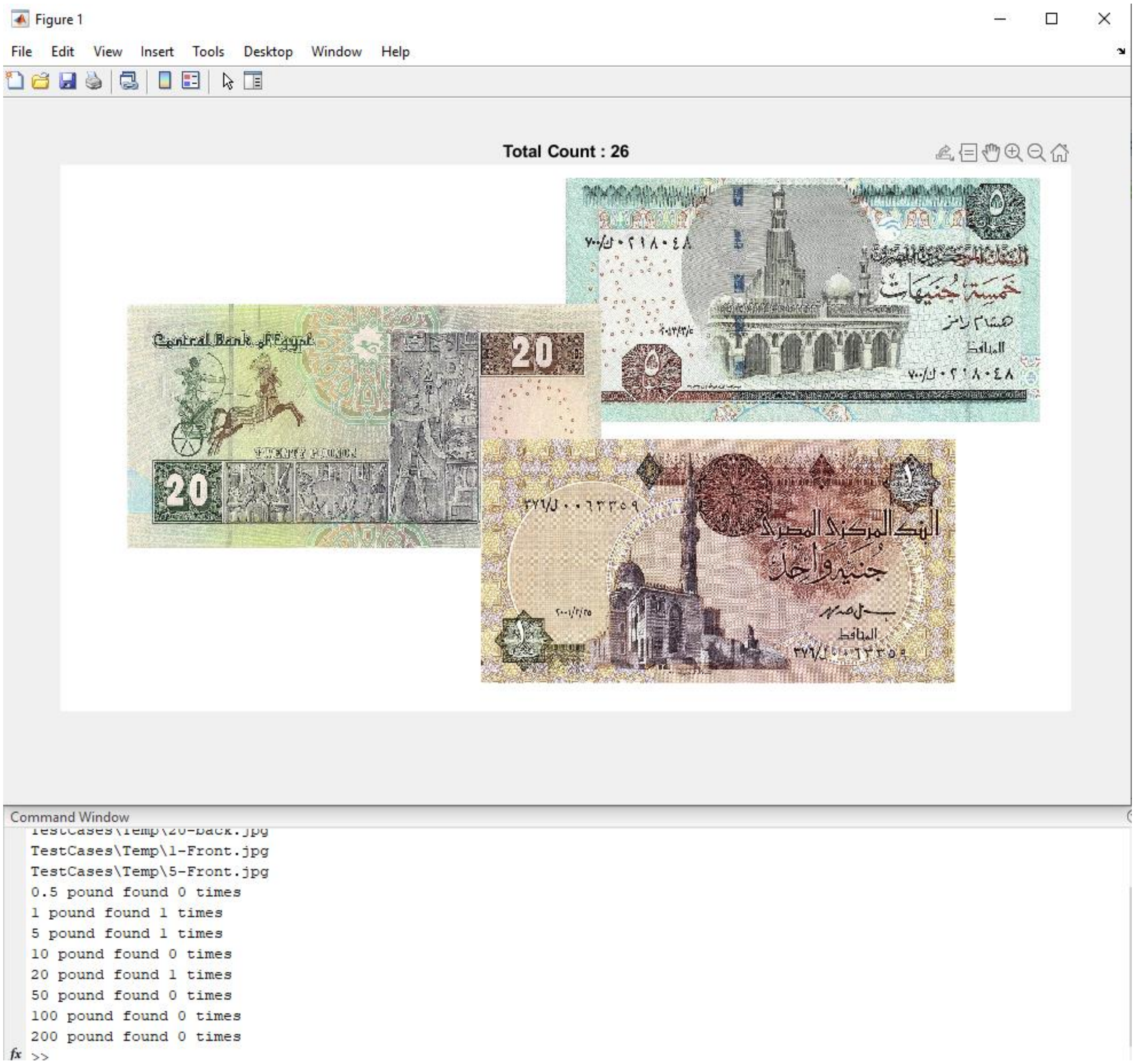


```
function [weightCounts] = InitializeWeightCounts()
weightCounts = containers.Map;
weightCounts('0.5') = 0;
weightCounts('1') = 0;
weightCounts('5') = 0;
weightCounts('10') = 0;
weightCounts('20') = 0;
weightCounts('50') = 0;
weightCounts('100') = 0;
weightCounts('200') = 0;
weightCounts('-1') = 0;
end
```

Images Results

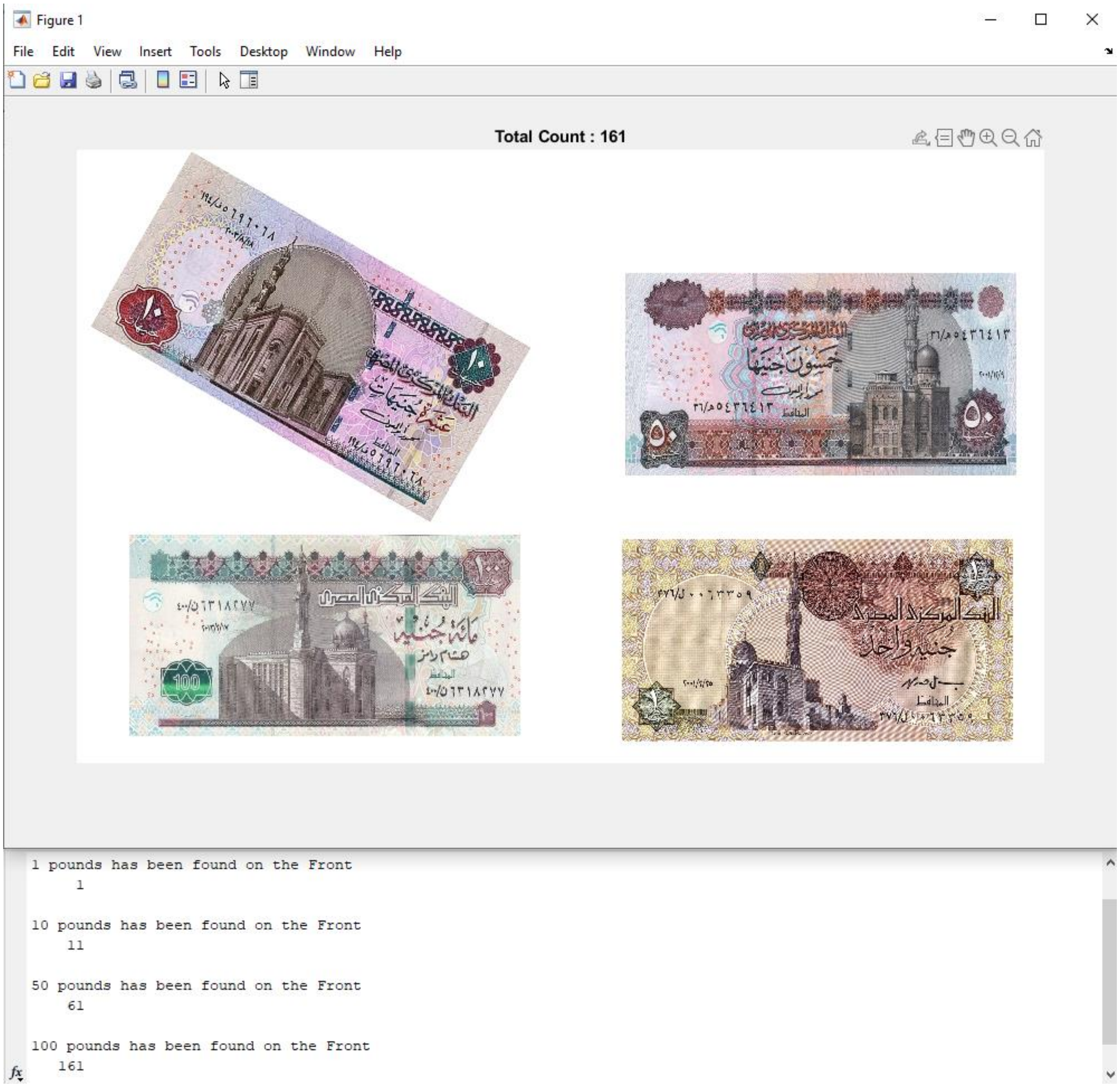


```
1 pounds has been found on the Front
     1

5 pounds has been found on the Front
     6

10 pounds has been found on the Front
    16

20 pounds has been found on the Back
    36

50 pounds has been found on the Front
    86

fx >>
```

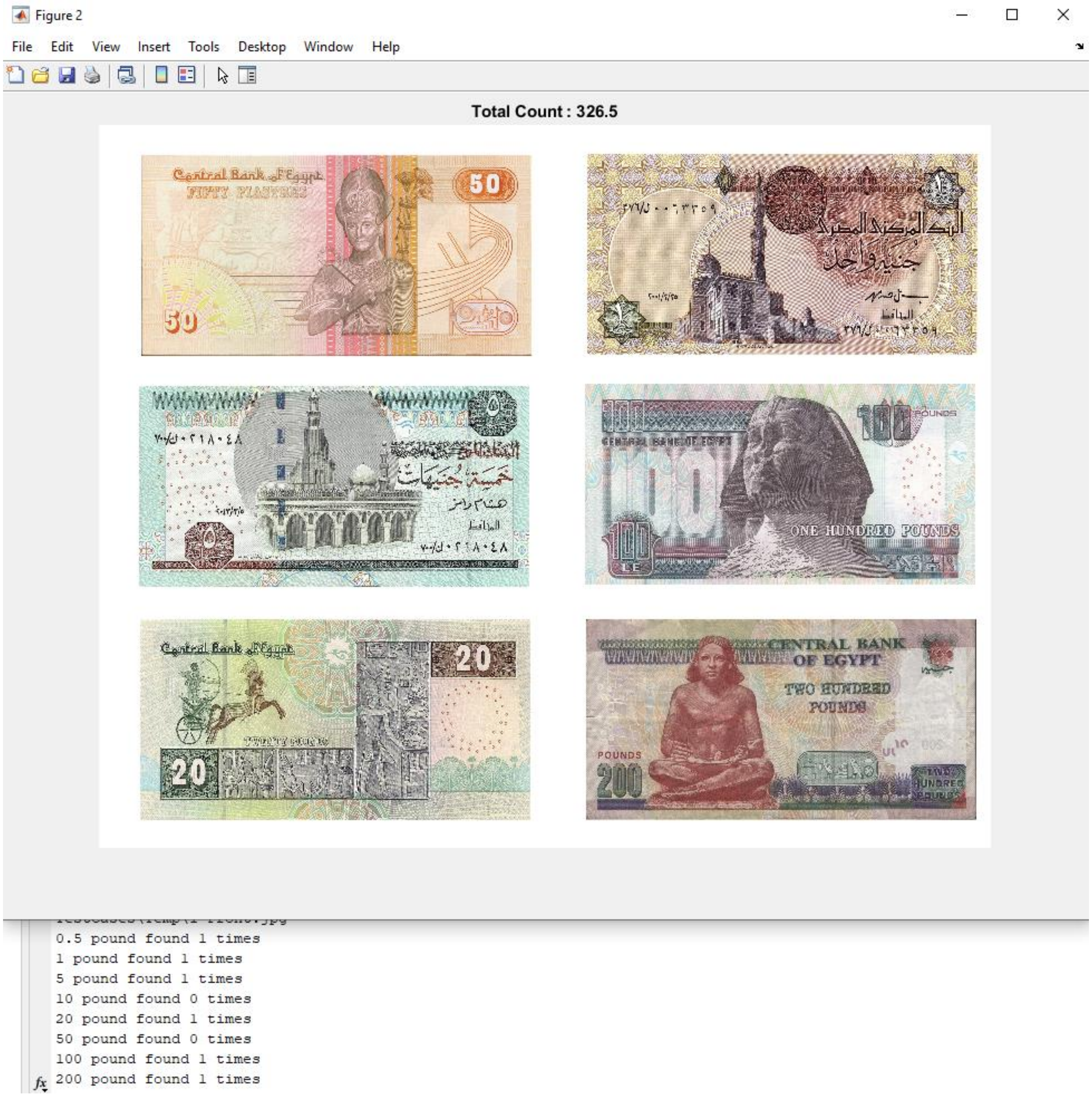


Total Count : 26

```
TestCases\Temp\20-back.jpg
TestCases\Temp\1-Front.jpg
TestCases\Temp\5-Front.jpg
0.5 pound found 0 times
1 pound found 1 times
5 pound found 1 times
10 pound found 0 times
20 pound found 1 times
50 pound found 0 times
100 pound found 0 times
200 pound found 0 times
fx >>
```

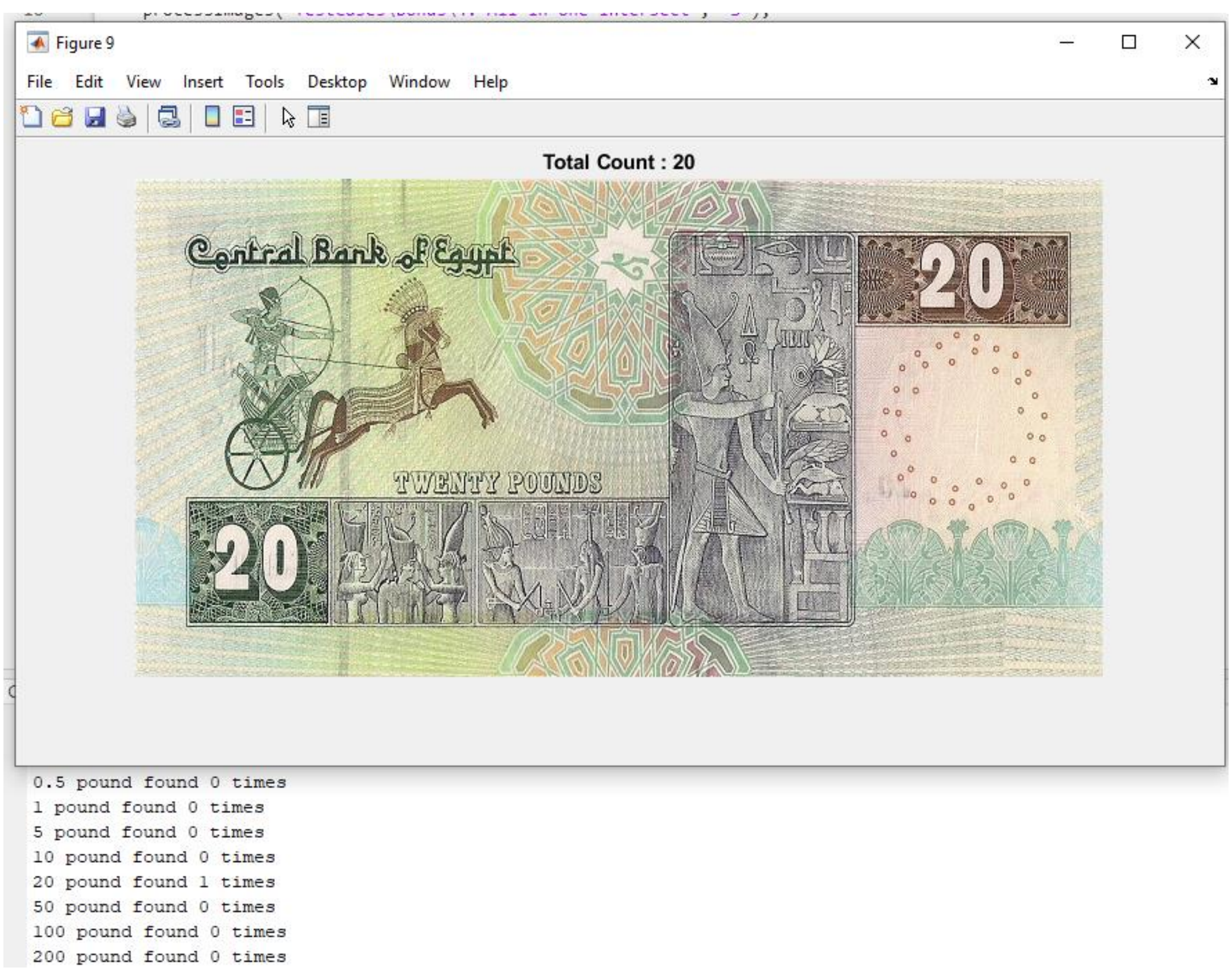


```
1 pounds has been found on the Front
     1

10 pounds has been found on the Front
    11

50 pounds has been found on the Front
    61

100 pounds has been found on the Front
   161
```



Total Count : 326.5

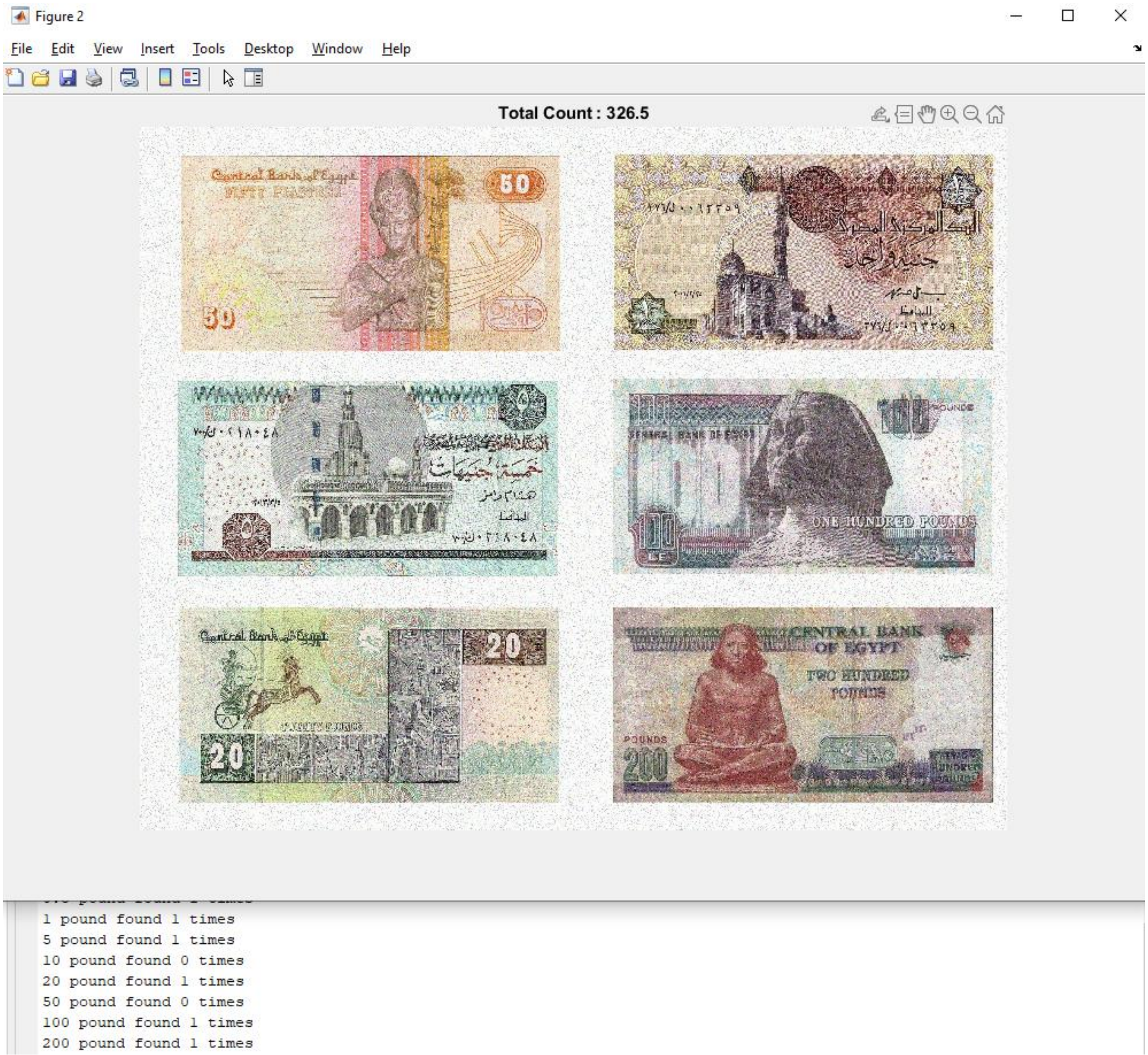
```
resources\temp\1-front.jpg
0.5 pound found 1 times
1 pound found 1 times
5 pound found 1 times
10 pound found 0 times
20 pound found 1 times
50 pound found 0 times
100 pound found 1 times
200 pound found 1 times
```



```
processImages('TestCases\Bonus\7. All in one intersect', '5');
```

Total Count : 20

```
0.5 pound found 0 times
1 pound found 0 times
5 pound found 0 times
10 pound found 0 times
20 pound found 1 times
50 pound found 0 times
100 pound found 0 times
200 pound found 0 times
```



Total Count : 326.5

```
0.5 pound found 1 times
1 pound found 1 times
5 pound found 1 times
10 pound found 0 times
20 pound found 1 times
50 pound found 0 times
100 pound found 1 times
200 pound found 1 times
```