ECE 8410: Computer Vision

Assignment 2

February 12, 2021

Mark Belbin (201618477)
Raylene Mitchell (201415247)
Rodney Winsor (200433886)

Code Explanation

The two test images, im1 and im2, were first imported to MATLAB and converted to grayscale to ensure they were the correct data type.

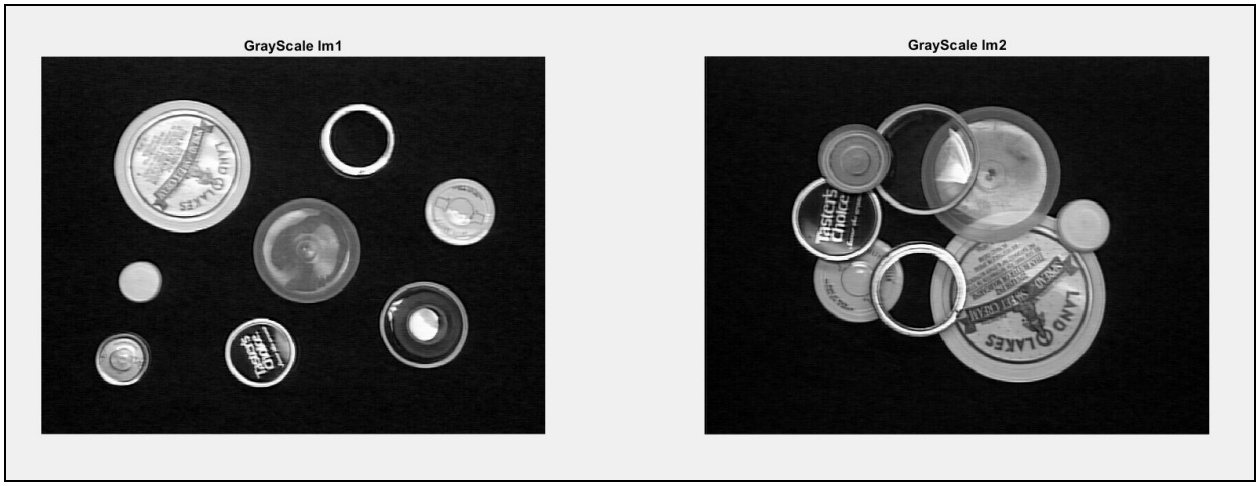

Figure 1: Original grayscale images of the coins.

Next, to clean up any noise, the script applied a gaussian blur on the images. After the blur, the images were passed into a canny edge detector to find their edges.

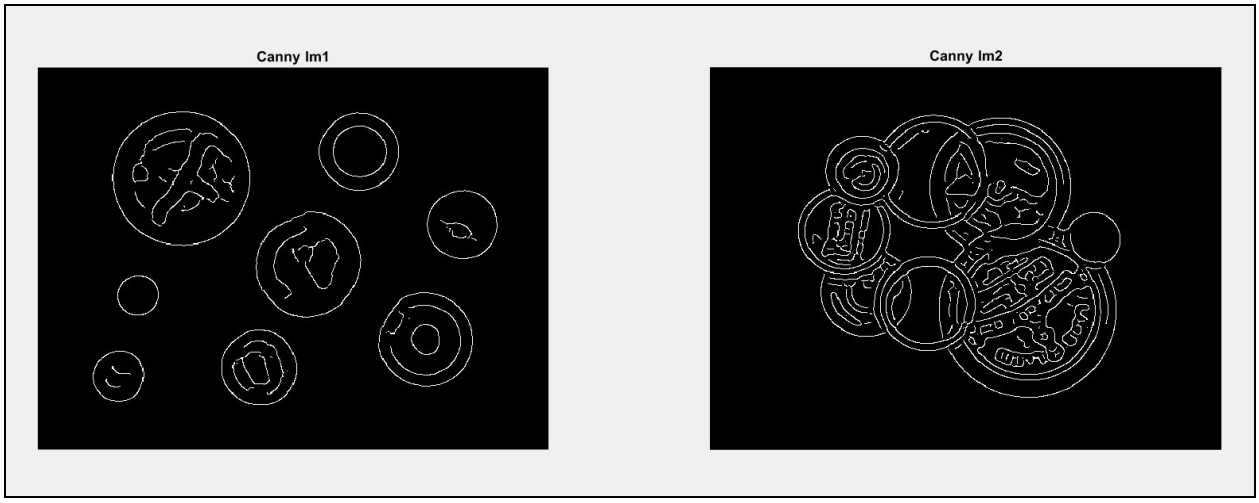

Figure 2: Binary images of the coins after gaussian blur and canny edge detection.

Once the binary edges were determined, the custom hough transform for circles of unknown radius is applied. We created a custom function for this, which takes in as inputs the binary edges image, the maximum radius that will be determined, and a global threshold value for the accumulator. The output of the function is a vector of circle locations and radii. The custom function is found in “HoughCircles.m”

In the function, the first step was to iterate through each pixel and possible radius and populate the accumulator 3D matrix. The code for this is shown below, and includes some out of bounds checking.

```
A = zeros(a,b,r); % 3D Accumulator Array

% Accumulator Lööp
for r = 1:1:r_max
    for y = 1:1:b
        for x = 1:1:a
            if edges(y,x) == 1
                for theta = 0:0.1:2*pi
                    % Calculate circle pixel locations
                    x0 = uint16(x - r * cos(theta));
                    y0 = uint16(y - r * sin(theta));

                    % Check if circle is within accumulator bounds
                    if (x0 < 1) || (x0 > a)
                        break
                    elseif (y0 < 1) || (y0 > b)
                        break
                    else
                        A(x0,y0,r) = A(x0,y0,r) + 1; % Add to accumulator
                    end
                end
            end
        end
    end
end
```

Figure 3: 3D Hough Accumulator loop code.

Next, the local maxima of the accumulator matrix was found using MATLAB's "islocalmax" function, which produces a binary result of the local max locations. Using these locations, as well as a global threshold accumulation value, initial circle size and locations are populated into a vector.

```
% Find local maxima in 3D array
maxima = islocalmax(A);

circle_count = 0;
circle_x = [];
circle_y = [];
circle_r = [];

% Create array of circles
for r = r_max:-1:20
    for y = 1:1:b
        for x = 1:1:a
            if (maxima(x,y,r) == 1) && (A(x,y,r) > thresh)
                circle_count = circle_count + 1;
                circle_x = [circle_x; x];
                circle_y = [circle_y; y];
                circle_r = [circle_r; r];
            end
        end
    end
end
```

Figure 4: Code for finding the local maxima, then thresholding them based on a global threshold value. This reduces redundant circles.

Next, to eliminate circles that are inside one another, the following code populates the circles into a new vector and compares their center location against circles in the original vector. If the center location is determined to be more than half of the old circle radius, then the new circle is considered to be unique and added to the new vector. Since the original vector of circles is ordered from biggest radius to smallest radius, the unique determined circles will always be the outermost circles, which would be the perimeter of the coins in im1 and im2.

```
% Eliminate redundant circles
for circ = 1:1:circle_count
    x0 = circles(circ,1);
    y0 = circles(circ,2);

    flag_similar = 0;
    for good_circ = 1:1:good_circle_count
        x1 = good_circles(good_circ, 1);
        y1 = good_circles(good_circ, 2);
        r1 = good_circles(good_circ, 3);

        d = sqrt((x1-x0)^2+(y1-y0)^2);

        if d <= 0.5*r1
            flag_similar = 1;
        end
    end
    % If circle is unique, add it to the good circle list
    if flag_similar == 0
        good_circles = [good_circles; circles(circ,:)];
        good_circle_count = good_circle_count + 1;
    end

end
```

Figure 5: Elimination of redundant circles. Circles with similar centers are compared, and the circle with the larger radius is kept while the other is discarded.

Final Results

The output list of unique circles found using the custom Hough Transform function was then used to draw the circles onto the original im1 and im2 images. The results are quite good, and are displayed below.

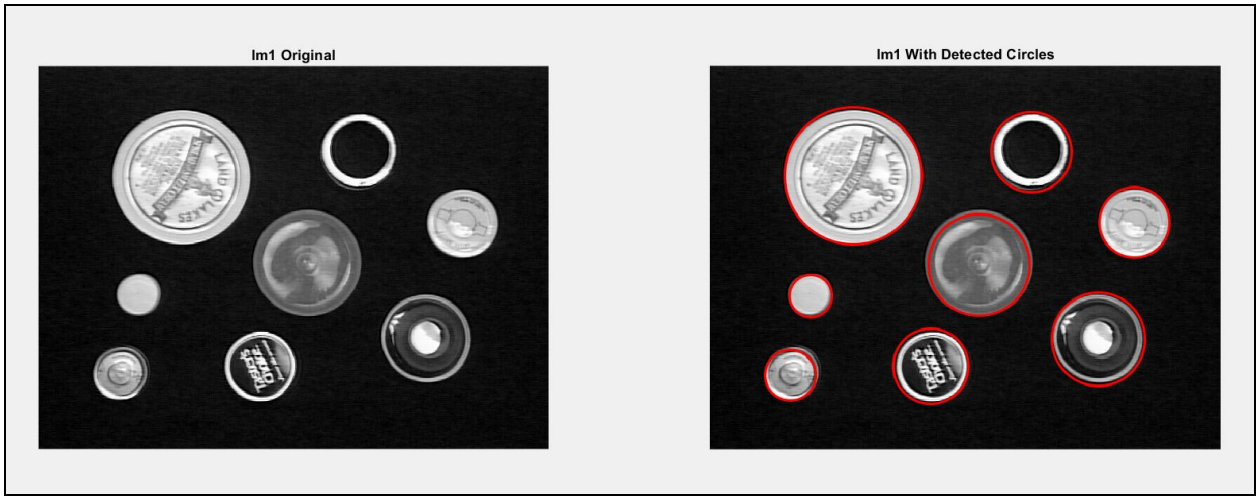

Figure 6: Final output result of the custom circle hough detection on im1.

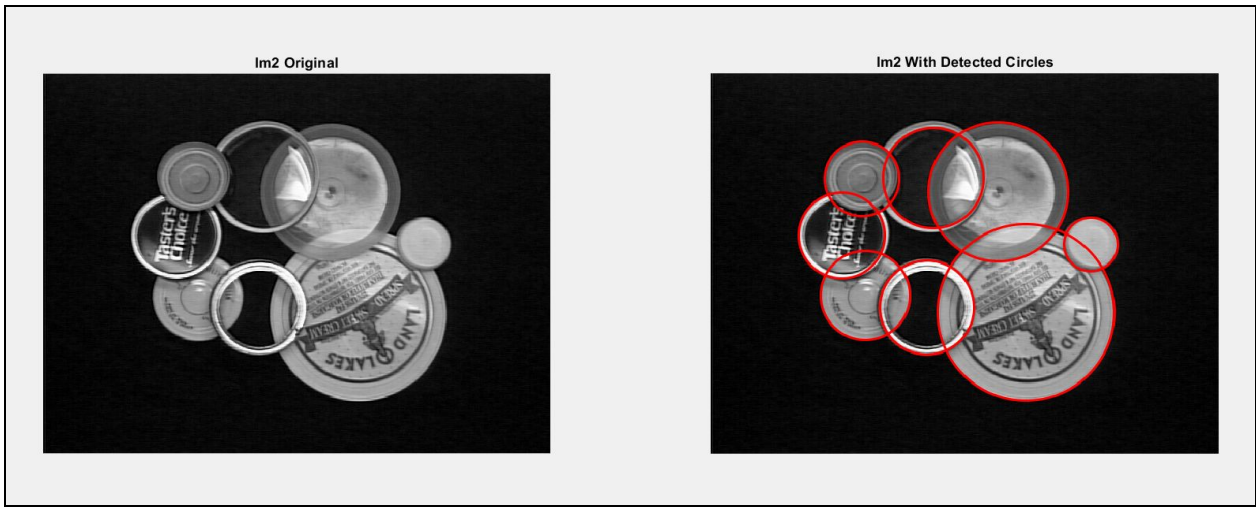

Figure 7: Final output result of the custom circle hough detection on im2. Even the overlapping coins pose no trouble for this algorithm. The outer perimeters are detected almost flawlessly!