CSC420 Assignment 2

Gabriel Luong
996268275

1

```
% Given an image, perform Harris corner detection and return the result image
function harris(im)
img = rgb2gray(im);

[Ix, Iy] = imgradientxy(img); % Compute the gradients Ix and Iy
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;

% Convolve the computed gradients with a Gaussian filter
gaussian = fspecial('gaussian');
gIx2 = imfilter(Ix2, gaussian, 'same', 'conv');
gIy2 = imfilter(Iy2, gaussian, 'same', 'conv');
gIxy = imfilter(Ixy, gaussian, 'same', 'conv');

height = size(im, 1);
width = size(im, 2);
alpha = 0.06;
R = zeros(height, width);

% Compute M and R for every pixel
for i = 1:height
    for j = 1:width
        M = [gIx2(i, j) gIxy(i, j); ...
             gIxy(i, j) gIy2(i, j)];
        detM = (gIx2(i, j) * gIy2(i, j)) - gIxy(i, j)^2;
        traceM = gIx2(i, j) + gIy2(i, j);
        R(i, j) = detM - (alpha * traceM ^ 2);
    end
end

% Keep track of the points to plot
X = [];
```

```
Y = [];

threshold = max(max(R)) * 0.03; % R threshold

% Perform non-maximum suppression by checking surrounding R values for a
% local maxima with a radius of 1
for i = 2:height - 1
    for j = 2:width - 1
        if R(i, j) > threshold && R(i, j) > R(i - 1, j + 1) && R(i, j) > R(i, j + 1) &&
                R(i, j) > R(i + 1, j + 1) && R(i, j) > R(i - 1, j) && R(i, j) > R(i + 1, j) &&
            R(i, j) > R(i - 1, j - 1) && R(i, j) > R(i, j - 1) && R(i, j) > R(i - 1, j + 1)
            X(end+1) = j;
            Y(end+1) = i;
        end
    end
end

% Display the results of the computations
figure, imshow(Ix, []), title('Ix')
figure, imshow(Iy, []), title('Iy')
figure, imshow(Ix2, []), title('Ix2')
figure, imshow(Iy2, []), title('Iy2')
figure, imshow(Ixy, []), title('Ixy')
figure, imshow(gIx2, []), title('gIx2')
figure, imshow(gIy2, []), title('gIy2')
figure, imshow(gIxy, []), title('gIxy')
figure, imshow(R, []), title('R');
figure, imshow(img), title('corners');
hold on;
plot(X, Y, 'R+');
end
```

(a) Compute I_x^2, I_y^2, $I_x I_y$ and the matrix M for every pixel.

```
img = rgb2gray(im);

[Ix, Iy] = imgradientxy(img); % Compute the gradients Ix and Iy
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;

% Convolve the computed gradients with a Gaussian filter
gaussian = fspecial('gaussian');
gIx2 = imfilter(Ix2, gaussian, 'same', 'conv');
gIy2 = imfilter(Iy2, gaussian, 'same', 'conv');
gIxy = imfilter(Ixy, gaussian, 'same', 'conv');

```

```
height = size(im, 1);
width = size(im, 2);
alpha = 0.06;
R = zeros(height, width);

% Compute M and R for every pixel
for i = 1:height
    for j = 1:width
        M = [gIx2(i, j) gIxy(i, j); gIxy(i, j) gIy2(i, j)];
    end
end
```

(b) Given the matrix A = $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$, the determinant $det(A) = ad - cb$, and $trace(A) = a + d$. The equation to compute the cornerness measure R in each pixel is $R = det(M) - \alpha trace(M)^2 = ad - cb - \alpha(a + d)^2$.

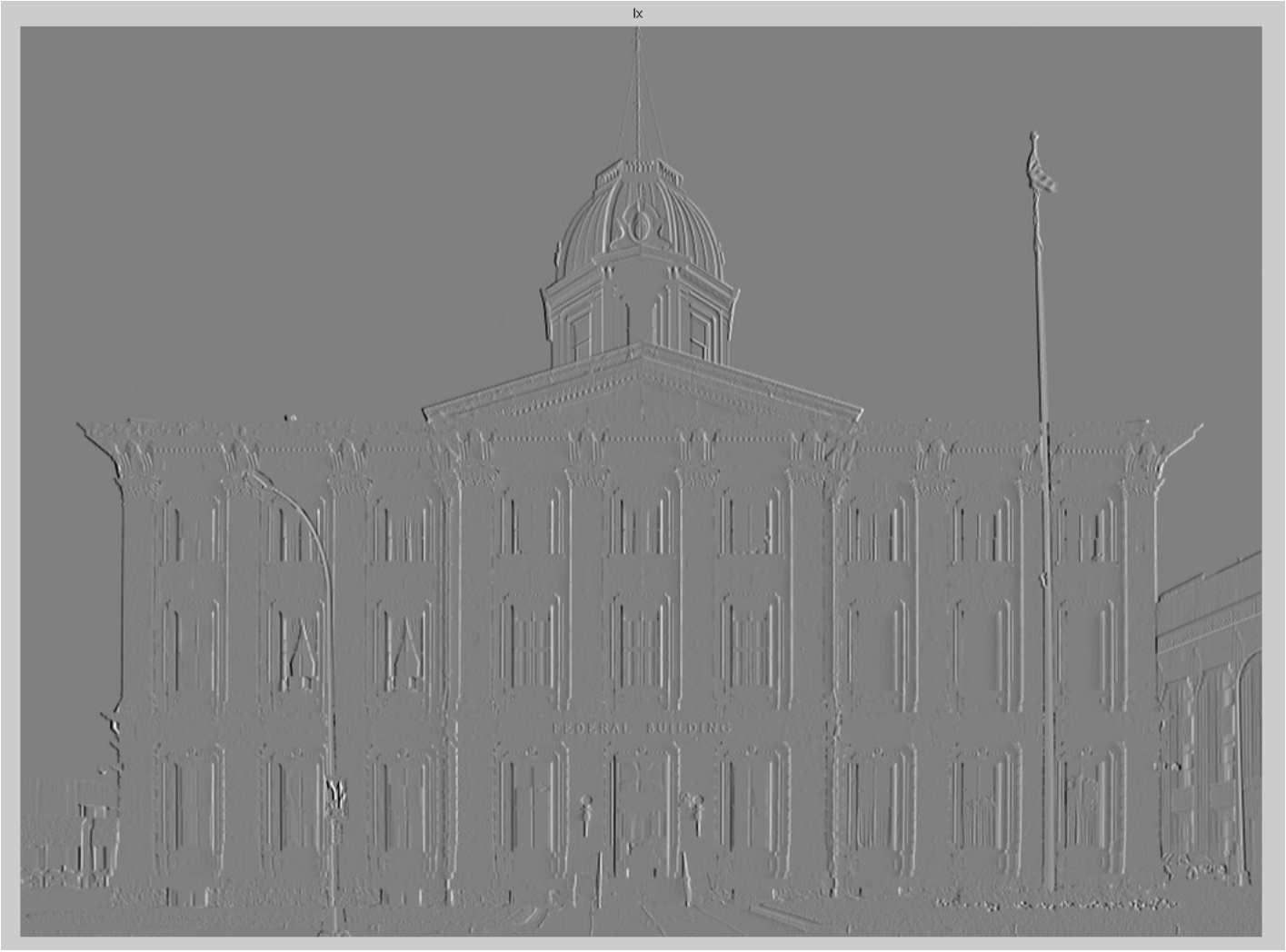
(c) Compute R for every pixel.

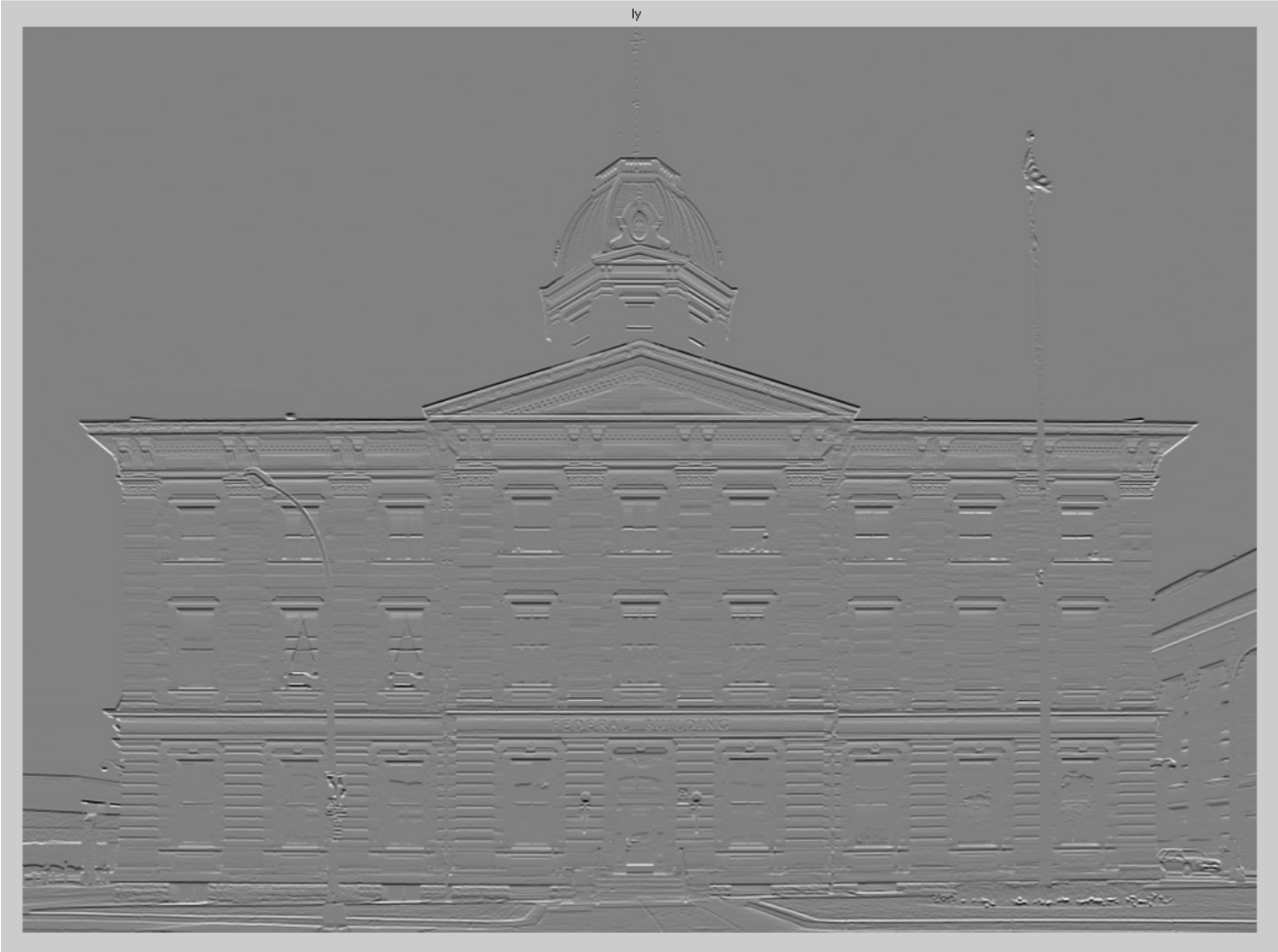
```
% Convolve the computed gradients with a Gaussian filter
gaussian = fspecial('gaussian');
gIx2 = imfilter(Ix2, gaussian, 'same', 'conv');
gIy2 = imfilter(Iy2, gaussian, 'same', 'conv');
gIxy = imfilter(Ixy, gaussian, 'same', 'conv');

height = size(im, 1);
width = size(im, 2);
alpha = 0.06;
R = zeros(height, width);

% Compute M and R for every pixel
for i = 1:height
    for j = 1:width
        M = [gIx2(i, j) gIxy(i, j); ...
             gIxy(i, j) gIy2(i, j)];
        detM = (gIx2(i, j) * gIy2(i, j)) - gIxy(i, j)^2;
        traceM = gIx2(i, j) + gIy2(i, j);
        R(i, j) = detM - (alpha * traceM ^ 2);
    end
end
```

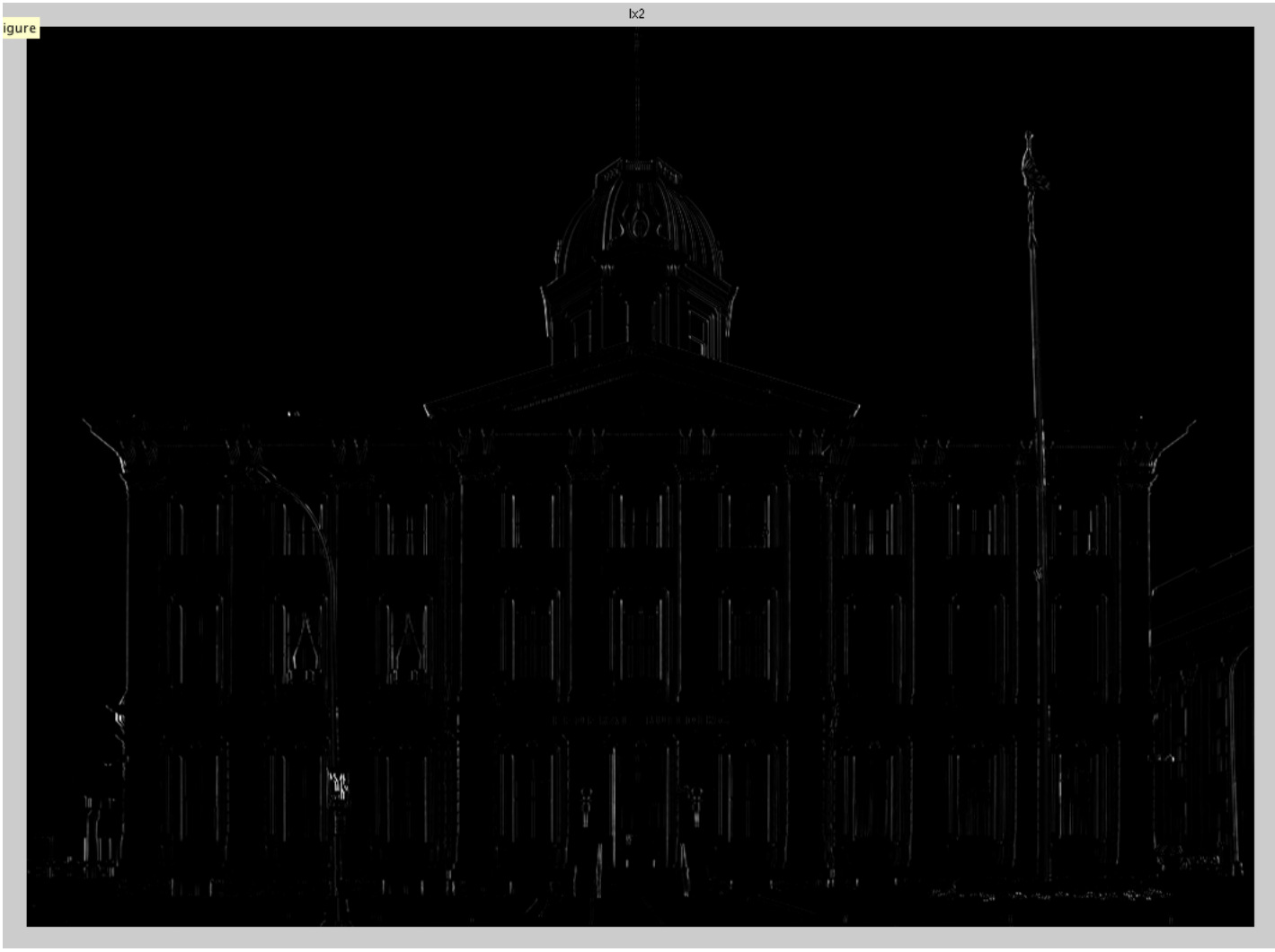
(d) Compute Threshold R and perform non-maxima suppression.

```
% Keep track of the points to plot
X = [];
Y = [];

```

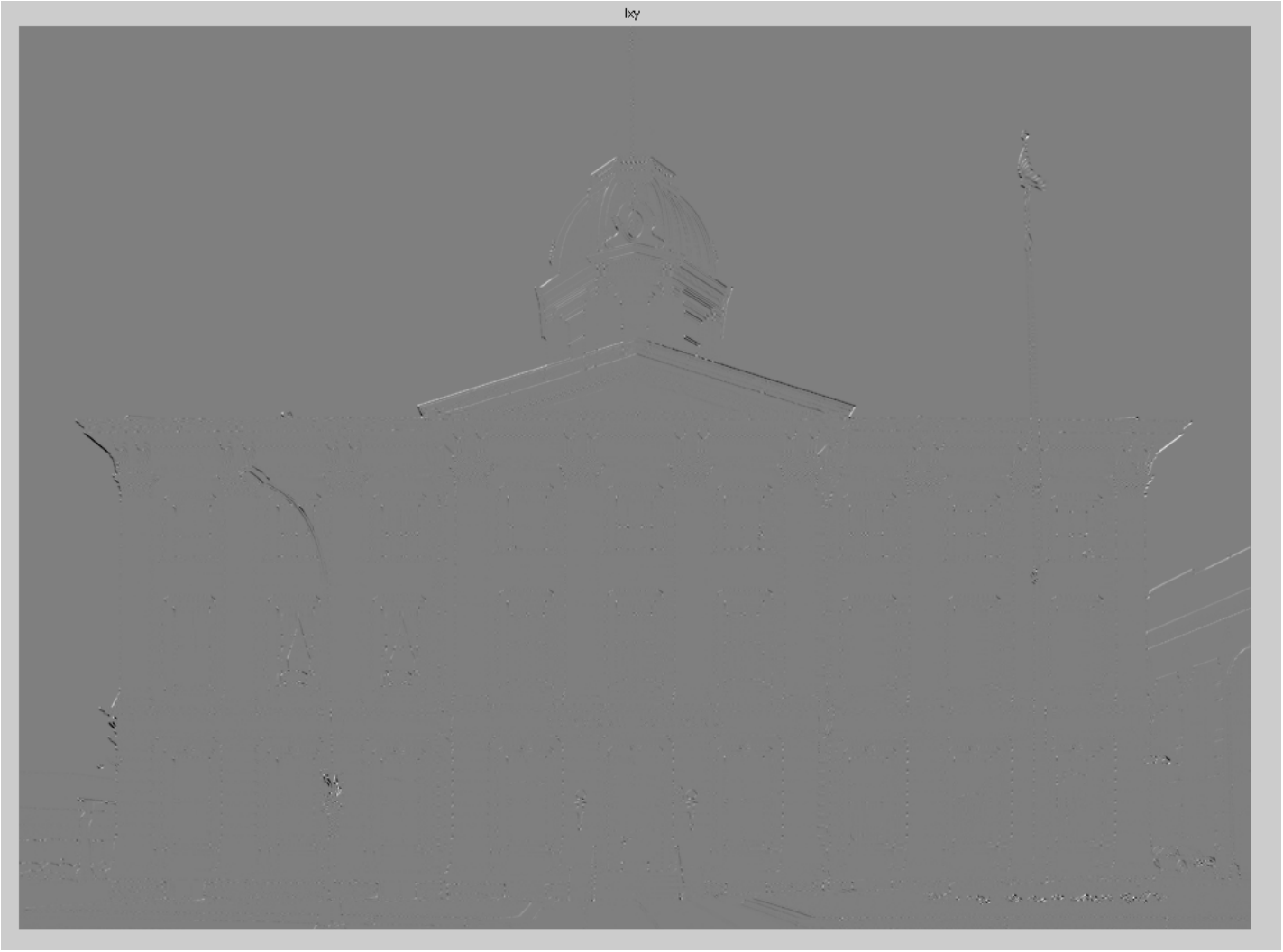
```
threshold = max(max(R)) * 0.03; % R threshold

% Perform non-maximum suppression by checking surrounding R values for a
% local maxima with a radius of 1
for i = 2:height - 1
    for j = 2:width - 1
        if R(i, j) > threshold && R(i, j) > R(i - 1, j + 1) && R(i, j) > R(i, j + 1) &&
                R(i, j) > R(i + 1, j + 1) && R(i, j) > R(i - 1, j) && R(i, j) > R(i + 1, j) &&
            R(i, j) > R(i - 1, j - 1) && R(i, j) > R(i, j - 1) && R(i, j) > R(i - 1, j + 1)
            X(end+1) = j;
            Y(end+1) = i;
        end
    end
end

figure, imshow(img), title('corners');
hold on;
plot(X, Y, 'R+');
```

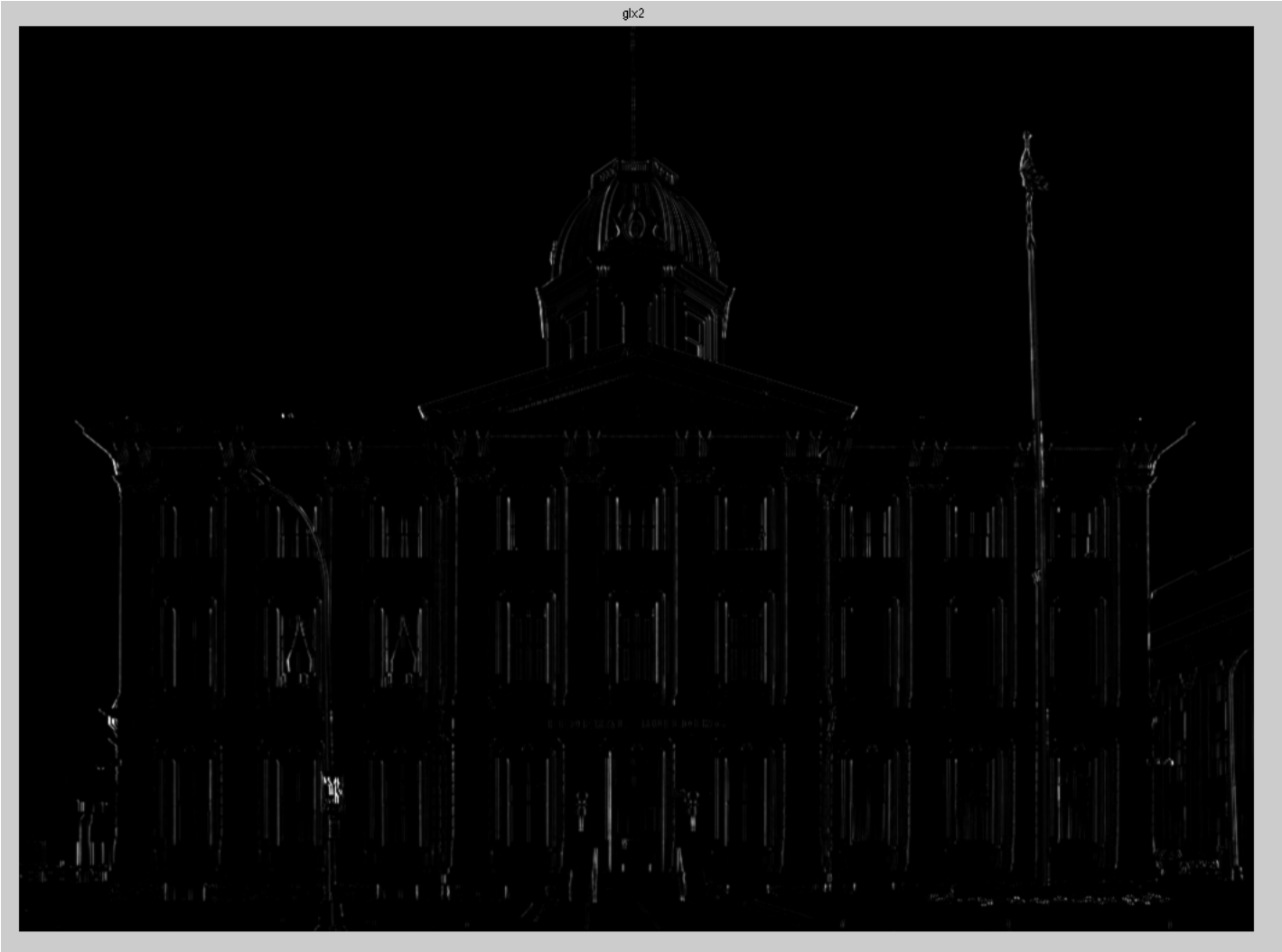


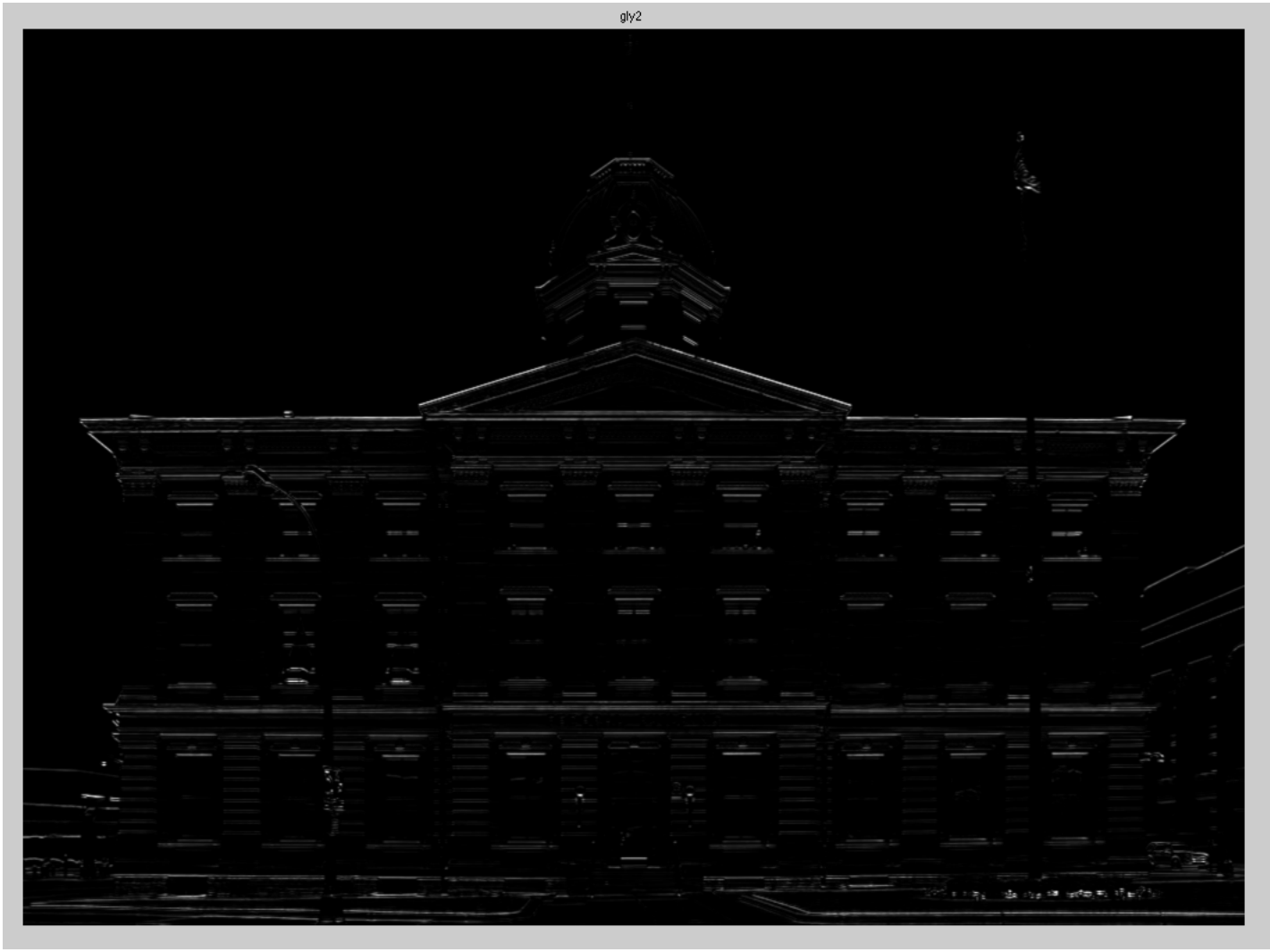

Iy

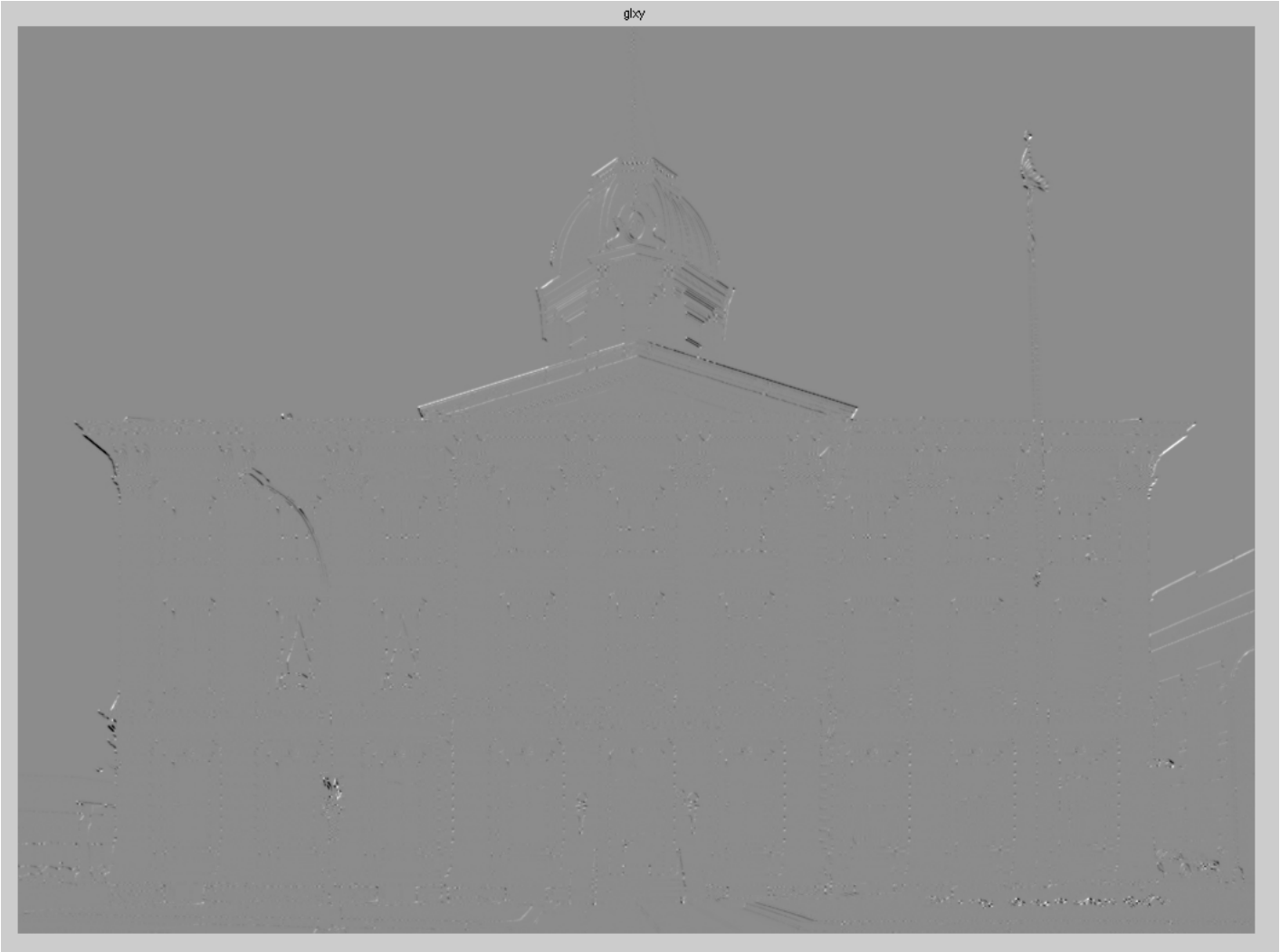

Ix2
igure

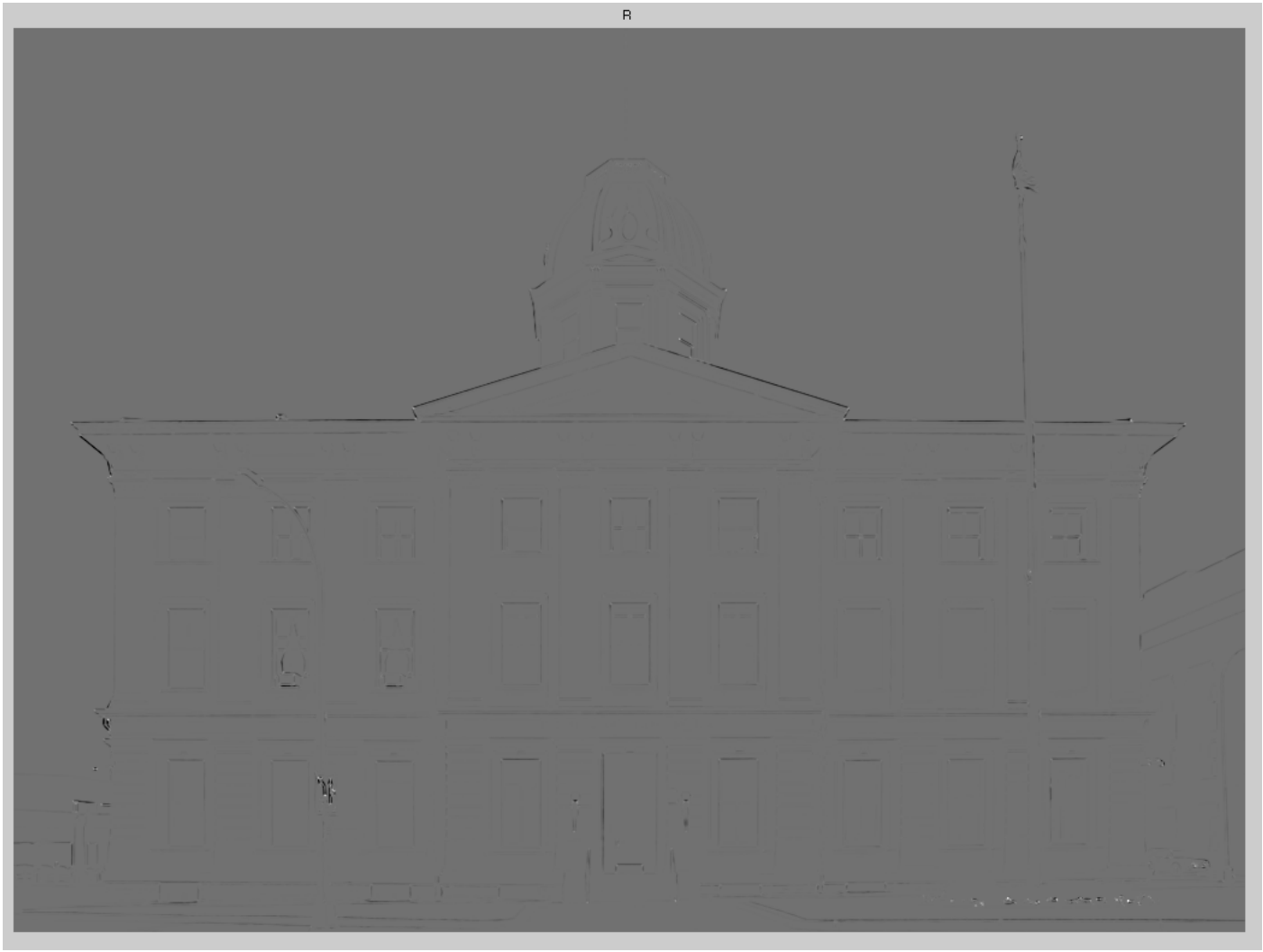

Iy2


Ixy


gIx2


gly2

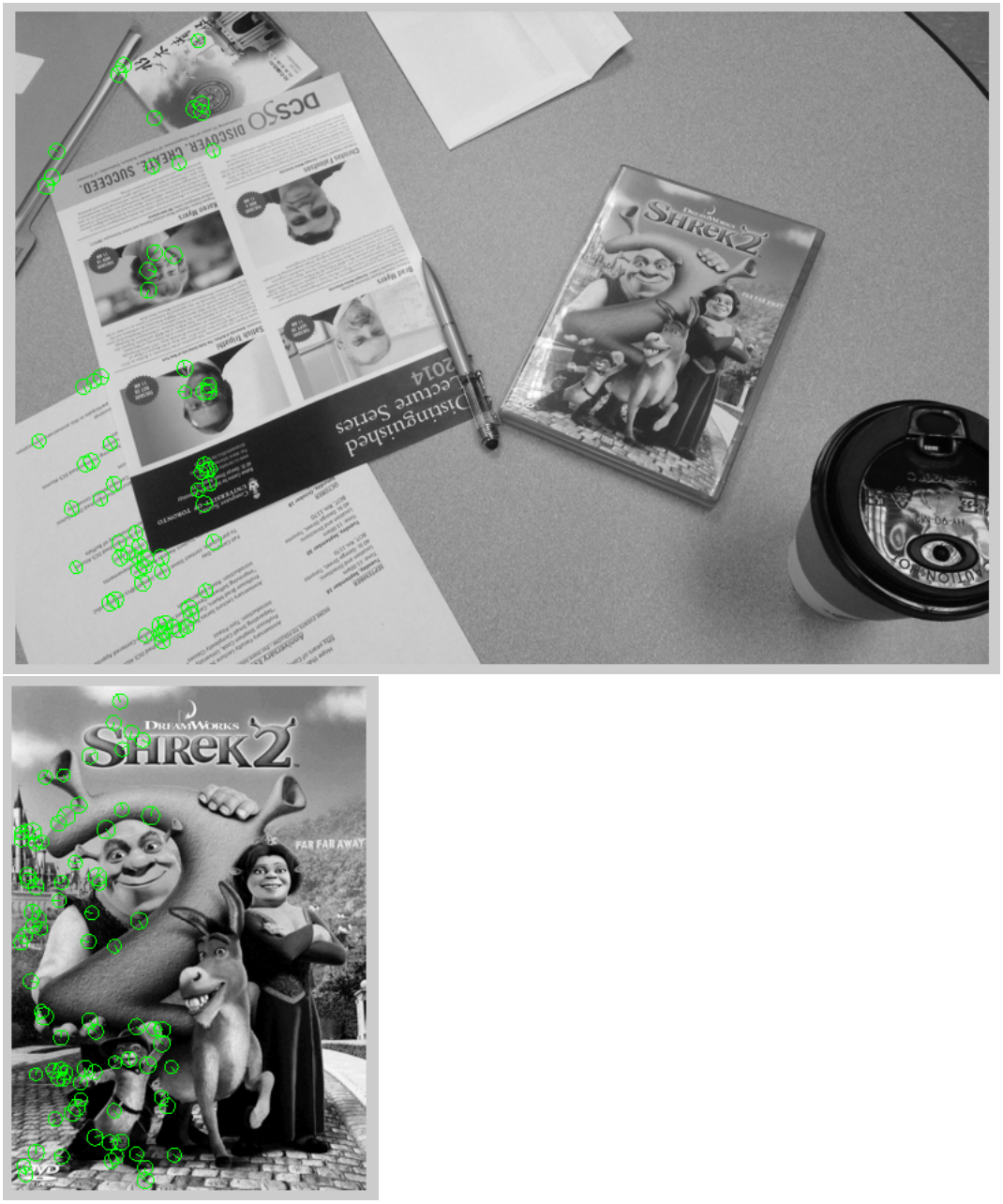

glxy


R

2

(a) **Feature extraction:**

```
im_test = imreadbw('test.png');
im_reference = imreadbw('reference.png');

[testFrames, testDescr] = sift(im_test);
[refFrames, refDescr] = sift(im_reference);

figure; imshow(im_test);
hold on;
h = plotsiftframe(testFrames(:,1:100)); set(h, 'LineWidth', 1, 'Color', 'g');

figure; imshow(im_reference);
hold on;
h = plotsiftframe(refFrames(:,1:100)); set(h, 'LineWidth', 1, 'Color', 'g');
```



(b) A simple matching algorithm to find the best feature matches is to compare all the features between the pair of images and compute the Euclidean distance, and find the closest match (min Euclidean distance). To ensure the results are reliable, compute the ratio between the closest and second closest match and determine that it meets a predetermined threshold.

```
1 im_test = imreadbw('test.png');
2 im_reference = imreadbw('reference.png');
3
4 [testFrames, testDescr] = sift(im_test);
```

```
[refFrames, refDescr] = sift(im_reference);

% Compare the Euclidean distances of all descriptor pairs
result = []; % Keeps track of the closest matches between the 2 image descriptors
                % and its computed ratio
threshold = 0.3;
for i = 1:size(testDescr, 2)
    minDist1 = inf; % closest match
    minDist2 = inf; % second closest match
    minIndex = 0;
    for j = 1:size(refDescr, 2)
        testDescriptor = testDescr(:,i);
        refDescriptor = refDescr(:,j);
        dist = dist2(testDescriptor, refDescriptor);

        if dist < minDist1
            minDist2 = minDist1;
            minDist1 = dist;
            minIndex = j;
        end
    end

    ratio = minDist1 / minDist2;
    if ratio > threshold
        result = [result ; i j ratio];
    end
end

result = sortrows(result, 3);
correspondences = result(1:3, :);
```