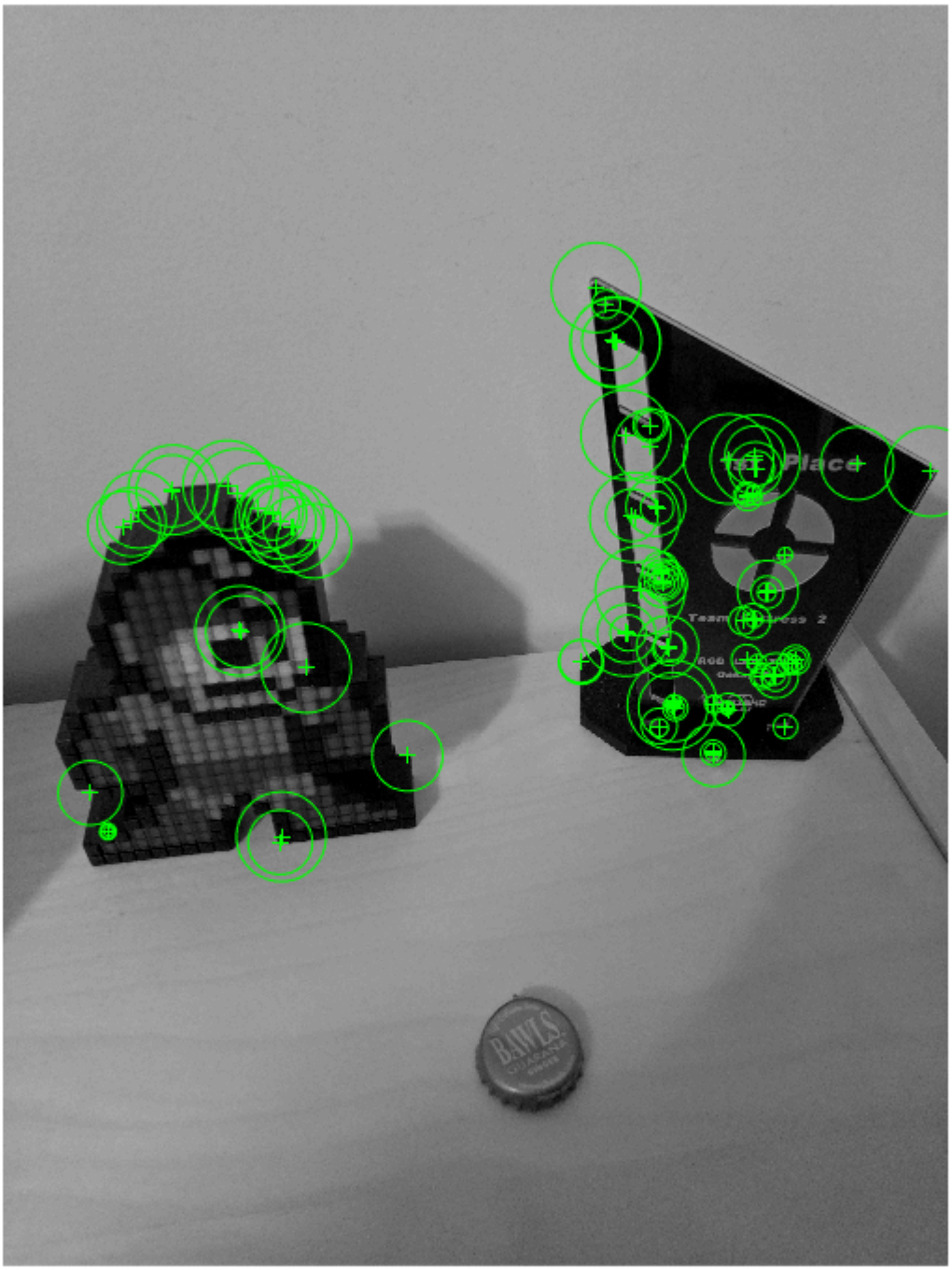
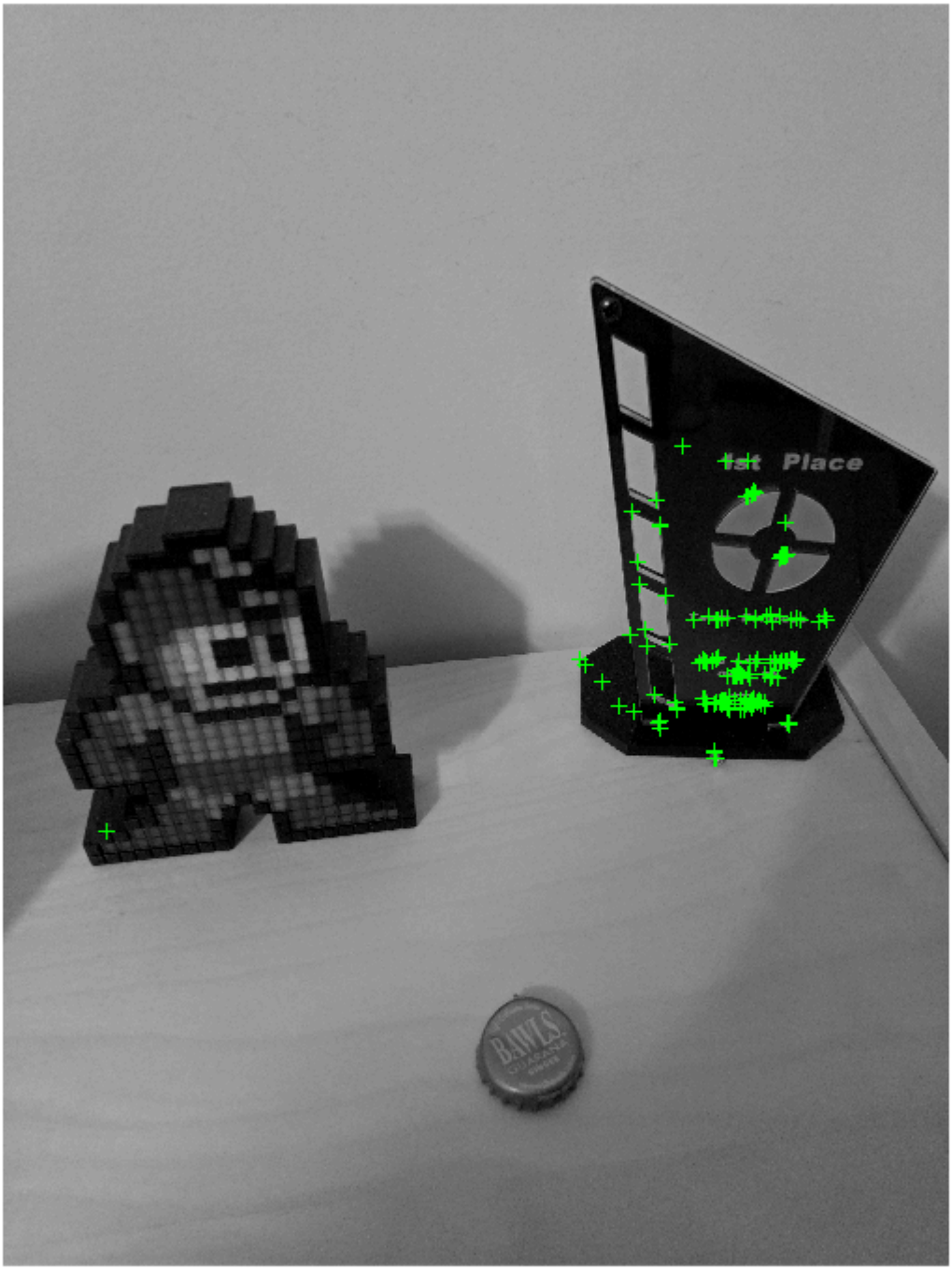
```
addpath 'D:'\homework\'checkerboard pics'\
%1
I = imread('megaman3.jpg');
I = rgb2gray(I);
%brisk
points = detectBRISKFeatures(I);
imshow(I); hold on;
plot(points.selectStrongest(100));
hold off
```



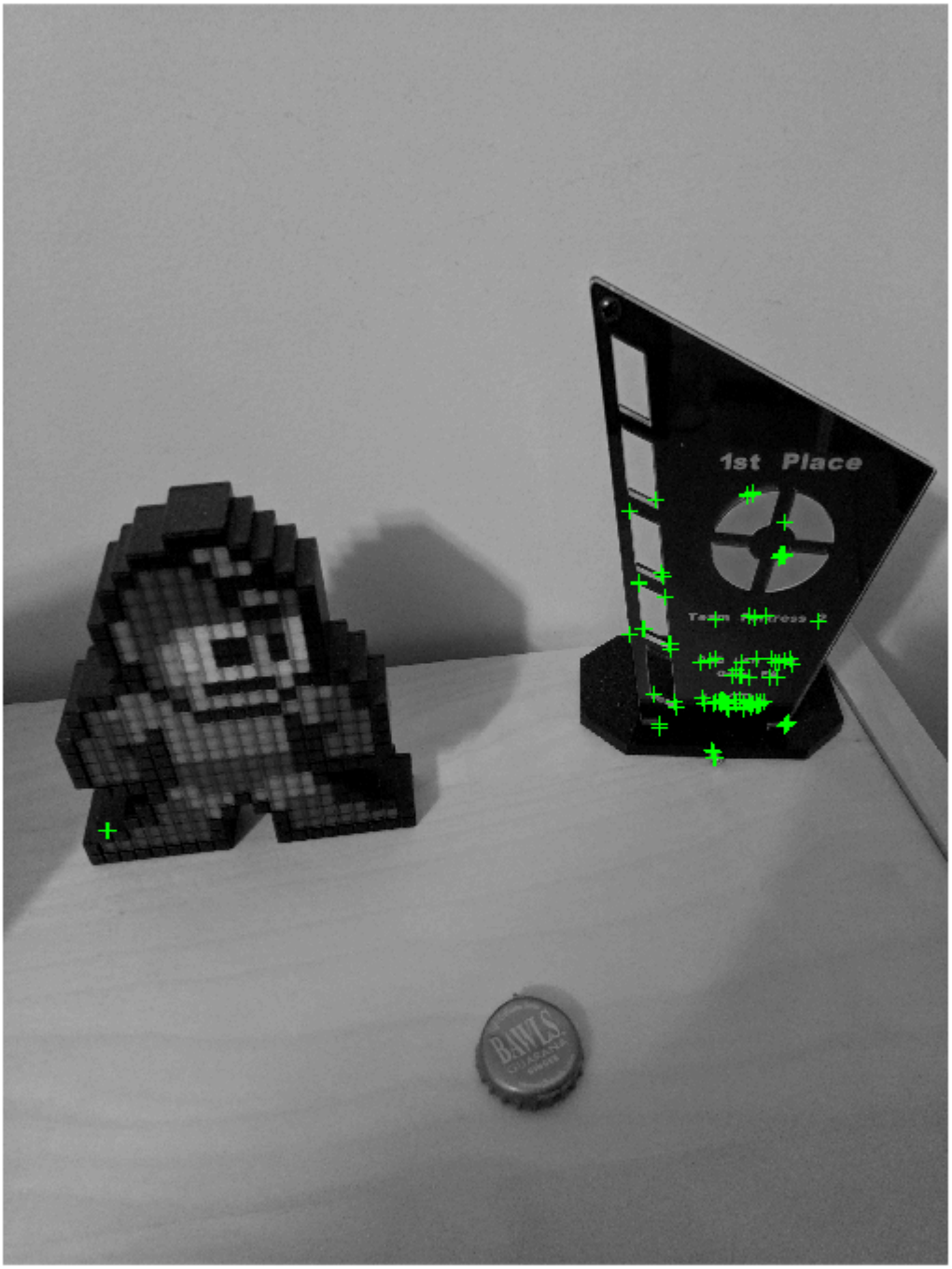
```
%a couple corners mistaken on the body of megaman, but mostly accurate.
%does not notice bottle cap

%fast
corners = detectFASTFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(1000));
```

```
hold off
```



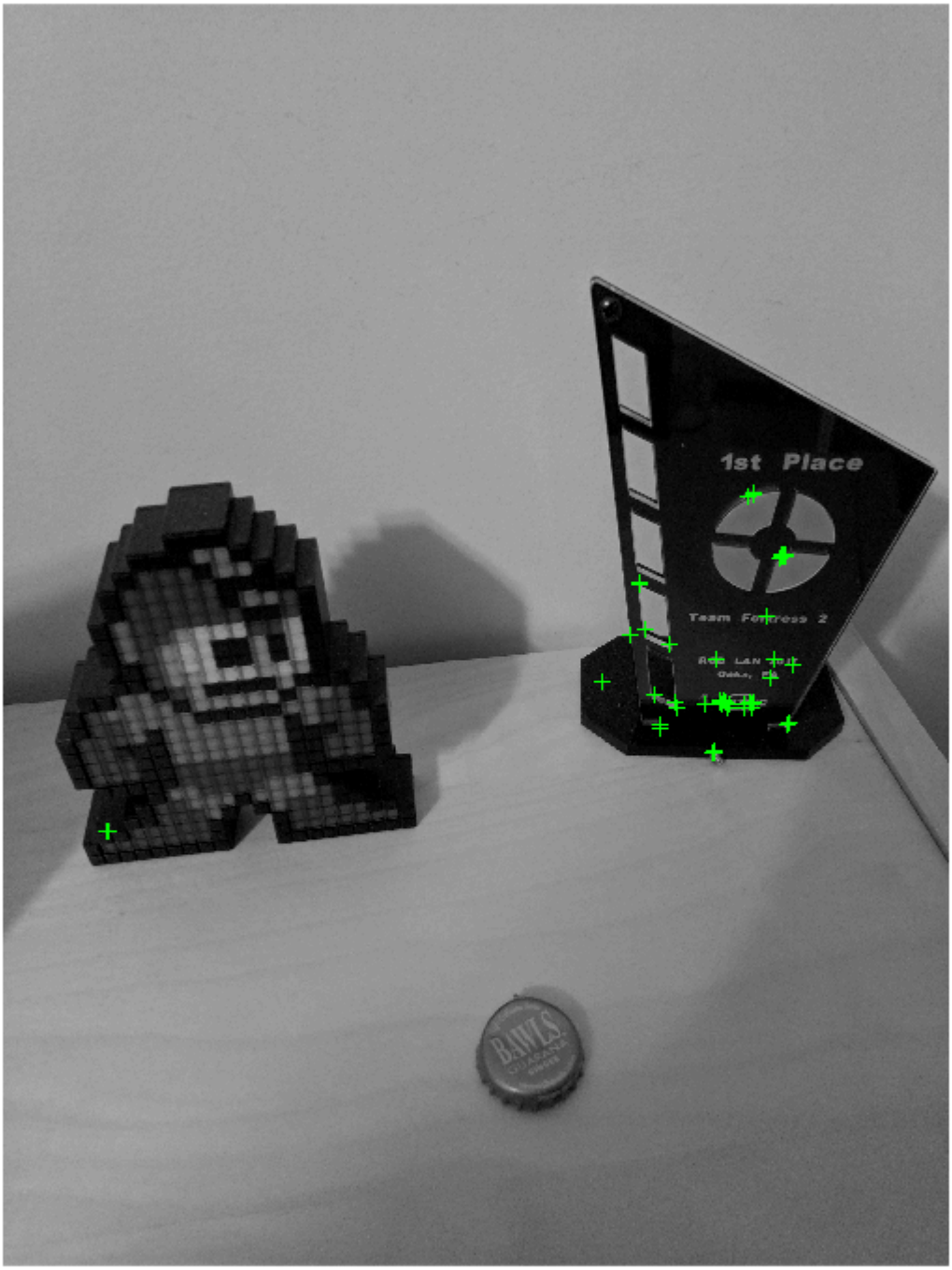

```
%even with 1000 points, only 1 for megaman and none for bottle cap

%harris
corners = detectHarrisFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(100));
```

```
hold off
```



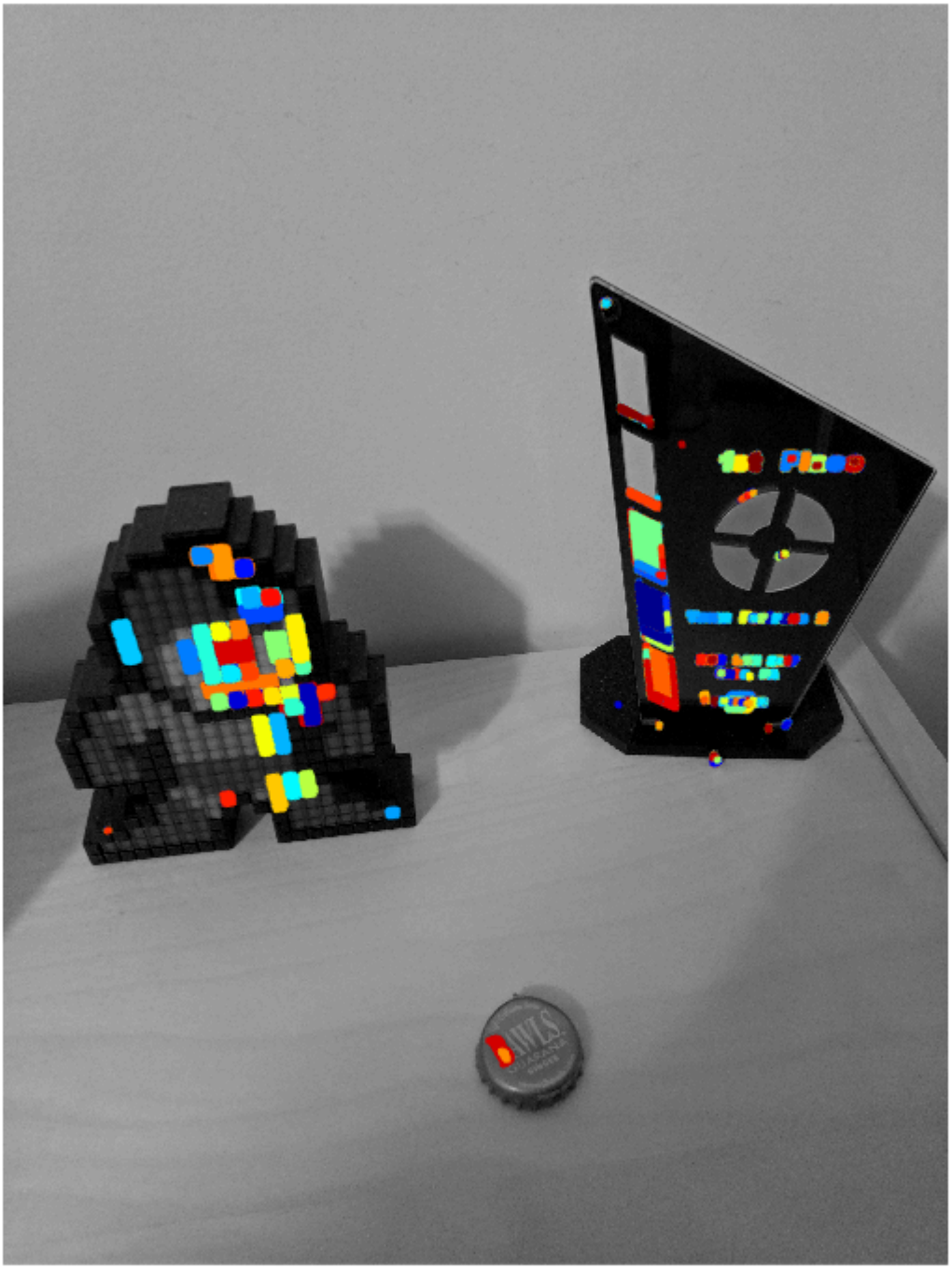

```
%similar results to fast

%mini eigen
corners = detectMinEigenFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

```
hold off
```



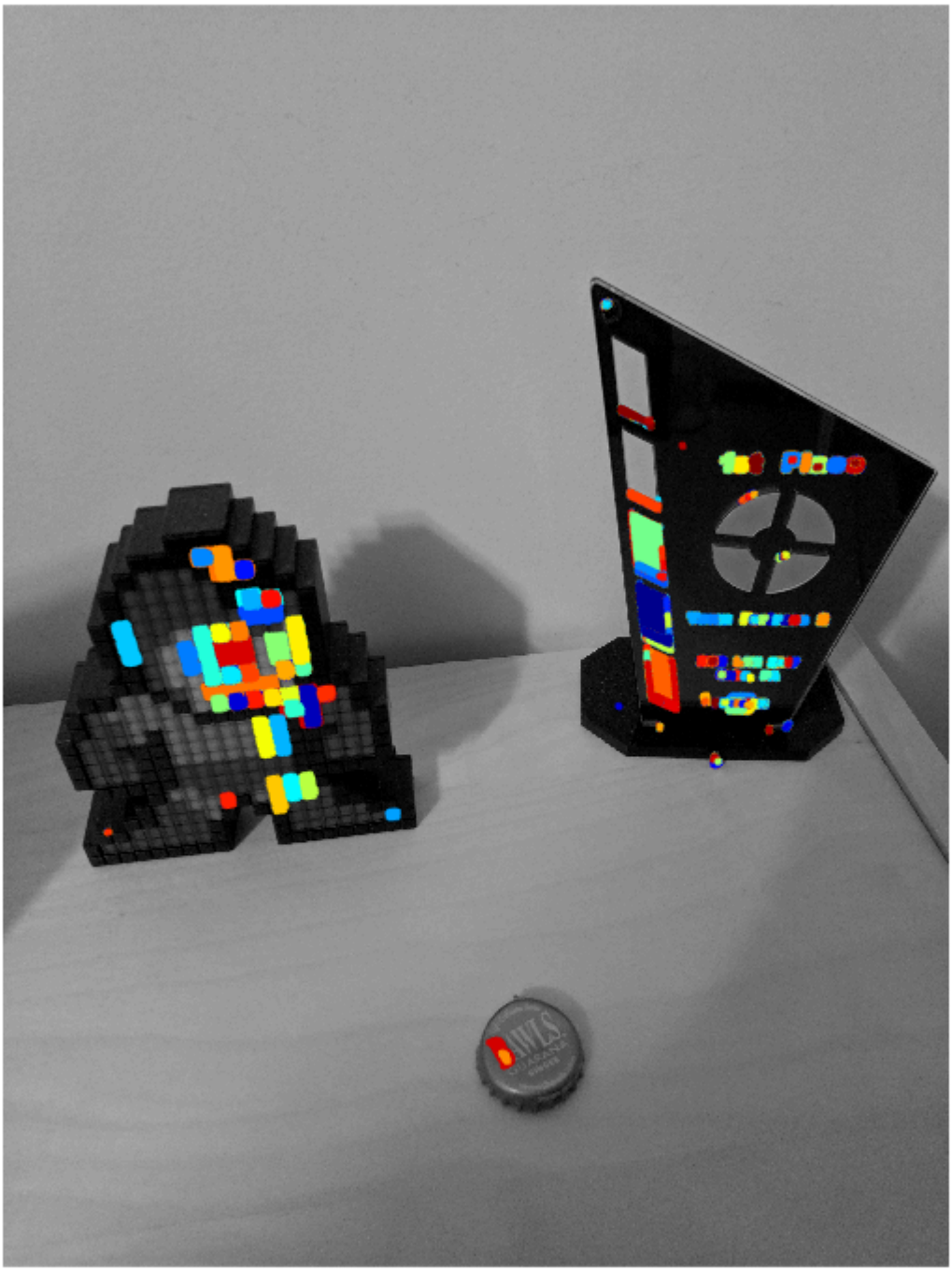

```
%also similar results

%mser
regions = detectMSERFeatures(I);
figure; imshow(I); hold on;
plot(regions, 'showPixelList',true,'showEllipses',false);
```

```
hold off
```



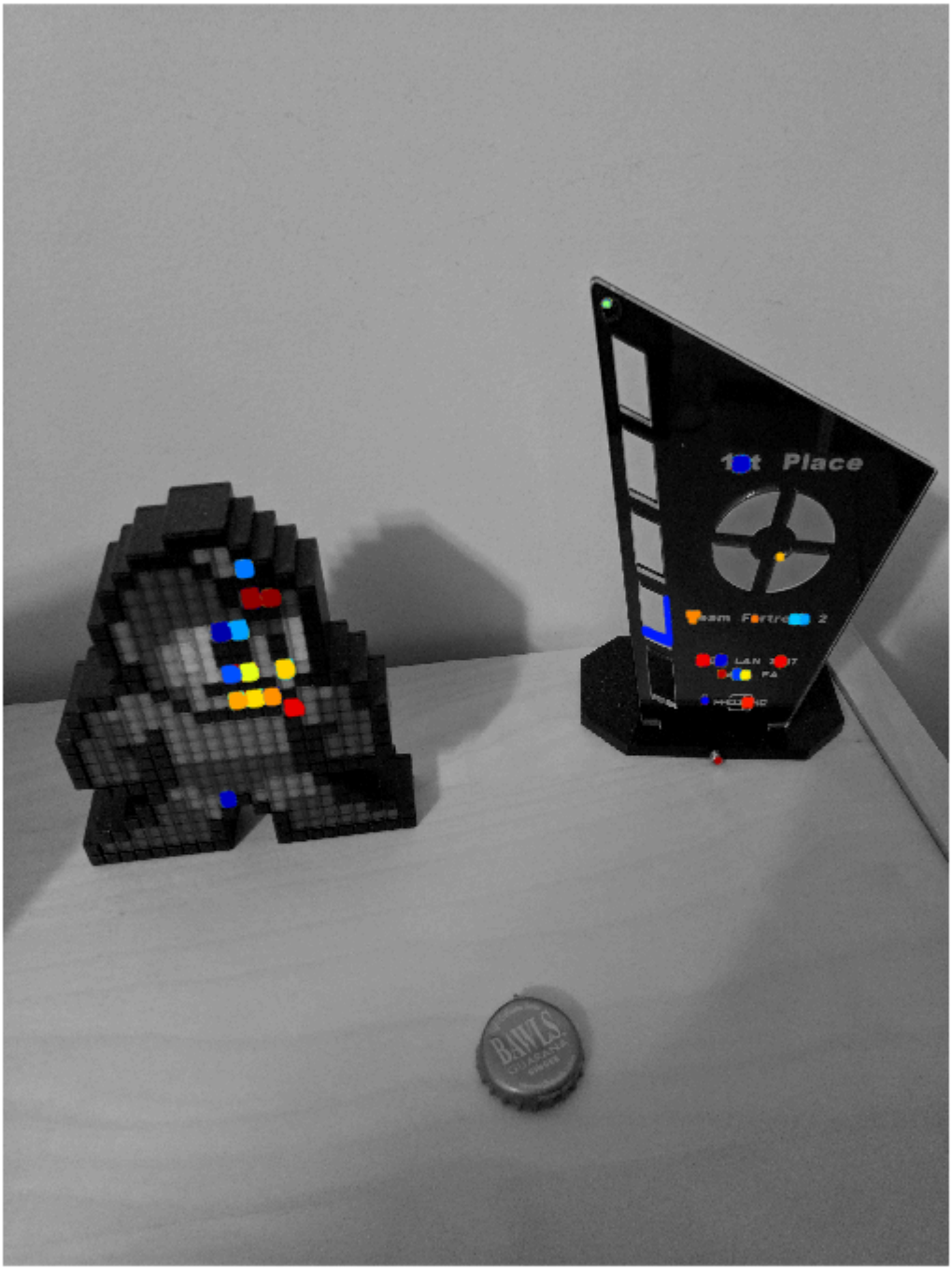
```
%recognizes the bottle cap! first to do so

[regions,mserCC] = detectMSERFeatures(I);
figure
imshow(I)
hold on
```

```
plot(regions,'showPixelList',true,'showEllipses',false);
hold off
```



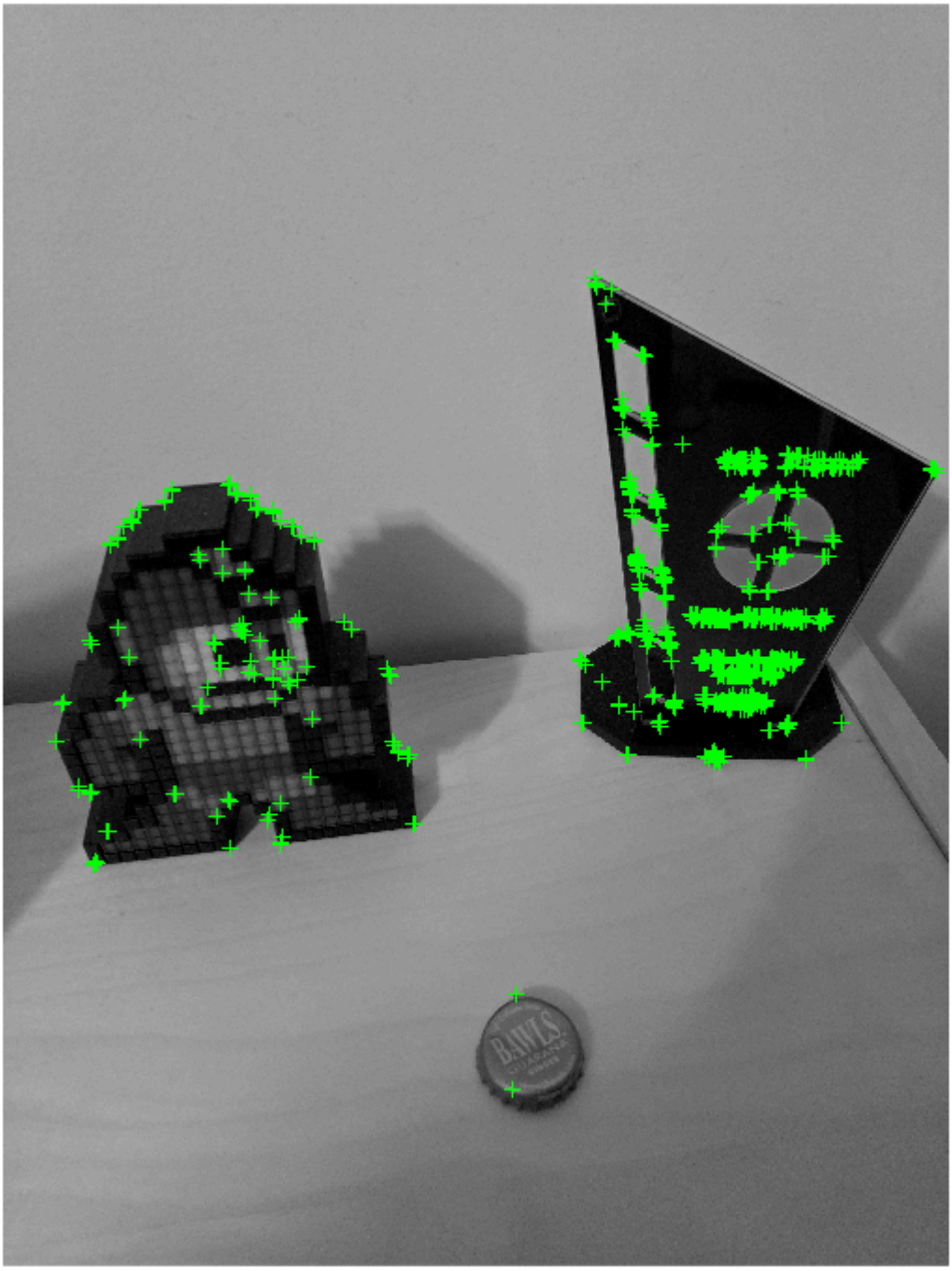
```
stats = regionprops('table',mserCC,'Eccentricity');
eccentricityIdx = stats.Eccentricity < 0.55;
circularRegions = regions(eccentricityIdx);
figure
```

```
imshow(I)
hold on
plot(circularRegions,'showPixelList',true,'showEllipses',false)
hold off
```



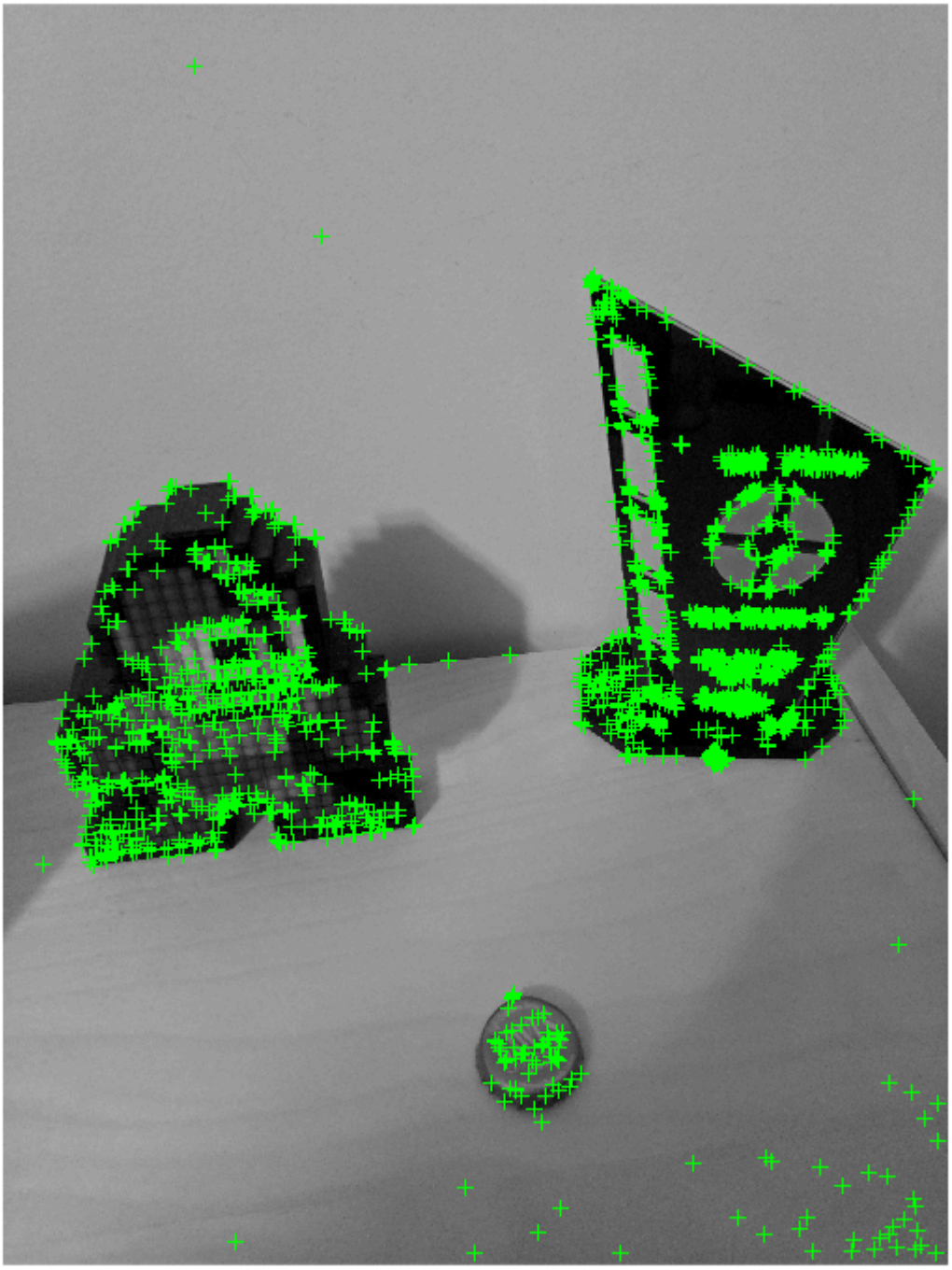

```
%a threshold applied to find most unique(?) points

%random, but works really well?
```

```
imshow(I)
hold on
plot(points,'ShowScale',false)
hold off
```



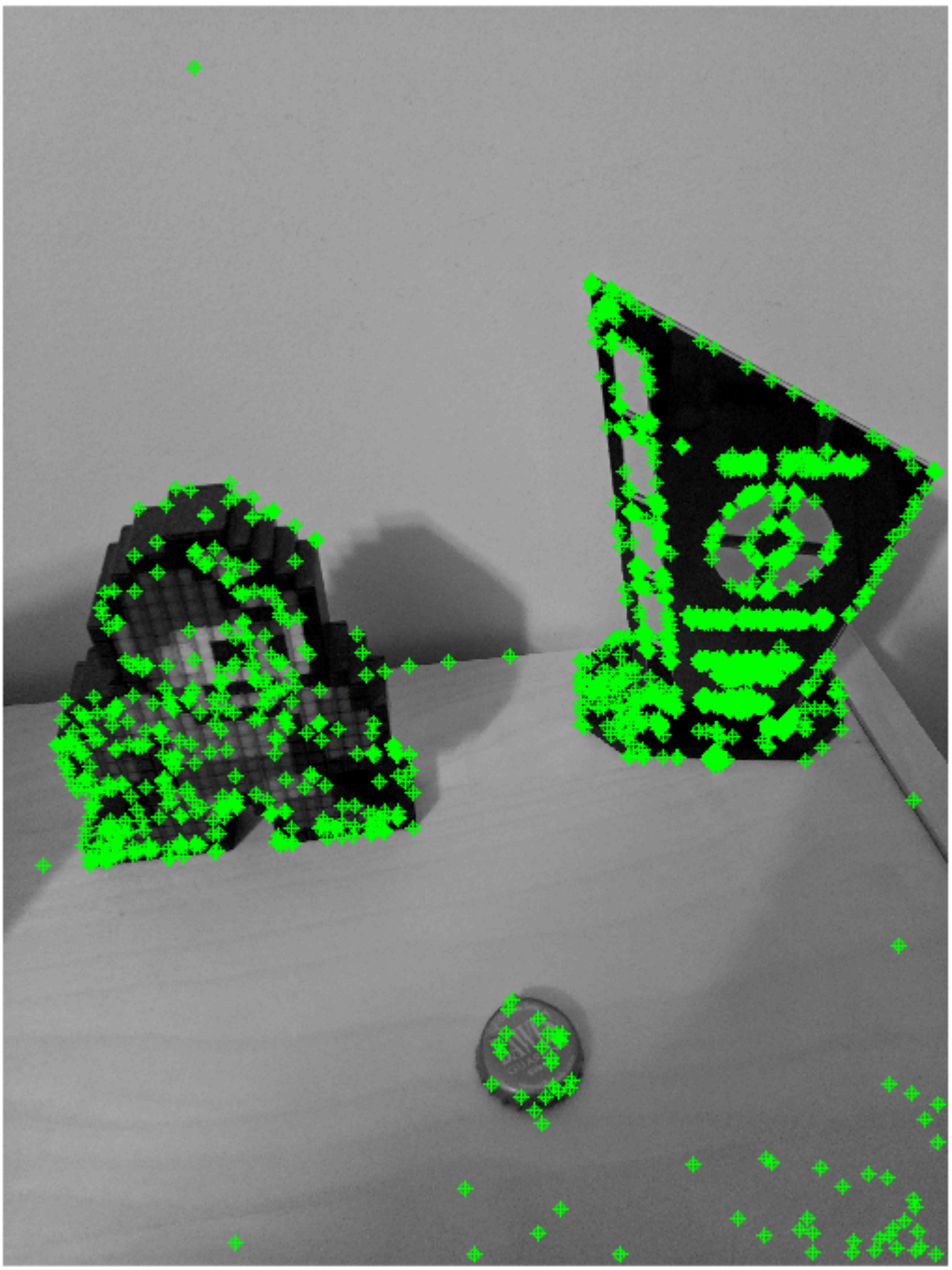
```
%orb
points = detectORBFeatures(I);
```

```
imshow(I)
hold on
plot(points,'ShowScale',false)
hold off
```



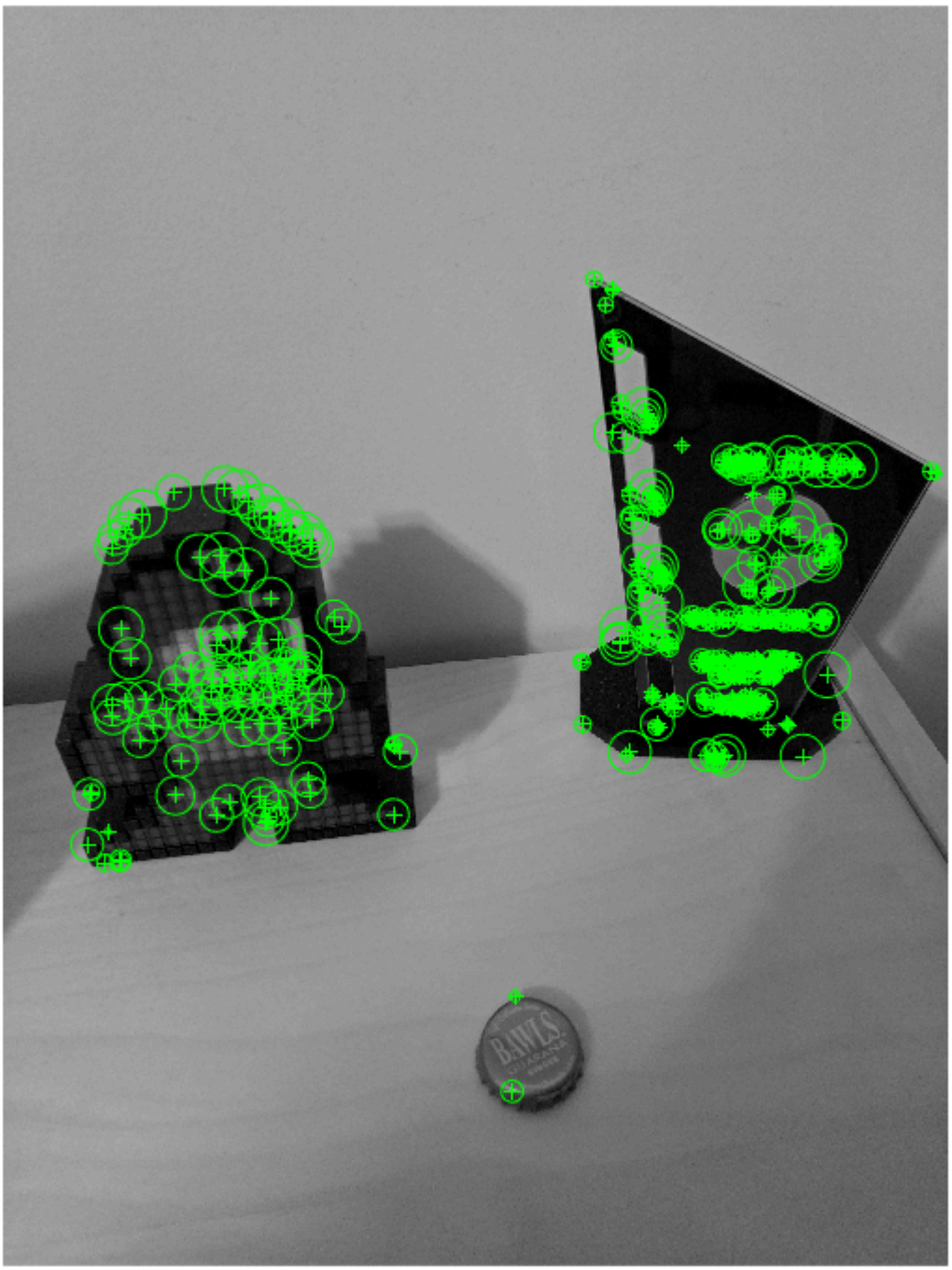
```
%a lot of points shown, happy it found the bottle cap but there are many
%useless points
```

```
points = detectORBFeatures(I,'ScaleFactor',1.01,'NumLevels',3);
imshow(I)
hold on
plot(points)
hold off
```



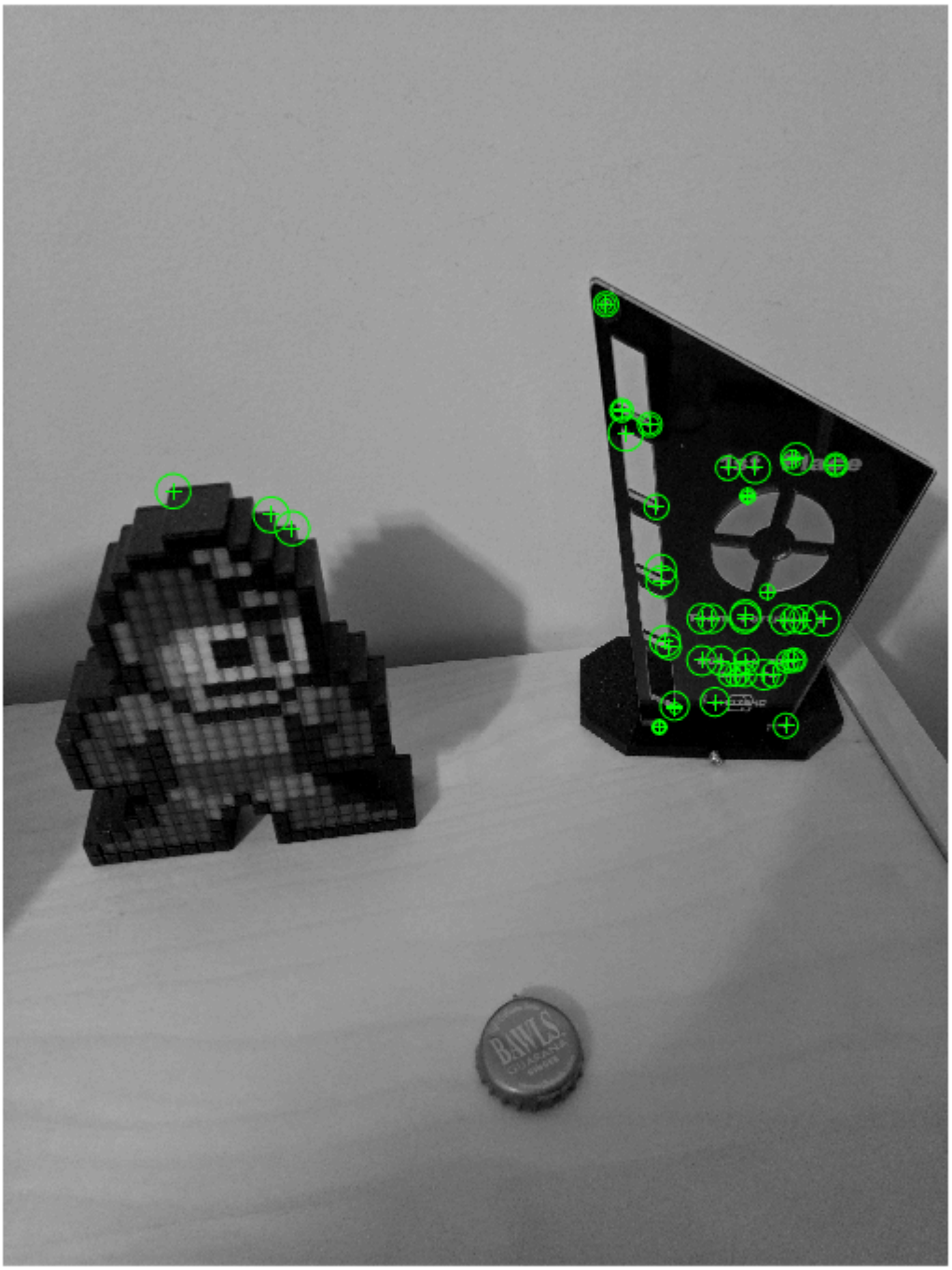
```
%surf
```

```
points = detectSURFFeatures(I);
imshow(I)
hold on
plot(points.selectStrongest(500));
hold off
```



```
%notices all! thanks surf you are great :)
%with 50 points, it's hard to get everything, 500 seems to do the job
```

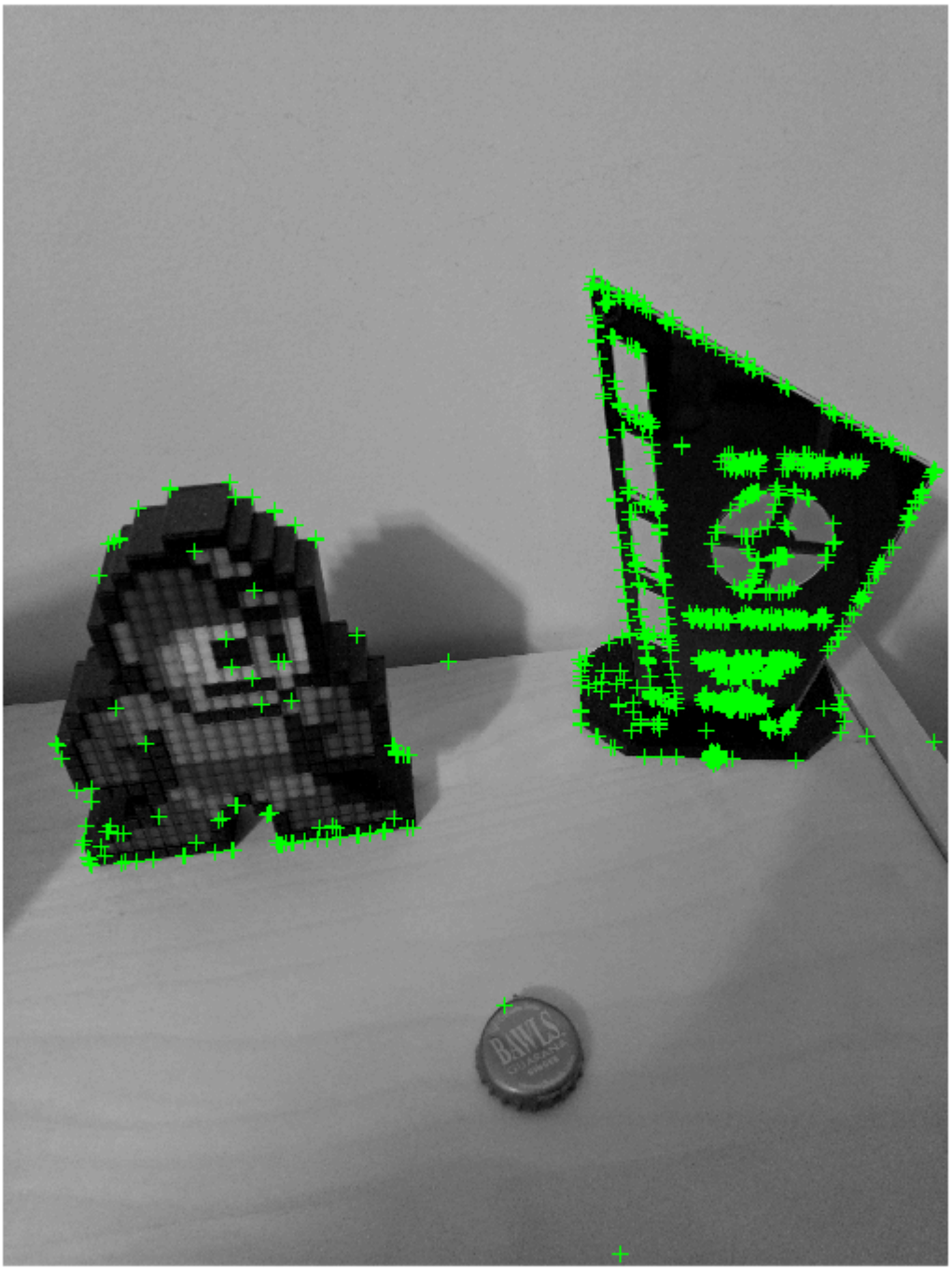
```
%kaze
points = detectKAZEFeatures(I);
imshow(I)
hold on
plot(selectStrongest(points,50))
hold off
```



```
%very similar to surf, with 10x more points it will find everything. kept
%at 50 to show difference

imshow(I)
hold on
plot(points(end-4:end));
hold off
```

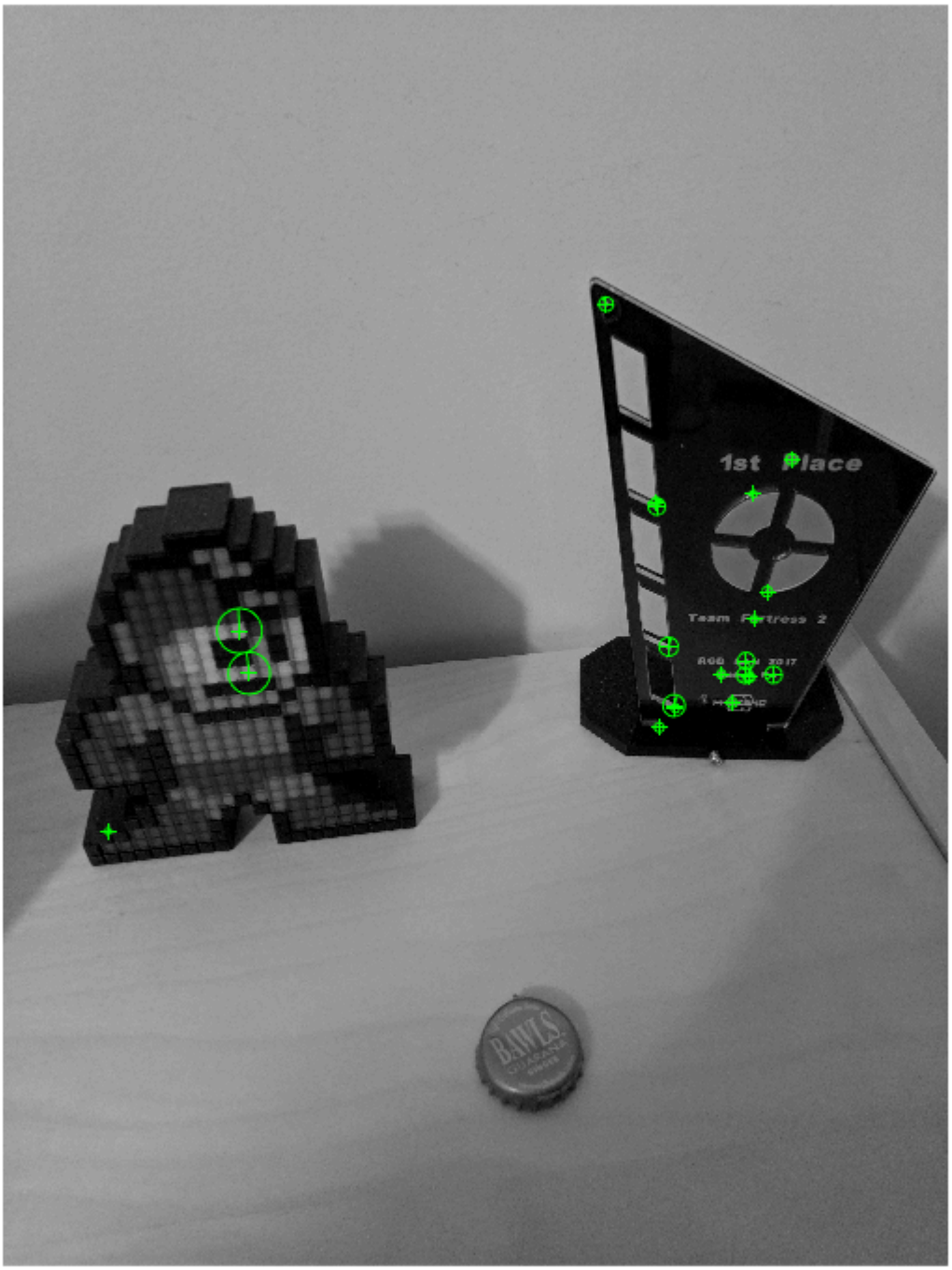
```
%only finds bottle cap for some reason
```

```
addpath 'D:'\homework\'checkerboard pics'\

I = imread('megaman3.jpg');
I = rgb2gray(I);

%harris
corners = detectHarrisFeatures(I);
[features, valid_corners] = extractFeatures(I, corners);
figure
imshow(I)
hold on
plot(valid_corners);
```



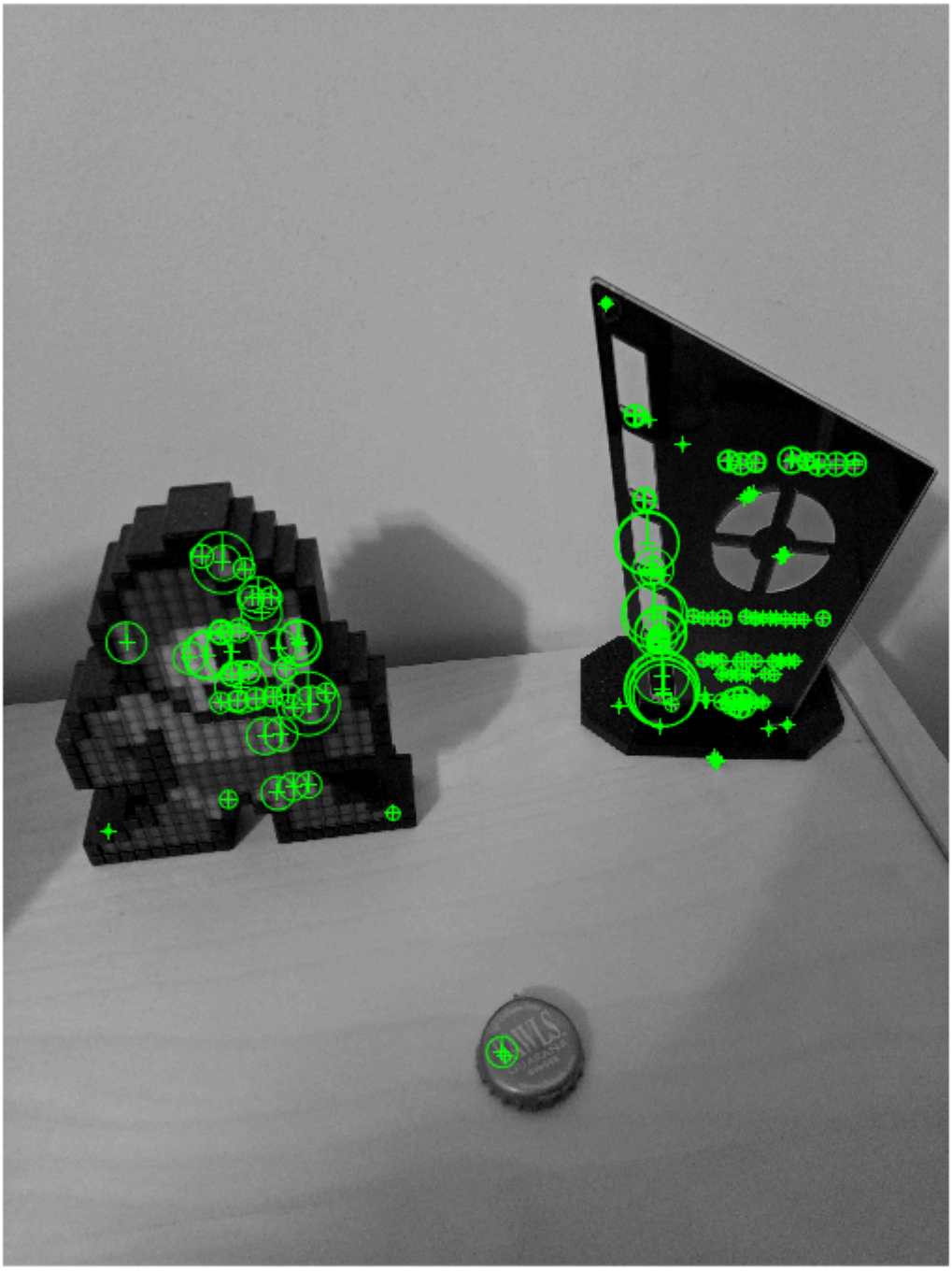

```
%good job at finding corners, again mistaken by the body of megaman. finds
%the bottle cap and even the edge of my bookshelf

%surf
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I, points);
figure
imshow(I)
```

```
hold on
plot(valid_points.selectStrongest(20),'showOrientation',true);
```



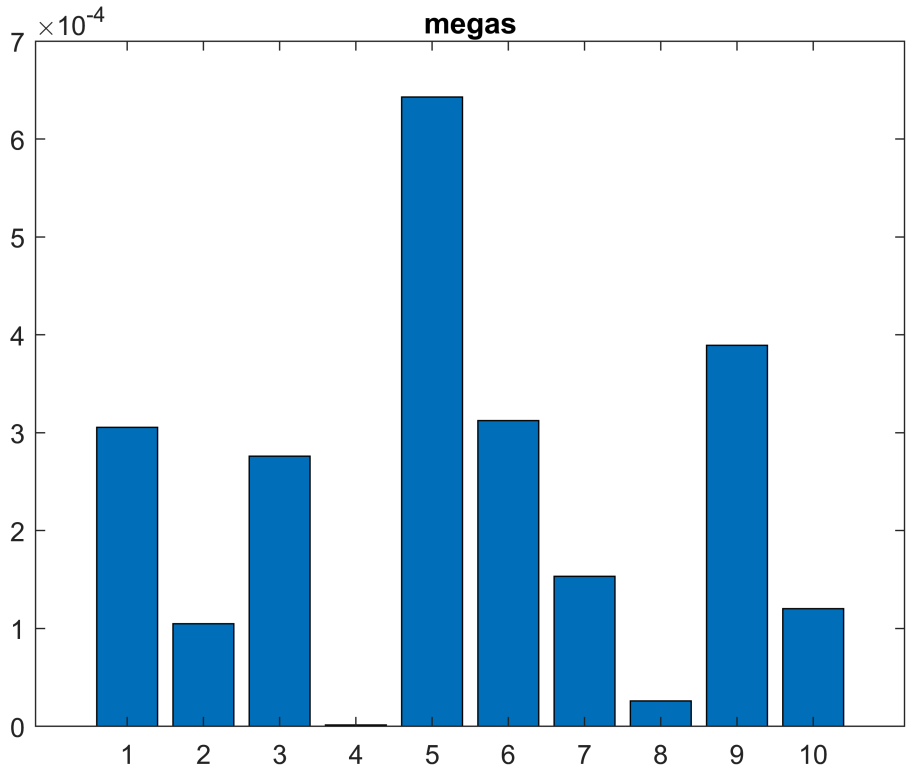

```
%small amount of points, doesn't find bottle cap

%mser
regions = detectMSERFeatures(I);
[features, valid_points] = extractFeatures(I,regions,'Upright',true);
```

```
figure
imshow(I)
hold on
plot(valid_points,'showOrientation',true);
```

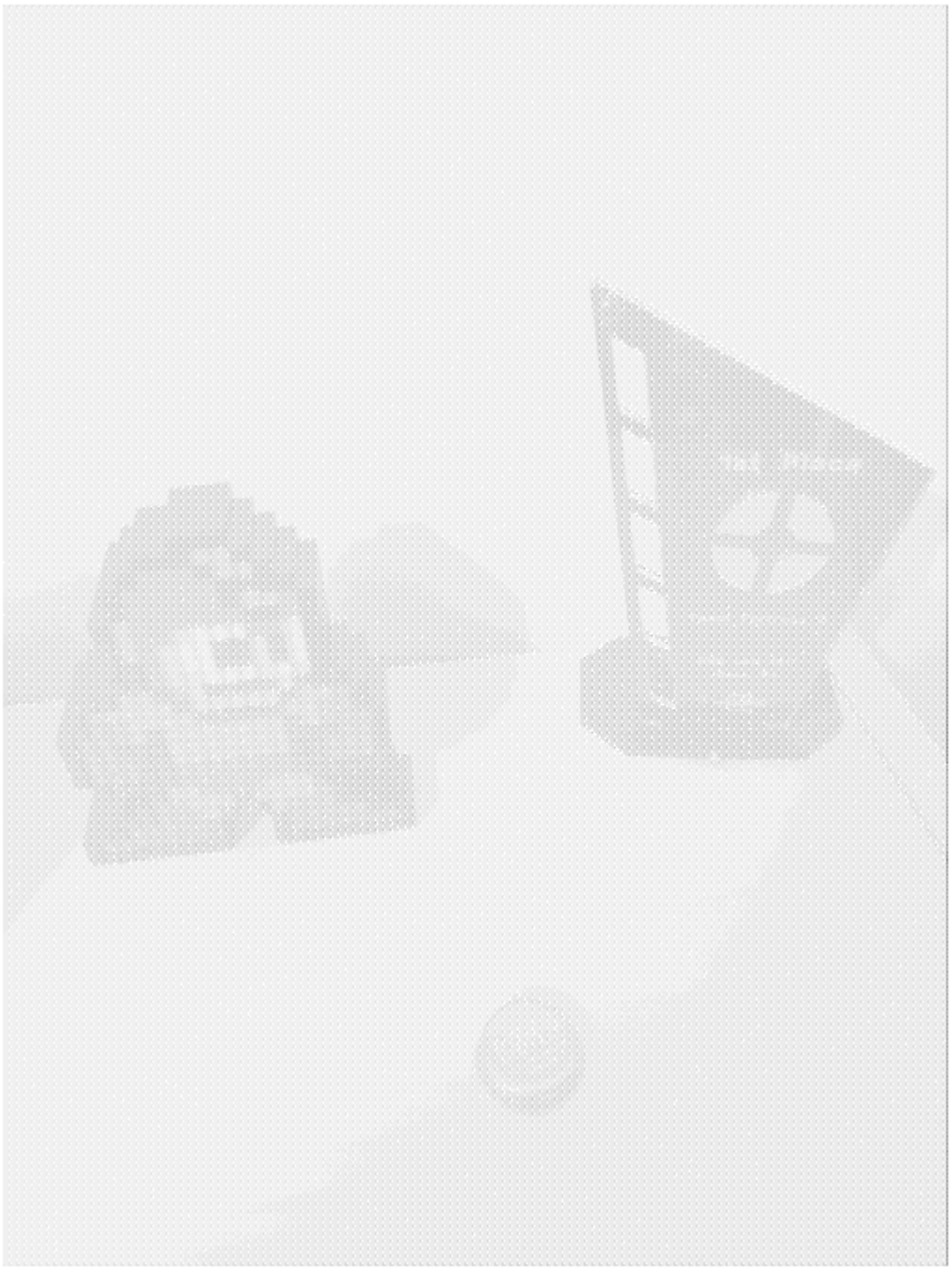


```
%perfect

%lbp
```

```
X = imread('megaman1.jpg');
X = rgb2gray(X);
lbp1 = extractLBPFeatures(I,'Upright',false);
lbp2 = extractLBPFeatures(X,'Upright',false);
megaVsMega = (lbp1 - lbp2).^2;

figure
bar([megaVsMega],'grouped')
title('megas')
```




```
%hog
[featureVector,hogVisualization] = extractHOGFeatures(I);
figure
imshow(I)
hold on
plot(hogVisualization)
```

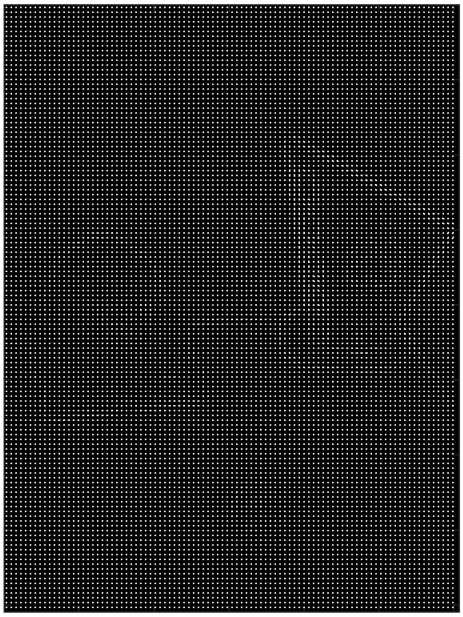


```
[hog1,visualization] = extractHOGFeatures(I,'CellSize',[32 32])
```

```
hog1 = 1×418500 single row vector
    0.1752    0.1527    0.1776    0.1701    0.1919    0.1521    0.1405    0.1529 ···
visualization =
Visualization

   Type plot(visualization) to visualize.
```

```
  Read-only properties:
              CellSize: [32 32]
             BlockSize: [2 2]
          BlockOverlap: [1 1]
               NumBins: 9
    UseSignedOrientation: 0
             BinCenters: [18×1 double]
```

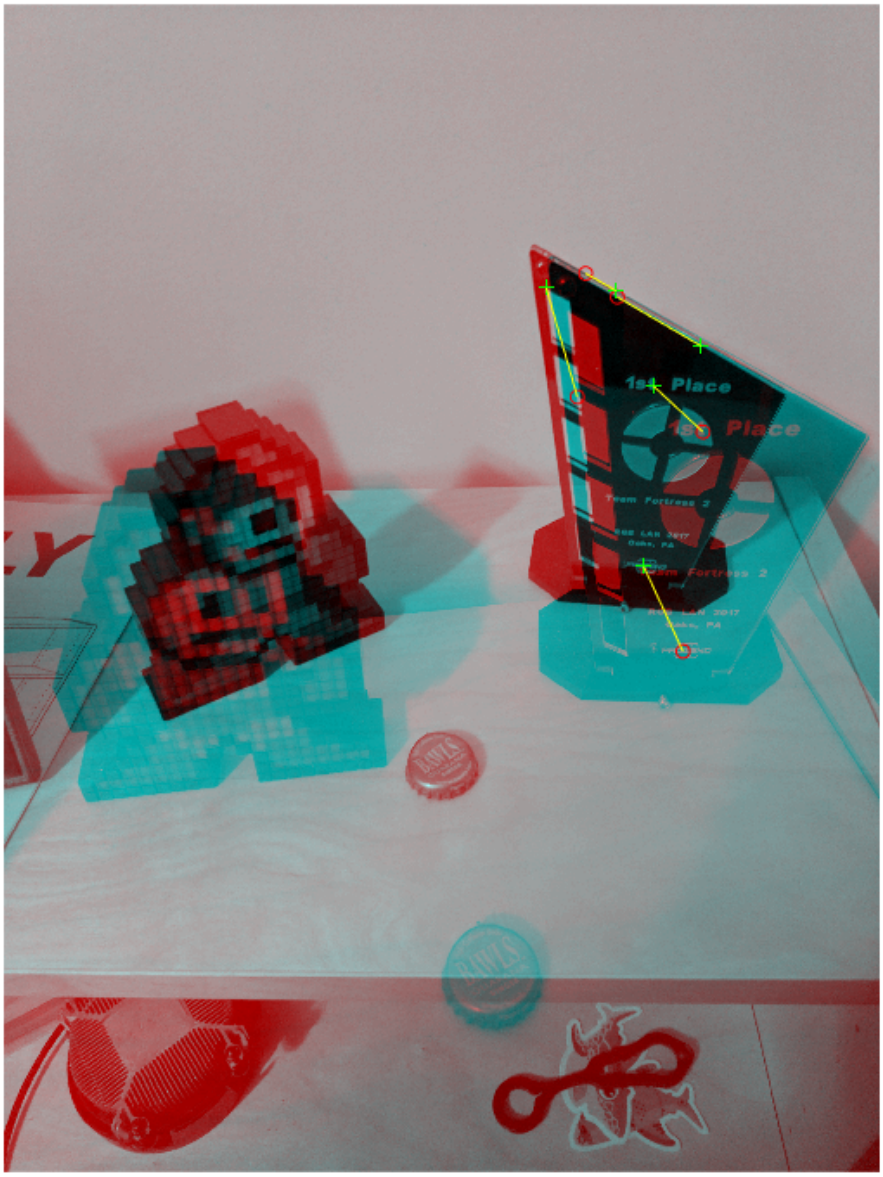
```
subplot(1,2,1);
imshow(I);
subplot(1,2,1);
plot(visualization);
```



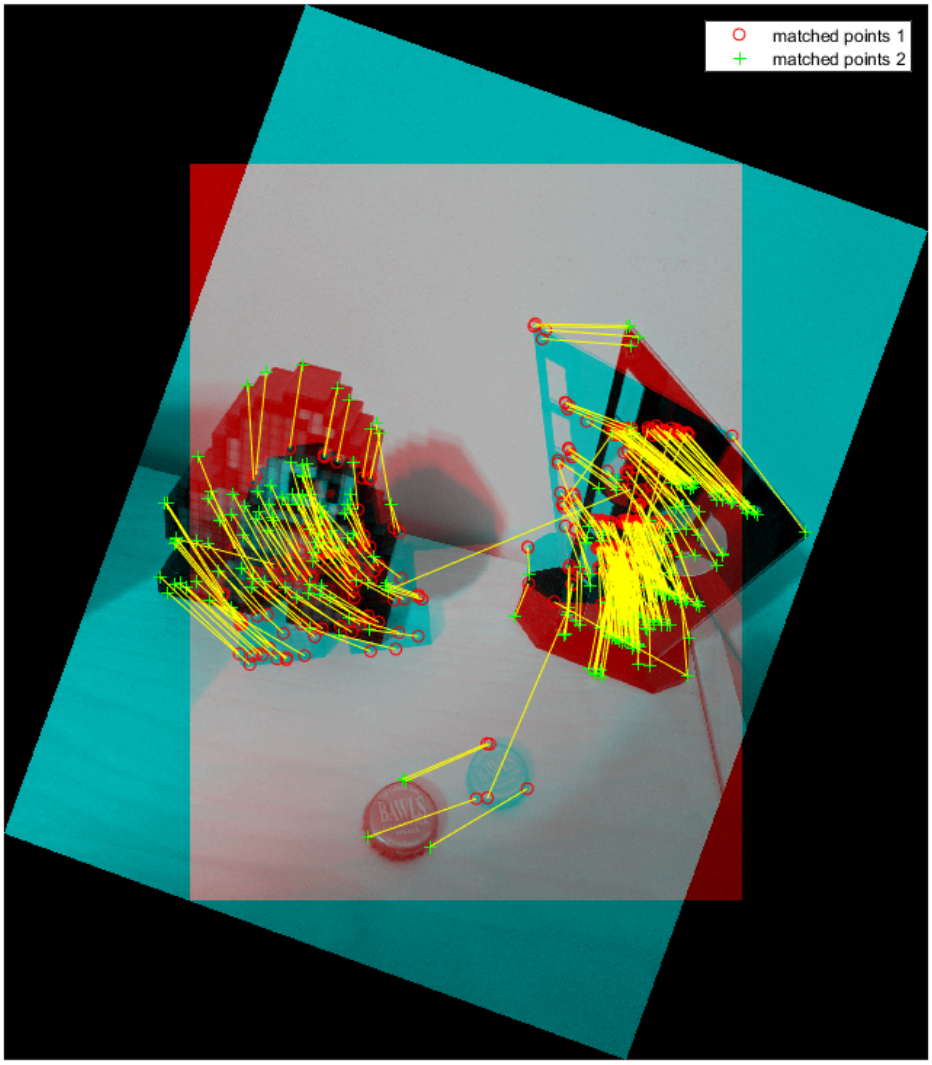
```
%fast
corners = detectFASTFeatures(I);
strongest = selectStrongest(corners,3);
[hog, validPoints,ptVis] = extractHOGFeatures(I,strongest);
figure
imshow(I)
hold on
```

```
plot(ptVis,'Color','green');
```

```
addpath 'D:'\homework\'checkerboard pics'\

I = imread('megaman3.jpg');
I = rgb2gray(I);


Y = imread('megaman2.jpg');
Y = rgb2gray(Y);
%match features
points1 = detectHarrisFeatures(I);
points2 = detectHarrisFeatures(Y);

[features1,valid_points1] = extractFeatures(I,points1);
[features2,valid_points2] = extractFeatures(Y,points2);

indexPairs = matchFeatures(features1,features2);

matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);

figure; showMatchedFeatures(I,Y,matchedPoints1,matchedPoints2);
```

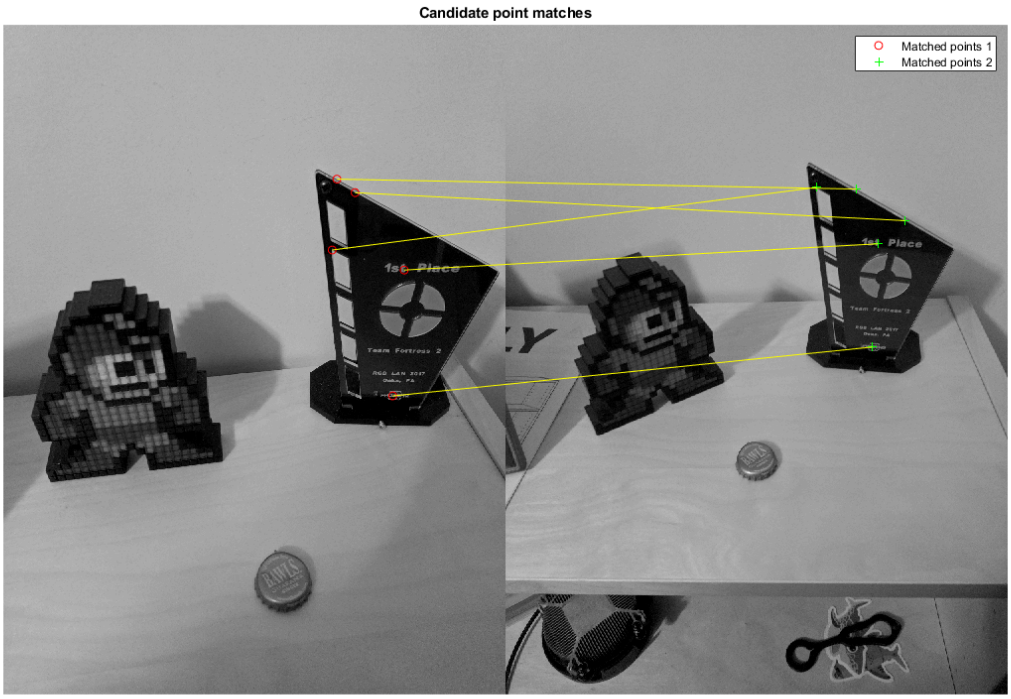



```
I2 = imresize(imrotate(I,-20),1.2);
points1 = detectSURFFeatures(I);
points2 = detectSURFFeatures(I2);
[f1,vpts1] = extractFeatures(I,points1);
[f2,vpts2] = extractFeatures(I2,points2);
indexPairs = matchFeatures(f1,f2) ;
matchedPoints1 = vpts1(indexPairs(:,1));
matchedPoints2 = vpts2(indexPairs(:,2));
figure; showMatchedFeatures(I,I2,matchedPoints1,matchedPoints2);
legend('matched points 1','matched points 2');
```




```
%tracks the base, logo, and top of the trophy, but neither of the other two
%objects

%showMatchedFeatures
points1 = detectHarrisFeatures(I);
points2 = detectHarrisFeatures(Y);
%tracks the location of all 3 objects post rotation. works really well in
%this case

[f1, vpts1] = extractFeatures(I, points1);
[f2, vpts2] = extractFeatures(Y, points2);

indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(1:5, 1));
matchedPoints2 = vpts2(indexPairs(1:5, 2));

figure; ax = axes;
showMatchedFeatures(I,Y,matchedPoints1,matchedPoints2,'montage','Parent',ax);
```

```
title(ax, 'Candidate point matches');
legend(ax, 'Matched points 1','Matched points 2');
```




```
%matches the trophy from one image to another, but neither of the other two
%objects
```

```
addpath 'D:'\homework\'checkerboard pics'\

I = imread('megaman3.jpg');
I = rgb2gray(I);

X = imread('megaman1.jpg');
X = rgb2gray(X);

%binary
%not entirely sure what values to put in for the binary features, just used
%those in the example
features1 = binaryFeatures(uint8([1 8 7 2; 8 1 7 2]));
features2 = binaryFeatures(uint8([8 1 7 2; 1 8 7 2]));

[indexPairs matchMetric] = matchFeatures(features1, features2)
```

```
indexPairs = 2×2 uint32 matrix
   1   2
   2   1
matchMetric = 2×1 single column vector
     0
     0
```

```
%brisk
points = detectBRISKFeatures(I);
location = [200:228;200:228]';
points = BRISKPoints(location);

strongest = points.selectStrongest(10);
imshow(I)
hold on
plot(strongest)
hold off
```


```
%garbage
strongest.Location
```

```
ans = 10×2 single matrix
   200   200
   201   201
   202   202
   203   203
   204   204
```

```
   205   205
   206   206
   207   207
   208   208
   209   209
```

```
%kaze
points = detectKAZEFeatures(I);
strongest = selectStrongest(points,10);
imshow(I)
hold on
plot(strongest)
hold off
```


```
%only finds trophy with this low amount of points, but with a higher amount
%of points it also does well
strongest.Location
```

```
ans = 10×2 single matrix
10^3 ×
    1.9378    0.9609
    1.9379    0.9609
```

```
    2.1579    2.2508
    2.1384    2.0617
    1.9376    0.9614
    2.0999    1.6118
    2.1159    1.8514
    2.4756    2.1507
    2.3831    1.9687
    2.3879    2.1060
```

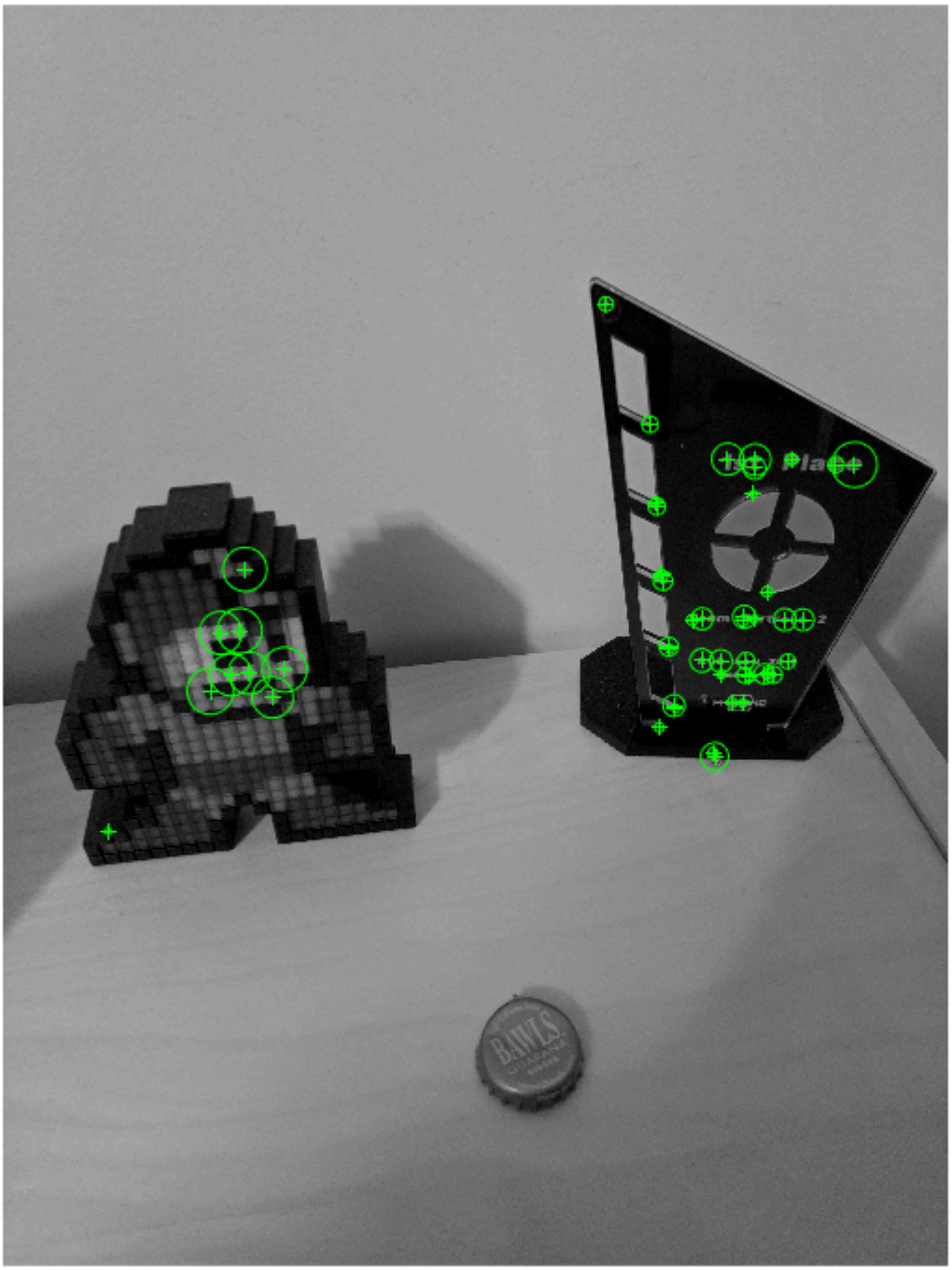
```
imshow(I)
hold on
plot(points(end-4:end)) %whenever i do this notation, i only find the bottlecap
hold off
```


```
%corner
points = detectHarrisFeatures(I);
strongest = selectStrongest(points,50);
imshow(I)
hold on
plot(strongest)
```

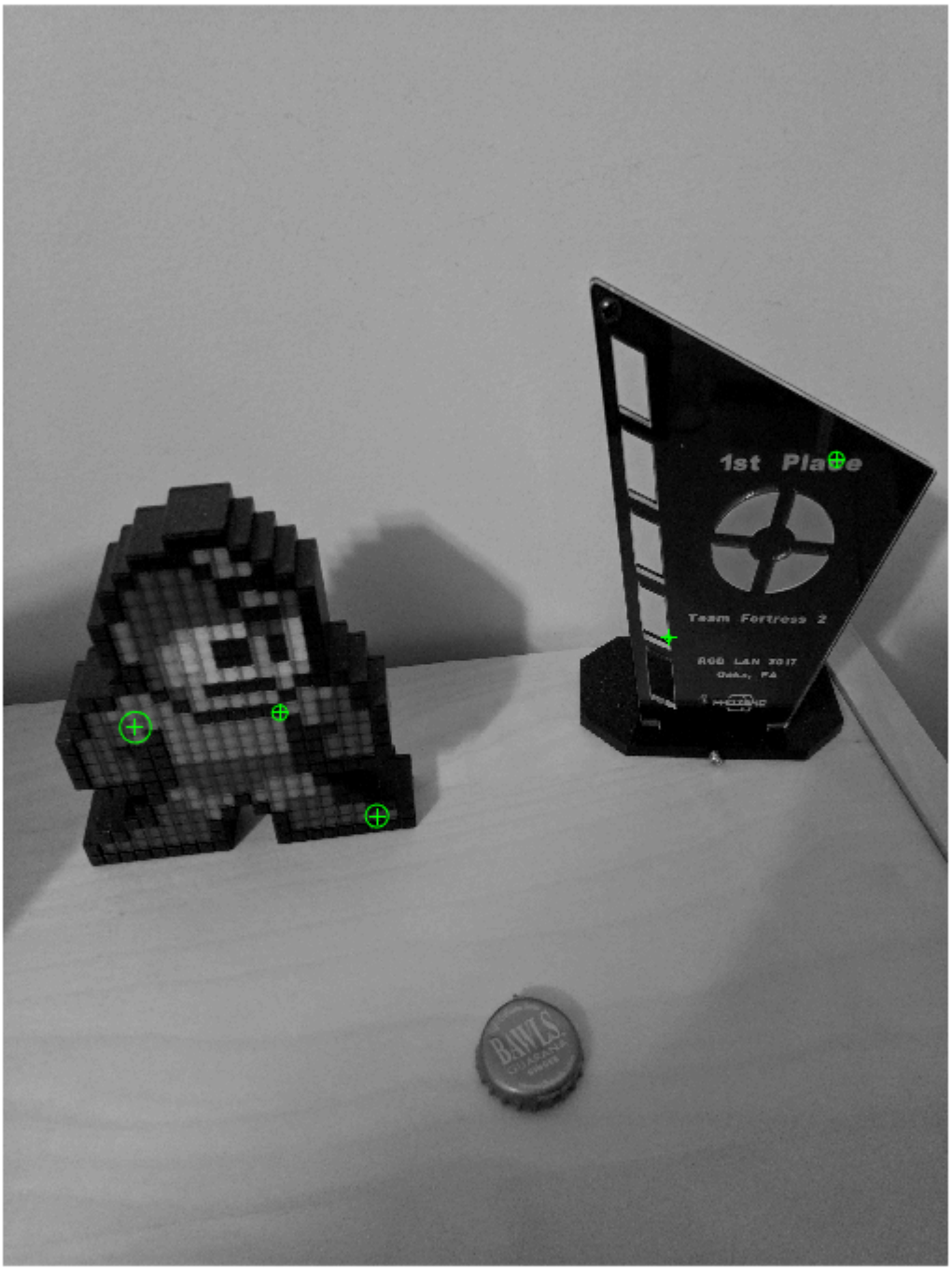
```
hold off
```


```
%surf
points = detectSURFFeatures(I);
strongest = points.selectStrongest(50);
imshow(I); hold on;
```

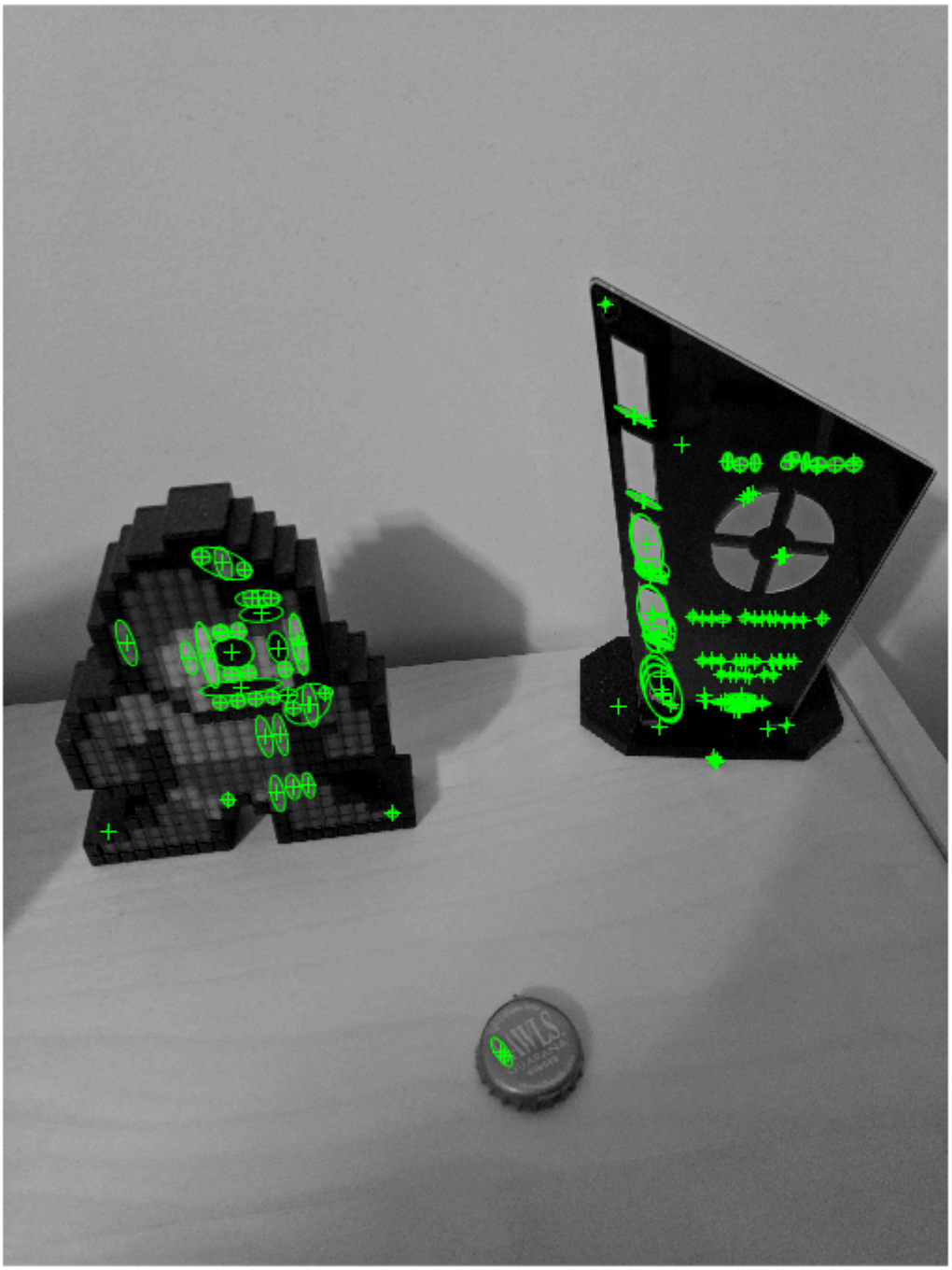
```
plot(strongest);
hold off
```



```
imshow(I); hold on;
plot(points(end-4:end));
hold off
```

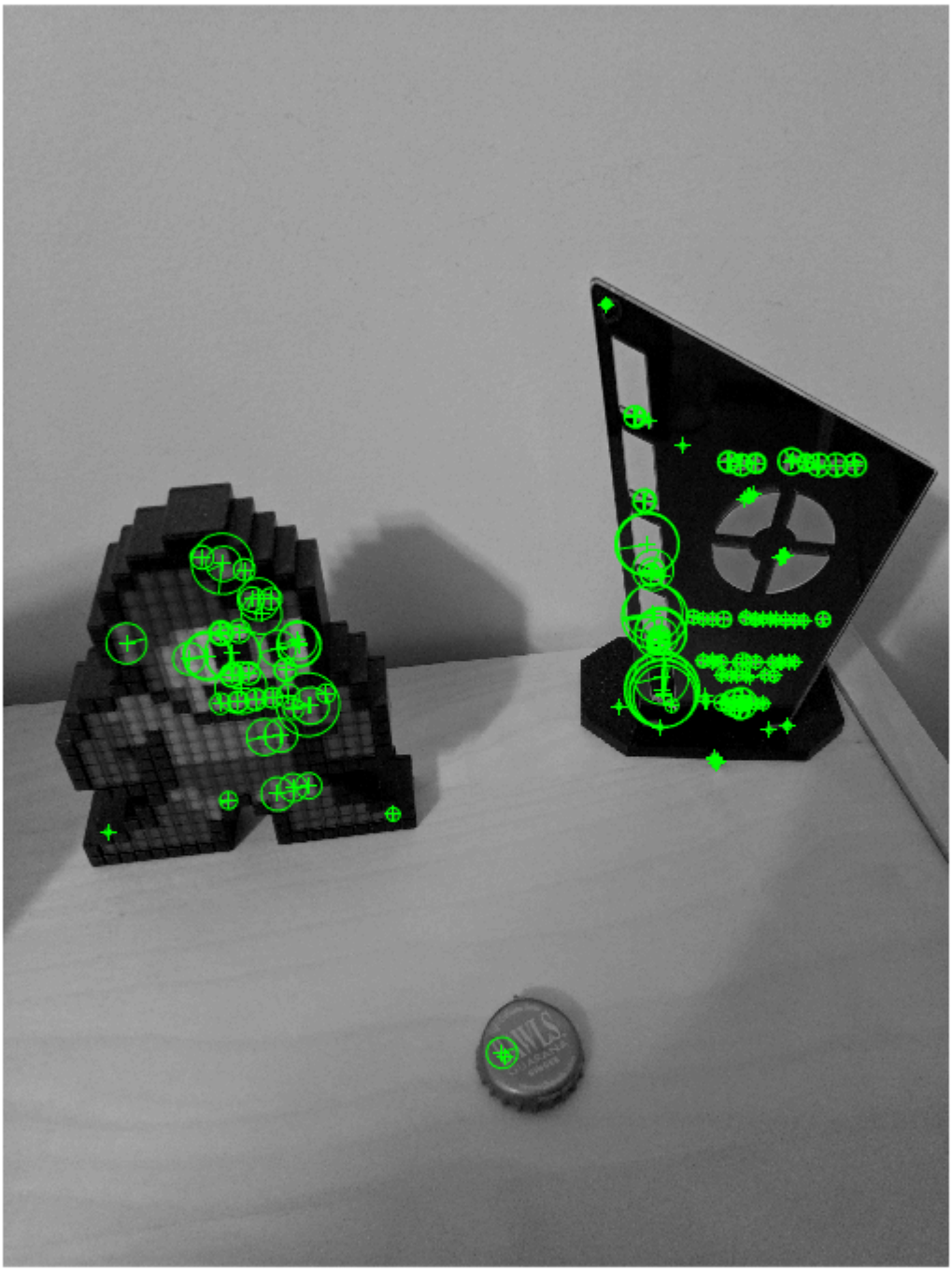



```
%mser
regions = detectMSERFeatures(I);
imshow(I); hold on;
plot(regions);
hold off
```



```
imshow(I); hold on;
plot(regions(1:10),'showPixelList', true);
hold off
```


```
regionsObj = detectMSERFeatures(I);
[features, validPtsObj] = extractFeatures(I, regionsObj);
imshow(I); hold on;
plot(validPtsObj,'showOrientation',true);
hold off
```

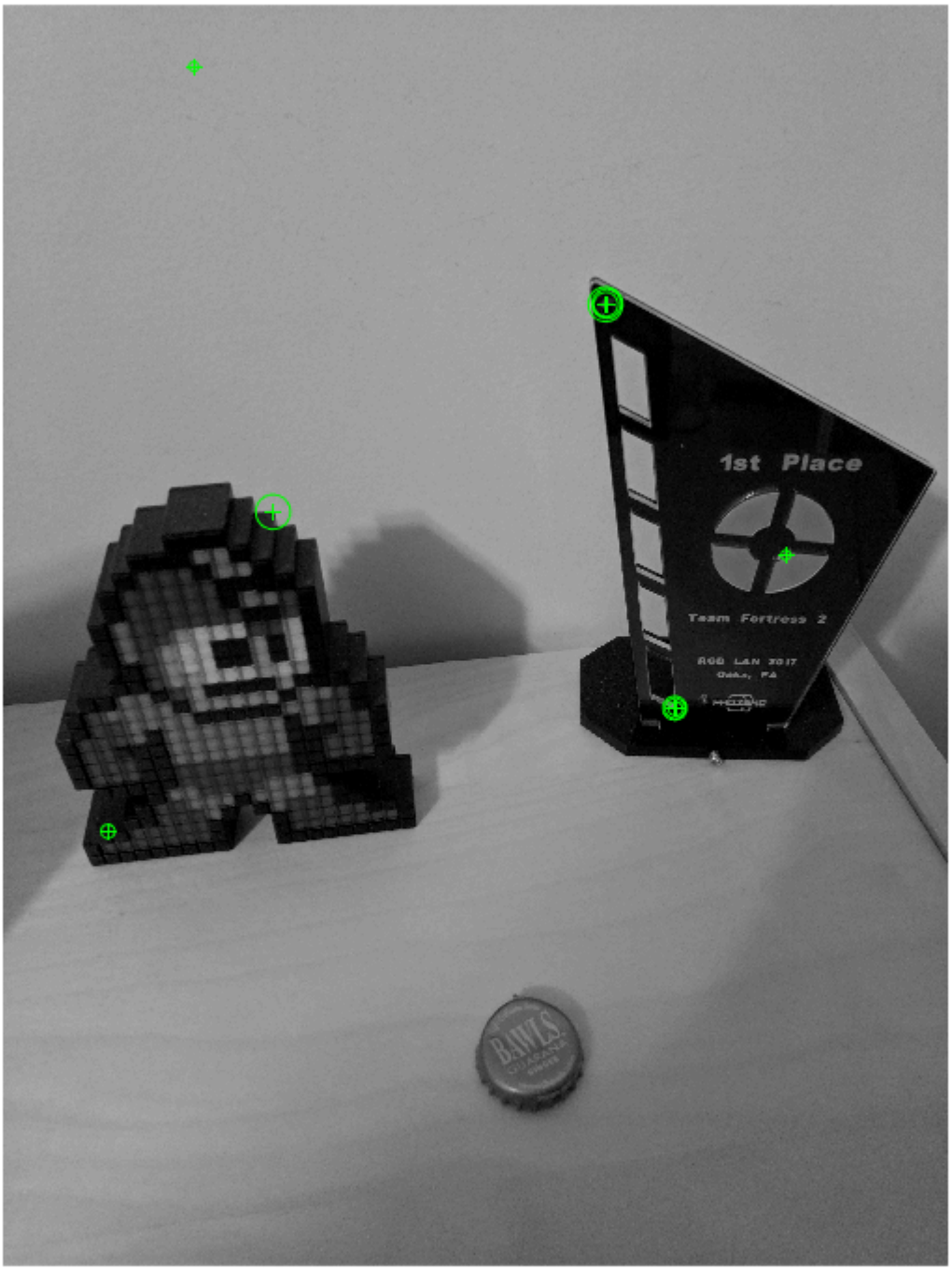


```
%orb
points = detectORBFeatures(I);
newPoints = selectUniform(points,10,size(I))
```

```
newPoints =
  10×1 ORBPoints array with properties:
```

```
       Location: [10×2 single]
         Metric: [10×1 single]
          Count: 10
          Scale: [10×1 single]
    Orientation: [10×1 single]
```

```
figure
imshow(I)
hold on
plot(newPoints)
hold off
```




```
%almost all of these needed more points to find the objects, and would
%require some kind of manual editing for the points that didn't belong to
%an object
```

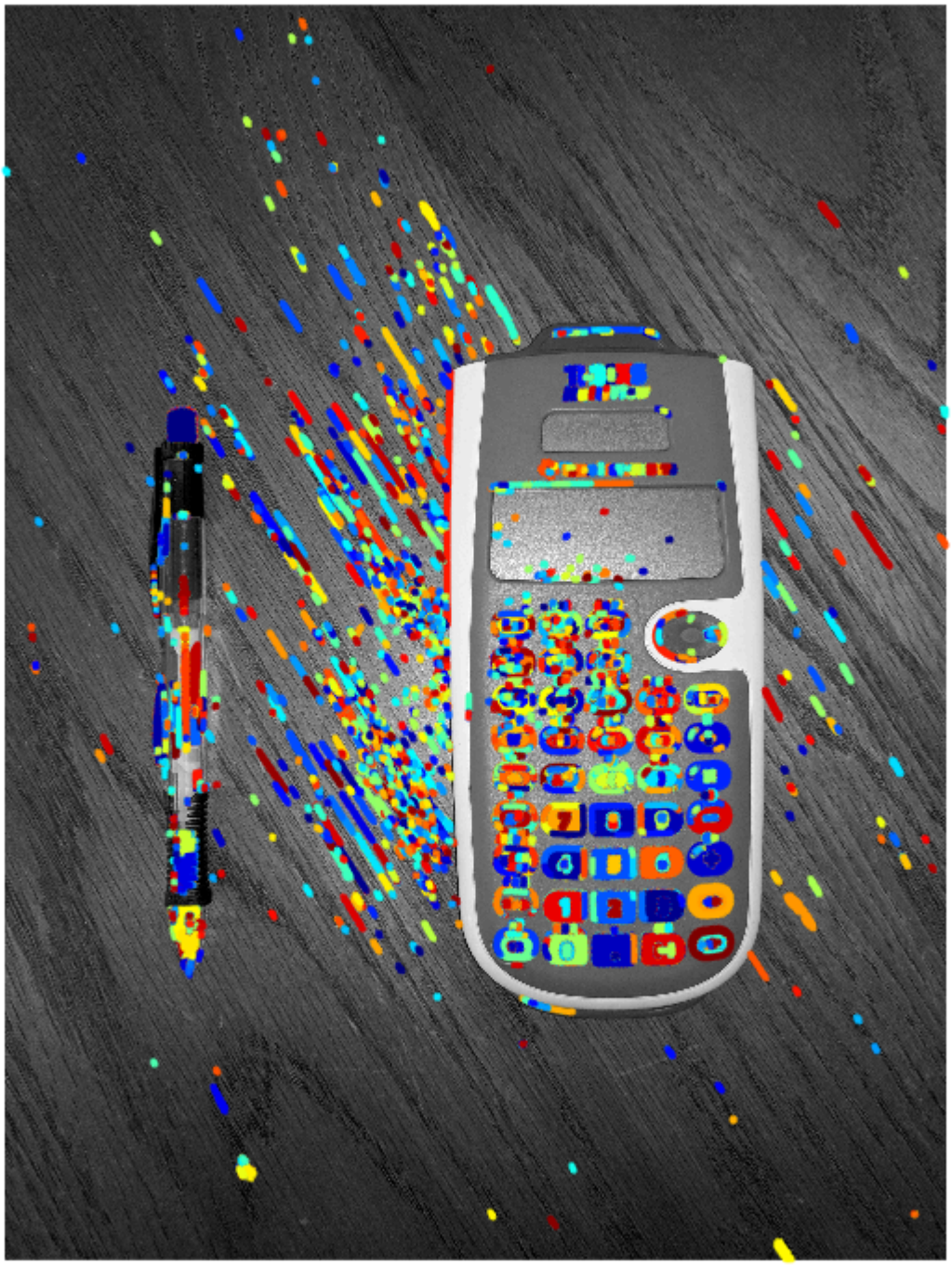
```
addpath 'D:'\homework\'comp vision'\'image pairs'\
%1
I = imread('calc_pencil_1.jpg');
I = rgb2gray(I);
%brisk
points = detectBRISKFeatures(I);
imshow(I); hold on;
plot(points.selectStrongest(100));
hold off
```

```
%fast
corners = detectFASTFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(1000));
hold off
```

```
%harris
corners = detectHarrisFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(100));
hold off
```

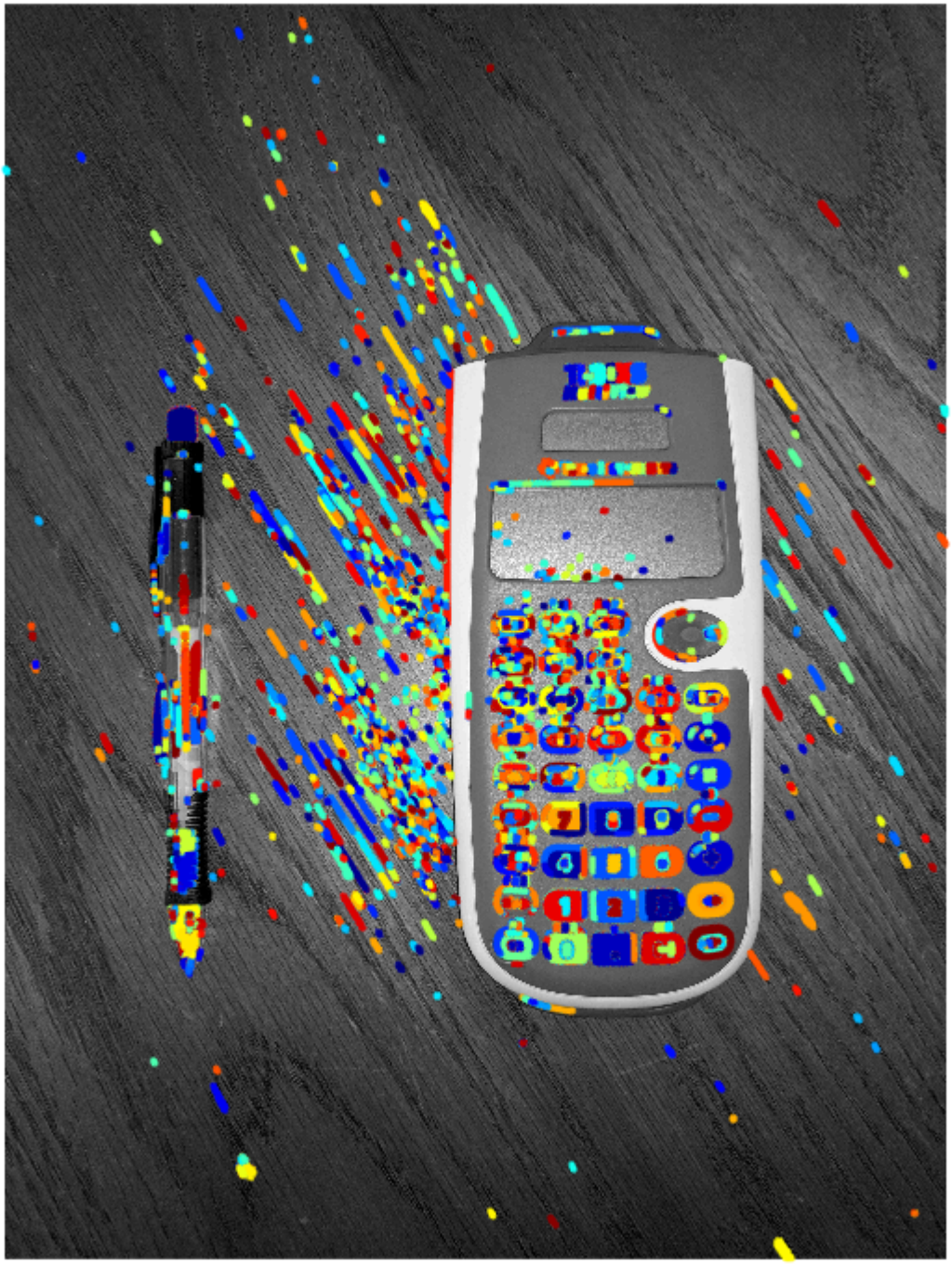
```
%similar results to fast

%mini eigen
corners = detectMinEigenFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(50));
hold off
```


```
%also similar results

%mser
regions = detectMSERFeatures(I);
figure; imshow(I); hold on;
plot(regions, 'showPixelList',true,'showEllipses',false);
hold off
```



```
[regions,mserCC] = detectMSERFeatures(I);
figure
imshow(I)
hold on
plot(regions,'showPixelList',true,'showEllipses',false);
```

```
hold off
```



```
stats = regionprops('table',mserCC,'Eccentricity');
eccentricityIdx = stats.Eccentricity < 0.55;
circularRegions = regions(eccentricityIdx);
figure
imshow(I)
```

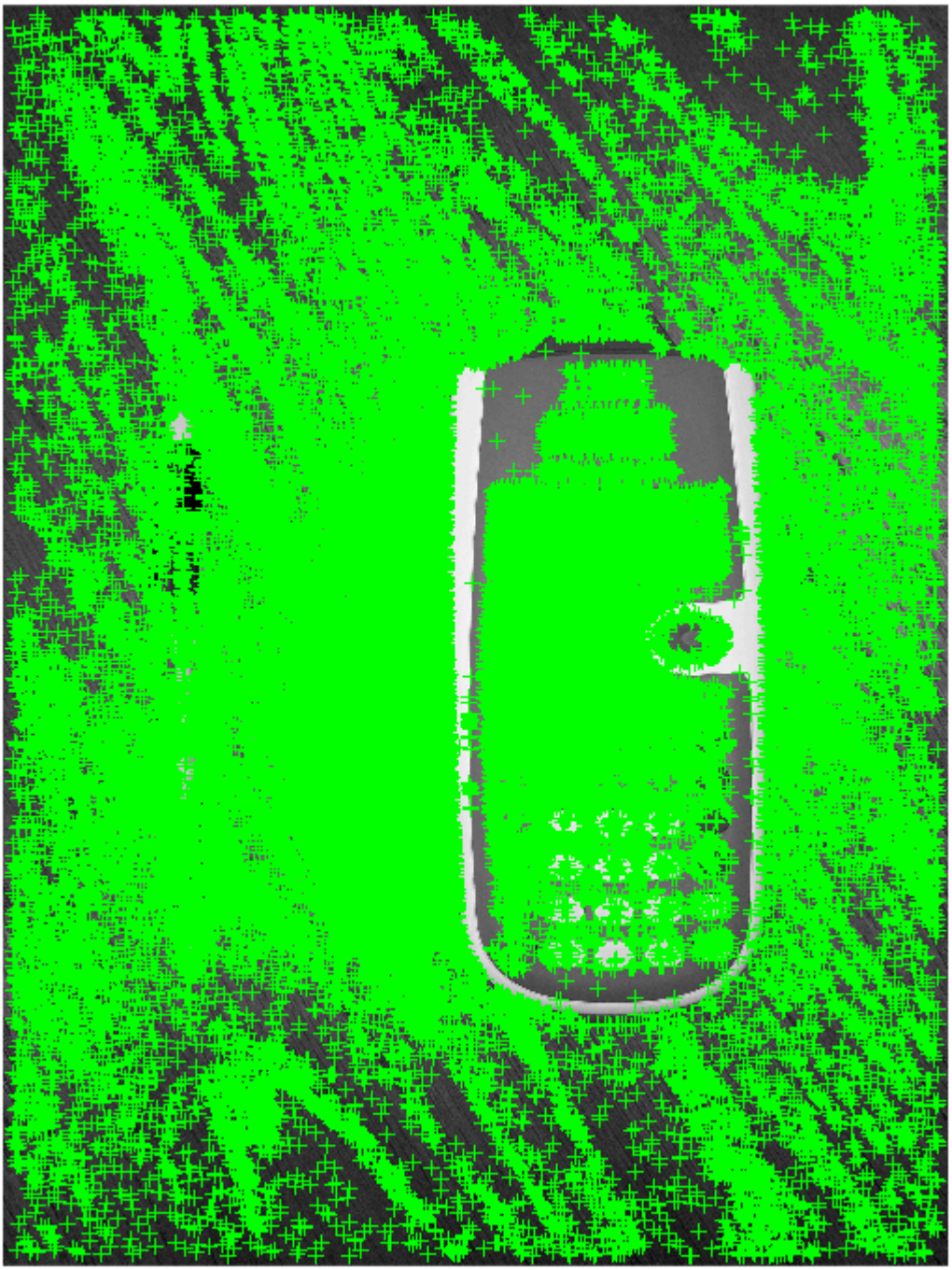
```
hold on
plot(circularRegions,'showPixelList',true,'showEllipses',false)
hold off
```


```
%a threshold applied to find most unique(?) points

%random, but works really well?
imshow(I)
```

```
hold on
plot(points,'ShowScale',false)
hold off
```

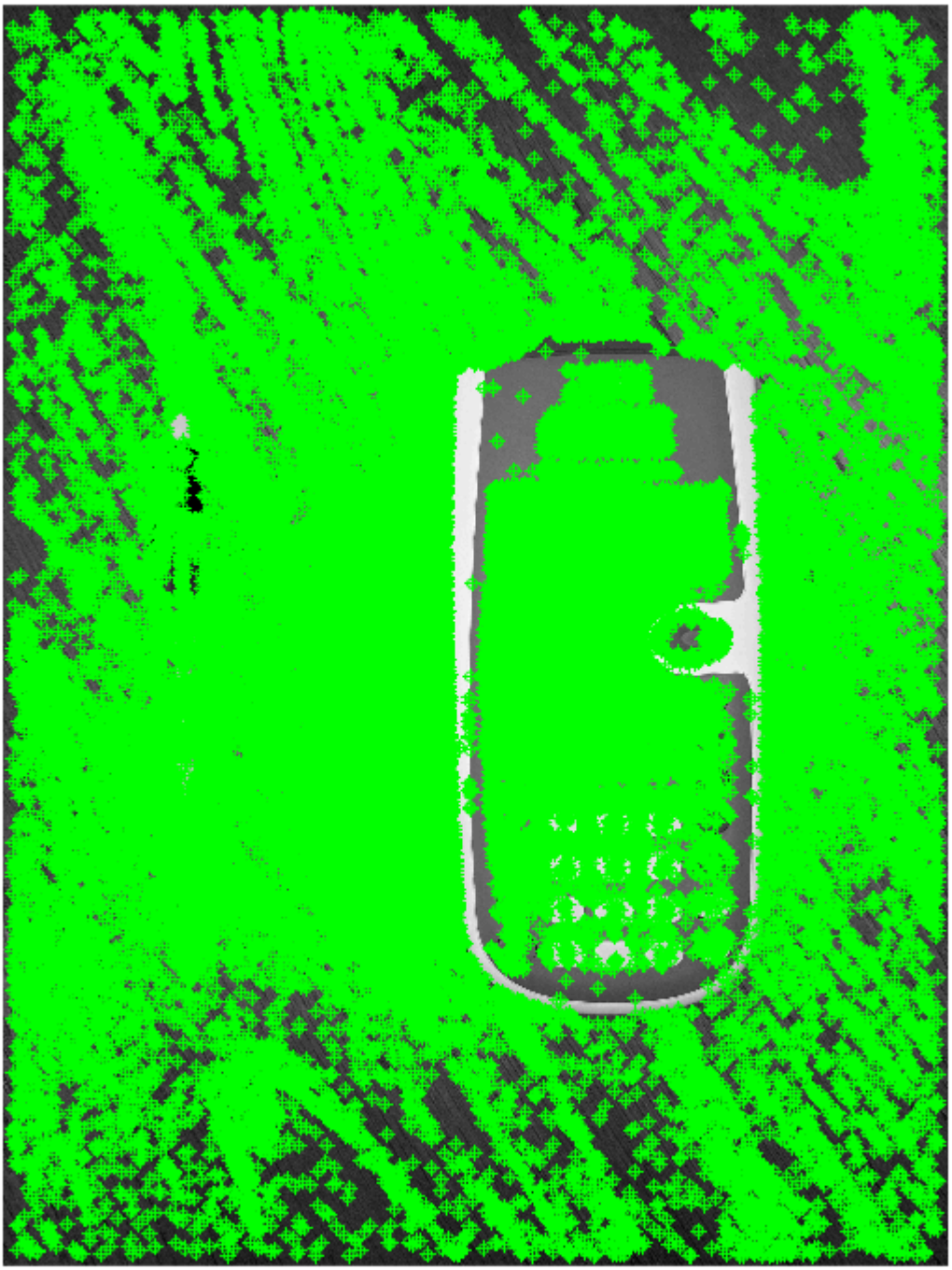
```
%orb
points = detectORBFeatures(I);
imshow(I)
```

```
hold on
plot(points,'ShowScale',false)
hold off
```



```
points = detectORBFeatures(I,'ScaleFactor',1.01,'NumLevels',3);
imshow(I)
```

```
hold on
plot(points)
hold off
```



```
%surf
points = detectSURFFeatures(I);
imshow(I)
```

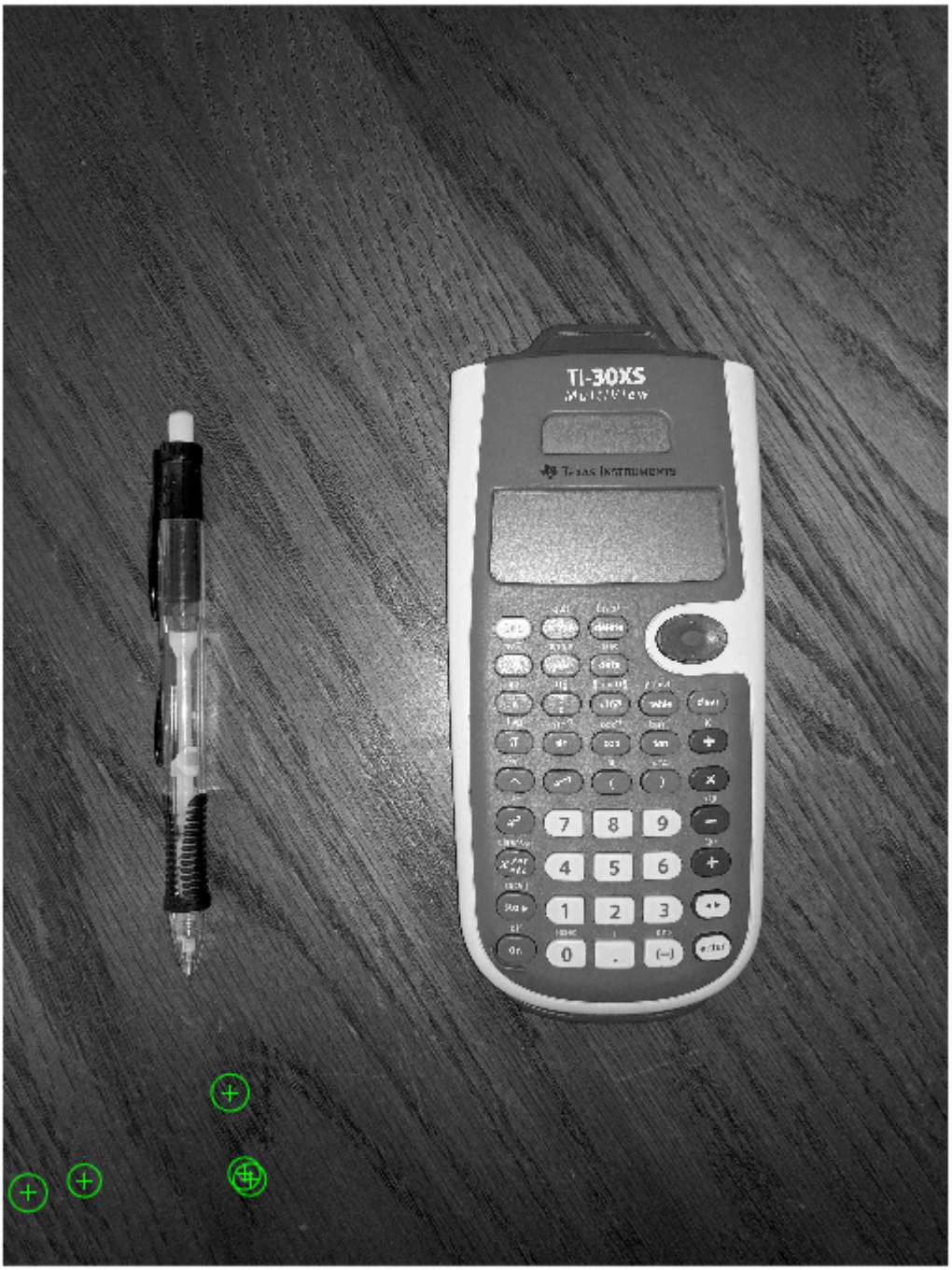
```
hold on
plot(points.selectStrongest(50));
hold off
```

```
%kaze
points = detectKAZEFeatures(I);
```

```
imshow(I)
hold on
plot(selectStrongest(points,50))
hold off
```

```
imshow(I)
```

```
hold on
plot(points(end-4:end));
hold off
```




```
%only finds bottle cap for some reason
```

```
addpath 'D:'\homework\'comp vision'\'image pairs'\

I = imread('calc_pencil_1.jpg');
I = rgb2gray(I);

%harris
corners = detectHarrisFeatures(I);
[features, valid_corners] = extractFeatures(I, corners);
figure
imshow(I)
hold on
plot(valid_corners);
```

```
%surf
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I, points);
figure
imshow(I)
hold on
```

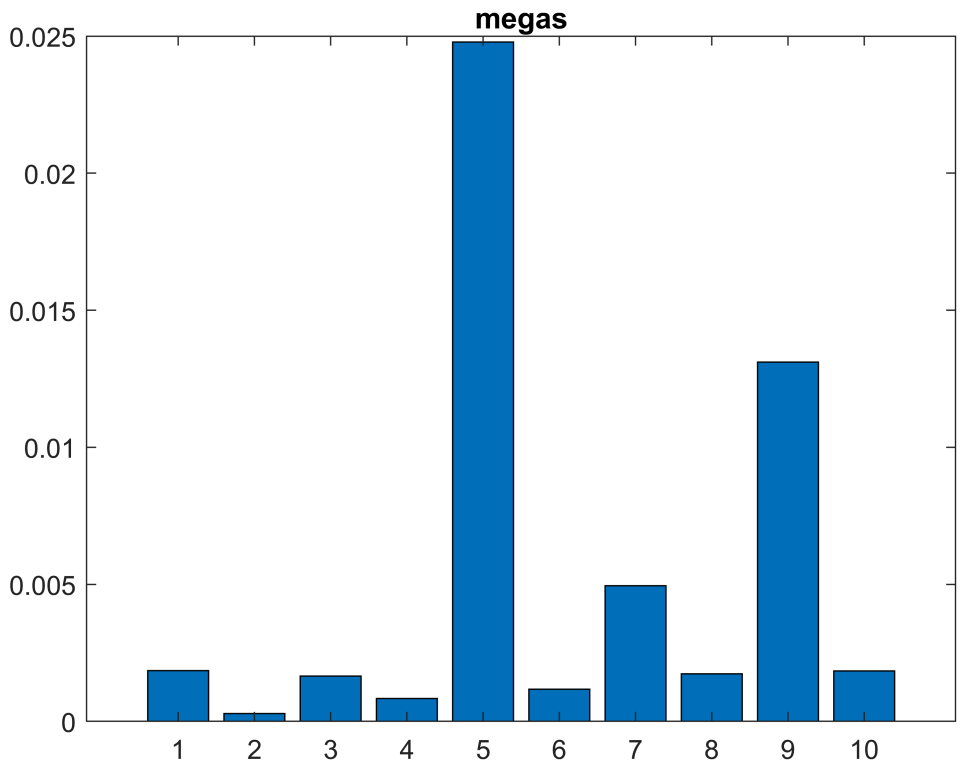
```
plot(valid_points.selectStrongest(20),'showOrientation',true);
```


```
%mser
regions = detectMSERFeatures(I);
[features, valid_points] = extractFeatures(I,regions,'Upright',true);
figure
```

```
imshow(I)
hold on
plot(valid_points,'showOrientation',true);
```

```
%perfect

%lbp
X = imread('calc_pencil_2.jpg');
```

```
X = rgb2gray(X);
lbp1 = extractLBPFeatures(I,'Upright',false);
lbp2 = extractLBPFeatures(X,'Upright',false);
megaVsMega = (lbp1 - lbp2).^2;

figure
bar([megaVsMega],'grouped')
title('megas')
```



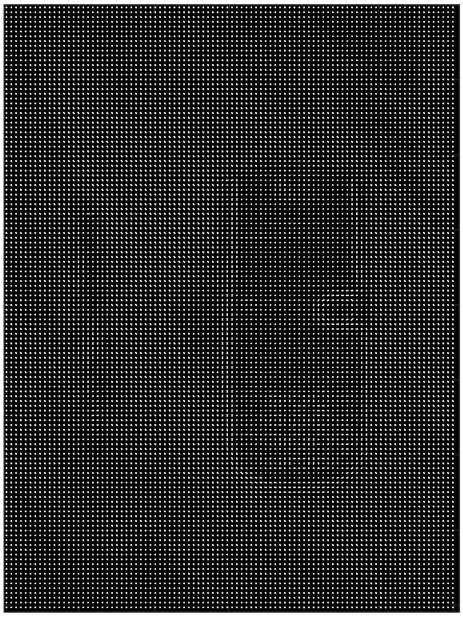

```
%hog
[featureVector,hogVisualization] = extractHOGFeatures(I);
figure
imshow(I)
hold on
plot(hogVisualization)
```

```
[hog1,visualization] = extractHOGFeatures(I,'CellSize',[32 32])
```

```
hog1 = 1×418500 single row vector
    0.1993    0.2112    0.2112    0.1665    0.1570    0.1259    0.1052    0.1260 ···
visualization =
Visualization

   Type plot(visualization) to visualize.
```

```
  Read-only properties:
              CellSize: [32 32]
             BlockSize: [2 2]
          BlockOverlap: [1 1]
               NumBins: 9
    UseSignedOrientation: 0
             BinCenters: [18×1 double]
```

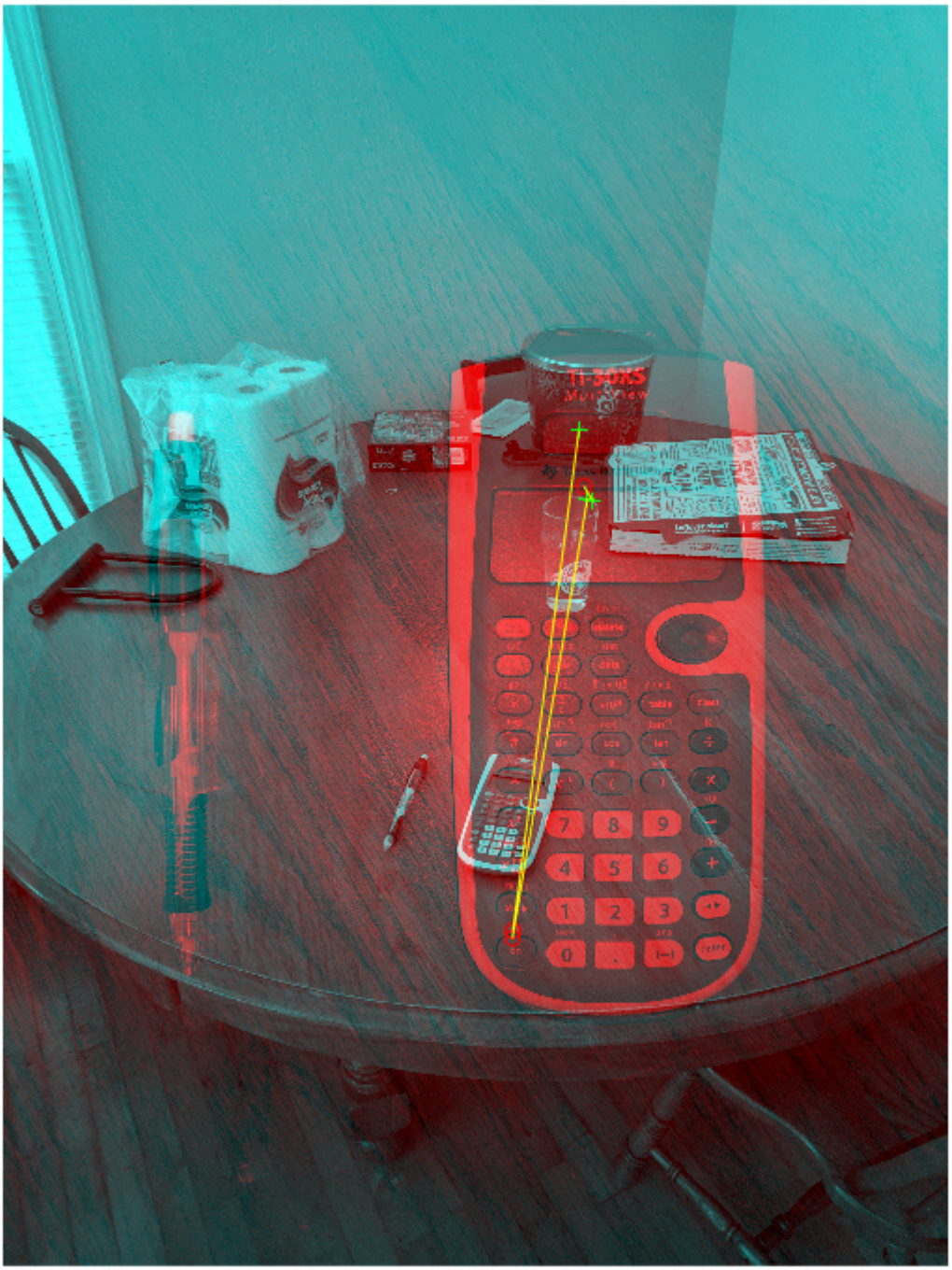
```
subplot(1,2,1);
imshow(I);
subplot(1,2,1);
plot(visualization);
```



```
%fast
corners = detectFASTFeatures(I);
strongest = selectStrongest(corners,3);
[hog, validPoints,ptVis] = extractHOGFeatures(I,strongest);
figure
imshow(I)
hold on
```

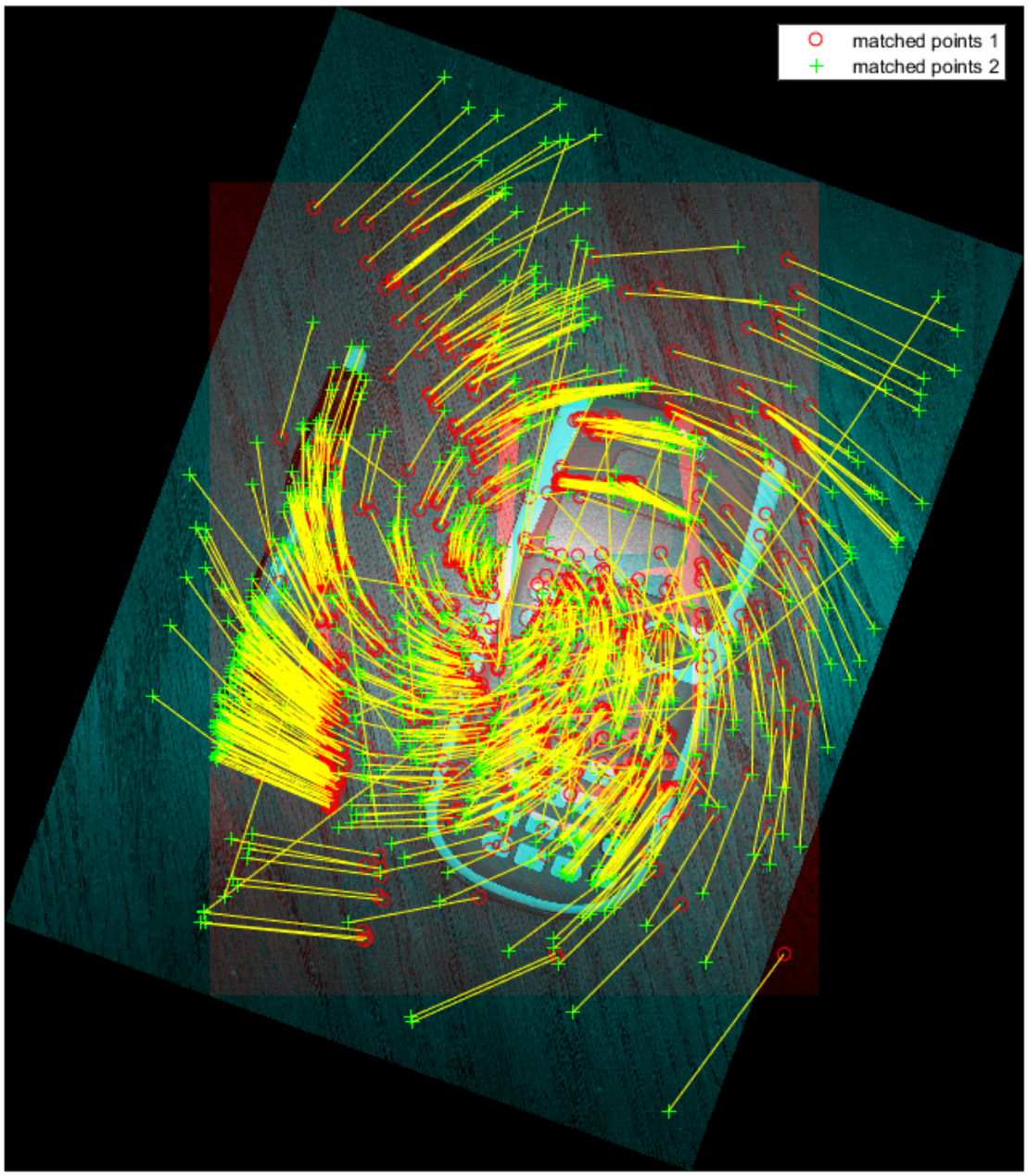
```
plot(ptVis,'Color','green');
```

```
addpath 'D:'\homework\'comp vision'\'image pairs'\

I = imread('calc_pencil_1.jpg');
I = rgb2gray(I);


Y = imread('calc_pencil_2.jpg');
Y = rgb2gray(Y);
%match features
points1 = detectHarrisFeatures(I);
points2 = detectHarrisFeatures(Y);

[features1,valid_points1] = extractFeatures(I,points1);
[features2,valid_points2] = extractFeatures(Y,points2);

indexPairs = matchFeatures(features1,features2);

matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);

figure; showMatchedFeatures(I,Y,matchedPoints1,matchedPoints2);
```



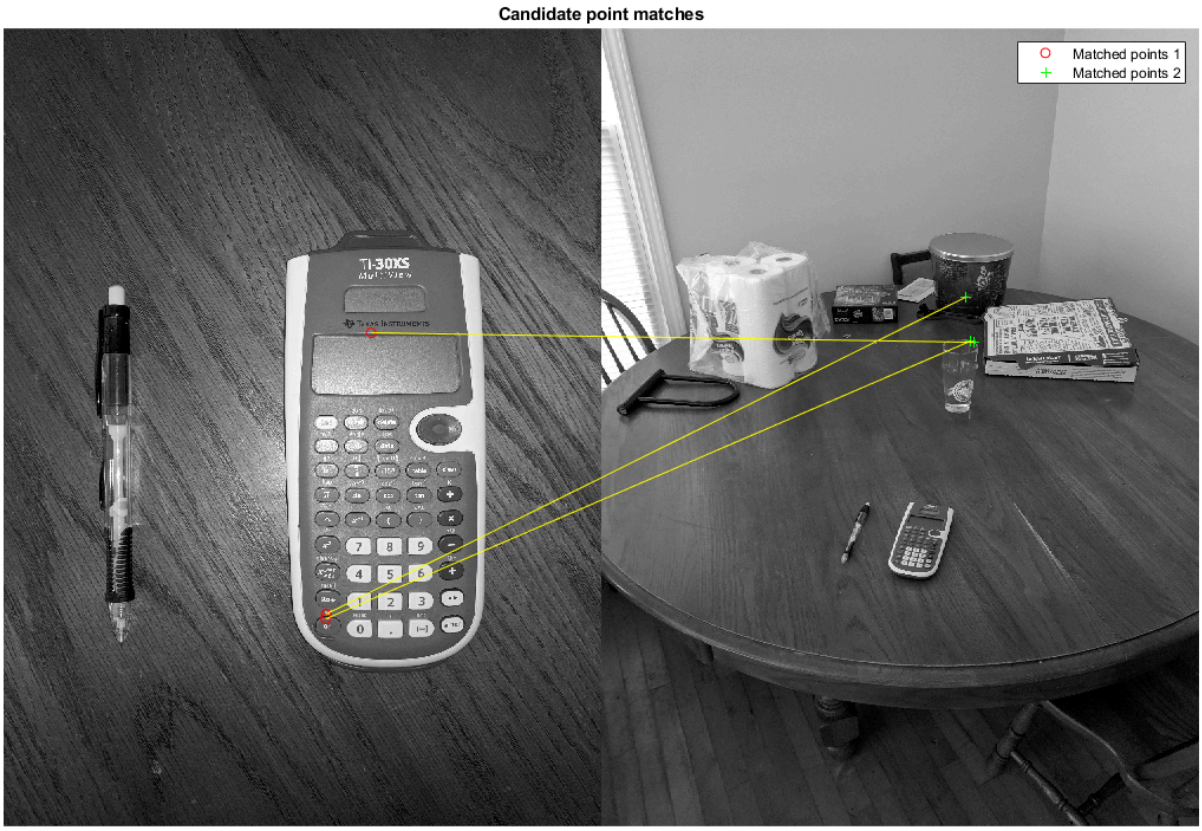
```
I2 = imresize(imrotate(I,-20),1.2);
points1 = detectSURFFeatures(I);
points2 = detectSURFFeatures(I2);
[f1,vpts1] = extractFeatures(I,points1);
[f2,vpts2] = extractFeatures(I2,points2);
indexPairs = matchFeatures(f1,f2) ;
matchedPoints1 = vpts1(indexPairs(:,1));
```

```
matchedPoints2 = vpts2(indexPairs(:,2));
figure; showMatchedFeatures(I,I2,matchedPoints1,matchedPoints2);
legend('matched points 1','matched points 2');
```




```
%showMatchedFeatures
points1 = detectHarrisFeatures(I);
points2 = detectHarrisFeatures(Y);


[f1, vpts1] = extractFeatures(I, points1);
```

```
[f2, vpts2] = extractFeatures(Y, points2);

indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(1:3, 1));
matchedPoints2 = vpts2(indexPairs(1:3, 2));

figure; ax = axes;
showMatchedFeatures(I,Y,matchedPoints1,matchedPoints2,'montage','Parent',ax);
title(ax, 'Candidate point matches');
legend(ax, 'Matched points 1','Matched points 2');
```

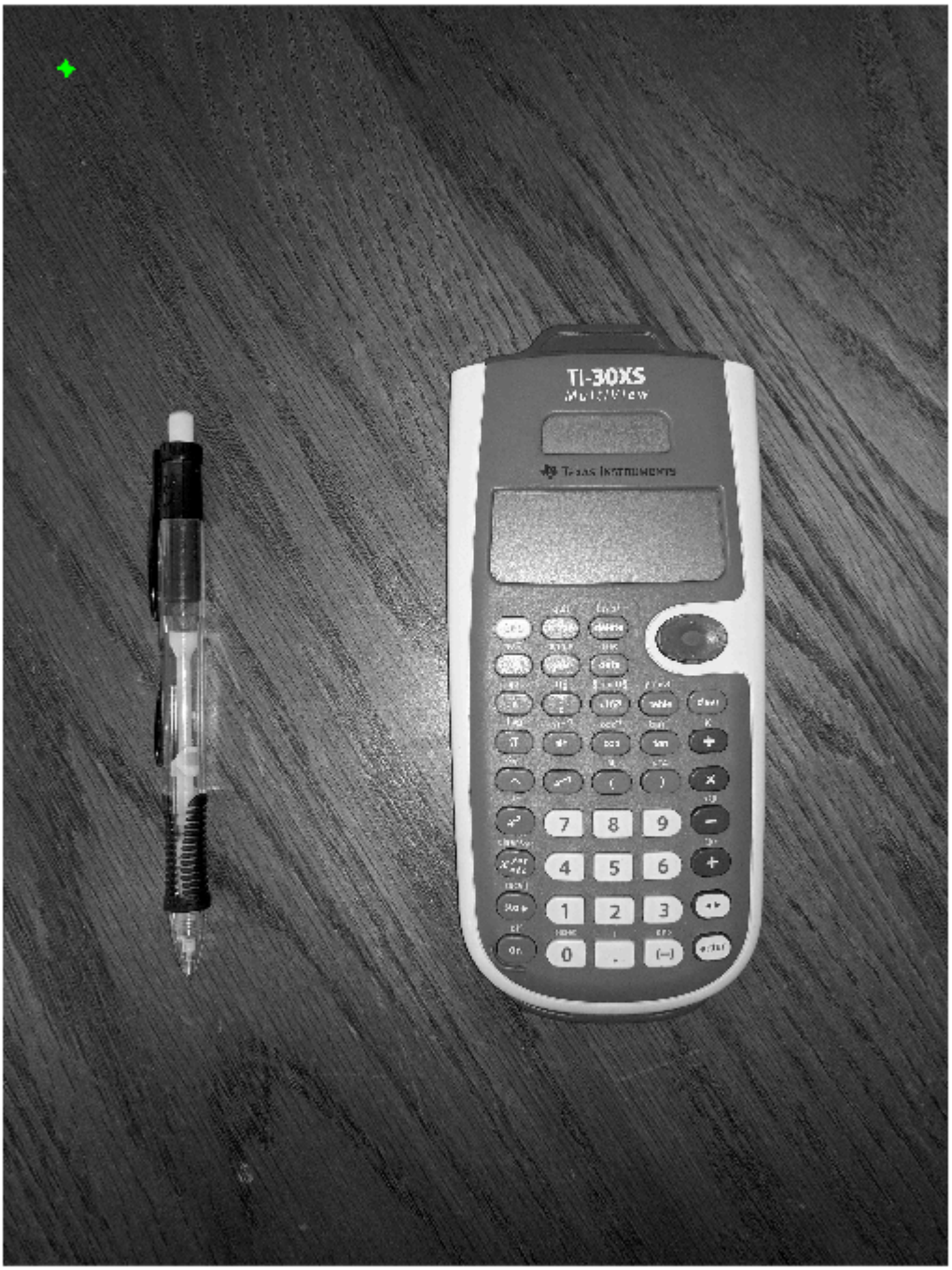



```
%doesn't work
```

```
addpath 'D:'\homework\'comp vision'\'image pairs'\

I = imread('calc_pencil_1.jpg');
I = rgb2gray(I);



%binary
%not entirely sure what values to put in for the binary features, just used
%those in the example
features1 = binaryFeatures(uint8([1 8 7 2; 8 1 7 2]));
features2 = binaryFeatures(uint8([8 1 7 2; 1 8 7 2]));

[indexPairs matchMetric] = matchFeatures(features1, features2)
```

```
indexPairs = 2×2 uint32 matrix
   1   2
   2   1
matchMetric = 2×1 single column vector
     0
     0
```

```
%brisk
points = detectBRISKFeatures(I);
location = [200:228;200:228]';
points = BRISKPoints(location);

strongest = points.selectStrongest(10);
imshow(I)
hold on
plot(strongest)
hold off
```




```
%garbage
strongest.Location
```

```
ans = 10×2 single matrix
   200   200
   201   201
   202   202
   203   203
   204   204
```

```
   205   205
   206   206
   207   207
   208   208
   209   209
```

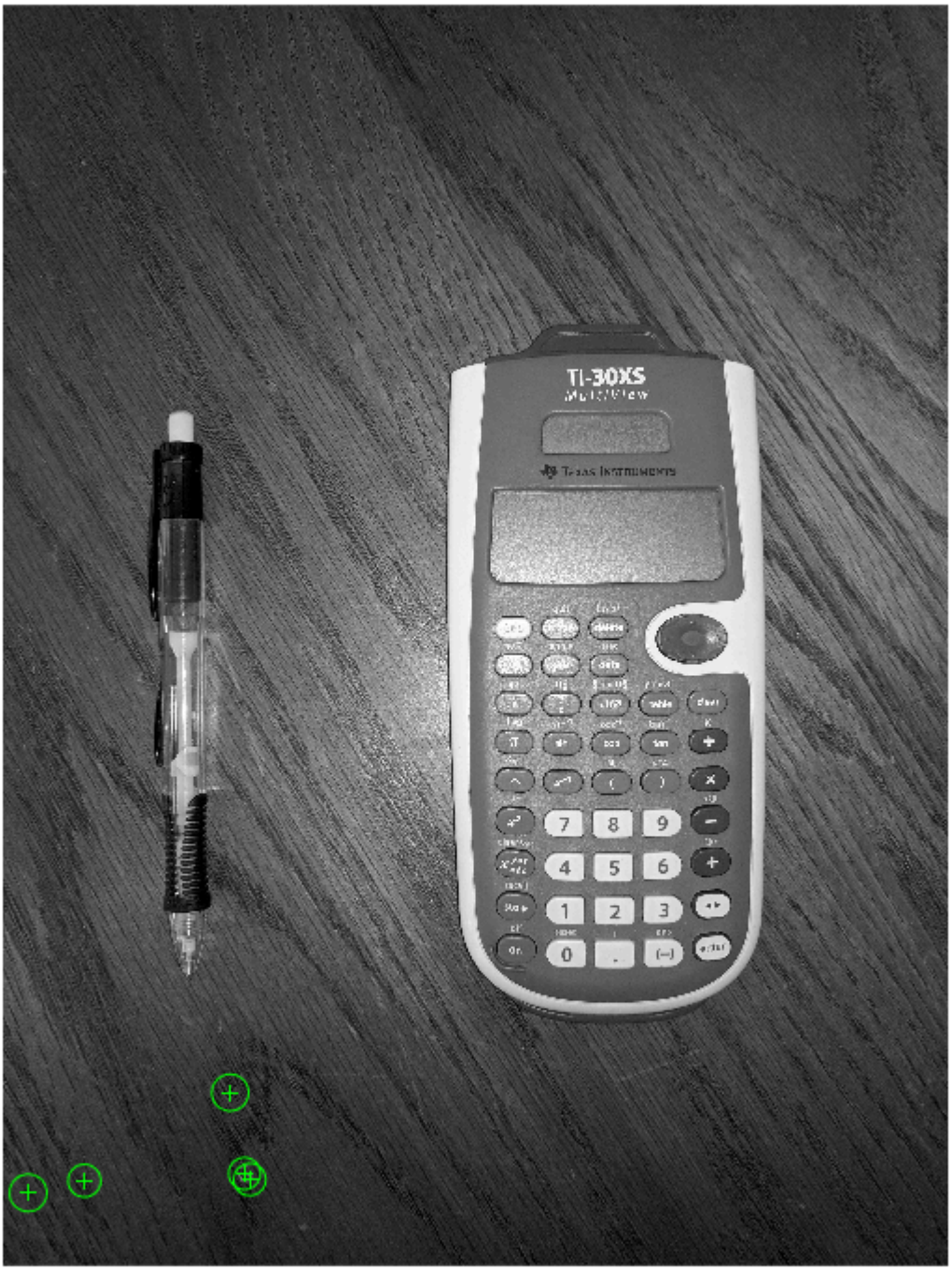
```
%kaze
points = detectKAZEFeatures(I);
strongest = selectStrongest(points,10);
imshow(I)
hold on
plot(strongest)
hold off
```


```
%only finds trophy with this low amount of points, but with a higher amount
%of points it also does well
strongest.Location
```

```
ans = 10×2 single matrix
10^3 ×
    0.4830    1.8931
    2.2694    2.8941
```

```
    0.4829    1.8932
    2.2952    2.8913
    2.2694    2.3777
    2.2696    2.3774
    2.2684    2.3521
    1.6751    2.9033
    2.1200    2.6219
    1.9623    2.6429
```

```
imshow(I)
hold on
plot(points(end-4:end)) %whenever i do this notation, i only find the bottlecap
hold off
```




```
%corner
points = detectHarrisFeatures(I);
strongest = selectStrongest(points,50);
imshow(I)
hold on
plot(strongest)
```

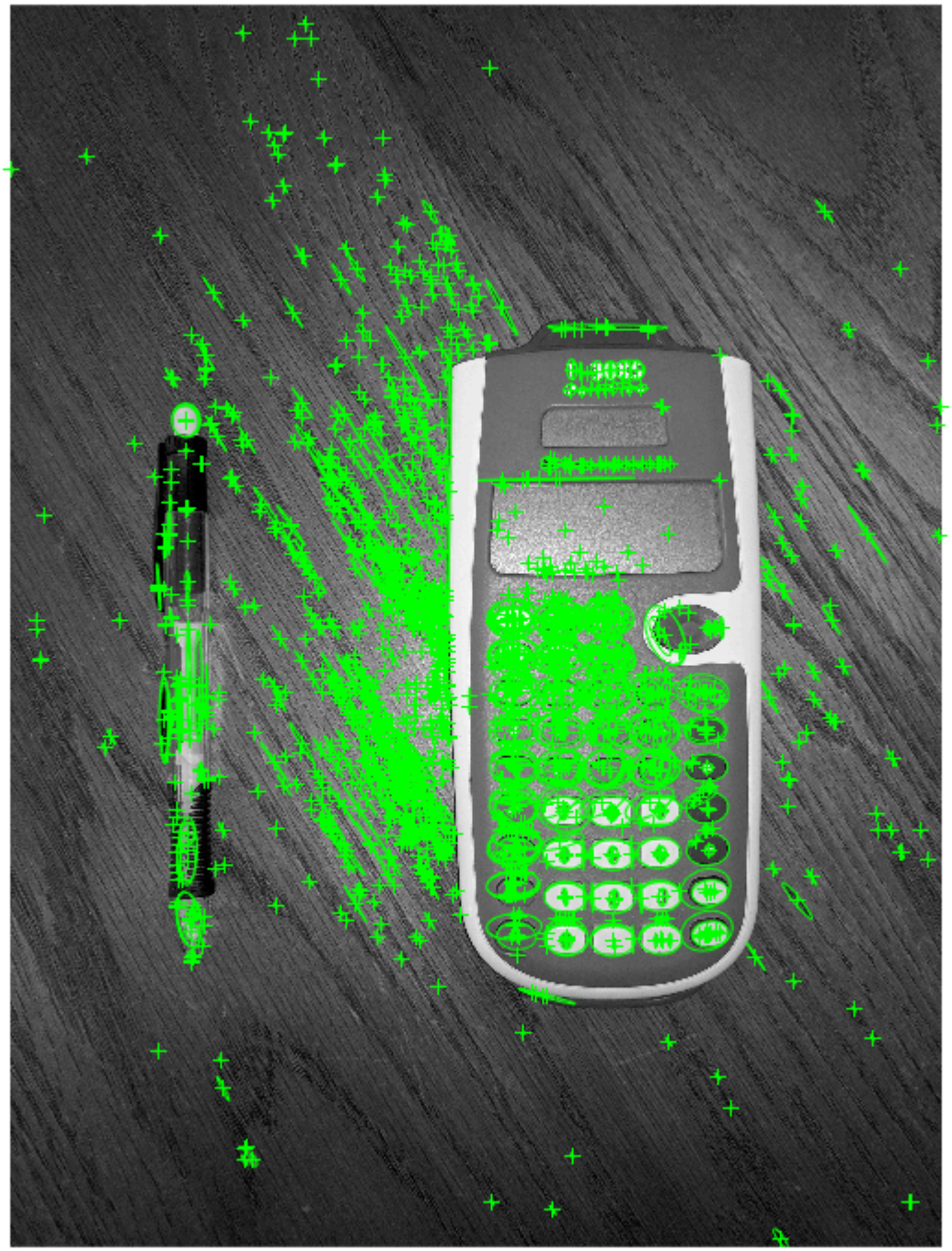
```
hold off
```

```
%surf
points = detectSURFFeatures(I);
strongest = points.selectStrongest(50);
imshow(I); hold on;
```

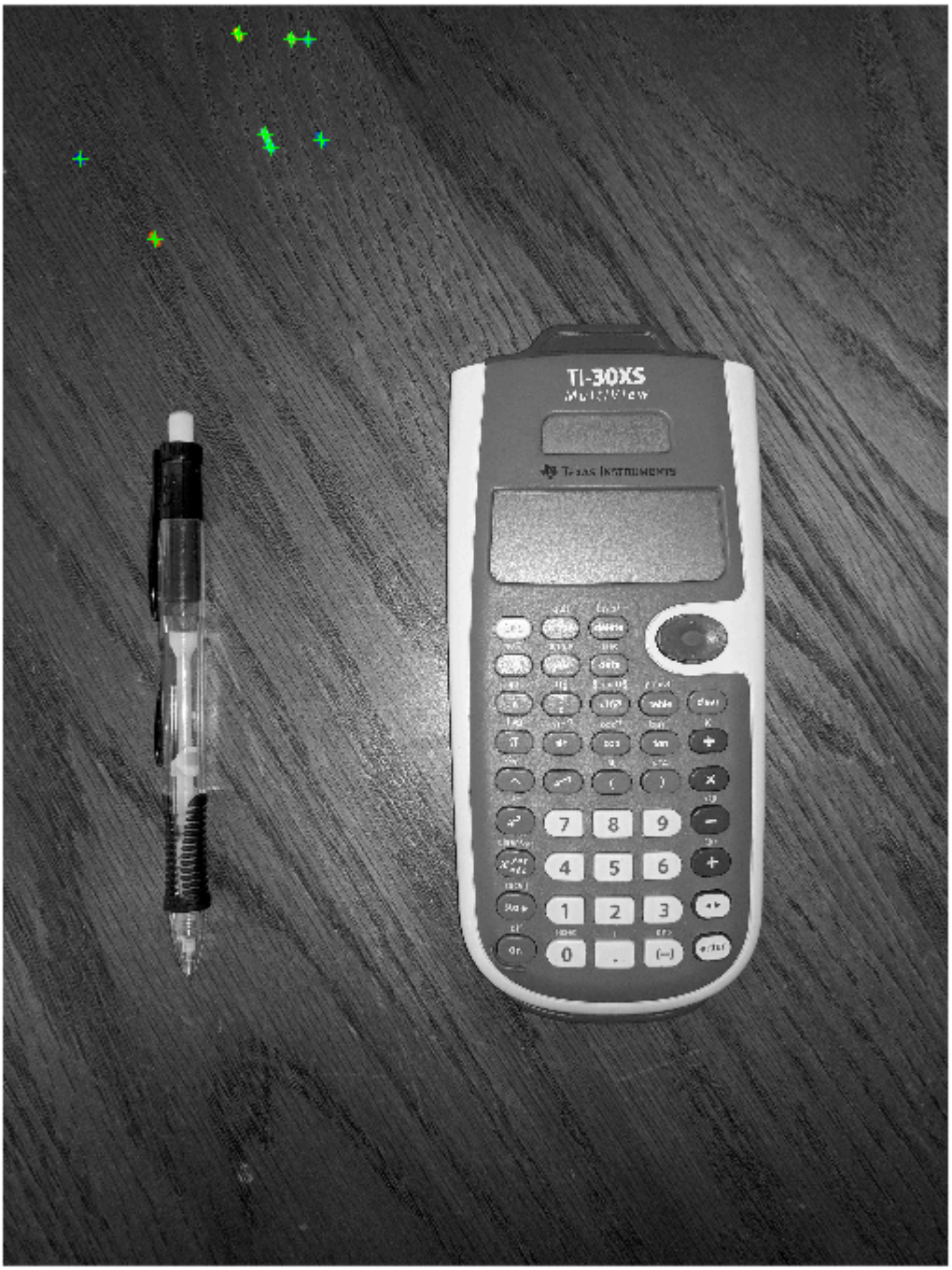
```
plot(strongest);
hold off
```

```
imshow(I); hold on;
plot(points(end-4:end));
hold off
```


```
%mser
regions = detectMSERFeatures(I);
imshow(I); hold on;
plot(regions);
hold off
```



```
imshow(I); hold on;
plot(regions(1:10),'showPixelList', true);
hold off
```




```
regionsObj = detectMSERFeatures(I);
[features, validPtsObj] = extractFeatures(I, regionsObj);
imshow(I); hold on;
plot(validPtsObj,'showOrientation',true);
hold off
```

```
%orb
points = detectORBFeatures(I);
newPoints = selectUniform(points,10,size(I))
```

```
newPoints =
  10×1 ORBPoints array with properties:
```

```
       Location: [10×2 single]
         Metric: [10×1 single]
          Count: 10
          Scale: [10×1 single]
    Orientation: [10×1 single]
```

```
figure
imshow(I)
hold on
plot(newPoints)
hold off
```


TI-30XS
Texas Instruments

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('cheeto_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Image of a Box

Read the target image containing a cluttered scene.

```
sceneImage = imread('cheeto_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

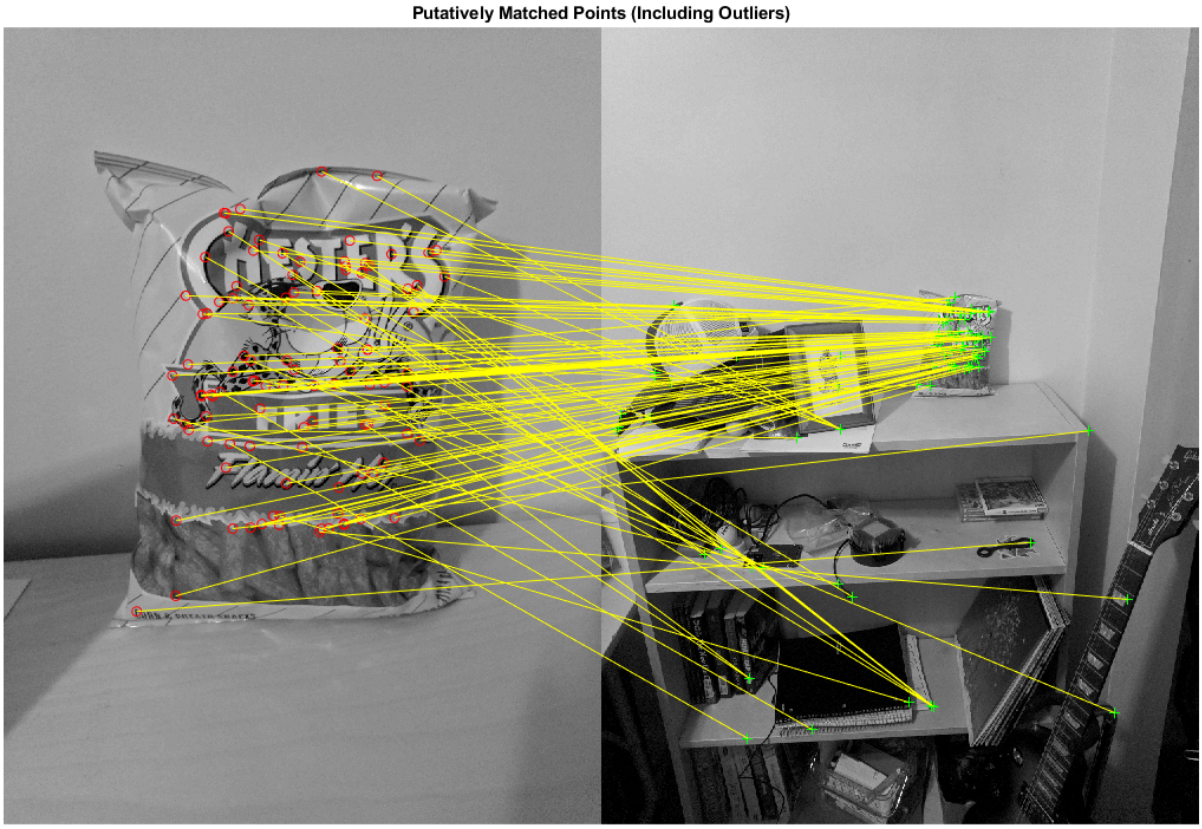
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



Step 5: Locate the Object in the Scene Using Putative Matches

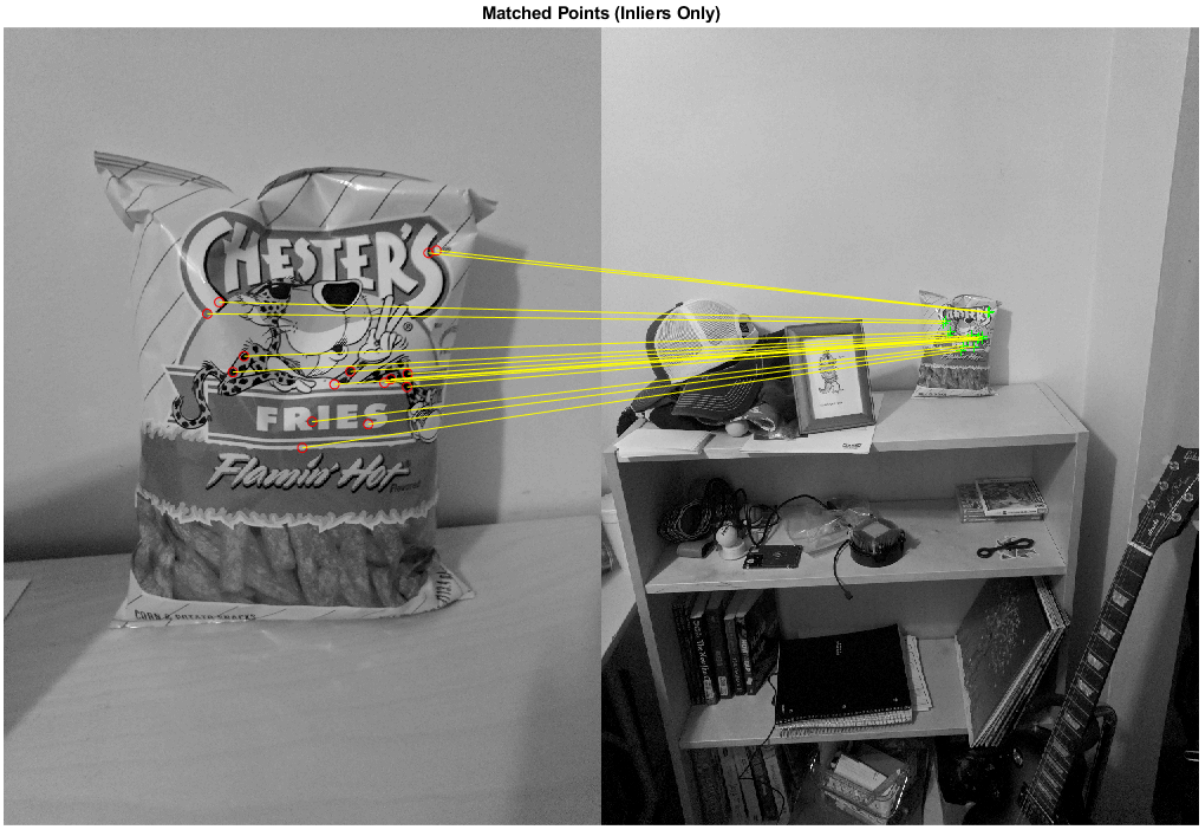
`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

```
Warning: Maximum number of trials reached. Consider increasing the maximum distance or decreasing the desired
 confidence.
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('coffee_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('coffee_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

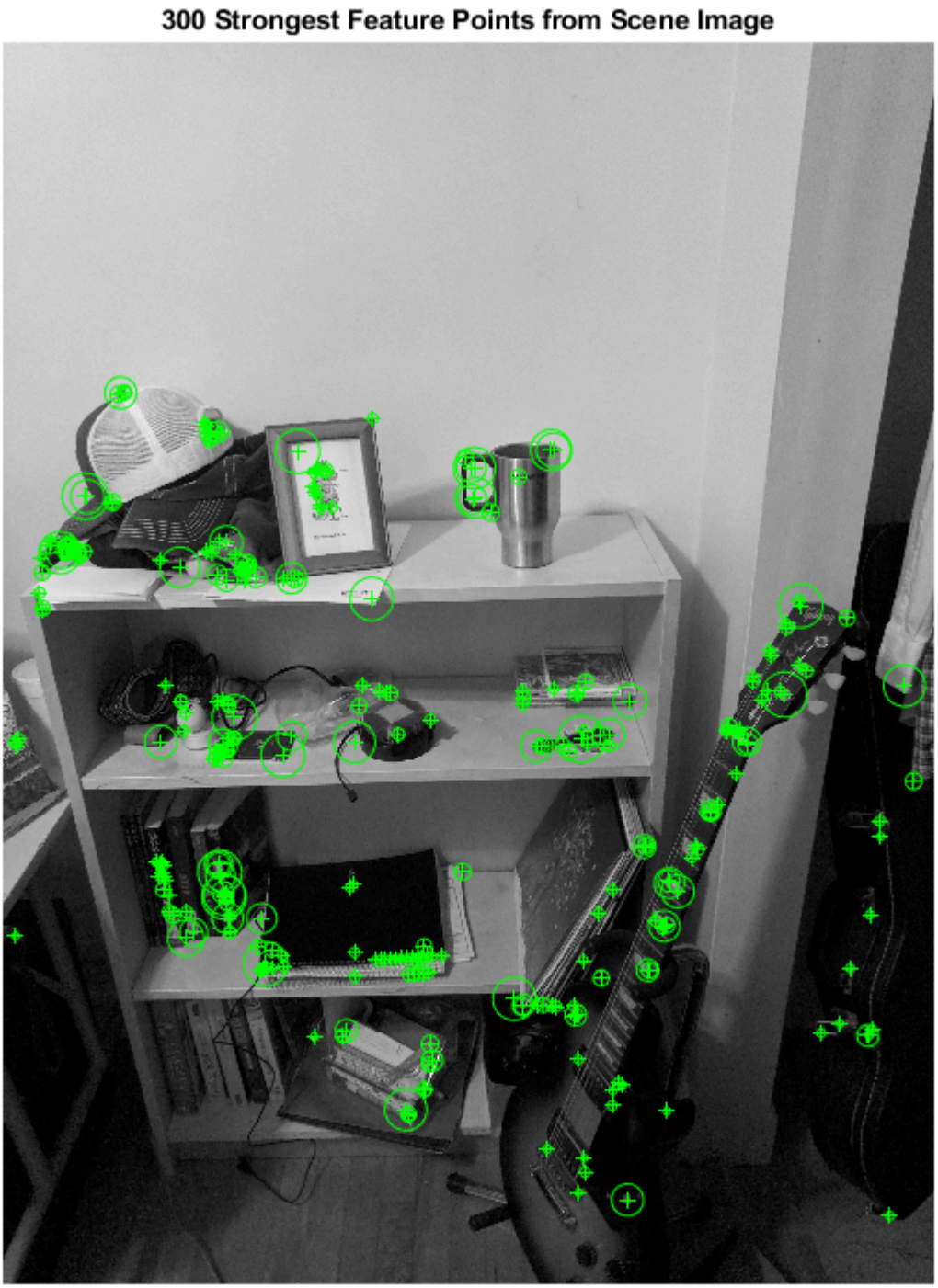
Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```



Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

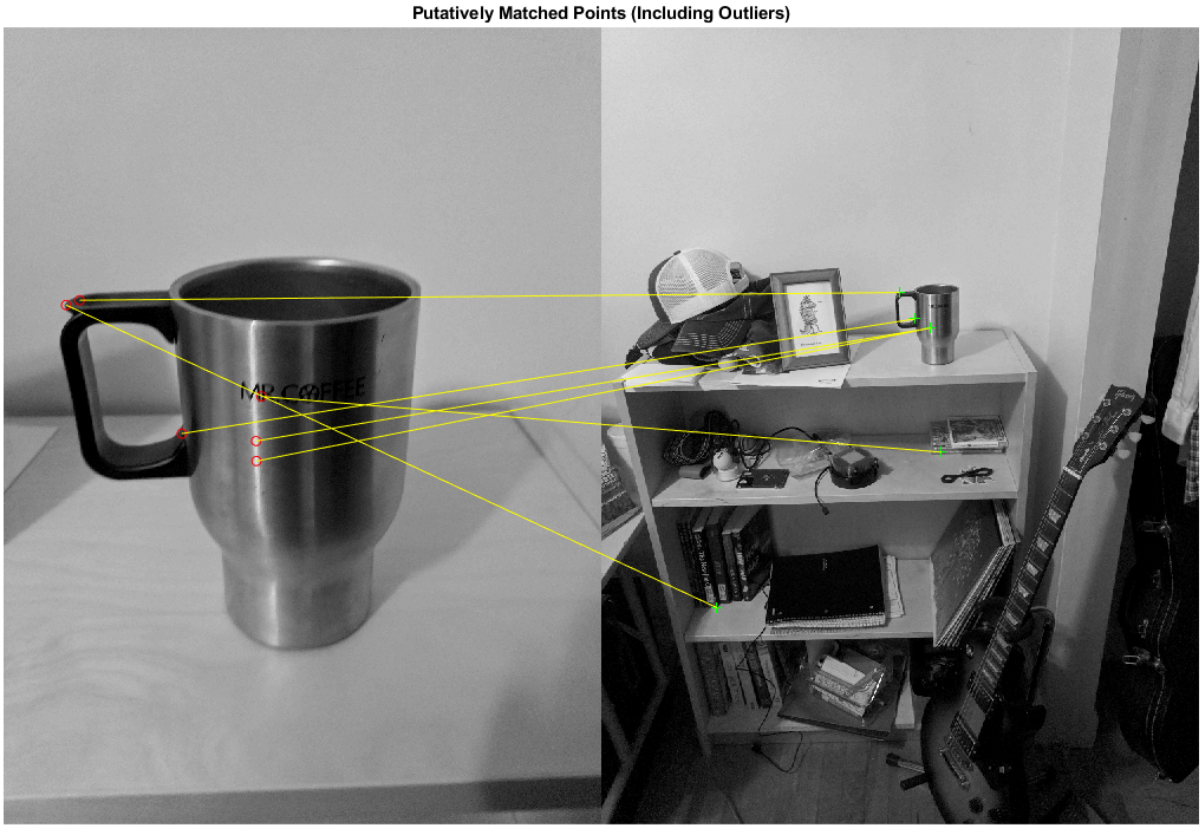
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



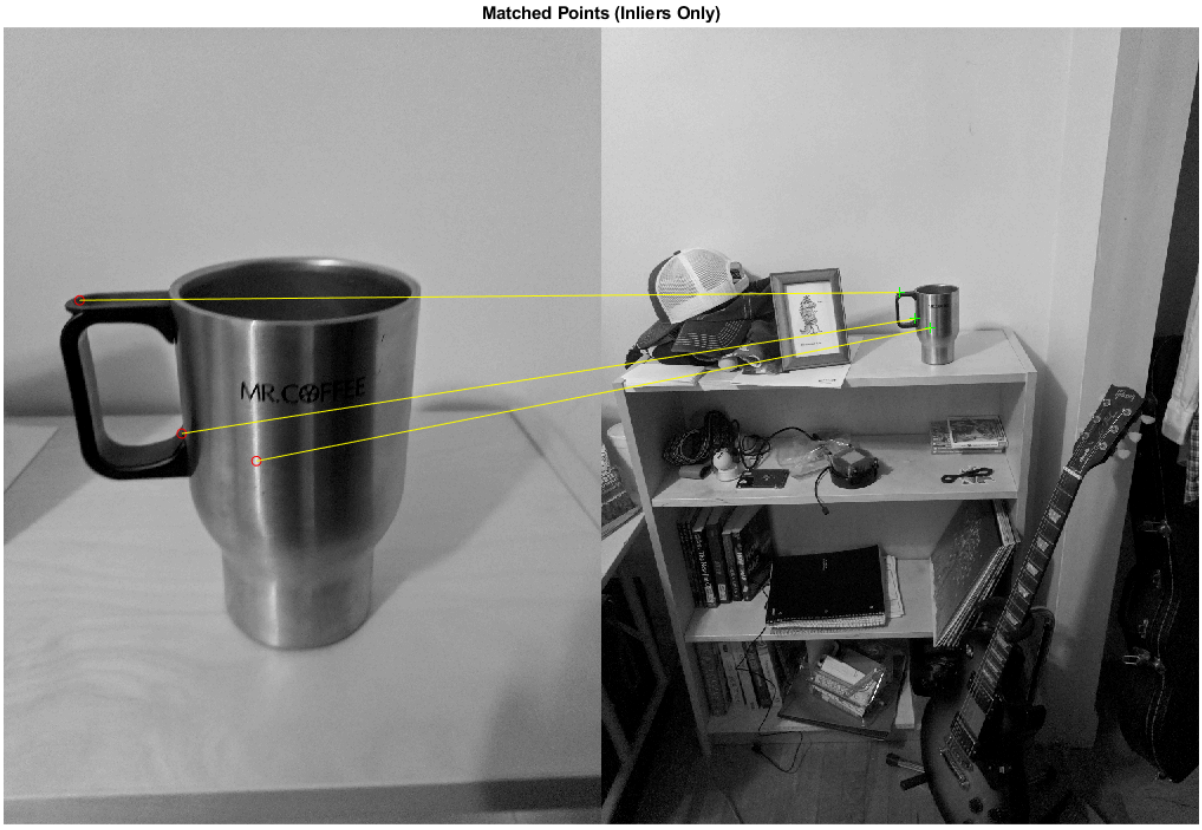
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

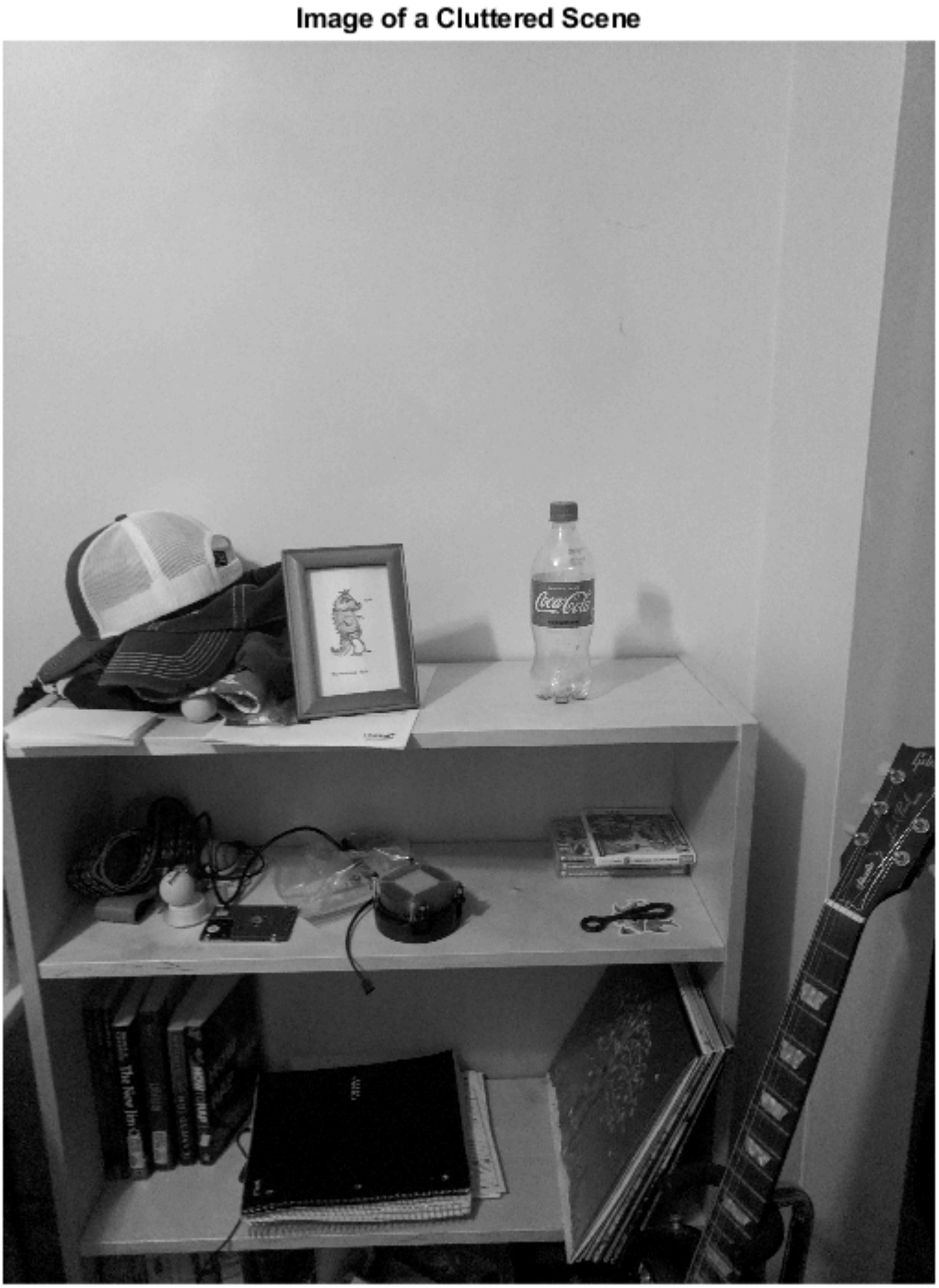
Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('coke_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('coke_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```



Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

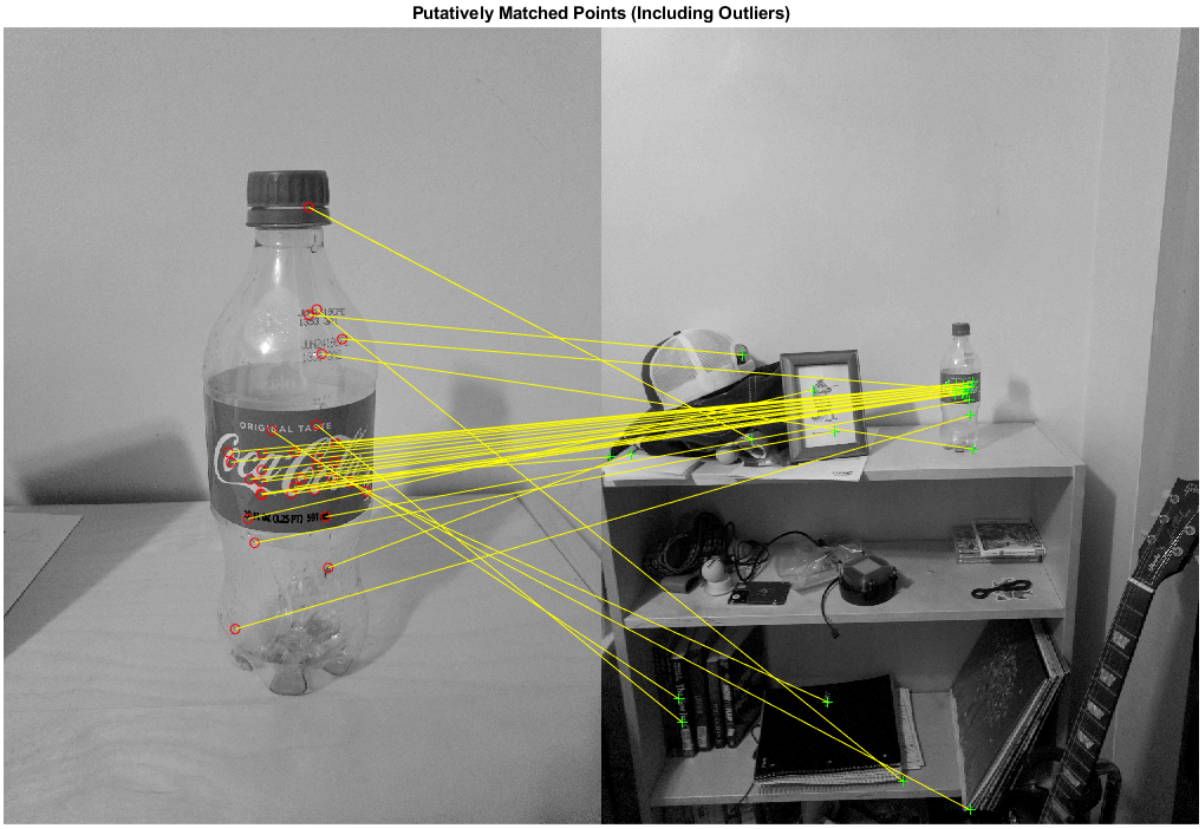
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



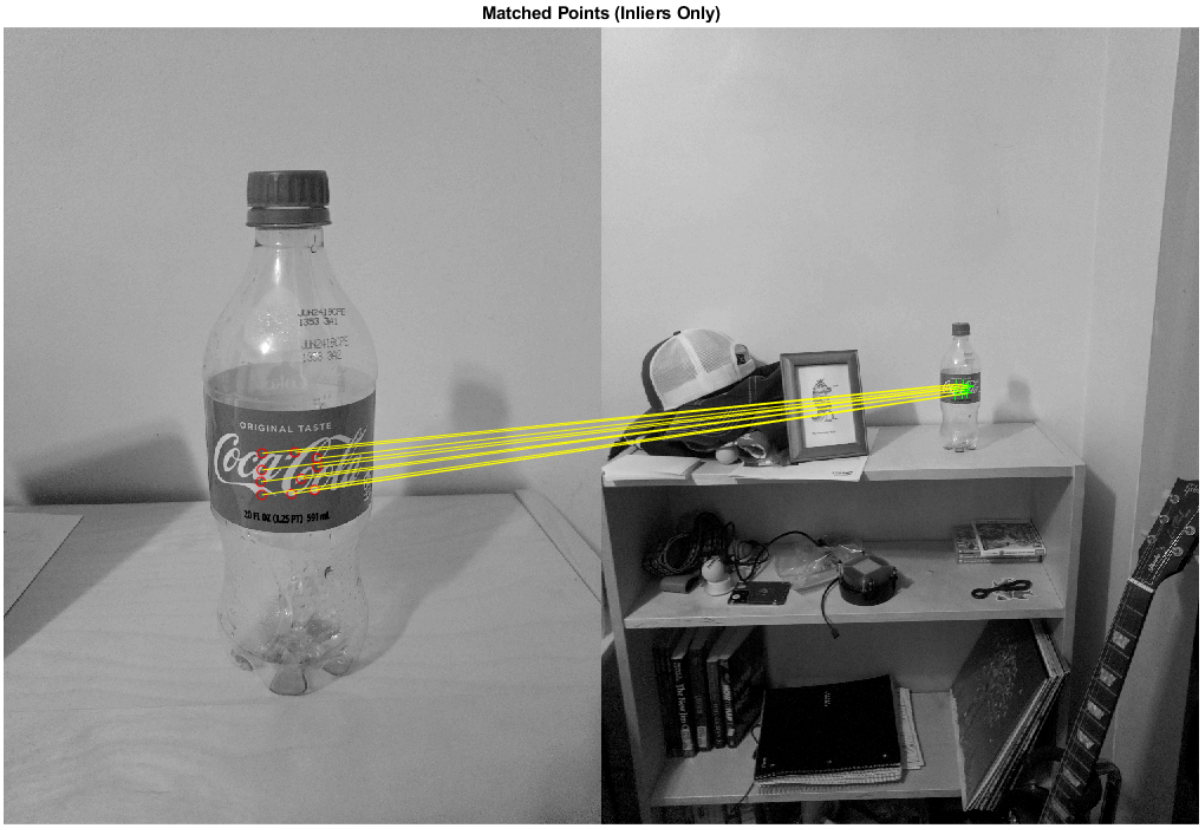
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('falco_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('falco_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

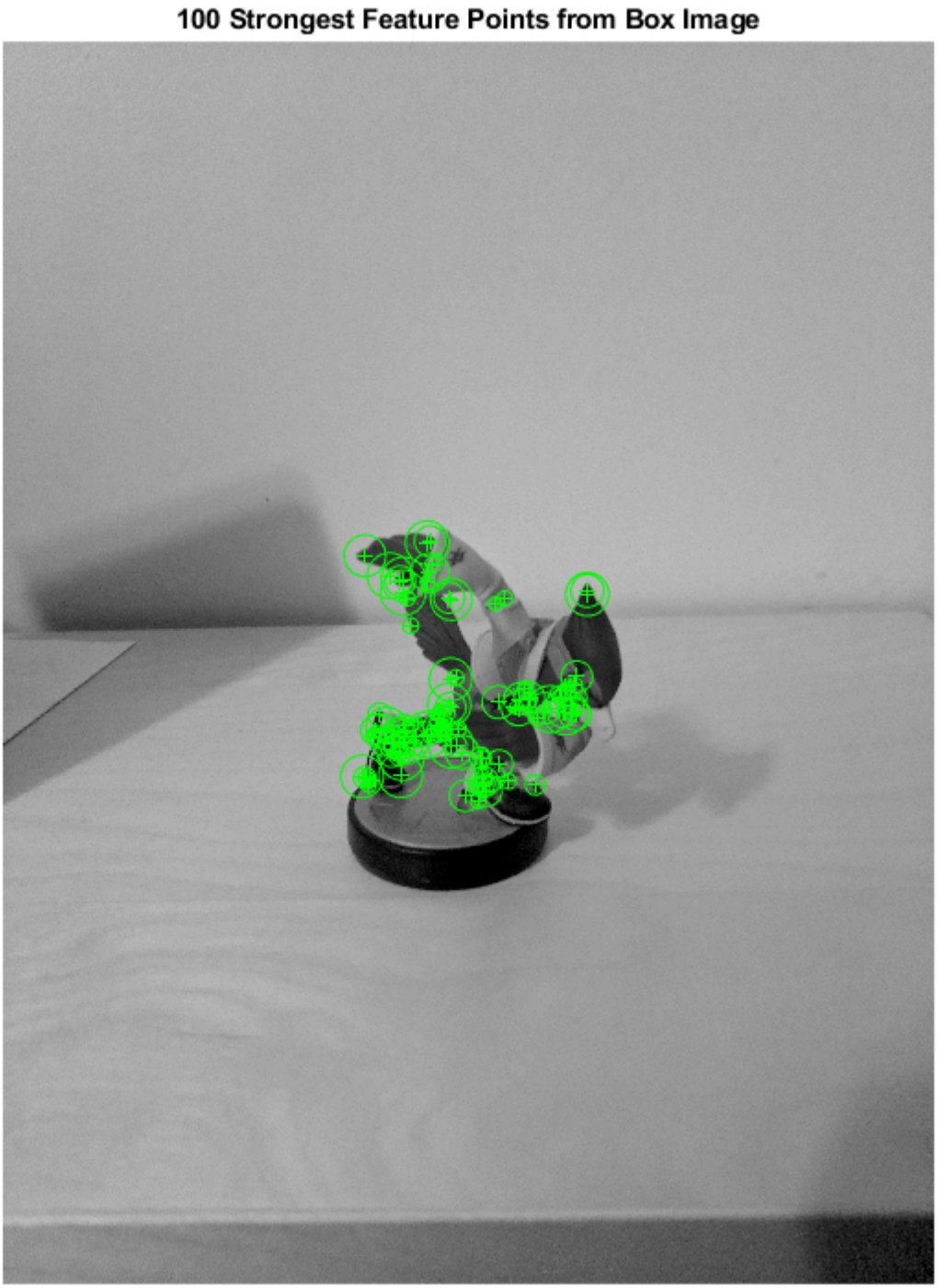
Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```



Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

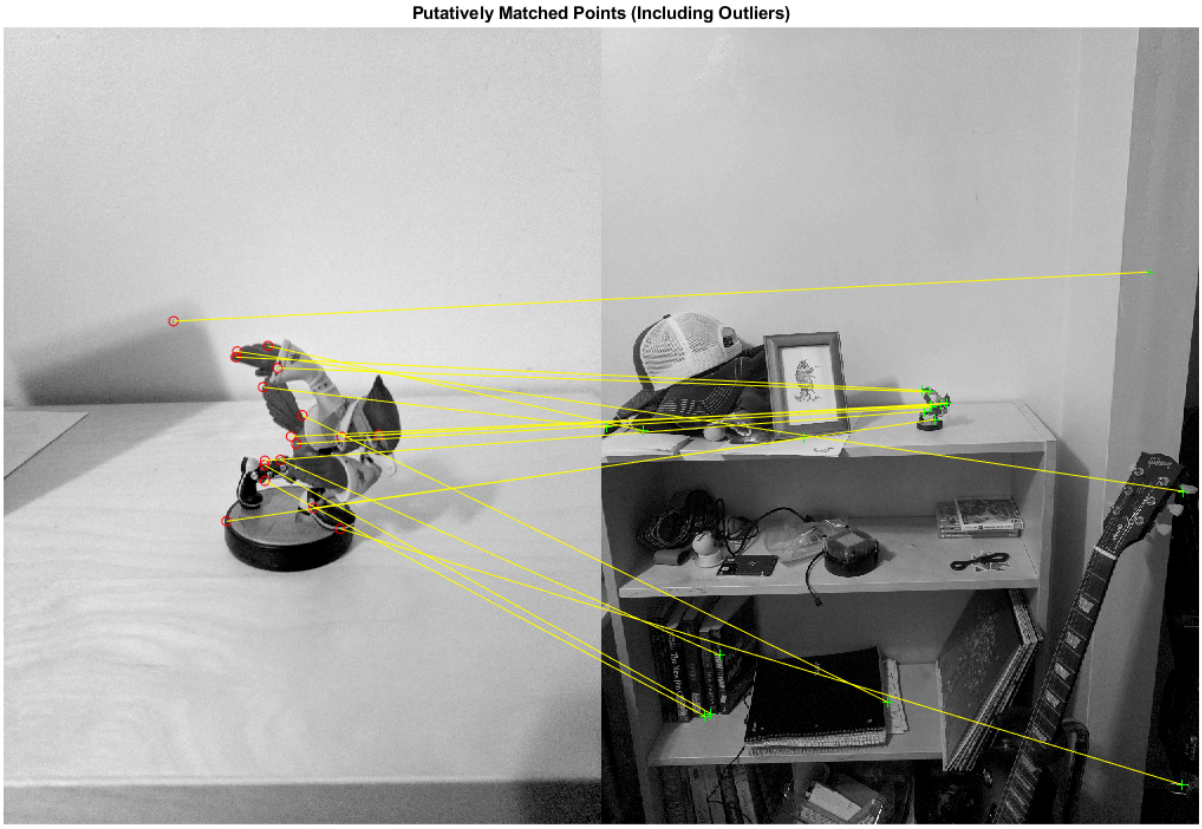
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



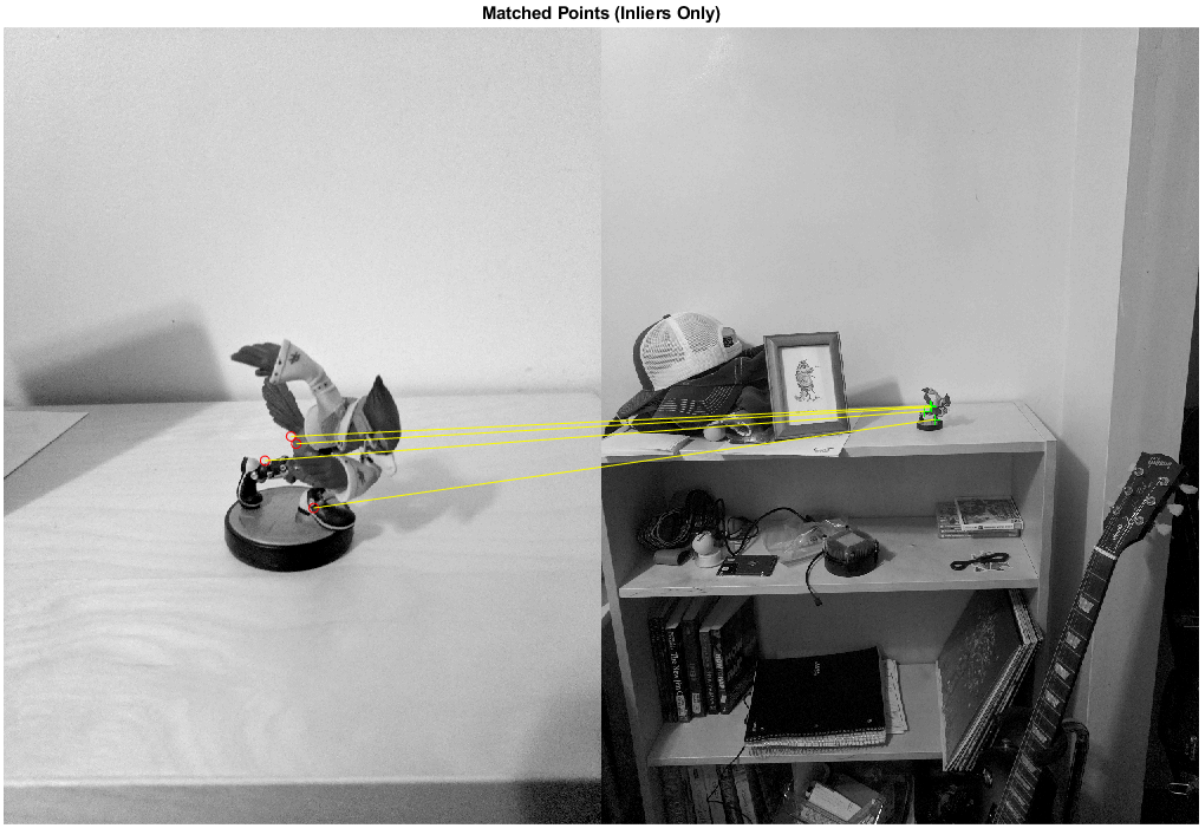
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

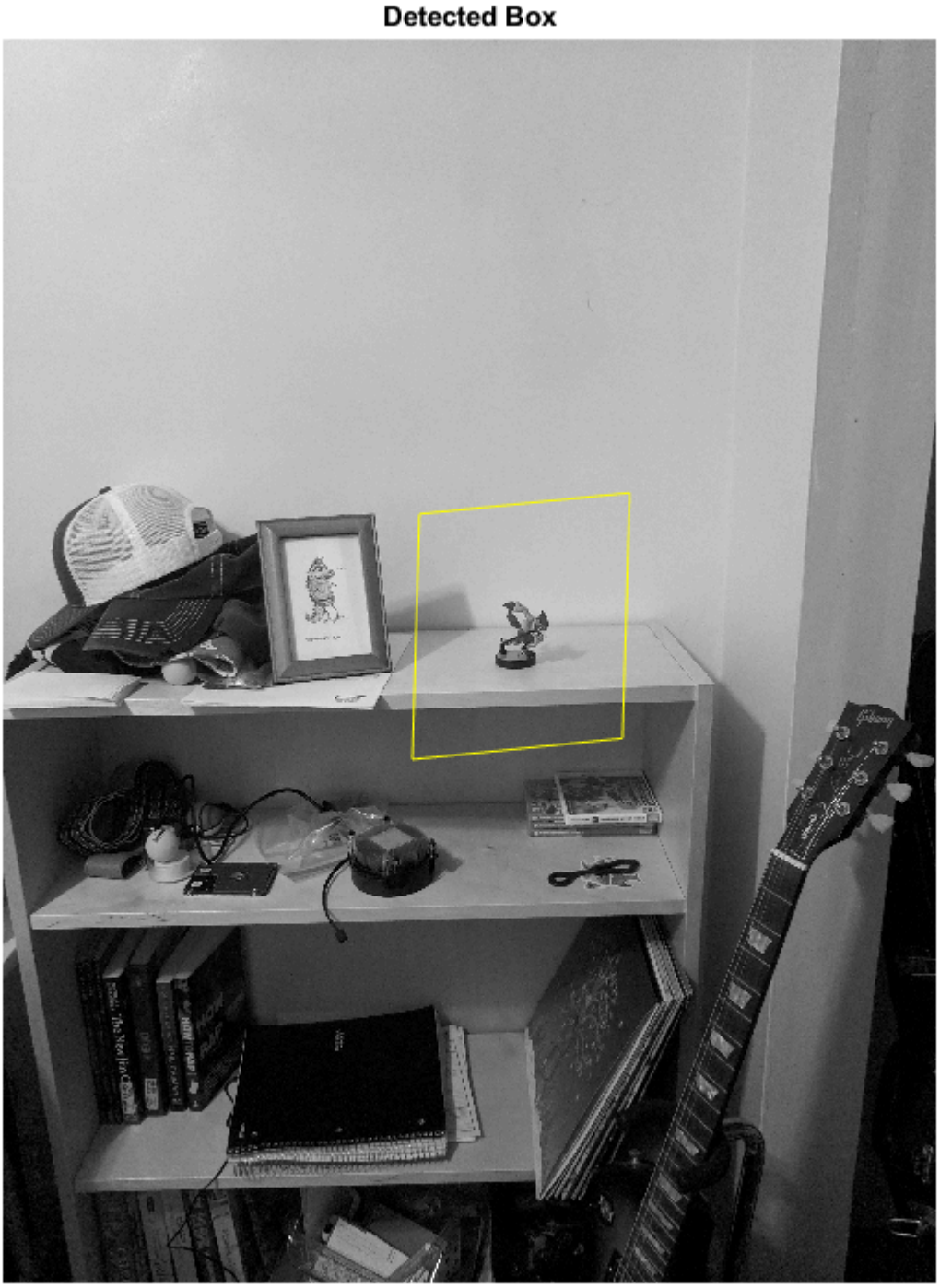
Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

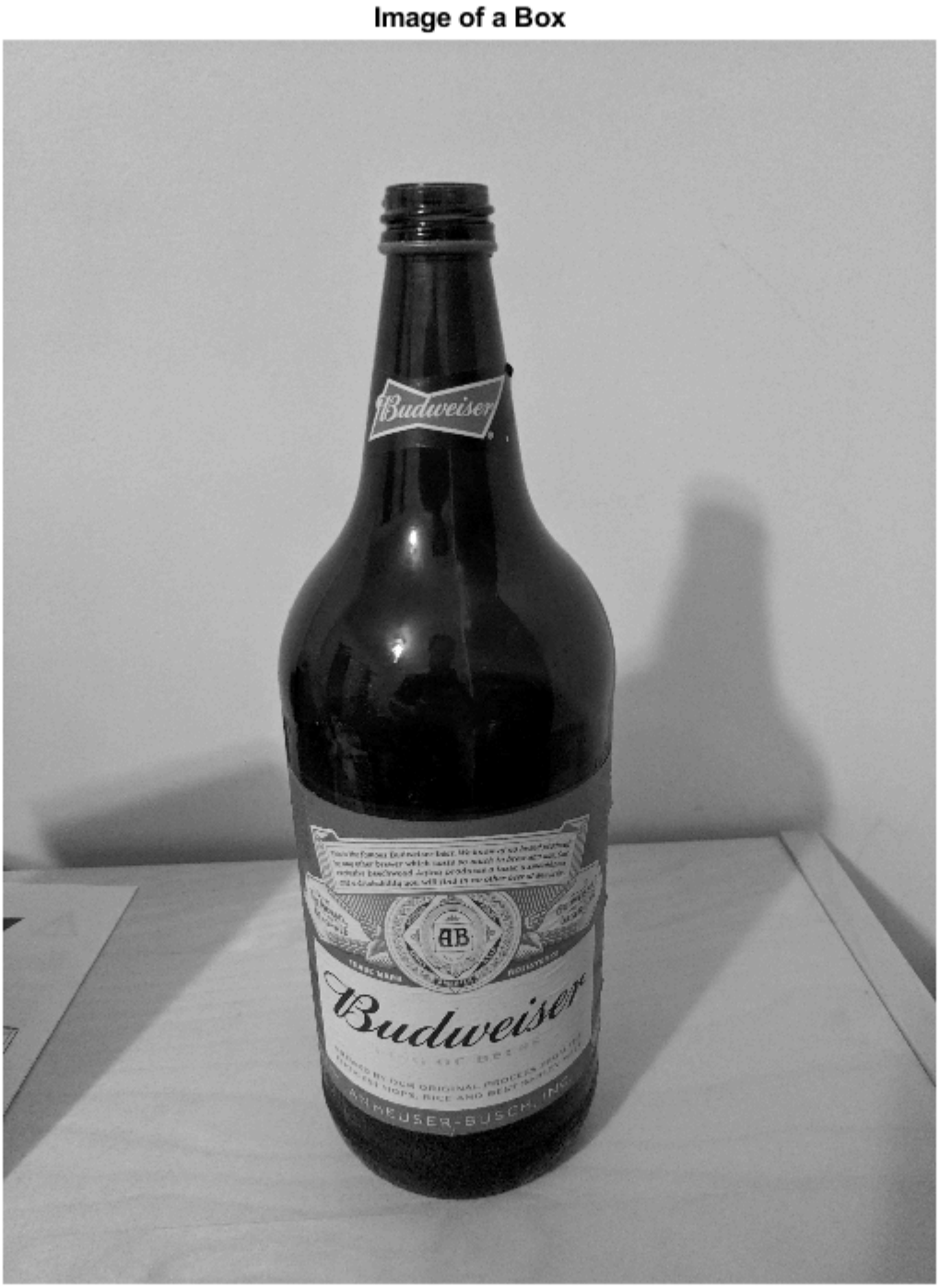


Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('40_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```



Read the target image containing a cluttered scene.

```
sceneImage = imread('40_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```

Step 5: Locate the Object in the Scene Using Putative Matches

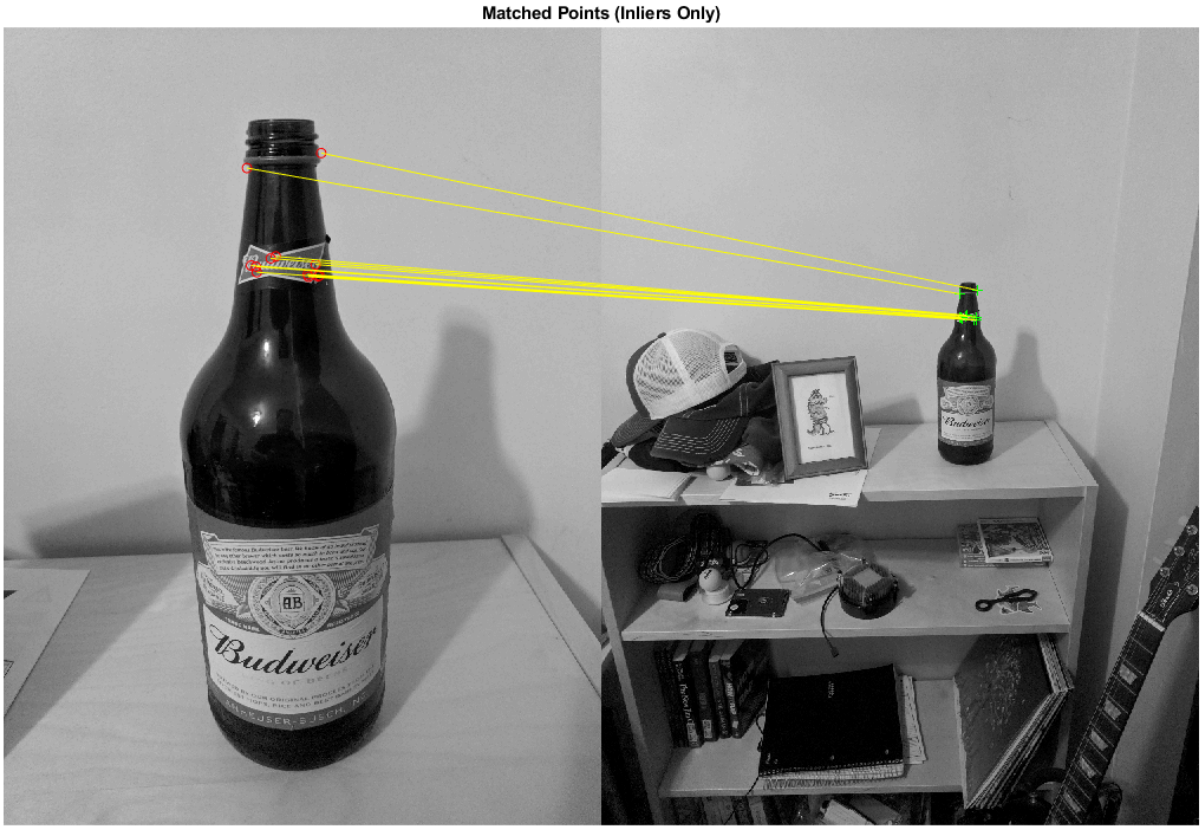
`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

```
Warning: Maximum number of trials reached. Consider increasing the maximum distance or decreasing the desired
 confidence.
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('jif_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Image of a Box

Read the target image containing a cluttered scene.

```
sceneImage = imread('jif_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

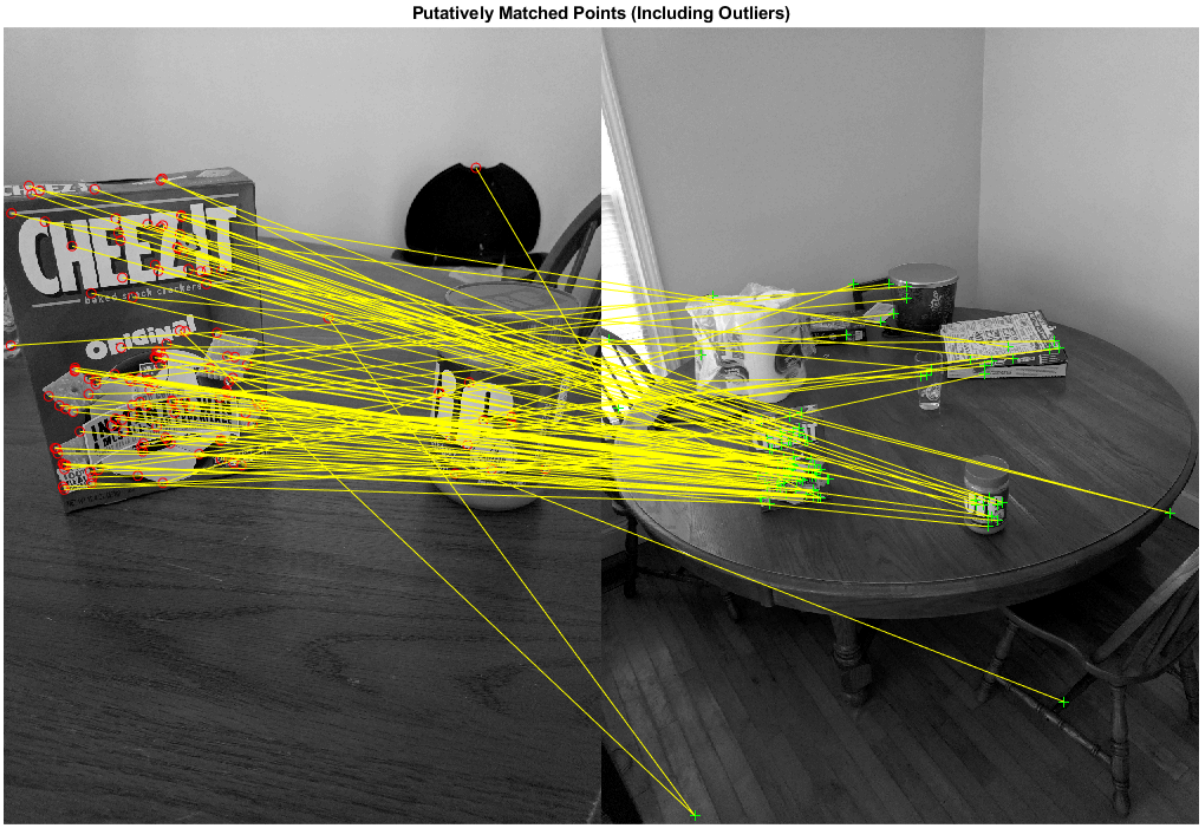
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



Step 5: Locate the Object in the Scene Using Putative Matches

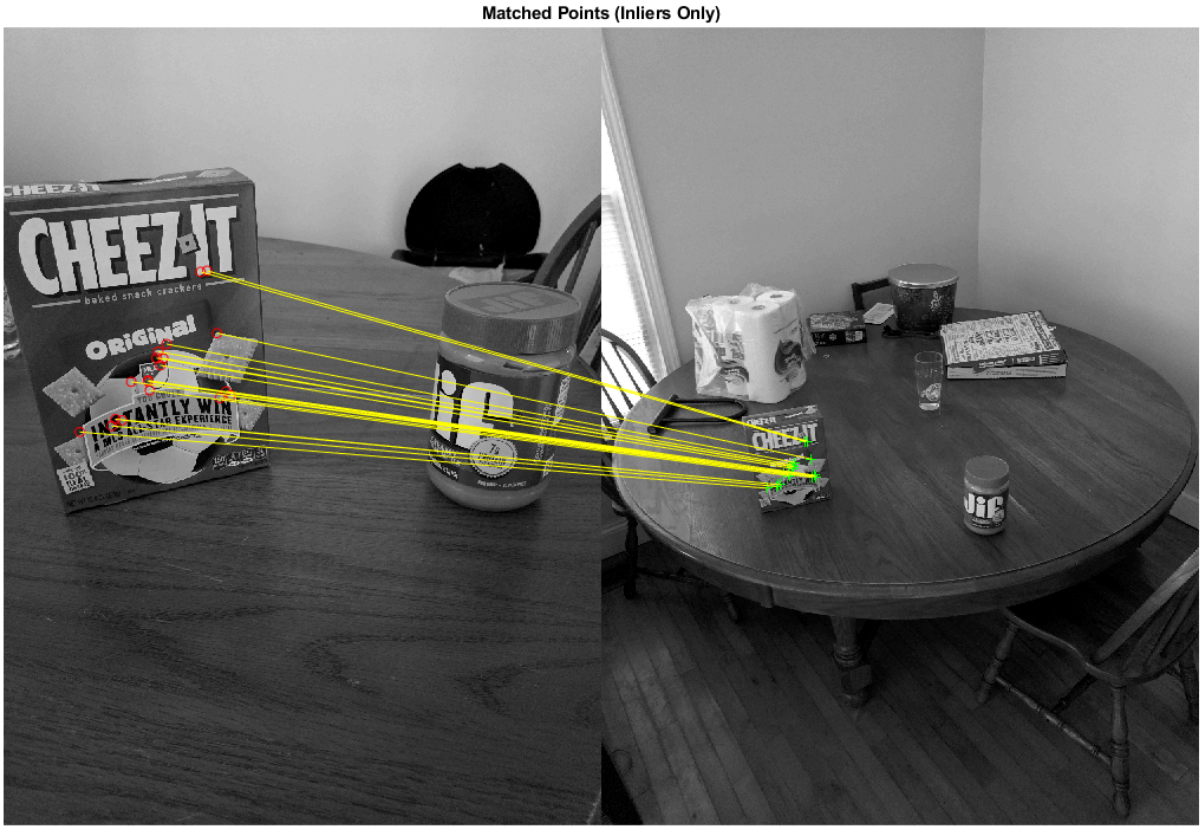
`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

```
Warning: Maximum number of trials reached. Consider increasing the maximum distance or decreasing the desired
 confidence.
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('kevin_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('kevin_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

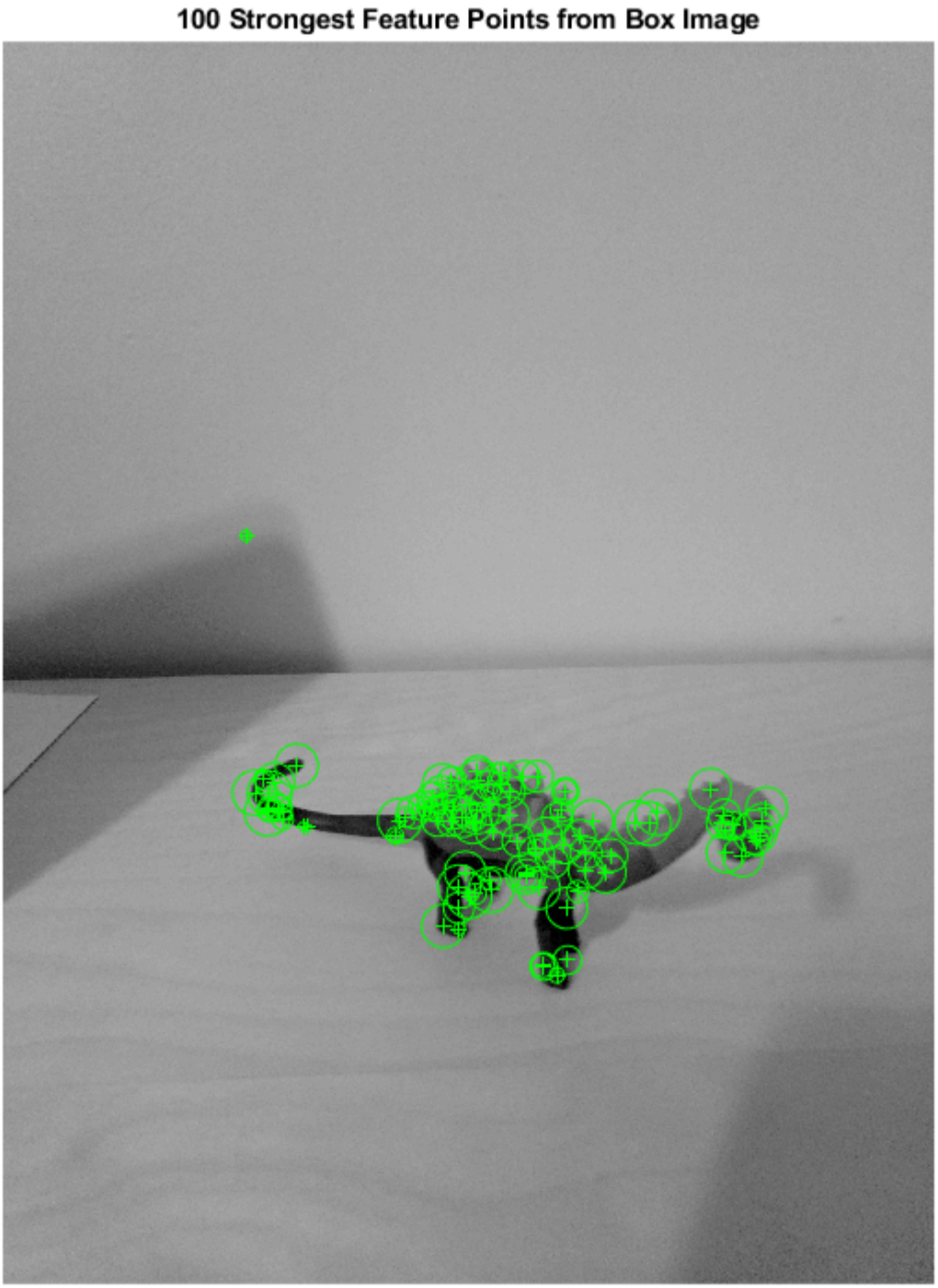
Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```



Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

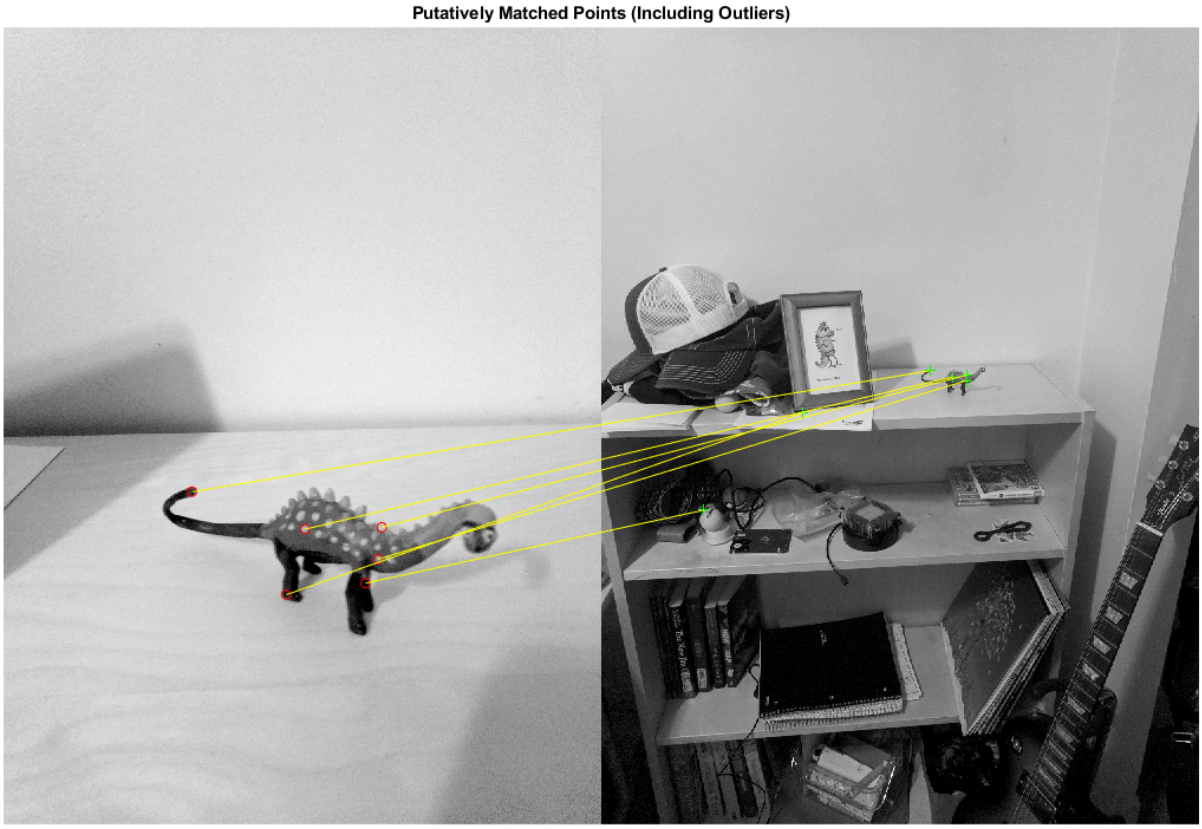
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



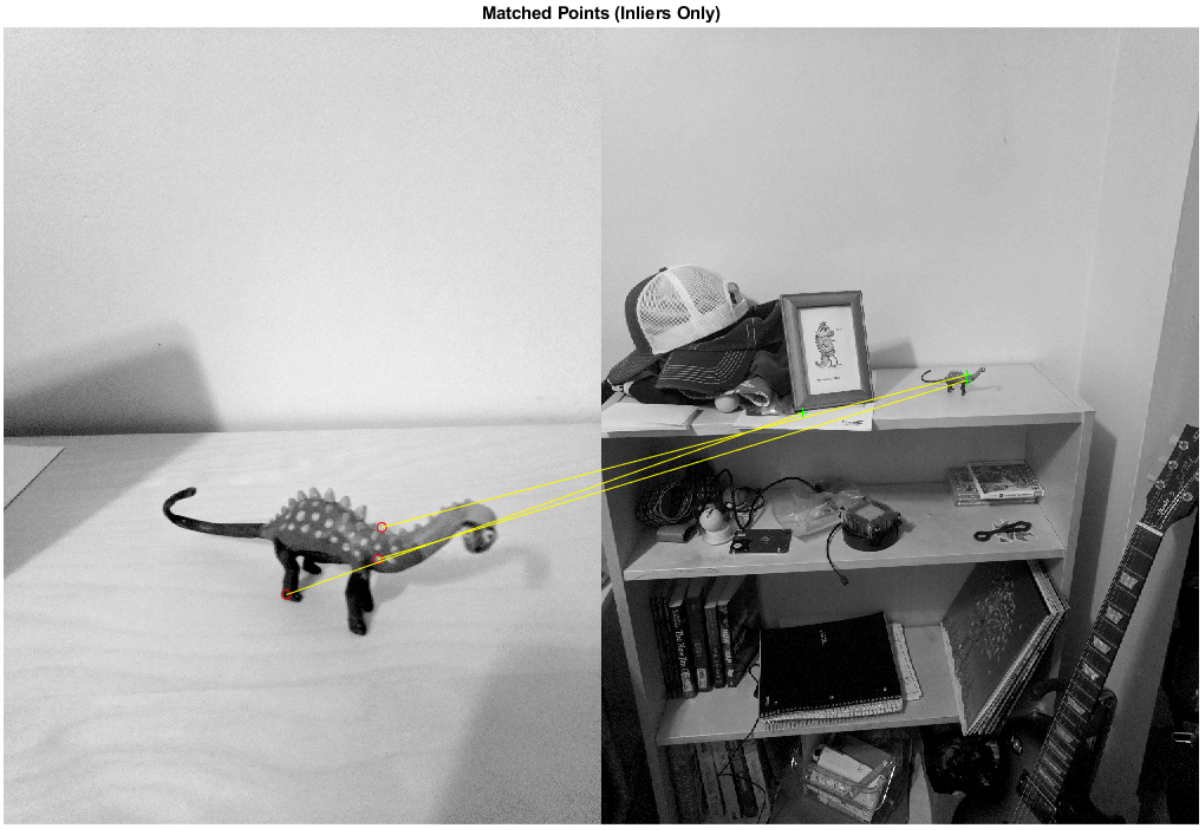
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

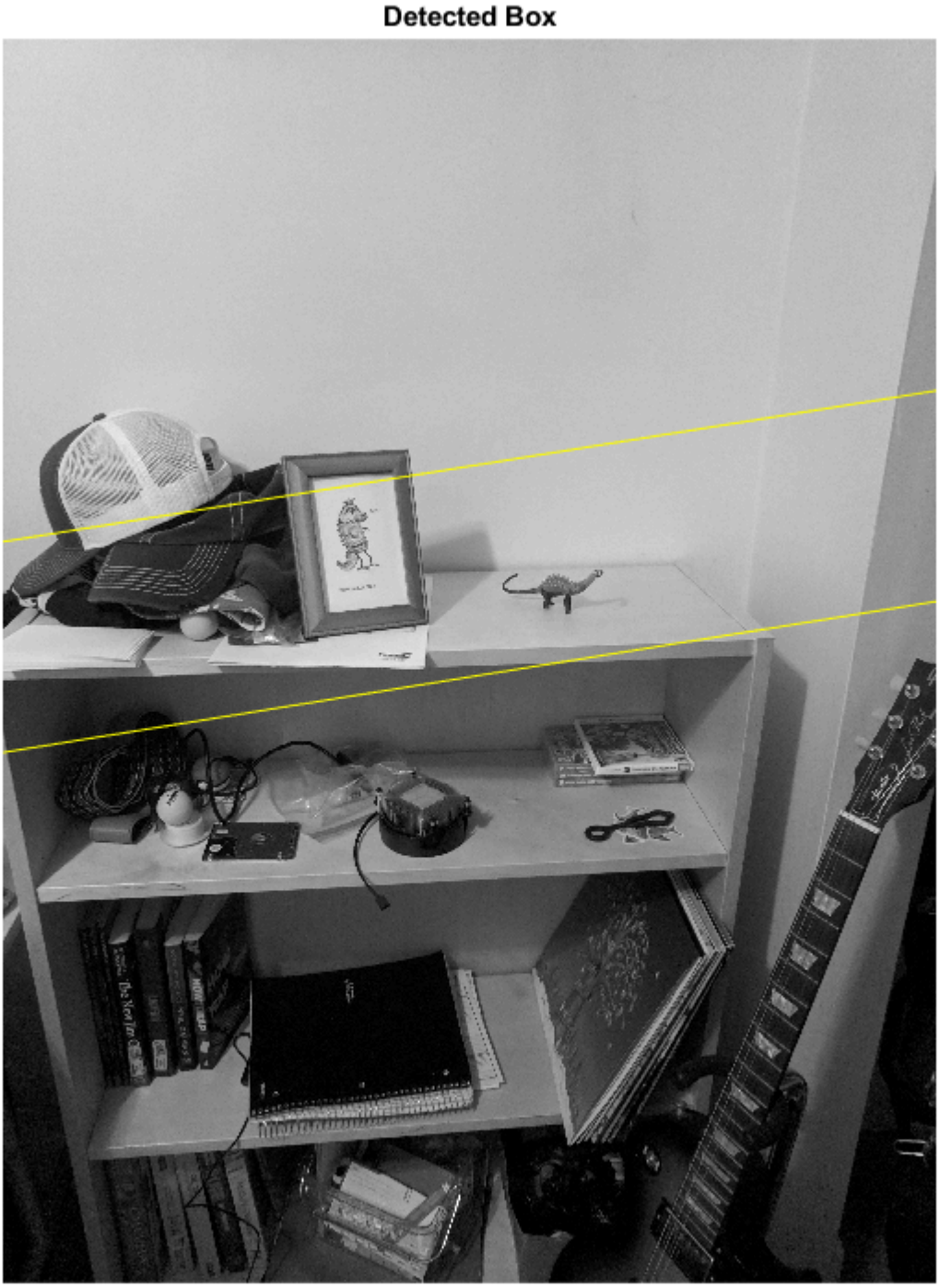
Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```



Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

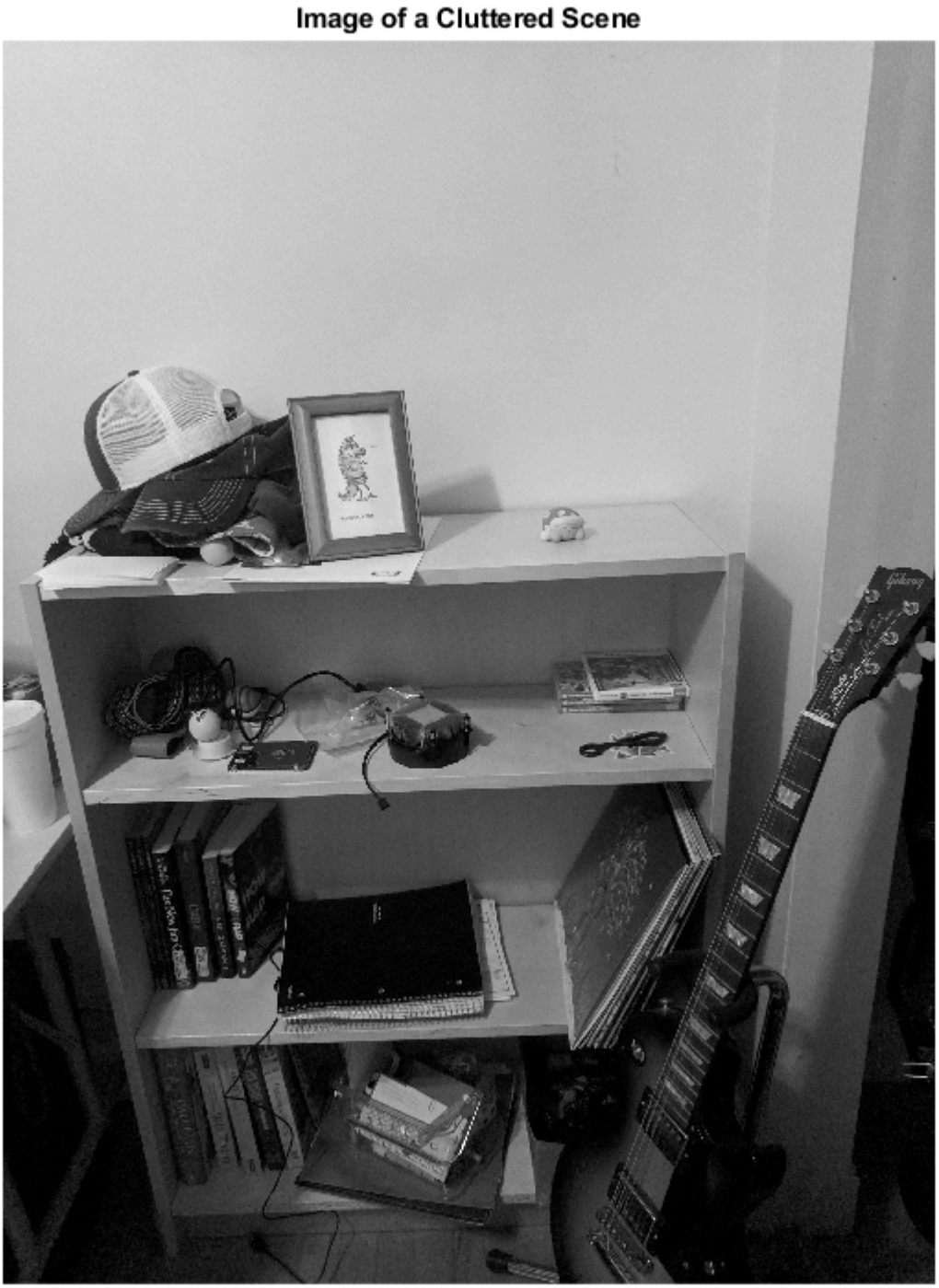
Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('kirby_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('kirby_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```



Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```

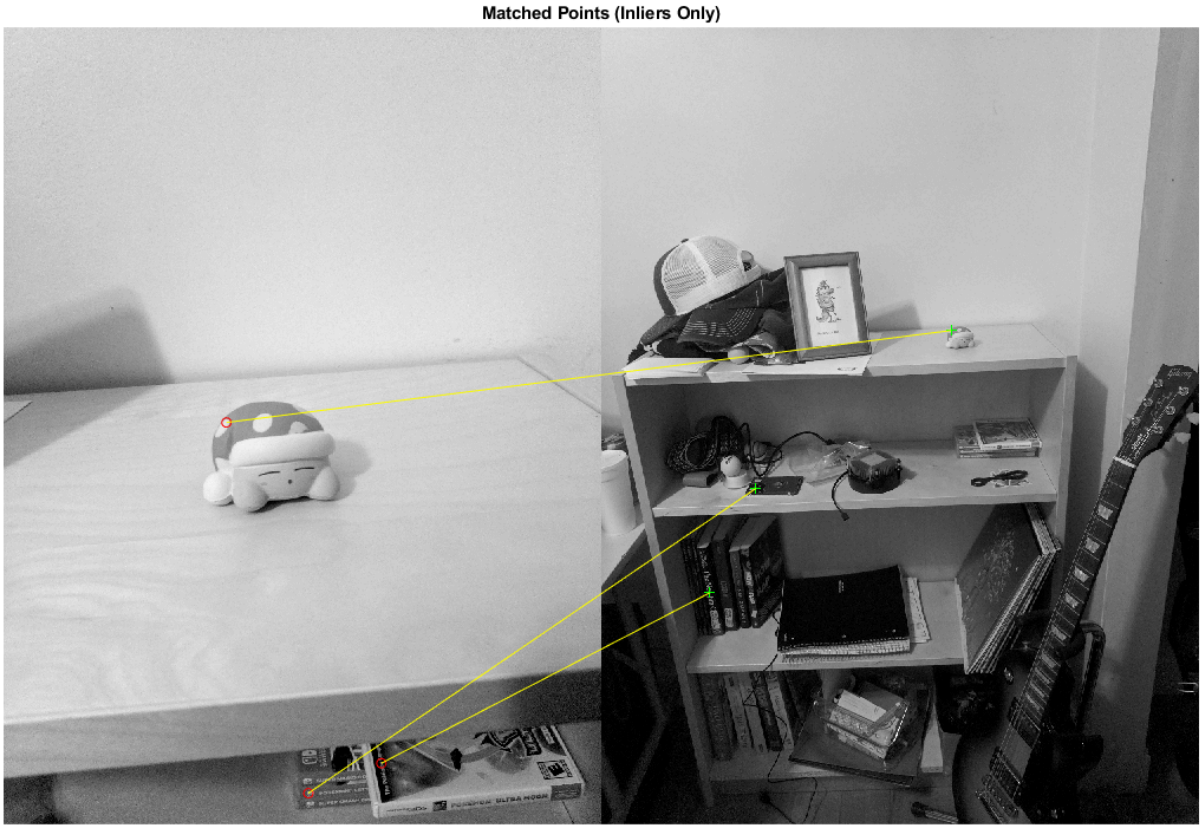
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

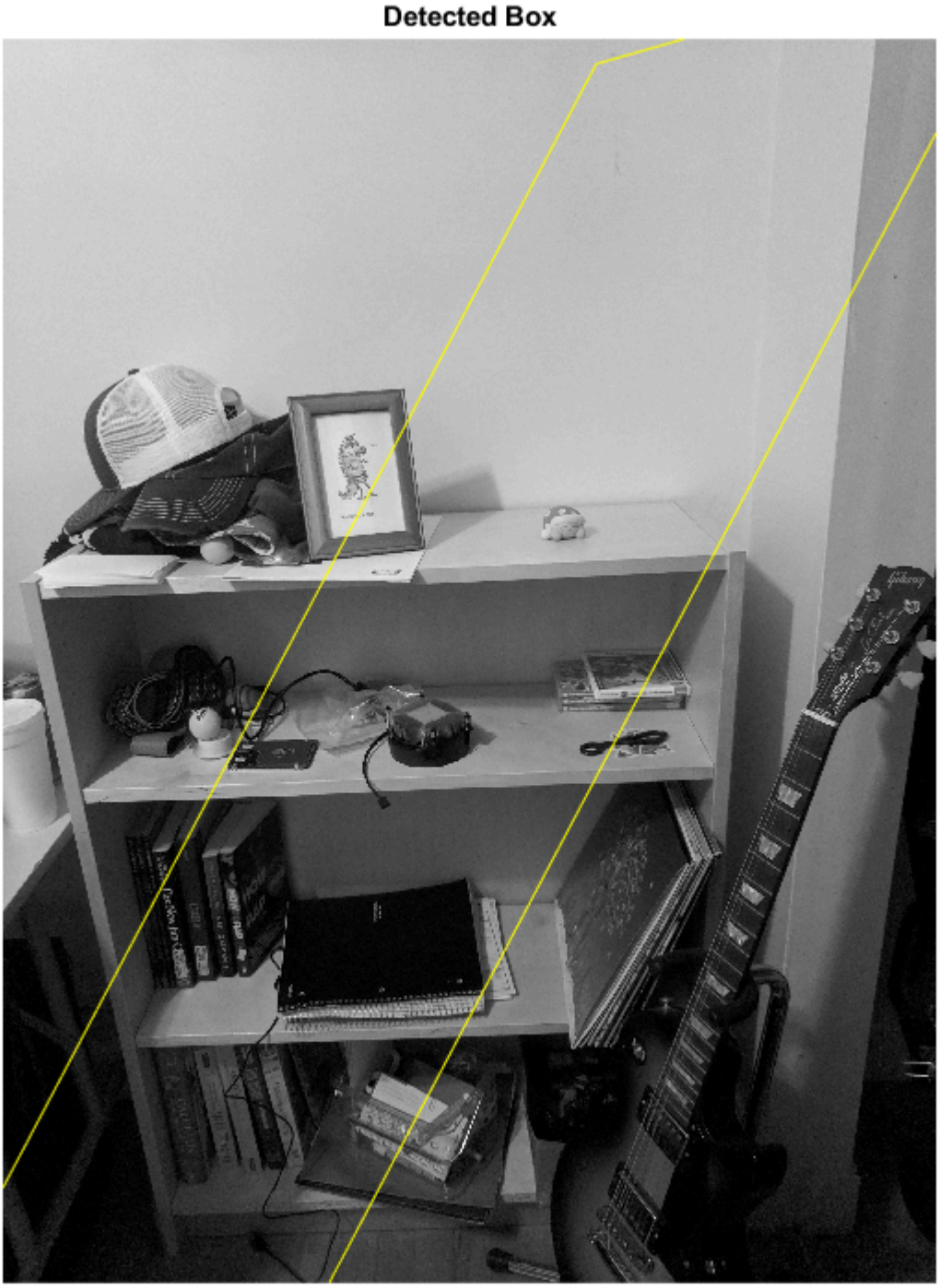
Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```



Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('pickle_1.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('pickle_2.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

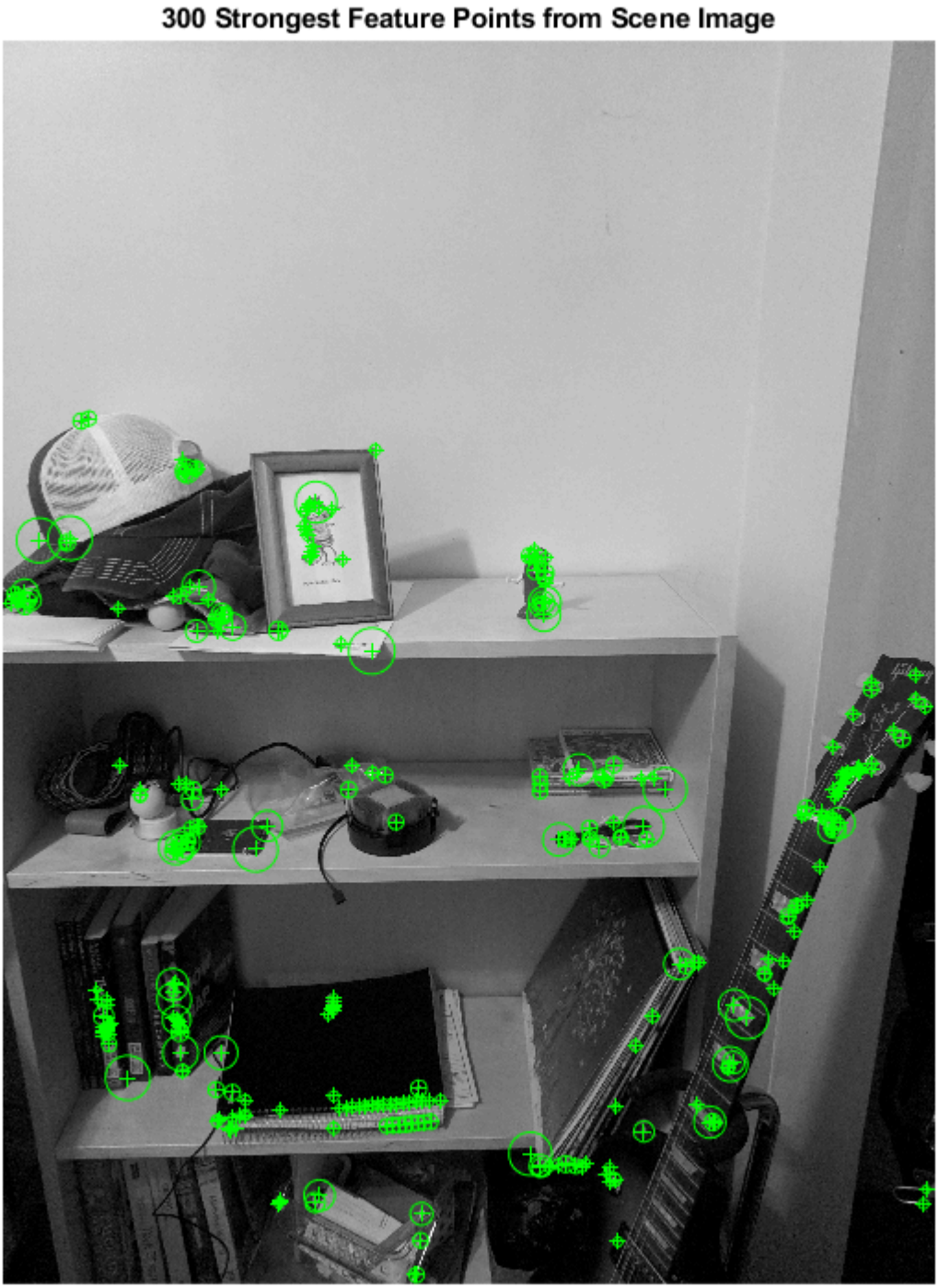
Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```

Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```



Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

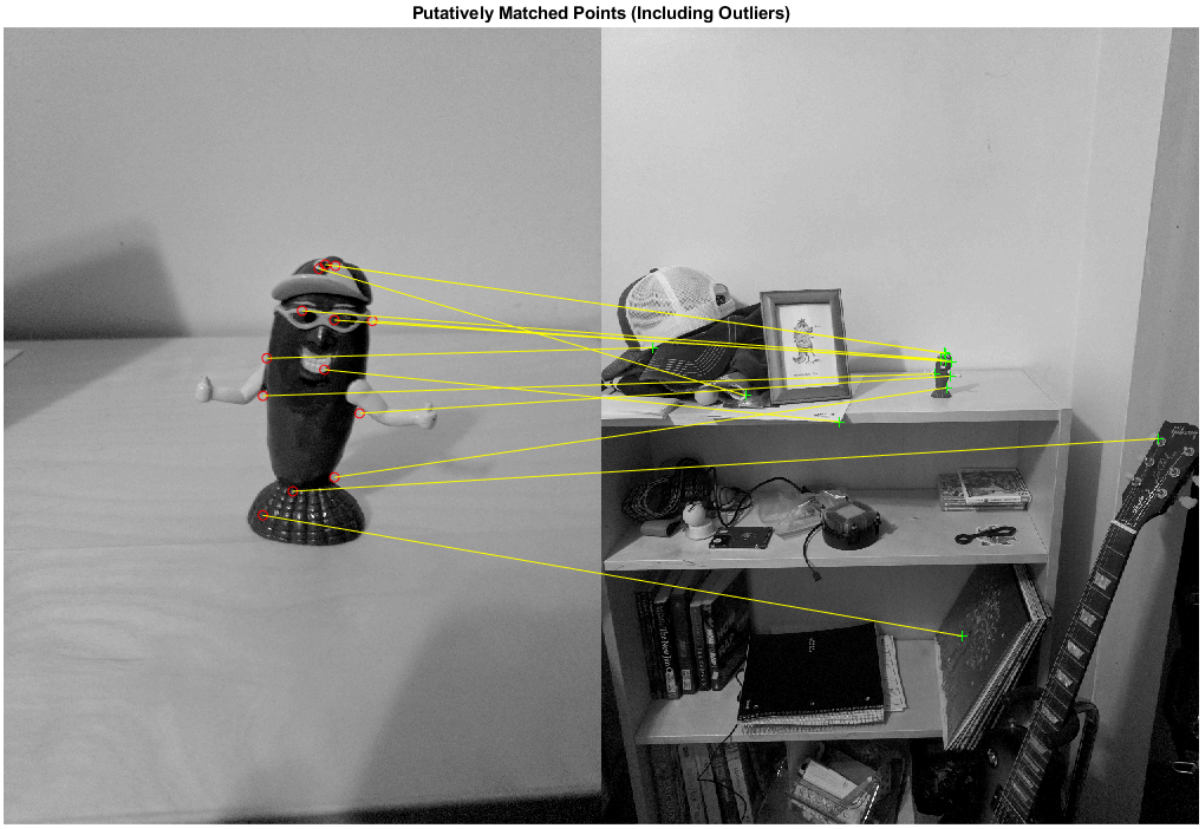
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



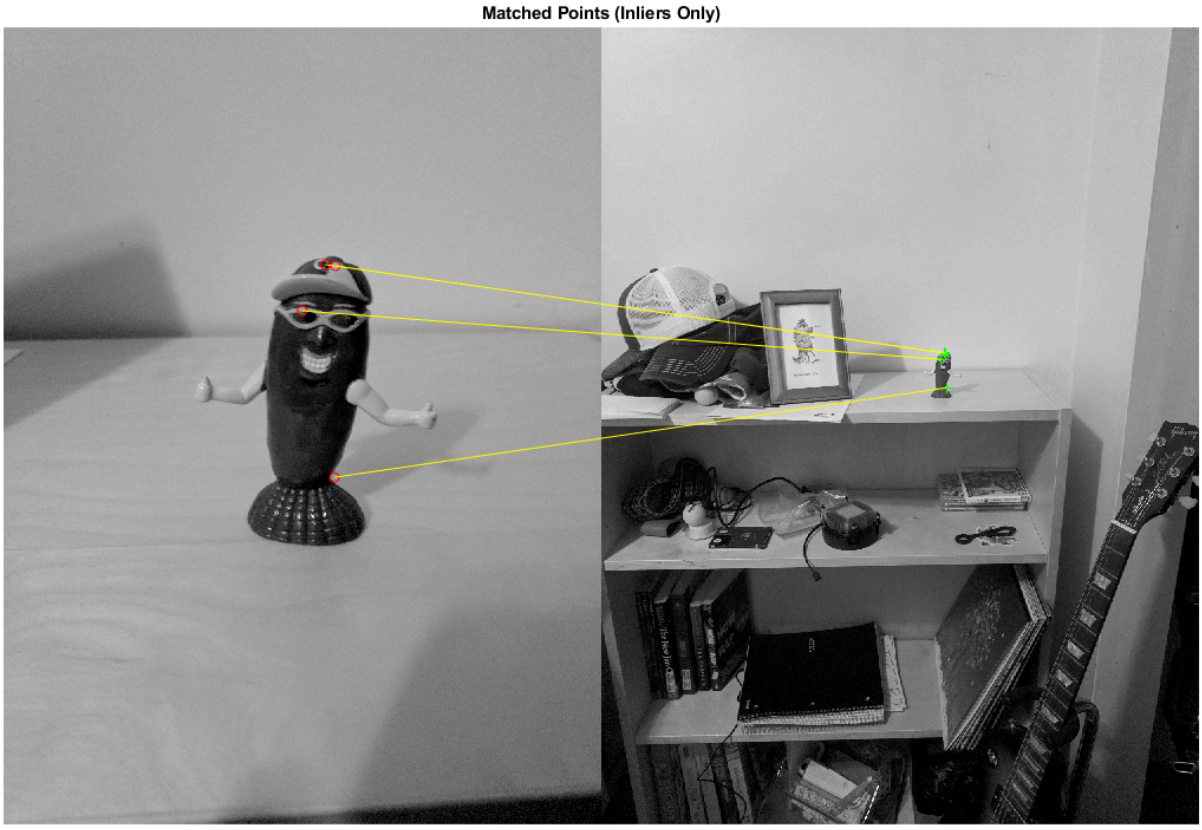
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

Object Detection in a Cluttered Scene Using Point Feature Matching

Step 1: Read Images

Read the reference image containing the object of interest.

```
addpath 'D:'\homework\'comp vision'\'image pairs'\
boxImage = imread('megaman3.jpg');
boxImage = rgb2gray(boxImage);
figure;
imshow(boxImage);
title('Image of a Box');
```

Read the target image containing a cluttered scene.

```
sceneImage = imread('megaman1.jpg');
sceneImage = rgb2gray(sceneImage);
figure;
imshow(sceneImage);
```

```
title('Image of a Cluttered Scene');
```

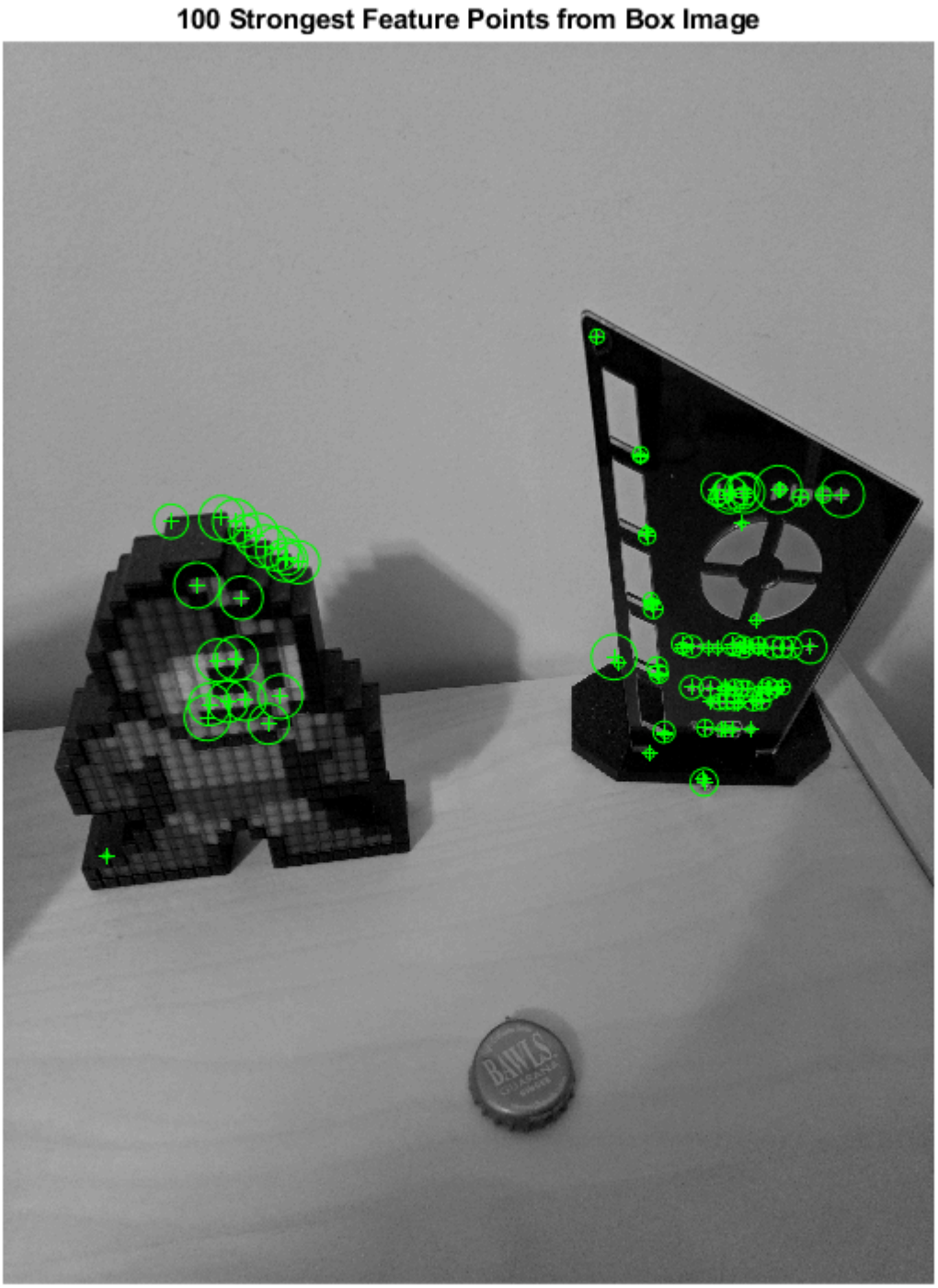
Step 2: Detect Feature Points

Detect feature points in both images.

```
boxPoints = detectSURFFeatures(boxImage);
scenePoints = detectSURFFeatures(sceneImage);
```

Visualize the strongest feature points found in the reference image.

```
figure;
imshow(boxImage);
title('100 Strongest Feature Points from Box Image');
hold on;
plot(selectStrongest(boxPoints, 100));
```



Visualize the strongest feature points found in the target image.

```
figure;
imshow(sceneImage);
title('300 Strongest Feature Points from Scene Image');
hold on;
```

```
plot(selectStrongest(scenePoints, 300));
```

Step 3: Extract Feature Descriptors

Extract feature descriptors at the interest points in both images.

```
[boxFeatures, boxPoints] = extractFeatures(boxImage, boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage, scenePoints);
```

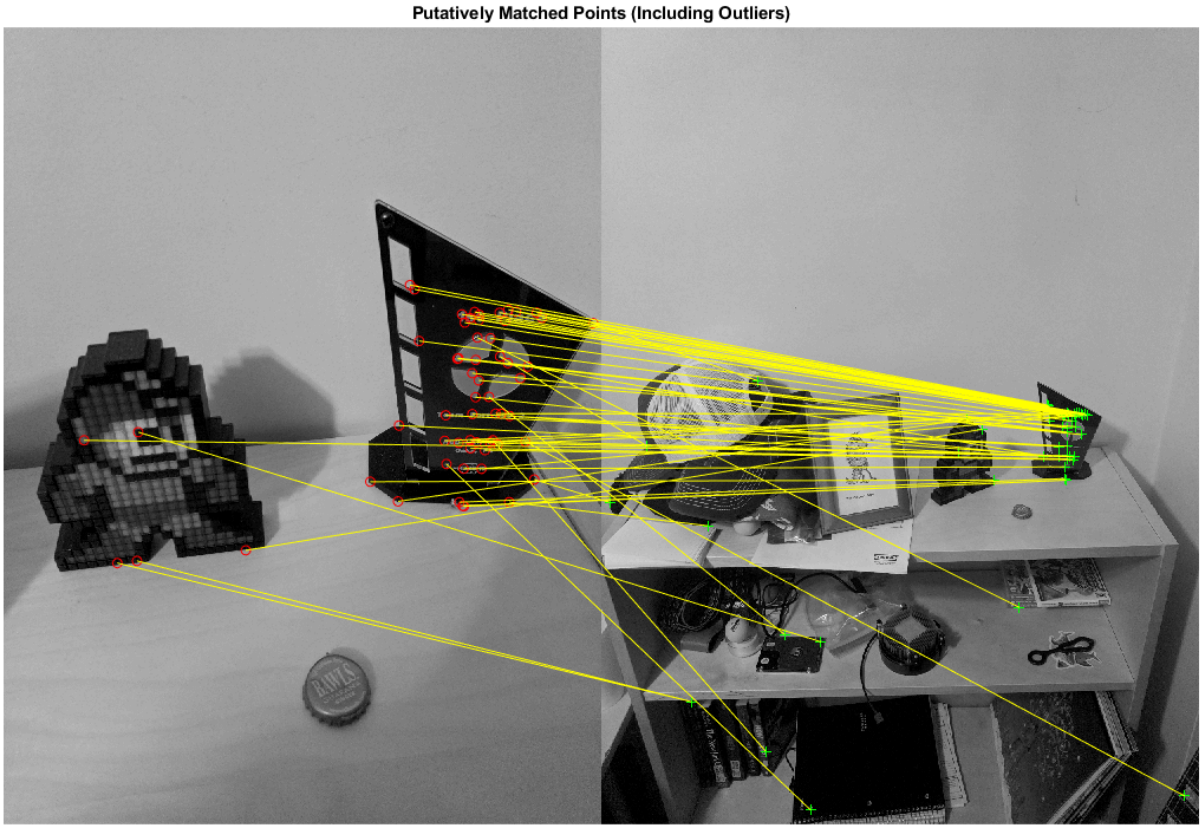
Step 4: Find Putative Point Matches

Match the features using their descriptors.

```
boxPairs = matchFeatures(boxFeatures, sceneFeatures);
```

Display putatively matched features.

```
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);
figure;
showMatchedFeatures(boxImage, sceneImage, matchedBoxPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');
```



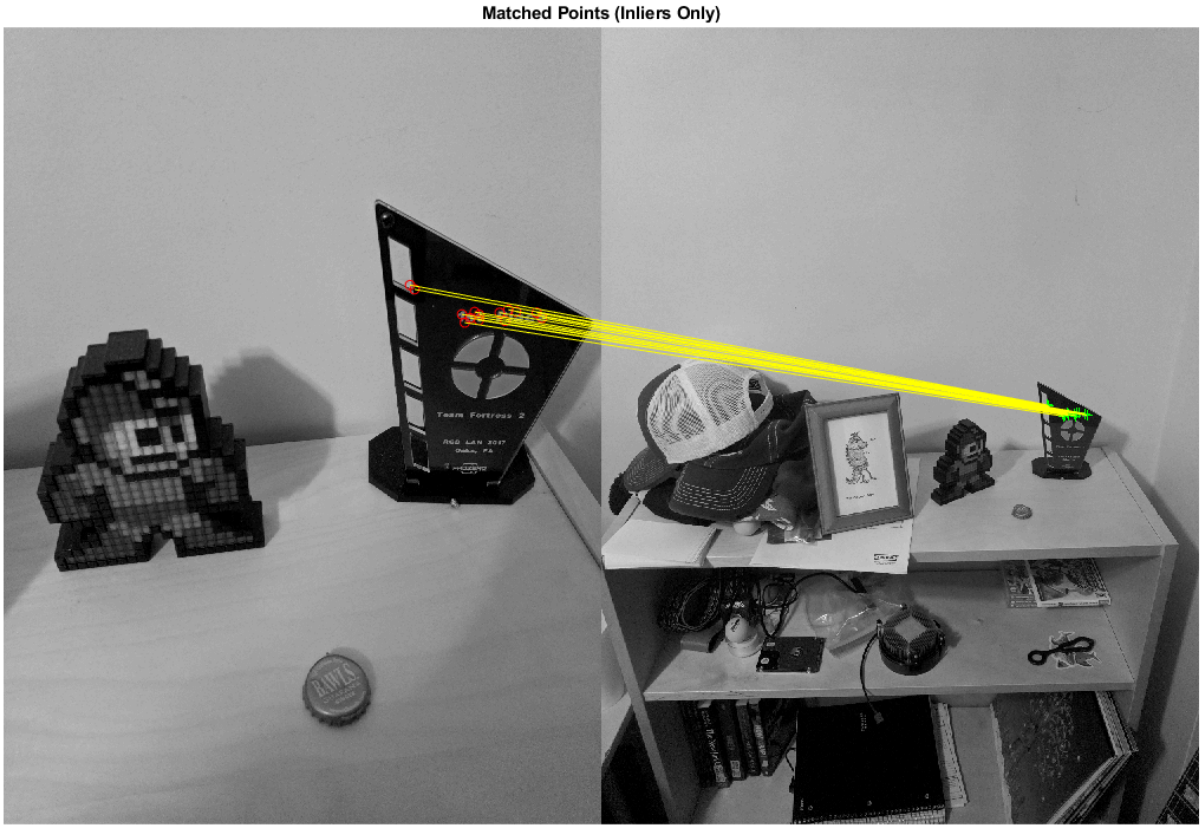
Step 5: Locate the Object in the Scene Using Putative Matches

`estimateGeometricTransform` calculates the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

```
[tform, inlierBoxPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedBoxPoints, matchedScenePoints, 'affine');
```

Display the matching point pairs with the outliers removed

```
figure;
showMatchedFeatures(boxImage, sceneImage, inlierBoxPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```



Get the bounding polygon of the reference image.

```
boxPolygon = [1, 1;...                           % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon
```

Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

```
newBoxPolygon = transformPointsForward(tform, boxPolygon);
```

Display the detected object.

```
figure;
```

```
imshow(sceneImage);
hold on;
line(newBoxPolygon(:, 1), newBoxPolygon(:, 2), 'Color', 'y');
title('Detected Box');
```

```
addpath 'D:'\homework\'comp vision'\'image pairs'\

I = imread('cheeto_2.jpg');
I = rgb2gray(I);

[J, rect] = imcrop(I);
```

```
T = imresize(J,[51 51]);
imshow(T)
```

```
ROI = [0 0 51 51];
tMatcher = vision.TemplateMatcher('ROIInputPort', true, ...
                                  'BestMatchNeighborhoodOutputPort', true);

[location,Nvals,Nvalid] = tMatcher(I,T,ROI);
location
```

```
location = 1×2 int32 row vector
   28   26
```

```
Nvals
```

```
Nvals = 3×3 uint8 matrix
     0     0     0
    85    73   112
   146   145   191
```

```
Nvalid
```

```
Nvalid = logical
   0
```

Kyle Bishop Quiz 1 - Assignment 4 002-24-8328

1 $\begin{bmatrix} x_{cam} \\ y_{cam} \\ w \end{bmatrix} = \begin{bmatrix} -f/s_x & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \left[\begin{array}{c|c} R & t \end{array}\right] \cdot \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ 1 \end{bmatrix}$

Camera points — Intrinsic Matrix — Extrinsic Matrix — World points

2 a) $R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$ $R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$ $R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

b) $R_x \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/2 & -\sqrt{3}/2 \\ 0 & \sqrt{3}/2 & 1/2 \end{bmatrix}$ R_y, $\theta = 60°$ $\begin{bmatrix} 1/2 & 0 & \sqrt{3}/2 \\ 0 & 1 & 0 \\ -\sqrt{3}/2 & 0 & 1/2 \end{bmatrix}$ =

$(1,1) = 1 \cdot \frac{1}{2} + 0 \cdot 0 + 0 \cdot (-\frac{\sqrt{3}}{2}) = \frac{1}{2}$
$(1,2) = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$
$(1,3) = 1 \cdot \frac{\sqrt{3}}{2} + 0 \cdot 0 + 0 \cdot \frac{1}{2} = \frac{\sqrt{3}}{2}$
$(2,1) = 0 \cdot \frac{1}{2} + \frac{1}{2} \cdot 0 + (-\frac{\sqrt{3}}{2}) \cdot (\frac{\sqrt{3}}{2}) = \frac{3}{4}$
$(2,2) = \frac{1}{2}$
$(2,3) = -\frac{\sqrt{3}}{4}$
$(3,1) = -\frac{\sqrt{3}}{4}$
$(3,2) = \frac{\sqrt{3}}{2}$
$(3,3) = \frac{1}{4}$

$R_x R_y$ $\begin{bmatrix} 1/2 & 0 & \sqrt{3}/2 \\ \frac{3}{4} & \frac{1}{2} & -\frac{\sqrt{3}}{4} \\ -\frac{\sqrt{3}}{4} & \frac{\sqrt{3}}{2} & \frac{1}{4} \end{bmatrix}$ R_z $\begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ =

$(1,1) = \frac{1}{4}$
$(1,2) = -\frac{\sqrt{3}}{4}$
$(1,3) = \frac{\sqrt{3}}{2}$
$(2,1) = \frac{3}{8} + \frac{\sqrt{3}}{4} = \frac{3+2\sqrt{3}}{8}$
$(2,2) = -\frac{3\sqrt{3}}{8} + \frac{1}{4} = \frac{2-3\sqrt{3}}{8}$
$(2,3) = -\frac{\sqrt{3}}{4}$
$(3,1) = -\frac{\sqrt{3}}{8} + \frac{3}{4} = \frac{6-\sqrt{3}}{8}$
$(3,2) = \frac{3}{8} + \frac{\sqrt{3}}{4} = \frac{3+2\sqrt{3}}{8}$
$(3,3) = \frac{1}{4}$

$R_x R_y R_z$ $\begin{bmatrix} \frac{1}{4} & -\frac{\sqrt{3}}{4} & \frac{\sqrt{3}}{2} \\ \frac{3+2\sqrt{3}}{8} & \frac{2-3\sqrt{3}}{8} & -\frac{\sqrt{3}}{4} \\ \frac{6-\sqrt{3}}{8} & \frac{3+2\sqrt{3}}{8} & \frac{1}{4} \end{bmatrix}$ R_y $\begin{bmatrix} 1/2 & 0 & \sqrt{3}/2 \\ 0 & 1 & 0 \\ -\sqrt{3}/2 & 0 & 1/2 \end{bmatrix}$ =

$(1,1) = \frac{1}{4} \cdot \frac{1}{2} + (-\frac{\sqrt{3}}{4}) \cdot 0 + \frac{\sqrt{3}}{2} \cdot (\frac{\sqrt{3}}{2}) = \frac{1}{8} + \frac{3}{4} = \frac{5}{8}$
$(1,2) = \frac{1}{4} \cdot 0 + (-\frac{\sqrt{3}}{4}) \cdot 1 + \frac{\sqrt{3}}{2} \cdot 0 = -\frac{\sqrt{3}}{4}$
$(1,3) = \frac{1}{4} \cdot \frac{\sqrt{3}}{2} + 0 + \frac{\sqrt{3}}{2} \cdot \frac{1}{2} = \frac{\sqrt{3}}{8} + \frac{\sqrt{3}}{4} = \frac{3\sqrt{3}}{8}$
$(2,1) = \frac{3+2\sqrt{3}}{8} \cdot \frac{1}{2} + \frac{2-3\sqrt{3}}{8} \cdot 0 - \frac{\sqrt{3}}{4} \cdot (\frac{\sqrt{3}}{2}) = \frac{3+2\sqrt{3}}{16} + \frac{3}{8} = \frac{9+2\sqrt{3}}{16}$
$(2,2) = 0 + \frac{2-3\sqrt{3}}{8} \cdot 1 + 0 = \frac{2-3\sqrt{3}}{8}$
$(2,3) = \frac{3+2\sqrt{3}}{8} \cdot \frac{\sqrt{3}}{2} + \frac{2-3\sqrt{3}}{8} \cdot 0 + (-\frac{\sqrt{3}}{4}) \cdot \frac{1}{2} = \frac{2+3\sqrt{3}}{16} - \frac{\sqrt{3}}{8} = \frac{2+\sqrt{3}}{16}$
$(3,1) = \frac{6-\sqrt{3}}{8} \cdot \frac{1}{2} + 0 + \frac{1}{4} \cdot (-\frac{\sqrt{3}}{2}) = \frac{6-\sqrt{3}}{16} - \frac{\sqrt{3}}{8} = \frac{6-3\sqrt{3}}{16}$
$(3,2) = 0 + \frac{3+2\sqrt{3}}{8} \cdot 1 + 0 = \frac{3+2\sqrt{3}}{8}$
$(3,3) = \frac{6-\sqrt{3}}{8} \cdot \frac{\sqrt{3}}{2} + 0 + \frac{1}{4} \cdot \frac{1}{2} = \frac{6\sqrt{3}-3}{16} + \frac{1}{8} = \frac{6\sqrt{3}-1}{16}$

$R_x R_y R_z R_y$ $\begin{bmatrix} -\frac{5}{8} & -\frac{\sqrt{3}}{4} & \frac{3\sqrt{3}}{8} \\ \frac{9+2\sqrt{3}}{16} & \frac{2-3\sqrt{3}}{8} & \frac{2+3\sqrt{3}}{16} \\ \frac{6-3\sqrt{3}}{16} & \frac{3+2\sqrt{3}}{8} & \frac{6\sqrt{3}-1}{16} \end{bmatrix}$

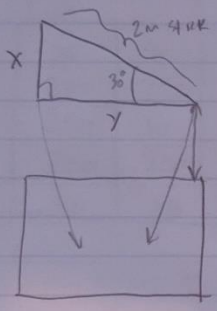
$x = \theta = \cos^{-1}\left(-\frac{5}{8}\right), y = \cos^{-1}\left(\frac{2-3\sqrt{3}}{8}\right), z = \cos^{-1}\left(\frac{6\sqrt{3}-1}{16}\right)$

Kyle Bishop Quiz 1 - Assignment 4 Continued 002-24-4326

3 a) ${}^{c}p = {}^{c}R_{w}\,{}^{w}p + {}^{c}t_{\rightarrow w}$

$${}^{c}p = \begin{bmatrix} -0.707 & 0.707 & 0 \\ 0 & 0 & 1 \\ 0.707 & 0.707 & 0 \end{bmatrix} {}^{w}p + \begin{bmatrix} 9 \\ 0 \\ -4 \end{bmatrix}$$

b)



$\frac{x}{2m} = \sin(30^\circ)$ $\frac{y}{2m} = \cos(30^\circ)$

$x = 2\sin(30^\circ)$ $y = 2\cos(30^\circ)$

$x' = \frac{fx}{z}$ $y' = \frac{fy}{z}$

$x' = \frac{(1000)(2\sin(30))}{4}$

$y' = \frac{(1000)(2\cos(30))}{4}$

Size of stick = x' x y' pixels

4 a) principal point = the origin of the camera in reference to the real world point (O_x, O_y)

b) $M = M_{\text{intrinsic}} \cdot M_{\text{extrinsic}}$

$$\begin{bmatrix} -f/s_x & 0 & O_x \\ 0 & f/s_y & O_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \left[\begin{array}{c:c} R & t \end{array}\right]$$

intrinsic extrinsic

f = focal length
(s_x, s_y) = size of pixels
(O_x, O_y) = principle point
R = rotation matrix
t = translation vector

Kyle Bishop Quiz 2 - Assignment 4 CSCI-24-8328

1. a)

Fig 1

250	50	250	50	250
50	250	50	250	50
250	50	250	50	250
50	250	50	250	50
250	50	250	50	250

Fig 2

250	255	250	240	250
101	98	102	120	240
250	110	250	99	250
250	250	101	100	250
250	255	250	255	250

Fig 3

50	50	50	50	50
50	250	250	250	50
50	250	10	250	50
50	250	250	250	50
50	50	50	50	50

AVG

183.33	116.33	183.33	113.33	183.33
67	199.33	134	206.66	113.33
183.33	136.66	170	133	183.33
116.66	250	133.66	200	116.66
183.33	116.33	183.33	116.33	183.33

3x3 Gaussian Kernel to $\sigma=1$

0.0585	0.0965	0.0585
0.0965	0.159	0.0965
0.0585	0.0965	0.0585

(row, column)

Convolve around AVG

(1,1) = 183.33 · 0.159 + 0.0965 · 116.33 + 67 · 0.0965 + 199.33 · 0.0585 =
= 29.15 + 11.42 + 6.47 + 11.66 = 58.7

(1,2) = 183.33 · 0.0965 + 0.159 · 116.33 + 0.0965 · 183.33 + 67 · 0.0585 + 199.33 · 0.0965 + 134 · 0.0585 =
= 17.69 + 11.42 + 17.69 + 3.92 + 19.24 + 7.84 = 77.8

(1,3) = 116.33 · 0.0965 + 183.33 · 0.159 + 113.33 · 0.0965 + 199.33 · 0.0585 + 134 · 0.0965 + 206.66 · 0.0585 =
= 11.42 + 29.15 + 10.94 + 11.66 + 12.93 + 12.09 = 88.19

(1,4) = 183.33 · 0.0965 + 113.33 · 0.159 + 183.33 · 0.0965 + 134 · 0.0585 + 206.66 · 0.0965 + 113.33 · 0.0585 =
= 17.69 + 18.02 + 17.69 + 7.84 + 19.94 + 6.63 = 88.03

(1,5) = 113.33 · 0.0965 + 183.33 · 0.159 + 206.66 · 0.0585 + 113.33 · 0.0965 =
= 10.94 + 29.15 + 12.09 + 10.94 = 63.12

(2,1) = 183.33 · 0.0965 + 116.33 · 0.0585 + 67 · 0.159 + 199.33 · 0.0965 + 183.33 · 0.0965 + 136.66 · 0.0585 =
= 17.69 + 6.92 + 10.65 + 19.24 + 17.69 + 7.99 = 80.18

(2,2) = 183.33 · 0.0585 + 116.33 · 0.0965 + 183.33 · 0.0585 + 67 · 0.0965 + 199.33 · 0.159 + 134 · 0.0965
+ 183.33 · 0.0585 + 136.66 · 0.0965 + 170 · 0.0585 = 10.72 + 11.42 + 10.72 + 6.47
+ 31.69 + 12.93 + 10.72 + 13.19 + 9.95 = 117.81

(2,3) = 116.33 · 0.0585 + 183.33 · 0.0965 + 113.33 · 0.0585 + 199.33 · 0.0965 + 134 · 0.159 + 206.66 · 0.0965
+ 136.66 · 0.0585 + 170 · 0.0965 + 133 · 0.0585 = 6.92 + 17.69 + 6.63 + 19.24 + 21.31 + 19.94
+ 7.99 + 16.41 + 7.78 = 123.91

Kyle Bishop QUIZ 2 - Assignment 4 Continued 902-24-8328

(2,4) = 183.33 · 0.0585 + 113.33 · 0.0965 + 183.33 · 0.0585 + 134 · 0.0965 + 206.66 · 0.159
10.72 + 10.94 + 10.72 + 12.93 + 32.86
+ 113.33 · 0.0965 + 170 · 0.0585 + 133 · 0.0965 + 183.33 · 0.0585 =
+ 10.94 + 9.95 + 12.83 + 10.72 = 122.66

(2,5) = 113.33 · 0.0585 + 183.33 · 0.0965 + 206.66 · 0.0965 + 113.33 · 0.159 + 133 · 0.0585 + 183.33 · 0.0965
= 6.63 + 17.69 + 19.94 + 18.02 + 7.78 + 17.69 = 87.75

(3,1) = 67 · 0.0965 + 199.33 · 0.0585 + 183.33 · 0.159 + 136.66 · 0.0965 + 116.66 · 0.0965 + 136.66 · 0.0585 =
6.47 + 11.66 + 29.15 + 13.19 + 11.26 + 7.99 = 79.72

(3,2) = 67 · 0.0585 + 199.33 · 0.0965 + 134 · 0.0585 + 183.33 · 0.0965 + 136.66 · 0.159 + 170 · 0.0965 + 116.66 · 0.0585
3.92 + 19.24 + 7.84 + 17.69 + 21.73 + 16.41 + 6.82
+ 250 · 0.0965 + 136.66 · 0.0585 = 155.77
24.13 + 7.99

(3,3) = 199.33 · 0.0585 + 134 · 0.0965 + 206.66 · 0.0585 + 136.66 · 0.0965 + 170 · 0.159 + 133 · 0.0965
11.66 + 12.93 + 12.09 + 13.19 + 27.03 + 12.83
+ 250 · 0.0585 + 133.66 · 0.0965 + 200 · 0.0585 = 128.9
14.63 12.9 11.7

(3,4) = 134 · 0.0585 + 206.66 · 0.0965 + 113.33 · 0.0585 + 170 · 0.0965 + 133 · 0.159 + 183.33 · 0.0965
7.84 + 19.94 + 6.63 + 16.41 + 21.15 + 17.69
+ 133.66 · 0.0585 + 200 · 0.0965 + 116.66 · 0.0585 = 123.6
7.82 + 19.3 + 6.82

(3,5) = 206.66 · 0.0585 + 113.33 · 0.0965 + 133 · 0.0965 + 183.33 · 0.159 + 200 · 0.0585 + 116.66 · 0.0965
12.09 + 10.94 + 12.83 + 29.15 + 11.7 + 11.26 = 87.97

(4,1) = 183.33 · 0.0965 + 136.66 · 0.0585 + 116.66 · 0.159 + 250 · 0.0965 + 183.33 · 0.0965 + 183.33 · 0.0585 =
17.69 + 7.82 + 18.55 + 24.13 + 17.89 + 6.92 = 92.8

(4,2) = 183.33 · 0.0585 + 136.66 · 0.0965 + 170 · 0.0585 + 116.66 · 0.0965 + 250 · 0.159 + 133.66 · 0.0965
10.72 + 13.19 + 9.95 + 11.26 + 39.75 + 12.9
+ 183.33 · 0.0585 + 113.33 · 0.0965 + 183.33 · 0.0585 = 130.63
10.72 11.42 10.72

(4,3) = 136.66 · 0.0585 + 170 · 0.0965 + 133 · 0.0585 + 250 · 0.0965 + 133.66 · 0.159 + 200 · 0.0965
7.82 + 16.41 + 7.78 + 24.13 + 21.25 + 19.3
+ 113.33 · 0.0585 + 183.33 · 0.0965 + 113.33 · 0.0585 = 128.22
6.92 17.69 6.92

(4,4) = 170 · 0.0585 + 133 · 0.0965 + 183.33 · 0.0585 + 133.66 · 0.0965 + 200 · 0.159 + 116.66 · 0.0965
9.95 + 12.83 + 10.72 + 13.19 + 31.8 + 11.26
+ 183.33 · 0.0585 + 113.33 · 0.0965 + 183.33 · 0.0585 = 122.61
10.72 11.42 10.72

(4,5) = 133 · 0.0585 + 183.33 · 0.0965 + 200 · 0.0965 + 116.66 · 0.159 + 113.33 · 0.0585 + 183.33 · 0.0965
= 7.78 + 17.69 + 19.3 + 18.55 + 6.92 + 17.69 = 87.93

Kyle Bishop · Quizz - Assignment 4 continued. 002-24-8328

$(5,1) = 116.66 \cdot 0.0965 + 250 \cdot 0.0585 + 183.33 \cdot 0.159 + 116.33 \cdot 0.965 =$
$11.26 + 14.625 + 29.15 + 11.42 = 66.46$

$(5,2) = 116.66 \cdot 0.0585 + 250 \cdot 0.0965 + 133.66 \cdot 0.0585 + 183.33 \cdot 0.0965 + 116.33 \cdot 0.159 + 183.33 \cdot 0.0965 =$
$6.82 + 24.13 + 7.82 + 17.69 + 16.81 + 17.69 = 92.96$

$(5,3) = 250 \cdot 0.0585 + 133.66 \cdot 0.0965 + 200 \cdot 0.0585 + 116.33 \cdot 0.0965 + 183.33 \cdot 0.159 + 116.33 \cdot 0.0965 =$
$14.63 + 12.9 + 11.7 + 11.42 + 29.15 + 11.42 = 91.22$

$(5,4) = 133.66 \cdot 0.0585 + 200 \cdot 0.0965 + 116.66 \cdot 0.0585 + 183.33 \cdot 0.0965 + 116.33 \cdot 0.159 + 183.33 \cdot 0.0965 =$
$7.82 + 19.3 + 6.82 + 17.69 + 16.81 + 17.69 = 88.13$

$(5,5) = 200 \cdot 0.0585 + 116.66 \cdot 0.0965 + 116.33 \cdot 0.0965 + 183.33 \cdot 0.159 =$
$11.7 + 11.26 + 11.42 + 29.15 = 63.53$

Avg w/ kernel applied (3x3; $\sigma = 1$)

58.7	77.8	88.19	88.08	63.12
80.18	117.81	123.91	122.66	87.75
79.72	155.77	128.9	123.6	87.97
92.8	130.63	128.22	122.81	87.93
66.46	92.96	91.22	88.13	63.53

b) The Algorithm will search through the image patch to find 3 corners, And if those corners connect to form a triangle (Edges)

1) corner Detection (Harris corner Detection):

$$x(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Where w is the window patch being tested,
u and v are the coordinates of the patch,
and x and y are the coordinates of the image

simplifies to: $M = \sum w(x,y)\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \Rightarrow E(u,v) = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$

($\downarrow \lambda_1$, $\downarrow \lambda_2$)

$\det M = \lambda_1 \lambda_2$

trace $M = \lambda_1 + \lambda_2$ then;

R (recognition) $\Rightarrow R = \det M - K(\text{trace } M)^2$

K is weighted value from image traversal

This will lead us to a value of R that is either positive (has corners) or negative (Not a corner)

Kyle Bishop Quiz 2 - Assignment 4 Continued 002-24-8328

R = +ve (has corner)
R = -ve (has edges)

If the algorithm never finds a R value that is positive (R=+ve) then there is NO triangle in the image. However if it is found, then we will shift to follow an edge value (R=-ve) until we reach another corner, followed by a 3rd corner, and finally back to the original. If the 3rd edge does not follow back to the original corner, then there is still no triangle.

In order to find an appropriate following corner, there is an additional step needed to calculate an Edge.

2) Canny Edge Detection:

1) the first step is to Apply a Gaussian filter to remove Noise

2) calculate the Angle of the edge heading away from the corner found: $\theta = \tan^{-1}\left(\frac{\frac{dI}{dy}}{\frac{dI}{dx}}\right)$

As well As the gradient: Magnitude $G = \sqrt{\left(\frac{dI}{dx}\right)^2 + \left(\frac{dI}{dy}\right)^2}$

3) Non-max suppression: Check the gradients of neighbouring points to ensure the direction of the Edge.
For point P, P_A & P_B (Neighbours). Array N for storing the values. If $G(P_A) < G(P) < G(P_B)$, then $N(P) = G(P)$
Else, $N(P) = 0$

4) Trace Edge With Hysteresis:
T_{min} and T_{max} (T is time). $N(P)$ must be $\geq T_{max}$
Plot all pixels where $N(P) \geq T_{min}$ on the resulting image to trace the Edge, And repeat when brought to Another corner.

Programming environment:

Kyle Bishop Quiz 3 - Assignment 4 CSCI-24-6328

1. Image filtering → RANGE
   EX: Gaussian Blur
   Image warping → Domain
   EX: Scaling by factor x = factor · u y = factor · v
   or Holistic tracking of an image patch

2. $I_x u + I_y v + I_t = 0$ $u(x,y) = u$ $V(x,y) = x - y$

$I_x u + I_y(x - y) + I_t = 0$

$I_x u + I_y x - I_y y + I_t = 0$

$u = \dfrac{-I_y x + I_y y - I_t}{I_x}$

(u: unknown)

3. $R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sqrt{2}/2 & -\sqrt{2}/2 \\ 0 & \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 \end{bmatrix}$

$\begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1600 & 0 & 960 \\ 0 & -1600 & 540 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ \sqrt{2}/2 & 0 & -\sqrt{2}/2 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

M

$M^{-1} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$