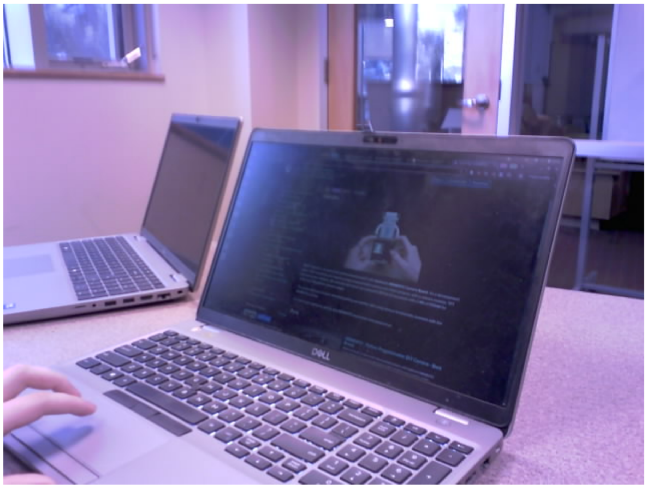
Assignment 5

Miranda Pietraski, Jiayuan Liu, Gia-Uyen Tran

11/08/2024

As always - choose 2 images.

```
clf
im1 = imread('../images/IMAGE001.jpg');
im2 = imread('../images/IMAGE003.jpg');
imshow(im1)
```



```
imshow(im2)
```

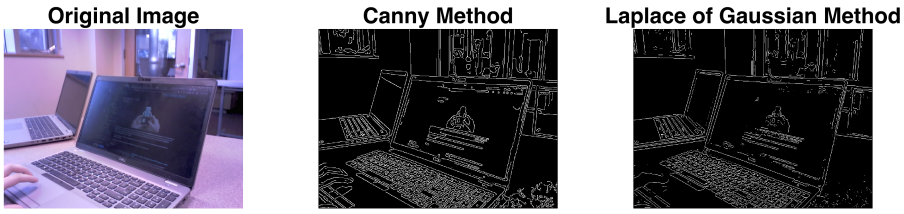
1) Second Order Gradient (total 40 points)

- Apply Laplace of Gaussian (LoG) to your images to find edges (10)
- Apply Canny edge detector to your images to detect edges (10)
- Comment on how well each of them work on your images (20)

Two second-order gradient edge detection methods, Canny and Laplacian of Gaussian (LoG), are applied to two images: one featuring people and another showing a computer screen.

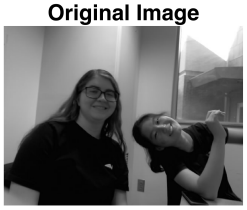
In both cases, the Canny Method results in better detection of weak edges, such as facial features and the keyboard's shadow. This is because the Canny Method uses two thresholds—one for strong edges and another for weak edges—and keeps weak edges that are connected to the strong edges in the final image. In contrast, the LoG Method does not include this double-threshold feature, it detects only zero-crossings of the second-order gradient after filtering the image with a LoG filter. The Canny Method is more effective when preserving detailed outlines from the original image is important (for instance, the smile on Miranda's face, which is only partially detected with the LoG Method) or in cases of low-contrast lighting. However, it is computationally more intensive. Also the inclusion of weaker edges can sometimes distract from the main contours of the image. For example, Jiayuan's arm in the first photo is more distinctly detected using the LoG Method.

```
% Convert to grayscale
I1 = rgb2gray(im1);
I2 = rgb2gray(im2);

% Image 1
C1 = edge(I1,'Canny');
L1 = edge(I1,'log');

figure
subplot(1,3,1)
imshow(im1)
title('Original Image')
subplot(1,3,2)
imshow(C1)
title("Canny Method")
subplot(1,3,3)
imshow(L1)
title("Laplace of Gaussian Method")
```

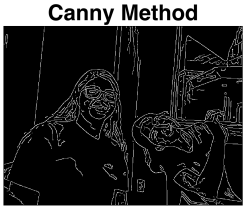



```
% Image 2
C2 = edge(I2,'Canny');
L2 = edge(I2,'log');

figure
subplot(1,3,1)
imshow(im2)
title('Original Image')
subplot(1,3,2)
imshow(C2)
title("Canny Method")
subplot(1,3,3)
imshow(L2)
title("Laplace of Gaussian Method")
```

Original Image



Canny Method

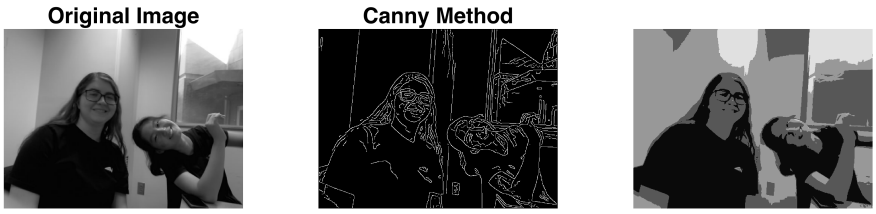


Laplace of Gaussian Method

2) Image Segmentation using clustering (total 30 points)

- Apply k-means algorithm to your images.(10)
- How many clusters seem to work well for your images? (10)
- What parameters did you use? (e.g. R, G, B, H, S, V....). Did these work well? (10)
- Just for your own learning: Can you apply mean-shift clustering to your images? (0 points)

```
[im1new, h1, s1, v1, p1] = kmeansHSV(im1, 4);
scatter3(h1(1:500:end), s1(1:500:end), v1(1:500:end), 6, p1(1:500:end))
xlabel("H")
ylabel("S")
zlabel("V")
imshow(im1new);
[im2new, h2, s2, v2, p2] = kmeansHSV(im2, 4);
scatter3(h2(1:500:end), s2(1:500:end), v2(1:500:end), 6, p2(1:500:end))
xlabel("H")
ylabel("S")
zlabel("V")
imshow(im2new);
```



I used HSV as the dimensions with which to find clusters. Looking at the raw scatterplot, there weren't very clear clusters of points. Using 4 means worked well in both images. In the first, it is easy to identify the computer screens and keyboards. In the second, the darkest color was reserved for the figures, and it was possible to discern facial features.

These images were not the most colorful, particularly the gray filtered one. HSV and RGB clustering would likely perform better with more color to discern objects. In this case, it may have been more helpful to use physical distance as a dimension.

```
function [newimg, h1, s1, v1, p1] = kmeansHSV(img, num)
    hsv1 = rgb2hsv(img);
    h1 = hsv1(:,:,1);
    h1 = reshape(h1.',1,[]);
    s1 = hsv1(:,:,2);
    s1 = reshape(s1.',1,[]);
    v1 = hsv1(:,:,3);
    v1 = reshape(v1.',1,[]);

    mean_idxs = randperm(length(h1), num);
    means = [h1(mean_idxs); s1(mean_idxs); v1(mean_idxs)]';
    p1 = zeros(size(h1));

    for i=1:10
        for j=1:length(p1)
            dists = sqrt((means(:,1) - h1(j)).^2 ...
```

```
                + (means(:,2) - s1(j)).^2 ...
                + (means(:,3) - v1(j)).^2);
            [maxd, idx] = min(dists);
            p1(j) = idx;
        end
        for k=1:length(means)
            group = (p1==k);
            sum_h = sum(group.*h1);
            sum_s = sum(group.*s1);
            sum_v = sum(group.*v1);
            means(k,1) = sum_h/sum(group);
            means(k,2) = sum_s/sum(group);
            means(k,3) = sum_v/sum(group);
        end
    end

    h1new = zeros(size(h1));
    s1new = zeros(size(s1));
    v1new = zeros(size(v1));
    for k=1:length(means)
        group = (p1==k);
        h1new = h1new + group.*means(k,1);
        s1new = s1new + group.*means(k,2);
        v1new = v1new + group.*means(k,3);
    end

    hsv1new = zeros(size(hsv1));
    for i=1:length(hsv1(:,1))
        for j=1:length(hsv1)
            hsv1new(i,j,1) = h1new( (i-1)*length(hsv1)+j );
            hsv1new(i,j,2) = s1new( (i-1)*length(hsv1)+j );
            hsv1new(i,j,3) = v1new( (i-1)*length(hsv1)+j );
        end
    end
    newimg = hsv2rgb(hsv1new);
end
```

3) Corner detection (total 20 points)

- try out the inbuilt corner detection functions in MATLAB and Python on both your images. (10 points)
- Did you need optimize the "number" of corners to be found for better accuracy? Explain. (10 points)

We used the `detectHarrisFeatures` function (computer vision toolbox) to identify corners.

```
% Image 1

% detect corners
points1 = detectHarrisFeatures(rgb2gray(im1));
% plot
```

```
figure;
subplot(1,3,1)
imshow(im1); hold on;
plot(points1);
title('All Corners')
subplot(1,3,2)
imshow(im1); hold on;
plot(points1.selectStrongest(180));
title("Strongest 180 Corners")
subplot(1,3,3)
imshow(im1); hold on;
plot(points1.selectStrongest(300));
title("Strongest 300 Corners")
```

474 corners were identified in image 1. The function mainly identified corners from the laptop keyboards as a well as some corners in the background. The "strongest" 180 corners roughly correspond to the closer laptop's keyboard. The strongest 300 begin to identify the further keyboard, but it also identifies features in the background. Changing the number of corners yielded different results, but their usefulness depends on the application. If our goal was to isolate the closer keyboard's keys, then limiting the corners to about 180 would be useful. Any more than that picks up significant noise in the background and only identifies some keys on the further keyboard.

```
% Image 2
```

```
% detect points
points2 = detectHarrisFeatures(rgb2gray(im2));
% plot
figure;
subplot(1,2,1)
imshow(im2); hold on;
plot(points2);
title('All Corners')
subplot(1,2,2)
imshow(im2); hold on;
plot(points2.selectStrongest(8));
title("Strongest 8 Corners")
```

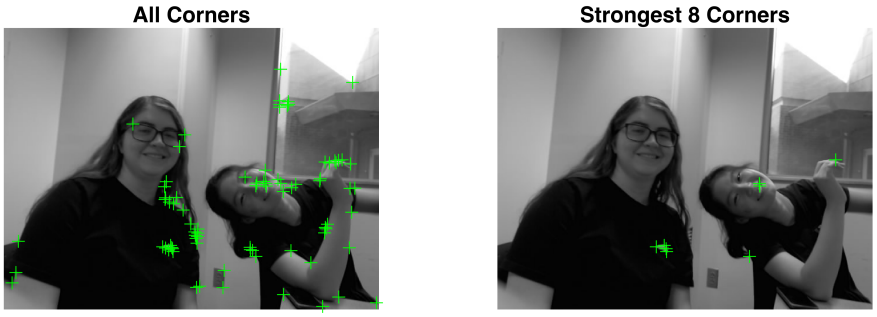
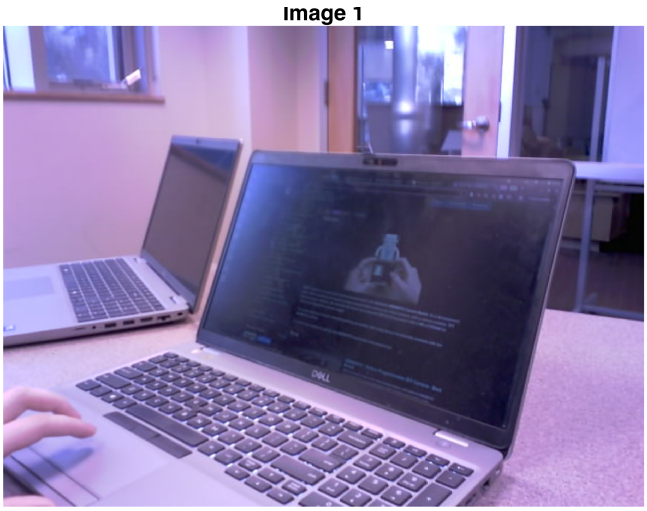


Image 2 was more difficult. Because the image is of Jiayuan and Miranda, there are more organic shapes and thus less clear corners. The function found 80 corners, with clusters around the logo on Miranda's shirt and Jiayuan's face and arms. Unsurprisingly, the most "artificial" shape in the foreground, the logo, had the strongest corners. The next strongest corners are on Jiayuan's face and arms. Adjusting the number of corners would be useful for locating the logo or identifying some anatomical features.

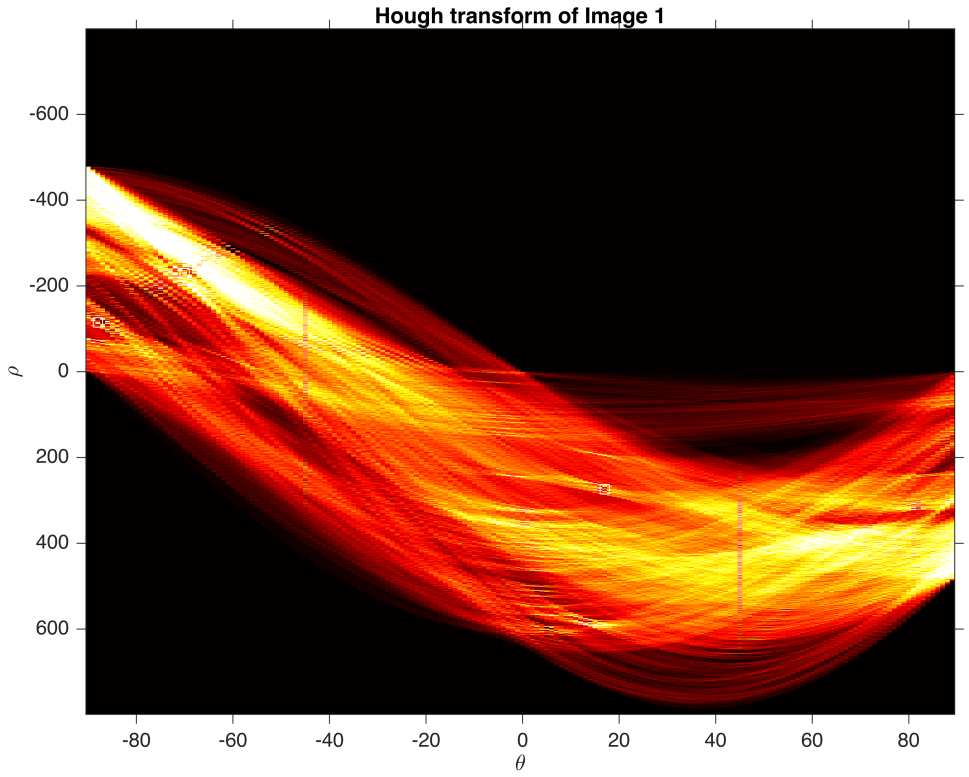
4) Hough transform (total 20 points)

- try out the inbuilt hough transform functions (in MATLAB they are: hough, houghpeaks, houghlines) to detect lines in your image. (20 points)
- What did you need to optimize to find the lines you wanted to find? (10 points)

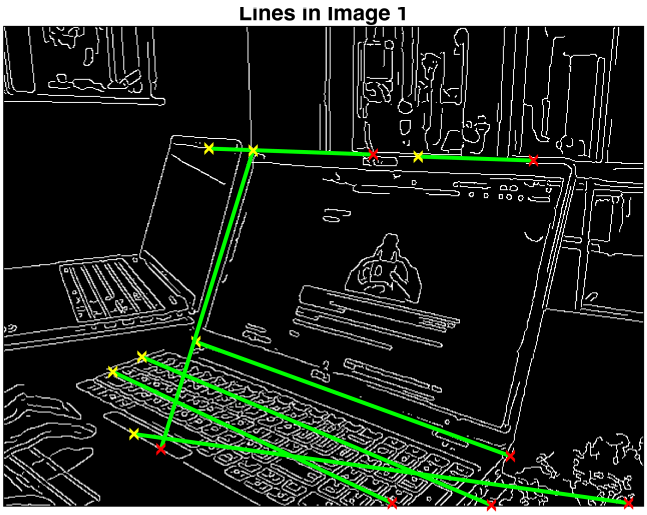
```
% Image 1
bw1 = edge(rgb2gray(im1),'canny');
[H,T,R] = hough(bw1);

figure;
imshow(im1);
title('Image 1');
```




```
figure;
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
title('Hough transform of Image 1');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,hot);

P  = houghpeaks(H,6);
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');
```




```
figure;
lines = houghlines(bw1,T,R,P,'FillGap', 30, 'MinLength',100);
imshow(bw1), hold on
max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;
   end
end
title('Lines in Image 1')
```

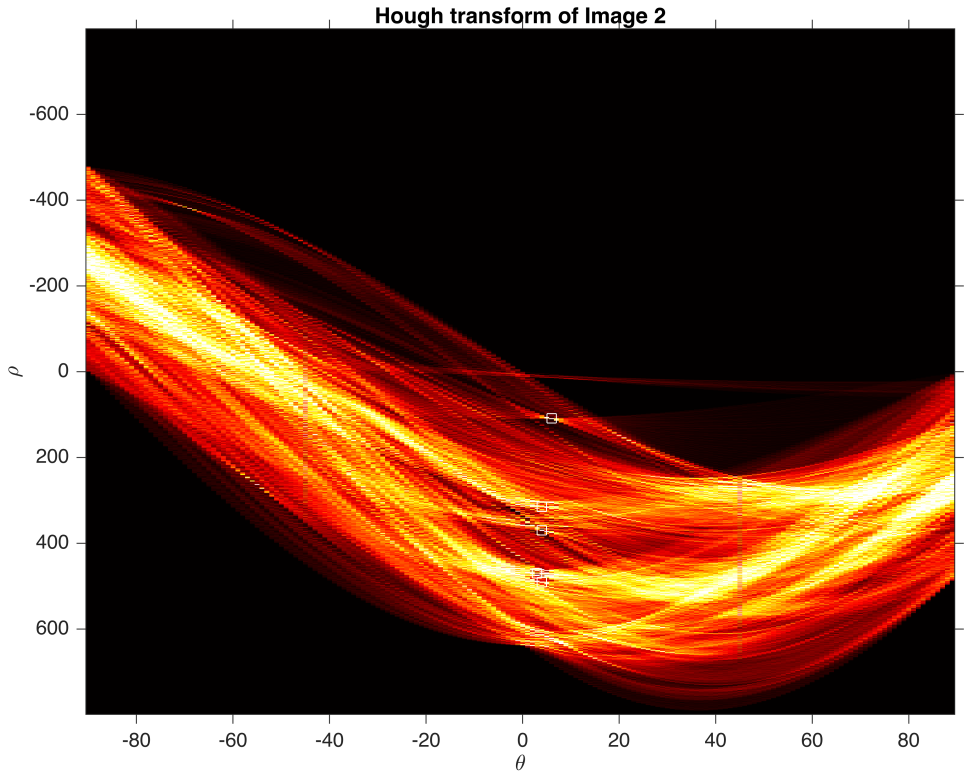


In image 1, we had to set a relatively high minimum length of 100 pixels and extracted 6 peaks. This is because the image contains many smaller lines that were not as important to extract. Additionally, we chose to use the `FillGap` parameter to continue lines if there was less than a 30 pixel gap because there were many "interruptions" to otherwise continuous lines.

```
% Image 2
bw2 = edge(rgb2gray(im2),'canny');
[H2,T2,R2] = hough(bw2);

% Plot image
figure;
imshow(im2);
title('Image 2');
```

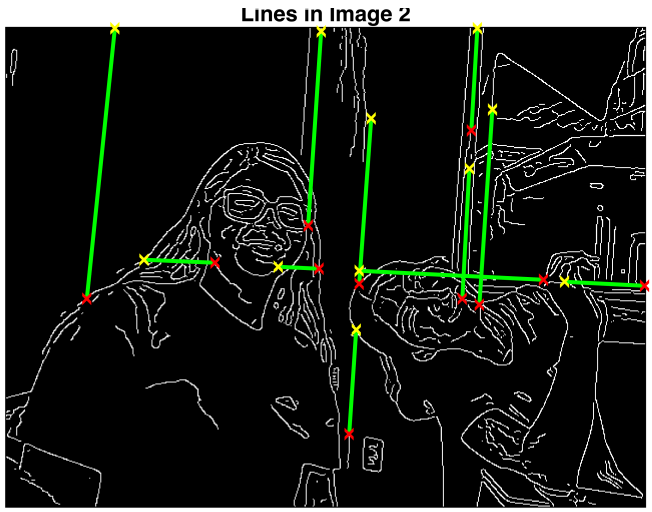

```
% Hough Transform
```

```
figure;
imshow(imadjust(rescale(H2)),'XData',T2,'YData',R2,...
       'InitialMagnification','fit');
title('Hough transform of Image 2');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,hot);

% Hough Peaks
P2  = houghpeaks(H2,6);
x2 = T2(P2(:,2)); y2 = R2(P2(:,1));
plot(x2,y2,'s','color','white');
```




```
% Identify Lines
figure;
lines = houghlines(bw2,T2,R2,P2);
imshow(bw2), hold on
max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

   % Plot beginnings and ends of lines
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
```

```
    % Determine the endpoints of the longest line segment
    len = norm(lines(k).point1 - lines(k).point2);
    if ( len > max_len)
       max_len = len;
       xy_long = xy;
    end
end
title('Lines in Image 2')
```



We did not have to place as many constraints on image 2. We simply set the number of peaks to 6, which yielded decent results. Any higher than 6 peaks identified extra lines from the window view, which were not useful for this prupose.

5) Applying various concepts together (40 points)

We have learned many concepts in image processing now. Use a combination of those concepts and methods to find letters in scrabble.jpg. (hint: it is a blurred image, letters have lines, corners, edges, the board has different colors on it).

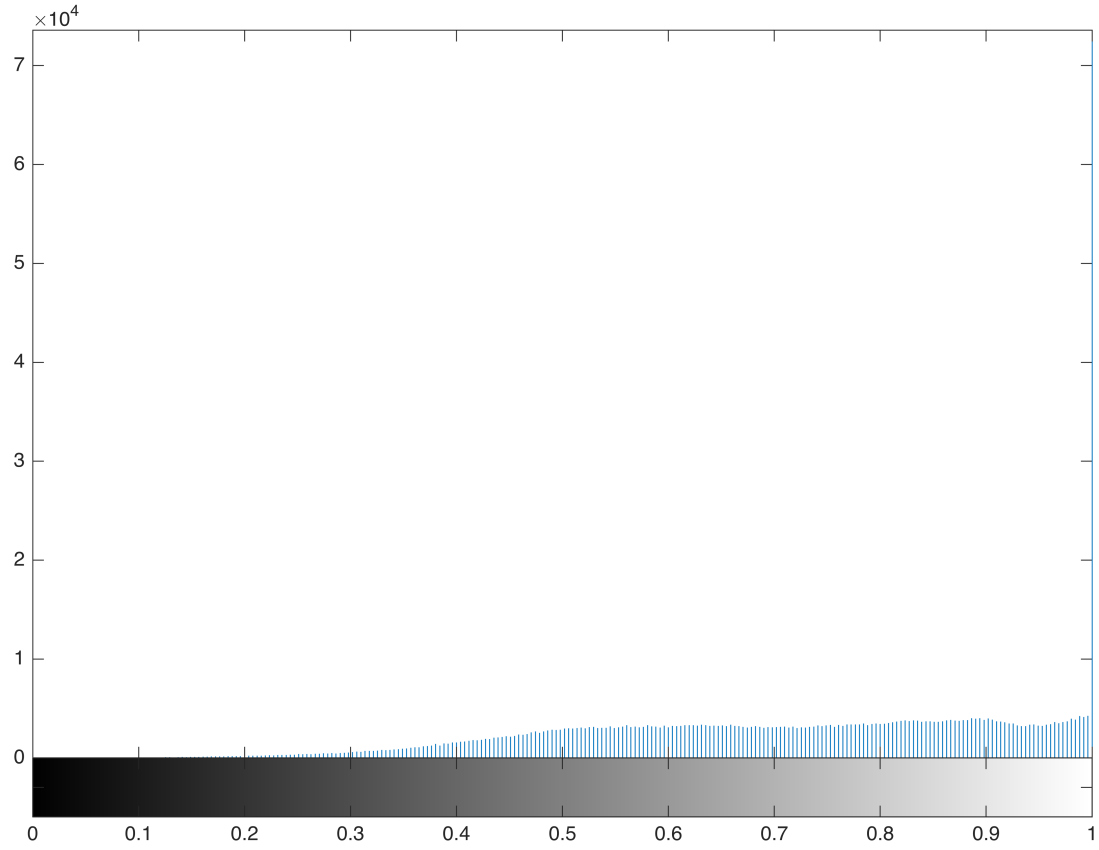
To identify the locations of the letters, we started with sharpening the image and converting into pure black and white.

```
I3 = imread("scrabble.jpg");
figure
imshow(I3)
title("Original Image")
```


```
% sharpening
k_size = 55;
k_center = floor(k_size/2) +1;
S1 = (-1/(k_size^2)) * ones(k_size);
S1(k_center,k_center) = 2;
I3_sharp = imfilter(I3, S1, 'conv');

figure;
imshow(I3_sharp)
title('Sharpened Image')
```

Sharpened Image

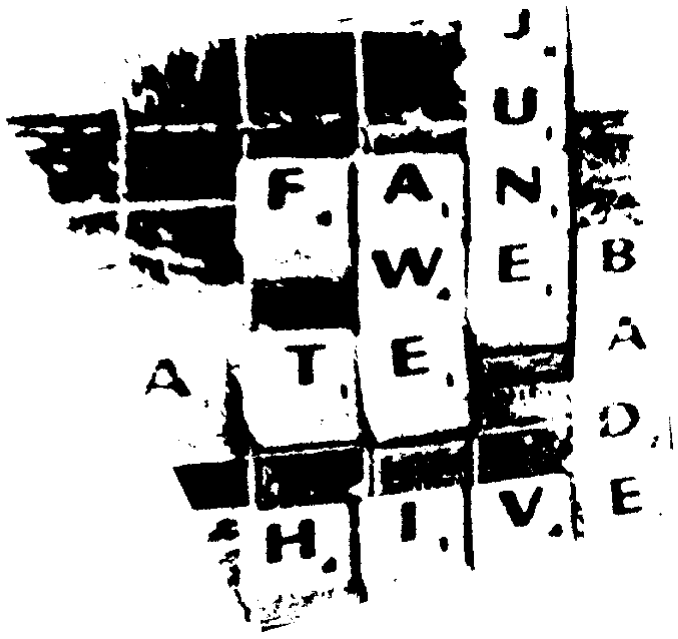

```
I3_sharp_copy = I3_sharp;

% hsv sparating
[h,s,v] = rgb2hsv(I3_sharp);
gray_I3 = rgb2gray(I3_sharp);
I3_sharp(:,:,2) = 0;
I3_sharp(:,:,3) = 0;


% histogram threshold
imhist(v)
```



Our plan to numerically find the letters included convolving with a mostly white frame approximately the size of a square tile. To be able to use a consistent square, we transformed the image to adjust for the slanted perspective.

```
threshold = 0.8;
bw_v = imbinarize(v, threshold);
a = 0.3;
b = -0.1;
T = affinetform2d([1 a 0; b 1 0; 0 0 1] );
R = makeresampler({'cubic','nearest'},'fill');
% B = imtransform(bw_v,T,R,'FillValues',[0,0,0]);
B = imwarp(bw_v,T, "FillValues",1);

% 824,222 622,217 576,581 808 566
col = [824 622, 576, 808]';
row = [222 217 581 566]';
base = [100 0; 0 0; 0 250; 100 230];

tf = fitgeotform2d([col row],base, "projective");

B = imwarp(B,tf, "FillValues",1);
```

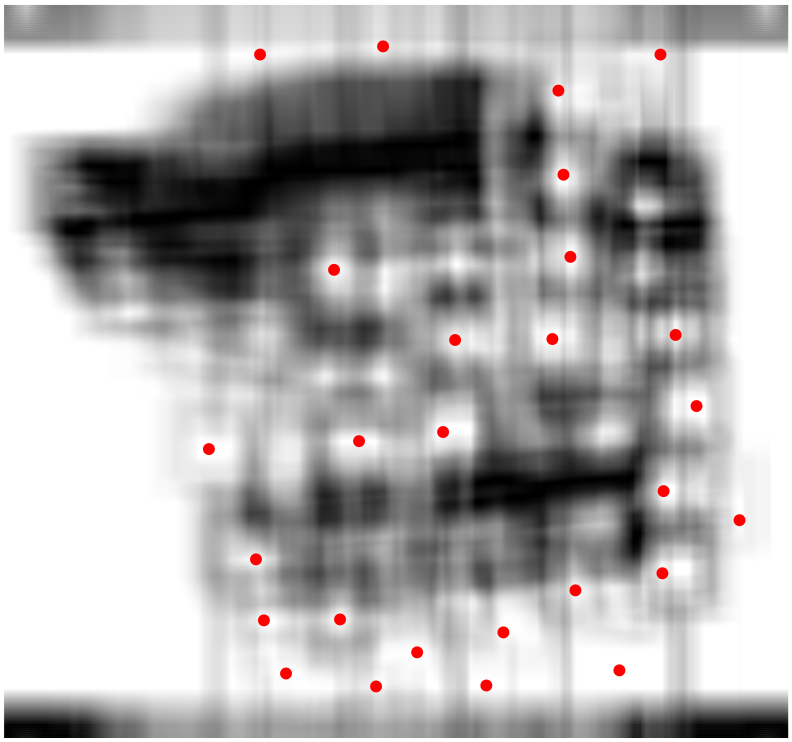
```
B = padarray(B,20,1);

clf
imshow(B);
```



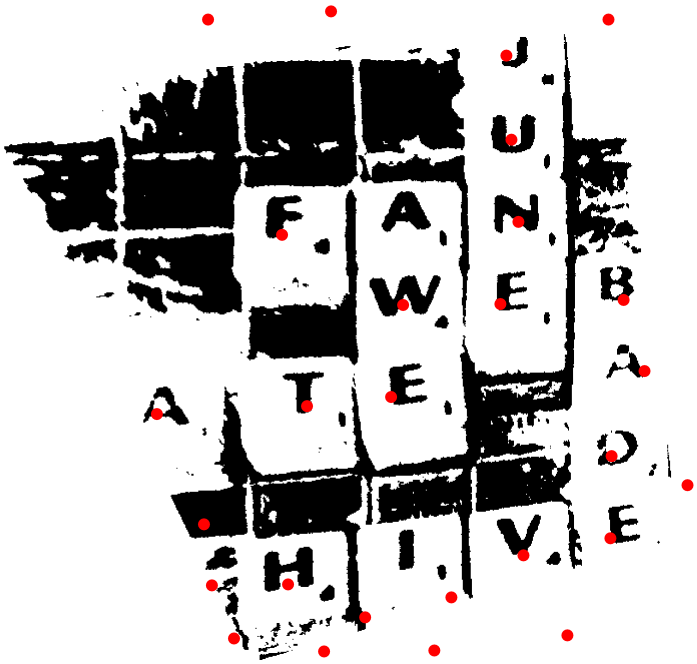
By adding a dark center to the square approximately the size of a letter, the frame would align best when centered on a letter, producing a high value compared to the surroundings. Identifying local maxima in the image created by the convolution was able to pinpoint some of the letters.

```
% 447 164 349 231
% 314 601 359 556
whitesz = 99;
frame = ones(whitesz);
blacksz = 45;
margins = floor((whitesz - blacksz)/2);
frame(margins+11:2:(whitesz+11-margins),margins+1:(whitesz-margins)) = 0;
clf

% frame(end-10:end,end-10:end) = zeros(11);

B_scale = double(B);
B_scale(B_scale < 0.5) = -1;
% valshift = 0.01;
% frame_scale = (frame-valshift)./max(frame-valshift);
frame_scale = frame;
frame_scale(frame_scale == 0) = -3;
```

```
frame_scale(frame_scale == 1) = 1;

C = conv2(B_scale,frame_scale,"same");
Cnorm = (C-min(C))./max(C-min(C));
clf
% mesh(C)

maxpoints = islocalmax2(Cnorm, "MinSeparation",50, "MinProminence",0.95,
"MaxNumExtrema",30);
[r,c]=find(maxpoints==1);
clf
imshow(Cnorm);
hold on
scatter(c,r, "red", "filled")
```



```
clf
imshow(B)
hold on
scatter(c,r, "red", "filled")
```



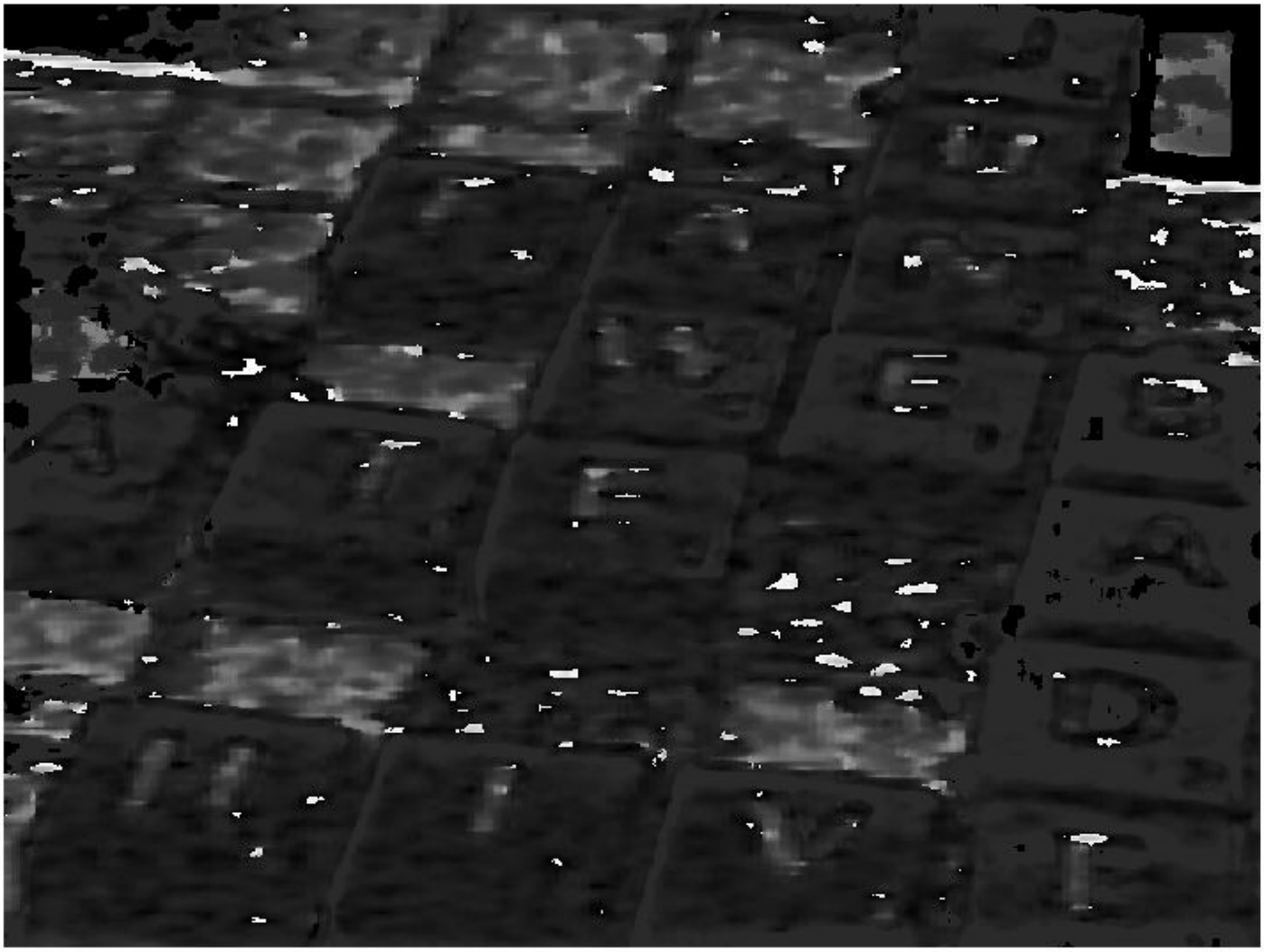
Ultimately, the accuracy was reasonable but not perfect. We were able to locate 15 out of 17 of the letters in the image this way, though there were many false positives particularly where zero padding made an effect and at the cut off edges of the original image. It may be possible to further refine this method by filtering results by the slope and width of identified maxima to select narrower peaks.

Alternatively, we found the bwareafilt() function extremely helpful in removing the noisy background of the image. Using this function eliminated the need for the frame and convolution to isolate letters and yeilded higher accuracy.

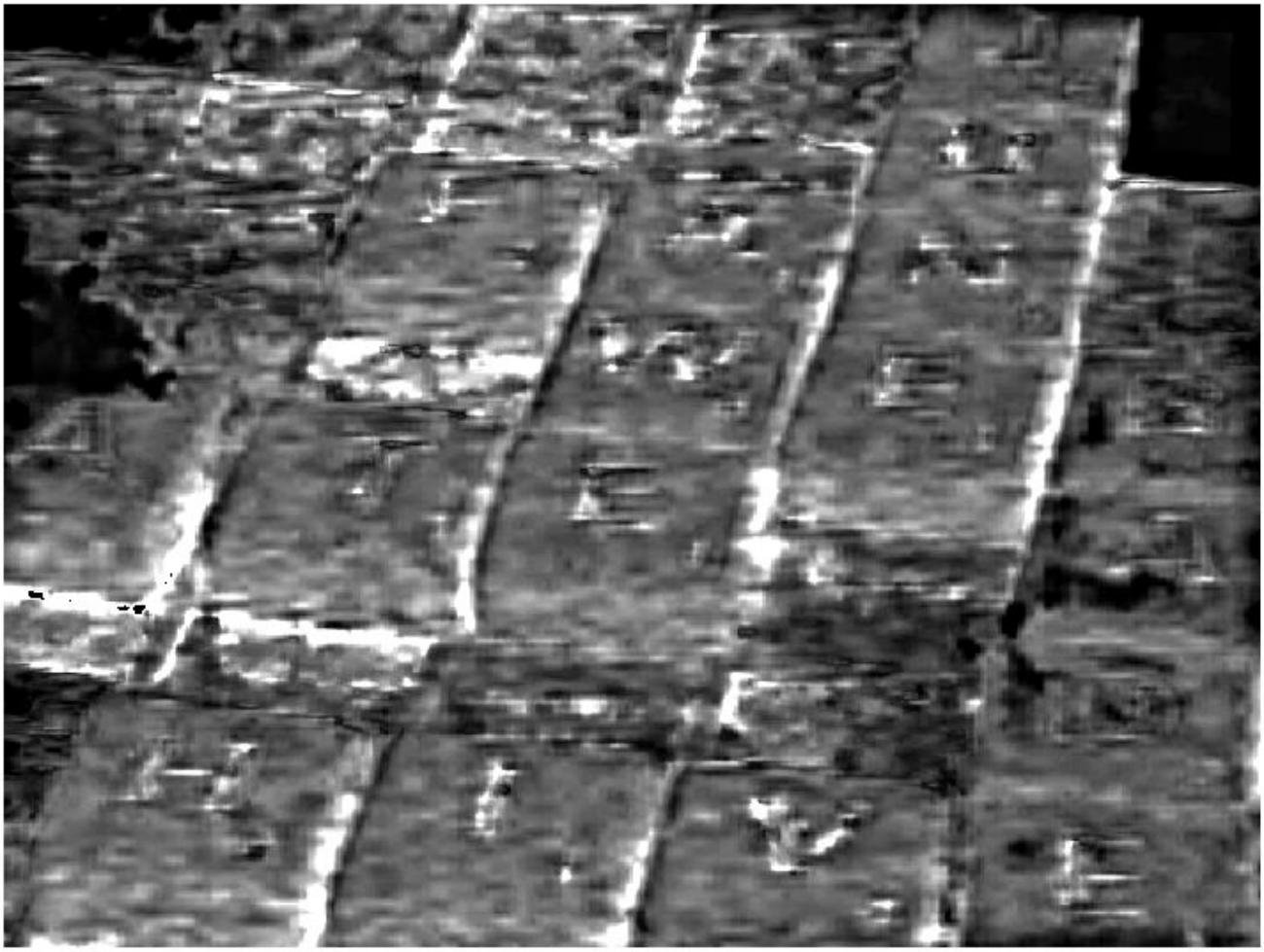
```
I3 = imread("scrabble.jpg");
figure
imshow(I3)
title("Original Image")
```


```
% sharpening
k_size = 55;
k_center = floor(k_size/2) +1;
S1 = (-1/(k_size^2)) * ones(k_size);
S1(k_center,k_center) = 2;
I3_sharp = imfilter(I3, S1, 'conv');
imshow(I3_sharp)
```


```
I3_sharp_copy = I3_sharp;

% hsv sparating
[h,s,v] = rgb2hsv(I3_sharp);
gray_I3 = rgb2gray(I3_sharp);
I3_sharp(:,:,2) = 0;
I3_sharp(:,:,3) = 0;
imshow(h)
```



```
imshow(s)
```

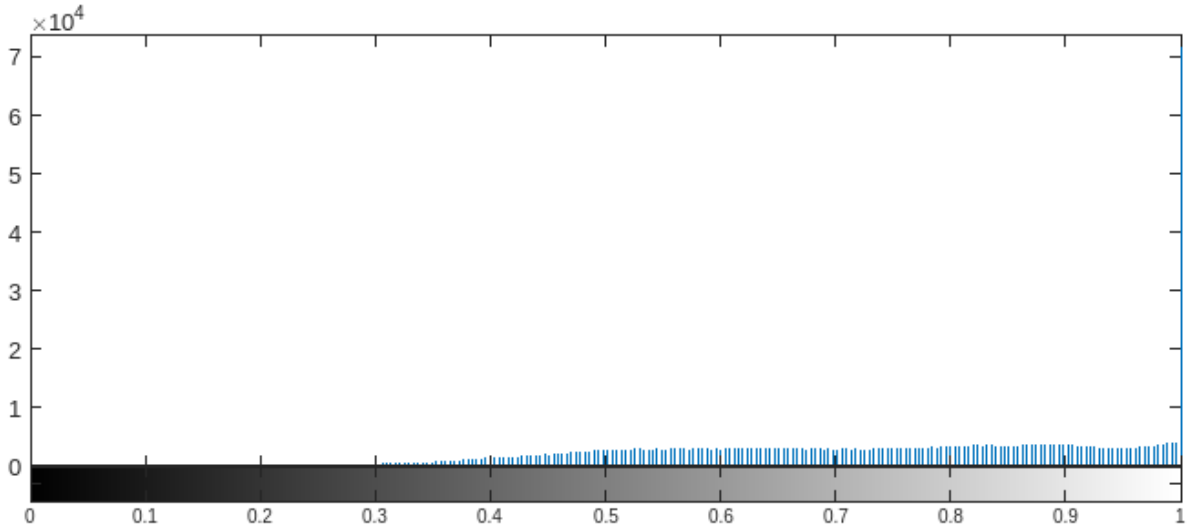


```
imshow(v)
```


```
imshowpair(gray_I3, v,"montage")
```


```
% histogram threshold
```

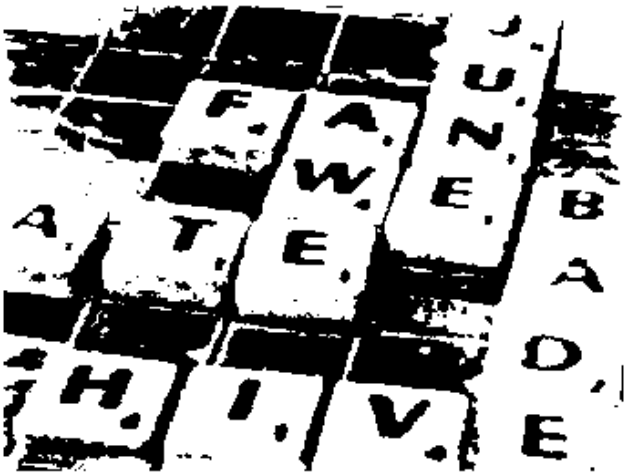
```
imhist(v)
```




```
threshold = 0.85
```

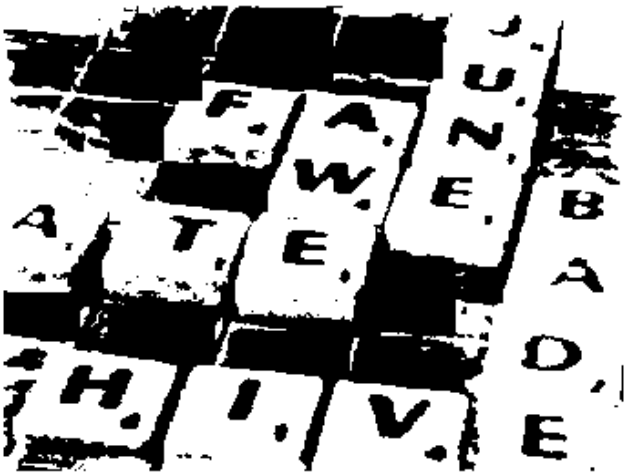
```
threshold = 0.8500
```

```
bw_v = imbinarize(v, threshold);
imshow(bw_v)
```

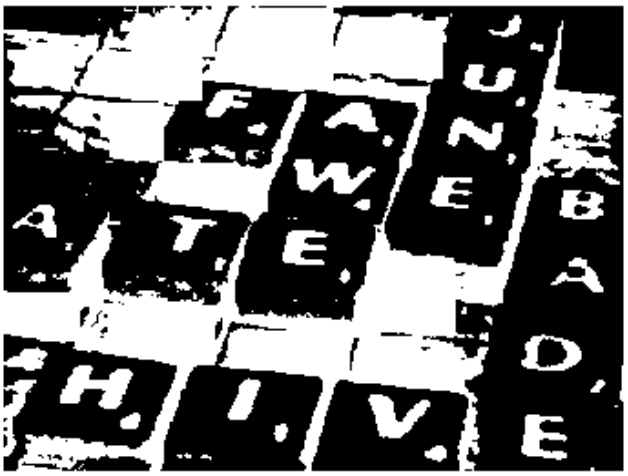


```
% remove background
dim = 50;
[w, l] = size(bw_v);
for i = 1 : w - dim
    for j = 1 : l - dim
        selected = bw_v(i : i + dim, j : j + dim);

        sum_pix = sum(selected, "all");
```

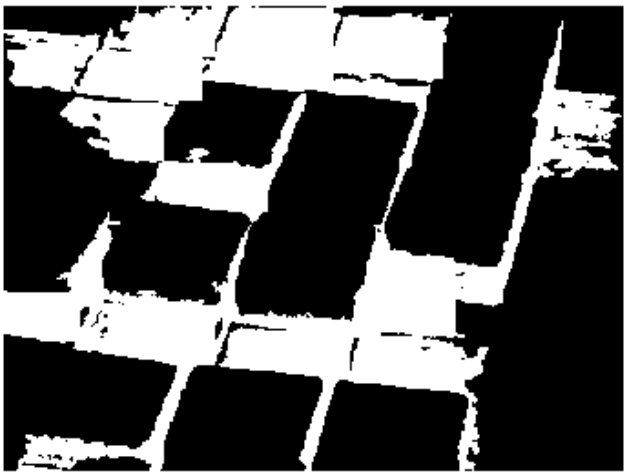
```
        if (sum_pix < 10)
            bw_v(i : i + dim, j : j + dim) = 0;
        end
    end
end
imshow(bw_v)
```



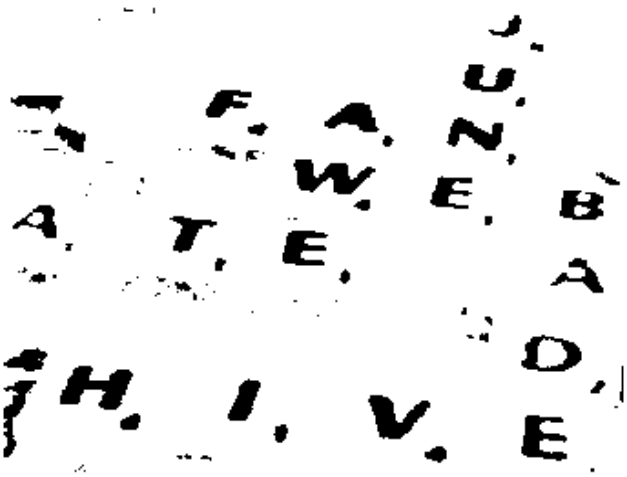

```
% binary extraction
bw_v_n = ~bw_v;
imshow(bw_v_n)
```




```
BW2 = bwareafilt(bw_v_n,2);
imshow(BW2)
```

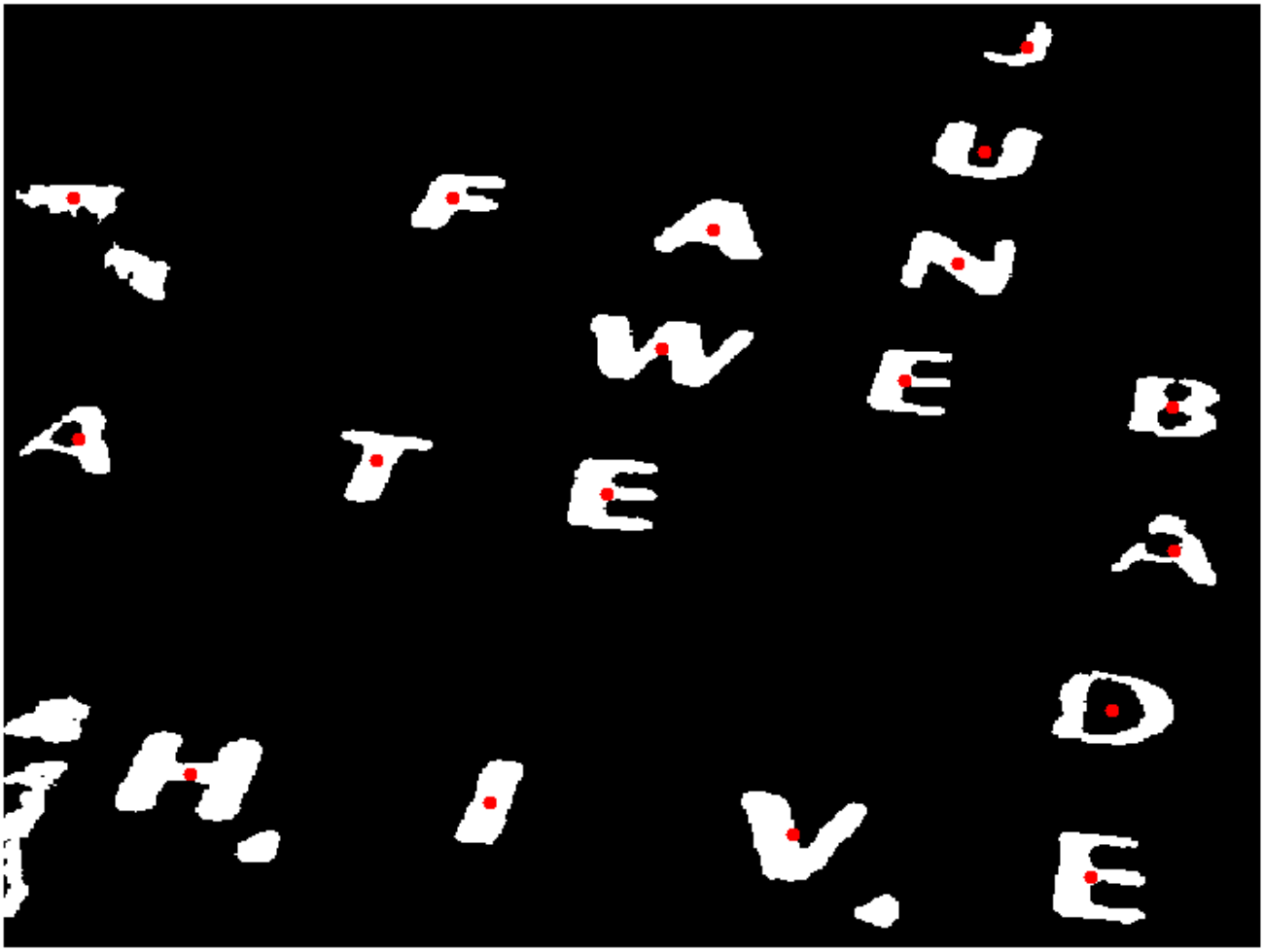


```
x = bw_v_n - BW2;
imshow(~x)
```



```
x = imbinarize(x);
BW_x = bwareafilt(x,23);
imshow(BW_x)
```

```
maxpoints = islocalmax2(BW_x, "MinSeparation",50, "MinProminence",0.95,
"MaxNumExtrema",30);
[r,c]=find(maxpoints==1);
clf
imshow(BW_x);
hold on
scatter(c,r, "red", "filled")
```




```
clf
```