Group Assignment 5

November 8, 2024

Mark Belanger, Carter Harris, Meagan Lipsman

As always - choose 2 images.

```
lighthouse = imread("lighthouse.jpg");
imshow(lighthouse);
title("Lighthouse")
```

Lighthouse

```
meagan = imread("meagan.jpg");
imshow(meagan);
```

```
title("Meagan (and Allan with consent)")
```

1) Second Order Gradient (total 40 points)

Apply Laplace of Gaussian (LoG) to your images to find edges (10)

```
sigma = 3;

lighthouse_guass = imgaussfilt(rgb2gray(lighthouse), sigma);
log = fspecial("laplacian");

lighthouse_conv = conv2(lighthouse_guass, log);

imshow(lighthouse_conv)

title("LoG Lighthouse")
```

LoG Lighthouse

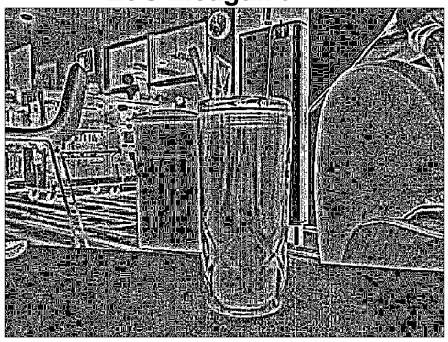
```
meagan_guass = imgaussfilt(rgb2gray(meagan), sigma);

meagan_guass2 = imgaussfilt(rgb2gray(meagan));

meagan_conv = conv2(meagan_guass, log);
meagan_conv2 = conv2(meagan_guass2, log);


imshow(meagan_conv)

title("LoG Meagan")
```

LoG Meagan

```
subplot(1,2,1)
imshow(meagan_conv)
title("LoG Meagan \sigma = 3")
subplot(1,2,2)
imshow(meagan_conv2)
title("LoG Meagan \sigma = 1")
```

LoG Meagan σ = 3

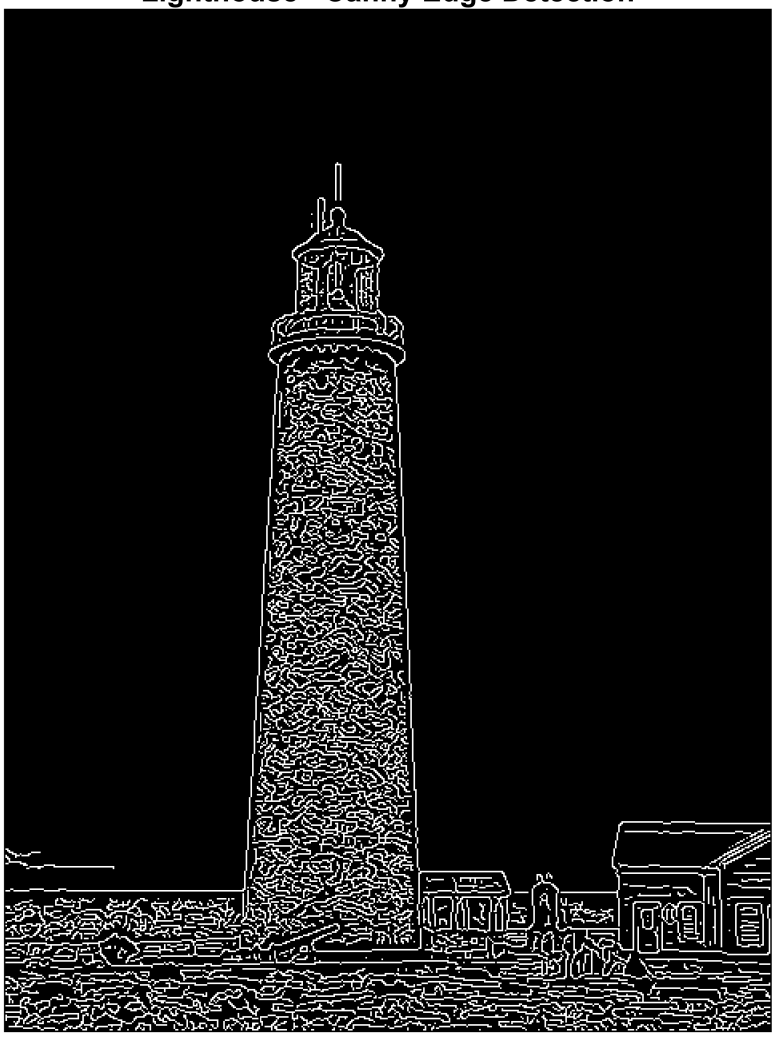
LoG Meagan σ = 1



Apply Canny edge detector to your images to detect edges (10)

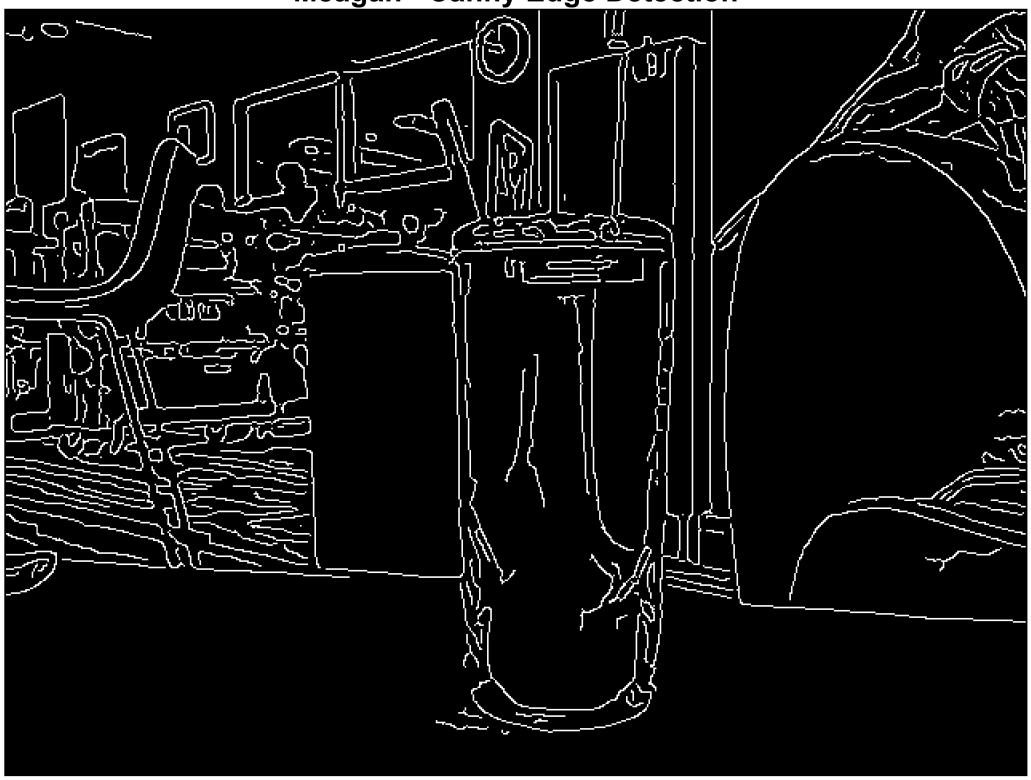
```
canny_lighthouse = edge(rgb2gray(lighthouse), "canny");
figure
imshow(canny_lighthouse)
title("Lighthouse - Canny Edge Detection")
```

Lighthouse - Canny Edge Detection



```
canny_meagan = edge(rgb2gray(meagan), "canny");
figure
imshow(canny_meagan)
title("Meagan - Canny Edge Detection")
```

Meagan - Canny Edge Detection



Comment on how well each of them work on your images (20)

The LoG edge detection works however there is a lot of noise for both output images. The lighthouse output shows artifacts in the sky that aren't apparent in the original image. The overall shape of the lighthouse is distinguishable though. The image of Meagan has even more noise. Her outline is apparent but it is surrouned by enough noise that its hard to interpret what is in the background.

2) Image Segmentation using clustering (total 30 points)

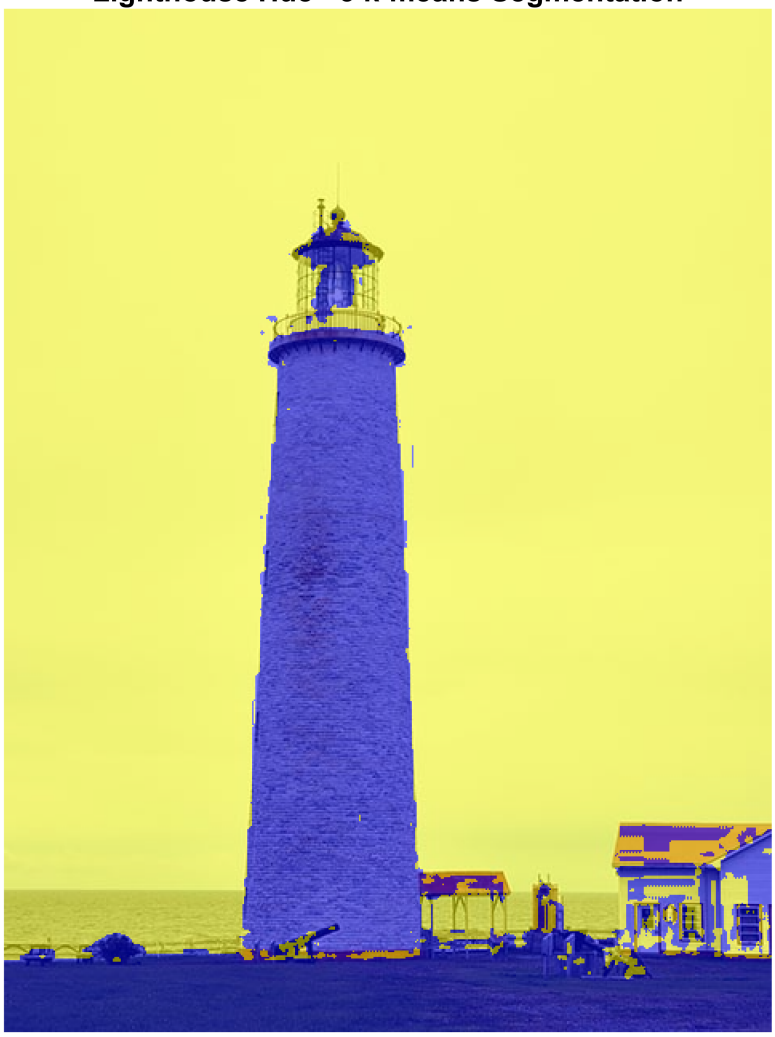
Apply k-means algorithm to your images.(10)

```
means = 3;
kmean_lighthouse = imsegkmeans(lighthouse, means);
kmean_lighthouse_overlay = labeloverlay(lighthouse, kmean_lighthouse);
figure
imshow(kmean_lighthouse_overlay)
title("Lighthouse - 3 k-means Segmentation")
```

Lighthouse - 3 k-means Segmentation

```
lighthouse_hsv = uint8(rgb2hsv(lighthouse));
kmean_lighthouse = imsegkmeans(lighthouse_hsv(:, :, 1), means);
kmean_lighthouse_overlay = labeloverlay(lighthouse, kmean_lighthouse);
figure
imshow(kmean_lighthouse_overlay)
title("Lighthouse Hue - 3 k-means Segmentation")
```

Lighthouse Hue - 3 k-means Segmentation



```
lighthouse_hsv = uint8(rgb2hsv(lighthouse));
kmean_lighthouse = imsegkmeans(lighthouse_hsv(:, :, 2), means);
kmean_lighthouse_overlay = labeloverlay(lighthouse, kmean_lighthouse);
figure
imshow(kmean_lighthouse_overlay)
title("Lighthouse Saturation - 3 k-means Segmentation")
```

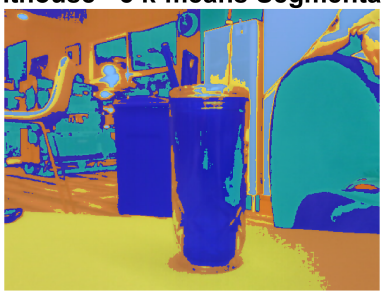
Lighthouse Saturation - 3 k-means Segmentation

```
lighthouse_hsv = uint8(rgb2hsv(lighthouse));
kmean_lighthouse = imsegkmeans(lighthouse_hsv(:, :, 3), means);
kmean_lighthouse_overlay = labeloverlay(lighthouse, kmean_lighthouse);
figure
imshow(kmean_lighthouse_overlay)
title("Lighthouse Value - 3 k-means Segmentation")
```


```
figure
kmean_meagan = imsegkmeans(meagan, 3);
kmean_meagan_overlay = labeloverlay(meagan, kmean_meagan);
subplot(1,2,1)
imshow(kmean_meagan_overlay)
title("Lighthouse - 3 k-means Segmentation")
kmean_meagan = imsegkmeans(meagan, 5);
kmean_meagan_overlay = labeloverlay(meagan, kmean_meagan);
subplot(1,2,2)
imshow(kmean_meagan_overlay)
title("Lighthouse - 5 k-means Segmentation")
```

Lighthouse - 3 k-means Segmentation

Lighthouse - 5 k-means Segmentation



How many clusters seem to work well for your images? (10)

For the lighthouse, 3 k-means worked well is highlighting the lighthouse, sky, house, and ground. Beyond 3 k-means, the lighhouse starts to get mistaken for other parts of the image

For the image of Meagan, we tried with values ranging from 1 to 10 and found that less is more. 3 k-means was sufficient and showing the important parts of the image and adding more means did not highlight any additional useful data.

What parameters did you use? (e.g. R, G, B, H, S, V....). Did these work well? (10)

We tried using images in RBG and HSV spaces for the lighthouse image. Overall, the RGB image came out the best. The image is still visable when processed in the HSV space however the color differences aren't as contrasted as with the RBG space.

3) Corner detection (total 20 points)

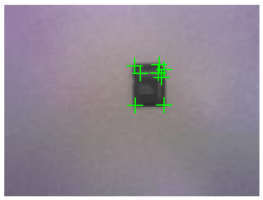
Try out the inbuilt corner detection functions in MATLAB and Python on both your images. (10 points)

```
arcaid = imread("arcaid.jpg");
figure
    corner_arcaid = detectHarrisFeatures(rgb2gray(arcaid));
imshow(arcaid); hold on
plot(corner_arcaid); hold off
title("Corners on the ARCAid Sign")
```

Corners on the ARCAid Sign


```
sd_card = imread("sd card.jpg");
figure
    corner_sd_card = detectHarrisFeatures(rgb2gray(sd_card));
imshow(sd_card); hold on
plot(corner_sd_card); hold off
title("Corners on the SD Card")
```

Corners on the SD Card



Did you need optimize the "number" of corners to be found for better accuracy? Explain. (10 points)

We tried and the function behaved unexpectedly at first however upon modifying the filter size, we were able to highlight the corners on the sd cars. Using a larger filter size decreased the number of found corners.

Decreasing the value of the minQuality input increases the number of corners. This value is a threshold that determines if a corner is present.

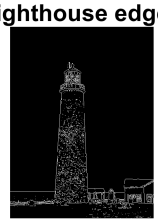
We tried the function on the image of the lighhouse and the ARCAid sign and the function struggled to find corners. In the lighthouse image, corners on the house were detected well but not on the lighthouse itself. For the ARCAid image, the lights were highlighted as being corners along with the wording on the posters but the corners on the ARCAid sign itself were not found well.

4) Hough transform (total 20 points)

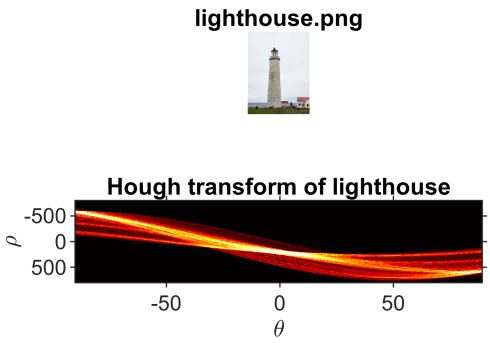
Try out the inbuilt hough transform functions (in MATLAB they are: hough, houghpeaks, houghlines) to detect lines in your image. (20 points)

```
lighthouse_edge = edge(rgb2gray(lighthouse), "sobel");
imshow(lighthouse_edge)
title("Lighthouse edges")
```

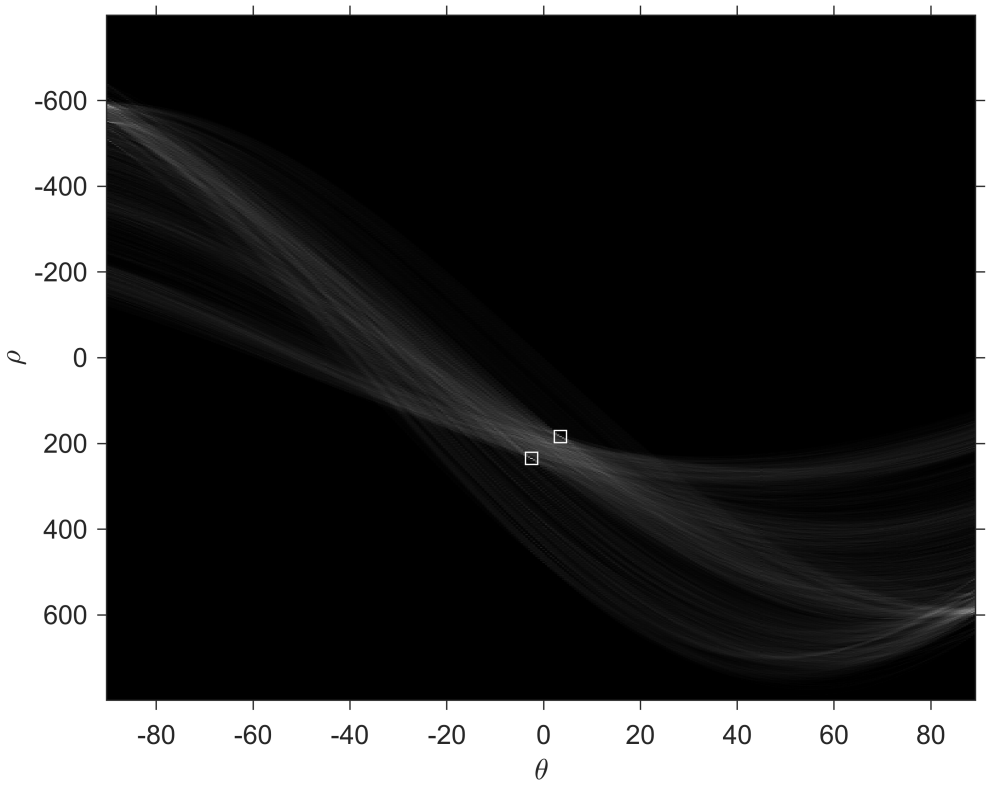
Lighthouse edges



```
[H,T,R] = hough(lighthouse_edge,'RhoResolution',1,'Theta',-90:0.5:89);

subplot(2,1,1);
imshow(lighthouse);
title('lighthouse.png');
subplot(2,1,2);
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
title('Hough transform of lighthouse');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,hot);
```




```
figure
P  = houghpeaks(H,2);
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color','white');
```




```
lines = houghlines(lighthouse_edge,T,R,P,'FillGap',10,'MinLength',1);
figure, imshow(lighthouse), 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("Lighthouse - Hough")
```

What did you need to optimize to find the lines you wanted to find? (10 points)

We started with the sample code given in the documentation for the hough functions and input our images. The starting code did not do a very good job at detecting lines and edges in the lighthouse image so we tried using the sd card image since it had better defined edges. This still did not work well and the code would only output horizontal lines. The starter code used canny edge detection which did not detect edges well in our images. We changed the edge detection to use sobel detection and this did far better at detecting edges and the final output from the hough functions better outlined the lines. We had to modify the RhoResolution input of the hough function for both sides of the lighthouse to be found.

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 the given image. (hint: it is a blurred image, letters have lines, corners, edges, the board has different colors on it).

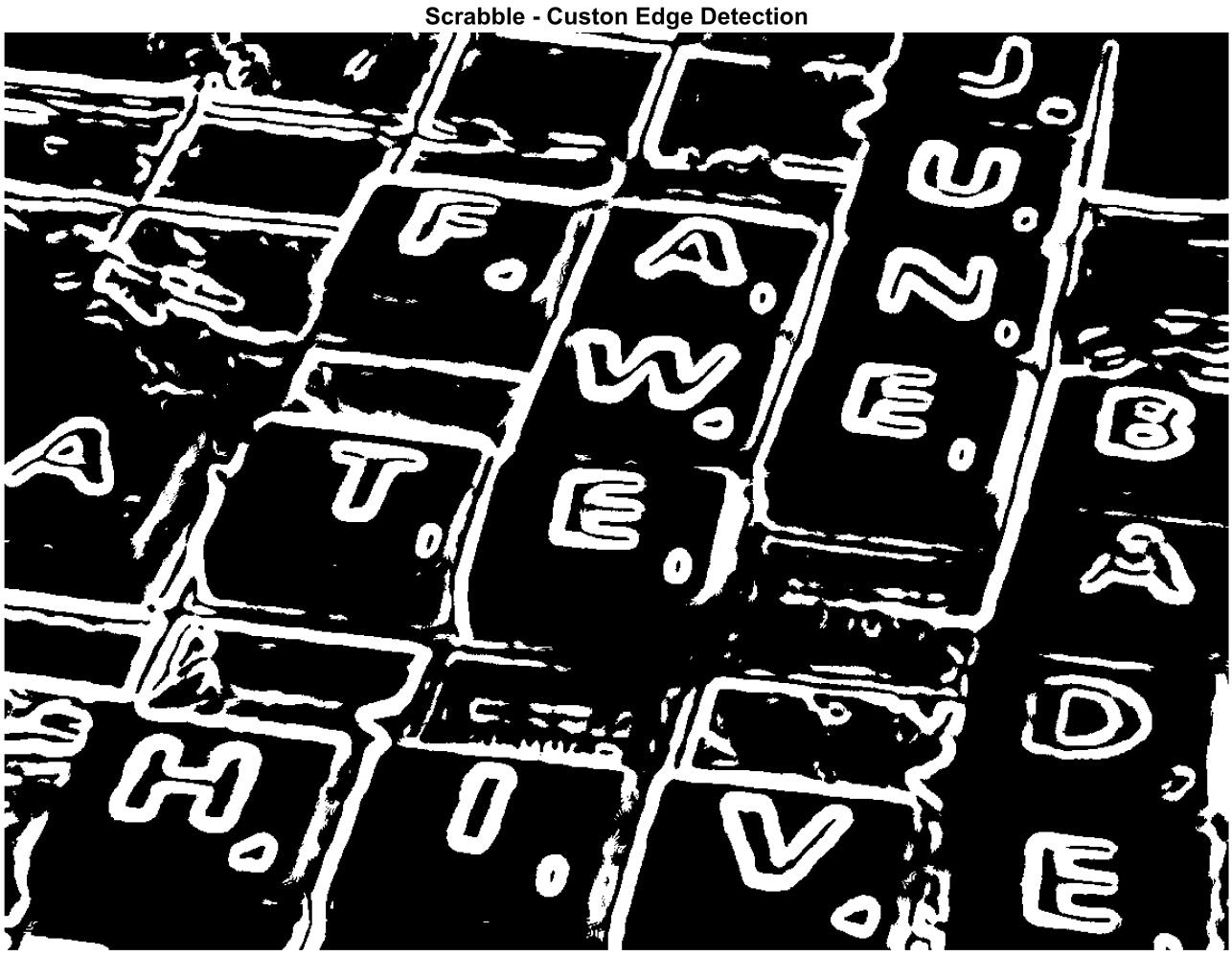
```
scrabble = imread("scrabble.jpg");
figure
imshow(rgb2gray(scrabble))
title("Scrabble")
```

```
%scrabble = rgb2gray(scrabble);

greeen_threshold = 105;

threshold = 130;

% for i = 1:size(scrabble, 1)
%     for j = 1:1:size(scrabble, 2)
%         if scrabble(i, j, 2) > greeen_threshold && ...
%             scrabble(i, j, 1) > threshold && ...
%             scrabble(i, j, 3) > threshold
%
%                 scrabble(i, j, 2) = threshold;
%         end
%     end
% end

% % Create logical masks for the conditions
% mask = (scrabble(:, :, 2) > greeen_threshold) & ...
%        (scrabble(:, :, 1) < threshold) & ...
%        (scrabble(:, :, 3) < threshold);
%
% % Apply the condition to the array
% scrabble(repmat(mask, [1, 1, 3])) = threshold;
%
%
%
% scrabble(scrabble > greeen_threshold) = mean(scrabble, "all");

% imshow(scrabble)
%
scrabble_gauss = imgaussfilt(rgb2gray(scrabble), 4);

imshow(scrabble_gauss)
title("Scrabble - Gauss Filtered")
```

Scrabble - Gauss Filtered

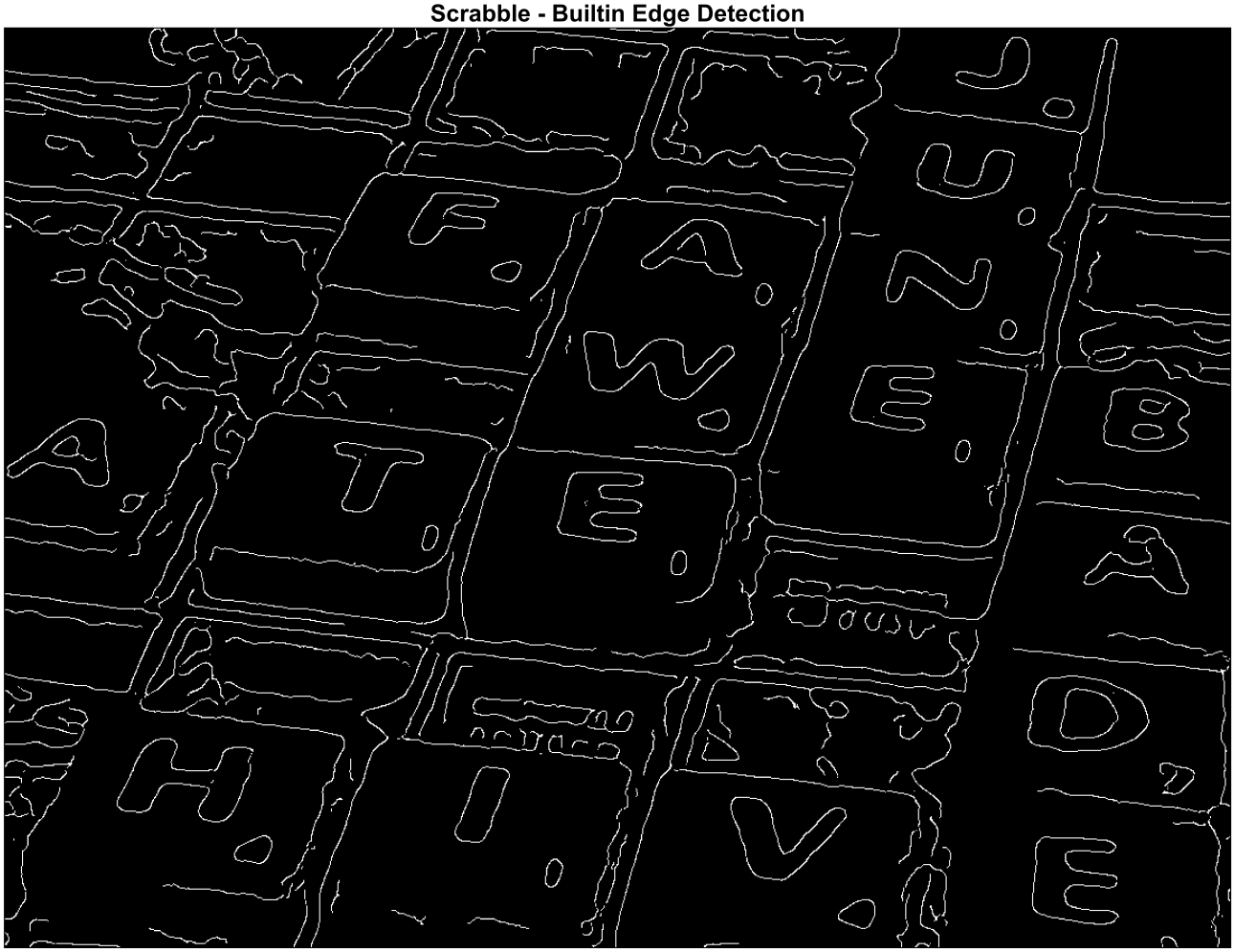

```
sobel_x = [
    -1, 0, 1;
    -2, 0, 2;
    -1, 0, 1;
];

sobel_y = sobel_x';

scrabble_sobel_x = conv2(scrabble_gauss, sobel_x);
scrabble_sobel_y = conv2(scrabble_gauss, sobel_y);

scrabble_sobel = sqrt(scrabble_sobel_x.^2 + scrabble_sobel_y.^2);

% The range of a double is quite large and some values are near 0
% but are not so this assumes anything less than the mean is a 0
% and everything greater than is a 1. This converts the image into
% binary and makes the edges more apparent.
scrabble_sobel = scrabble_sobel > mean(scrabble_sobel, "all");
```

```
figure
imshow(scrabble_sobel)
title("Scrabble - Custon Edge Detection")
```



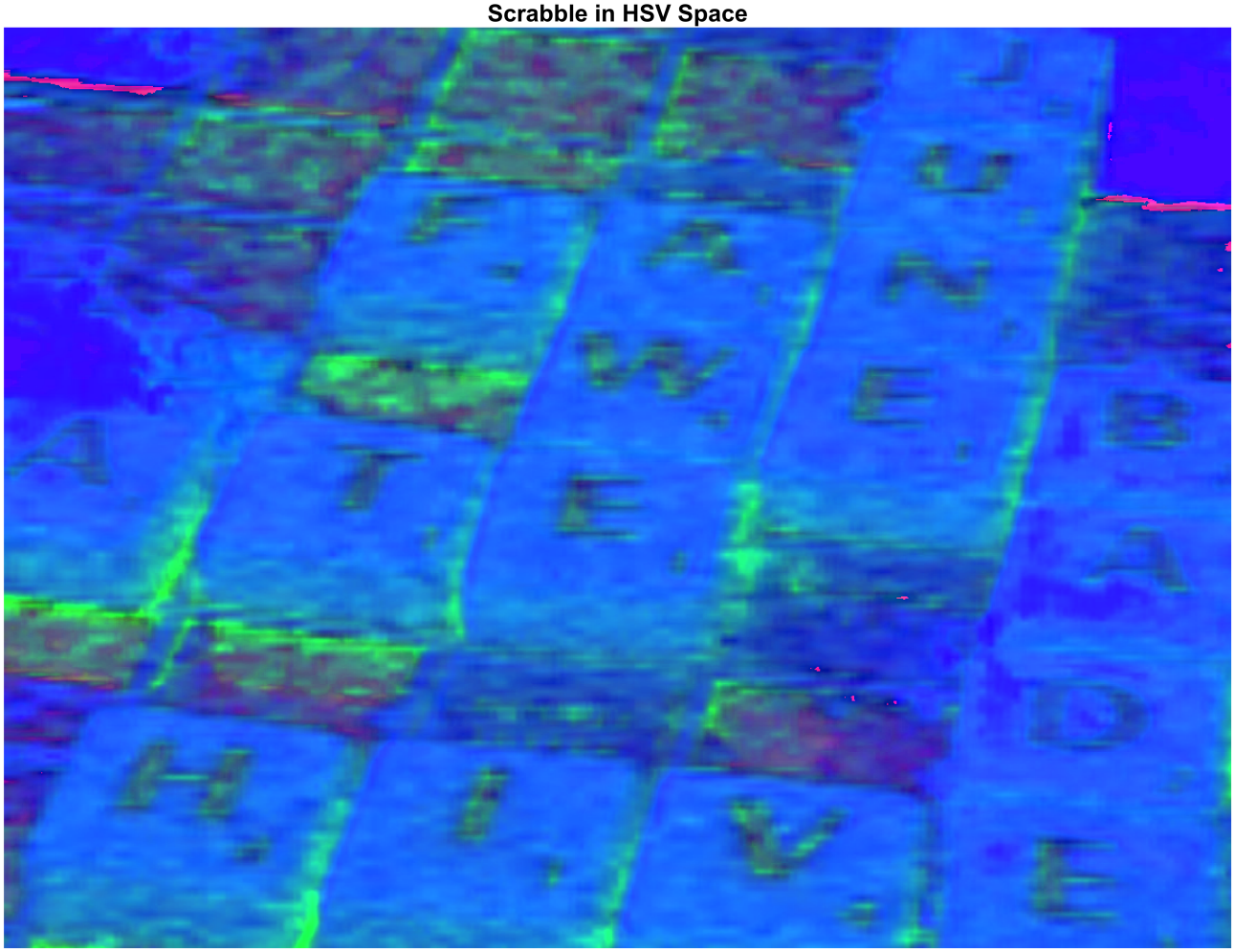

```
scrabble_edge = edge(scrabble_gauss, "canny");

imshow(scrabble_edge)
title("Scrabble - Builtin Edge Detection")
```

Scrabble - Builtin Edge Detection



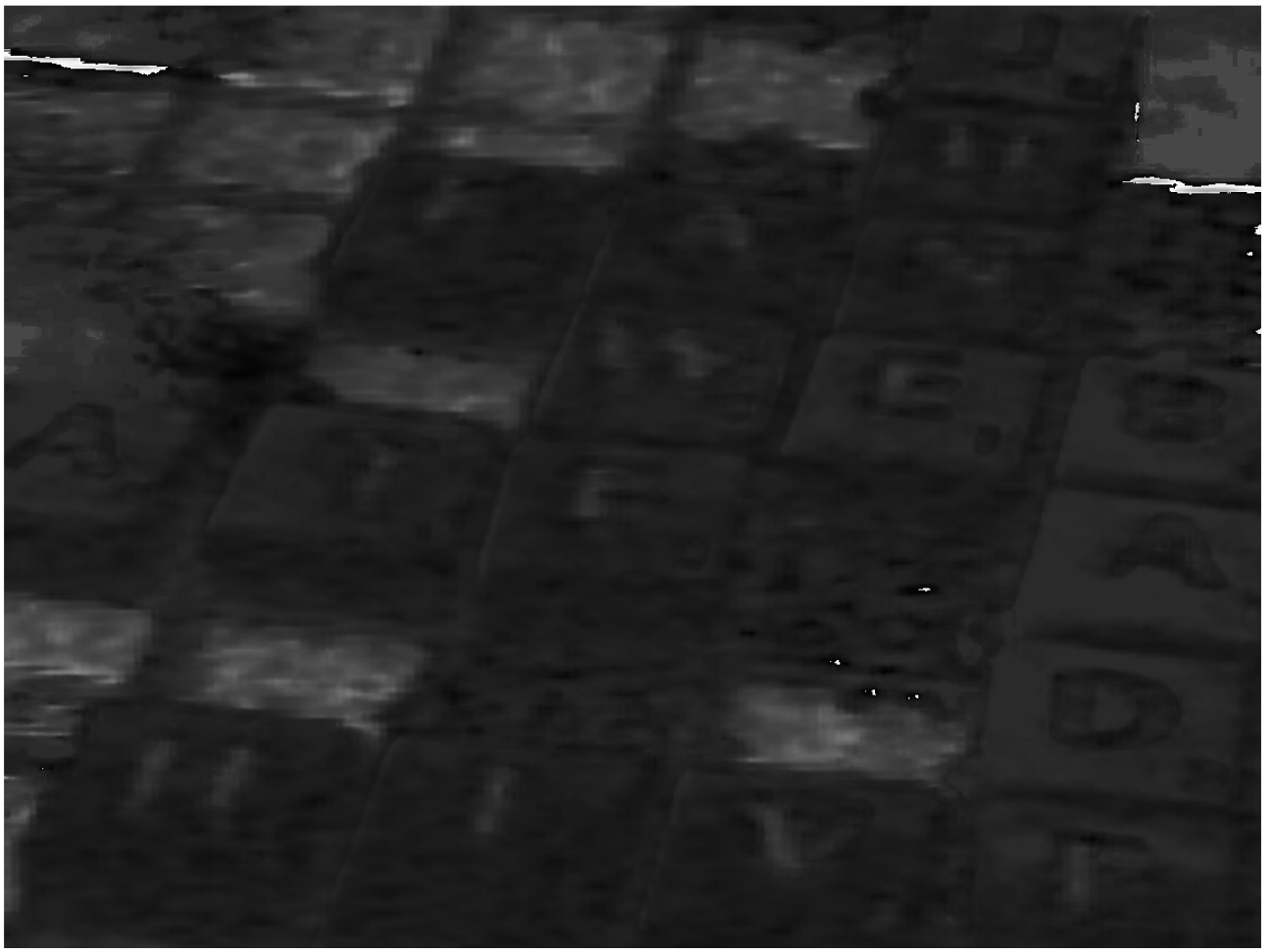
```
hsv_scrabble = rgb2hsv(scrabble);

imshow(hsv_scrabble)
title("Scrabble in HSV Space")
```

Scrabble in HSV Space



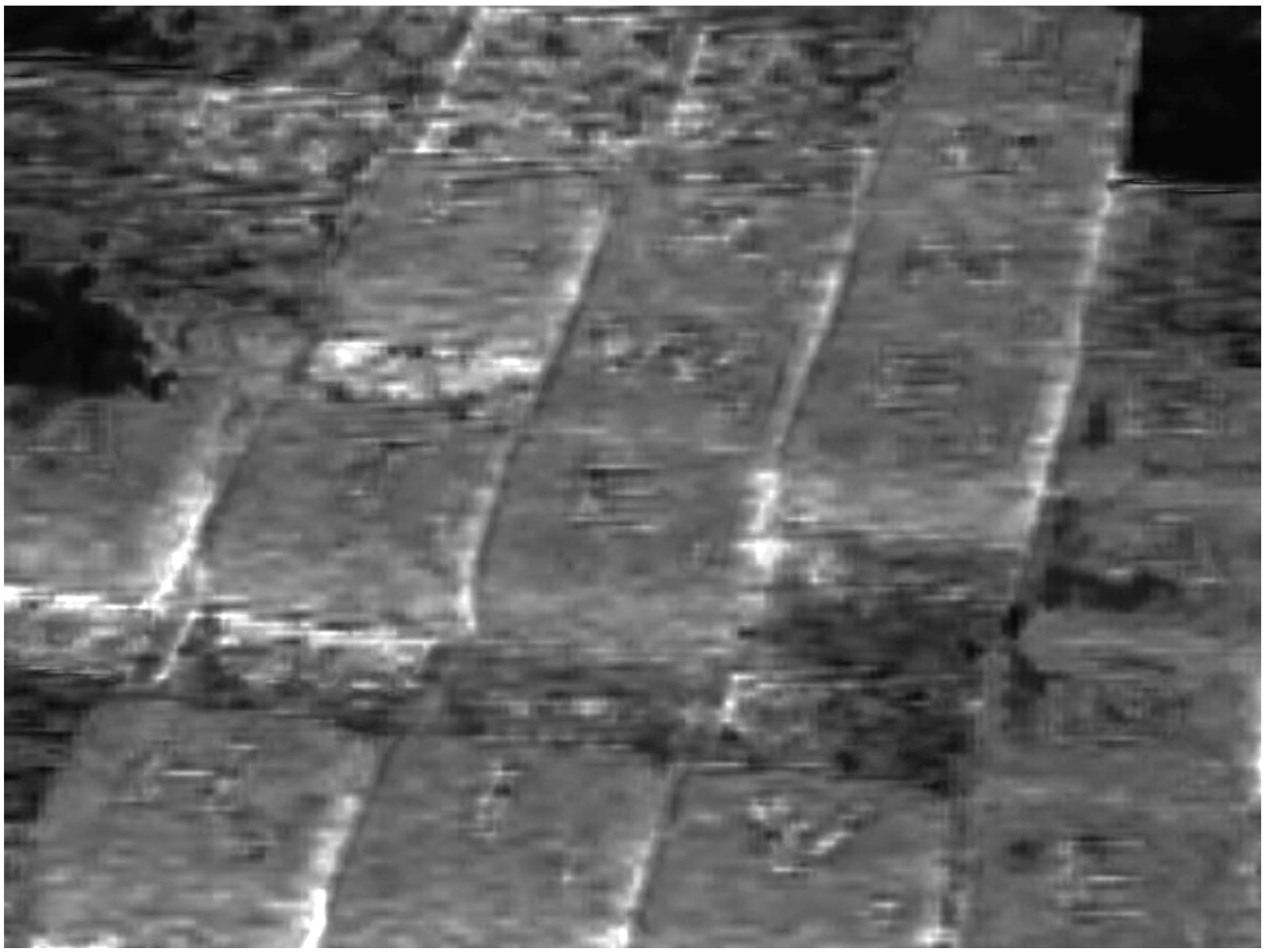
```
imshow(hsv_scrabble(:,:,1))
title("Scrabble - Hue")
```

Scrabble - Hue



```
imshow(hsv_scrabble(:,:,2))
title("Scrabble - Saturation")
```

Scrabble - Saturation

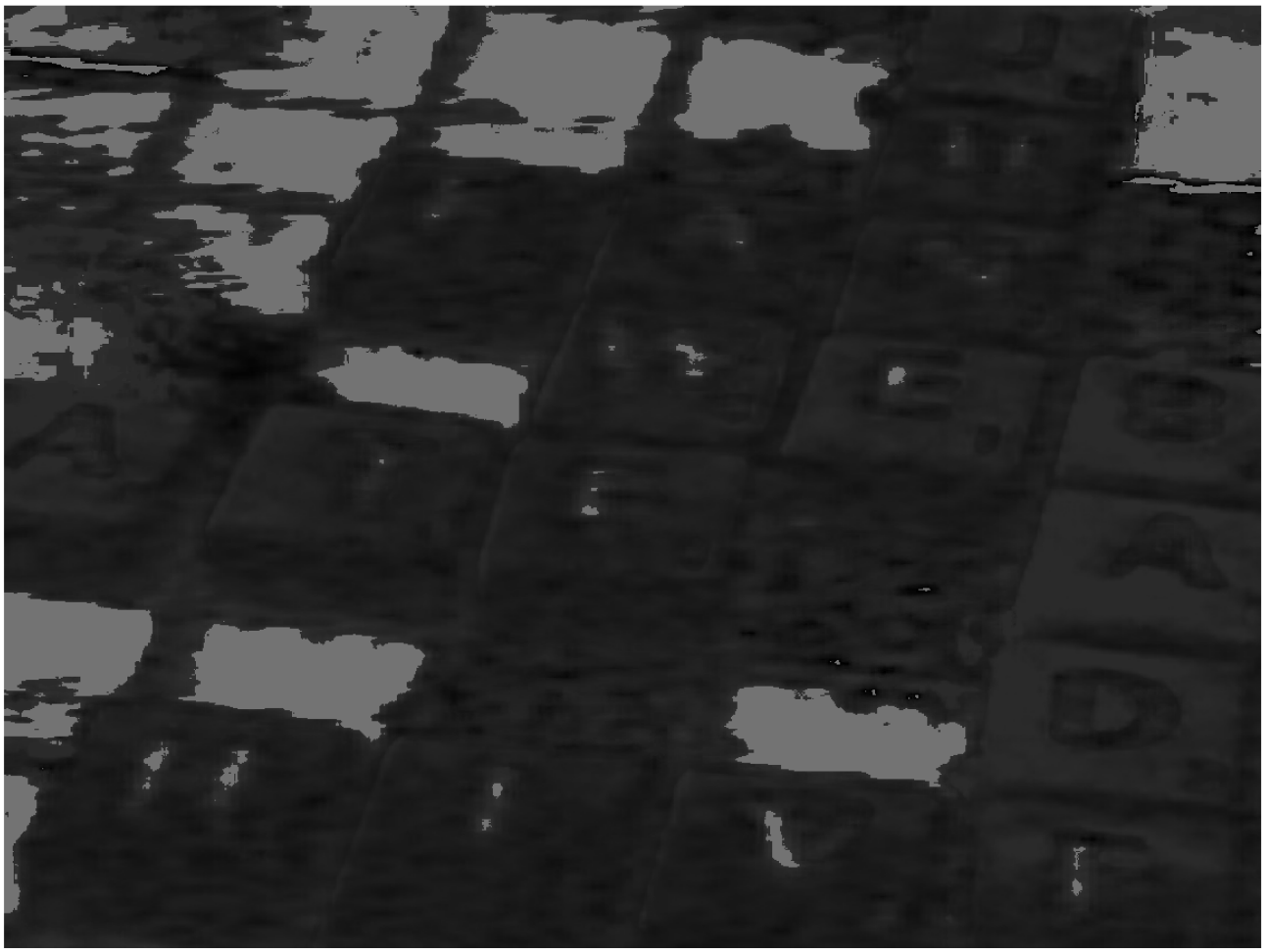


```
imshow(hsv_scrabble(:,:,3))
title("Scrabble - Value")
```

Scrabble - Value


```
hue_scrabble = hsv_scrabble(:,:,1);

sat_scrabble = hsv_scrabble(:,:,2);

value_scrabble = hsv_scrabble(:,:,3);



hue_scrabble(hue_scrabble > 0.19) = 0.45;

sat_scrabble(hue_scrabble == 0.45) = 0.4;

value_scrabble(hue_scrabble == 0.45) = 0.78;



imshow(hue_scrabble)
title("Scrabble - Filtered Hue")
```

Scrabble - Filtered Hue



```
hsv_scrabble(:,:,1) = hue_scrabble;

hsv_scrabble(:,:,2) = sat_scrabble;


hsv_scrabble(:,:,3) = value_scrabble;


rgb_scrabble = hsv2rgb(hsv_scrabble);

figure
imshow(rgb_scrabble)
title("Scrabble - HSV transfered back to RGB")
```

Scrabble - HSV transfered back to RGB


```
bw_scrabble = rgb2gray(rgb_scrabble);

imshow(bw_scrabble)
title("Scrabble - RGB to Grayscale")
```

Scrabble - RGB to Grayscale


```
bw_guass_scrabble = imgaussfilt(bw_scrabble, 7);

imshow(bw_guass_scrabble)
title("Scrabble - Gauss Filtered Grayscale")
```

Scrabble - Gauss Filtered Grayscale

```
sobel_x = [
    -1, 0, 1;
    -2, 0, 2;
    -1, 0, 1;
];

sobel_y = sobel_x';

scrabble_sobel_x = conv2(bw_guass_scrabble, sobel_x);
scrabble_sobel_y = conv2(bw_guass_scrabble, sobel_y);

scrabble_sobel = sqrt(scrabble_sobel_x.^2 + scrabble_sobel_y.^2);

% The range of a double is quite large and some values are near 0
% but are not so this assumes anything less than the mean is a 0
% and everything greater than is a 1. This converts the image into
% binary and makes the edges more apparent.
scrabble_sobel = scrabble_sobel > mean(scrabble_sobel, "all");
```

```
figure
imshow(scrabble_sobel)
title("Scrabble - Sobel Edge Detection of Modified HSV")
```


```
figure
subplot(1,2,1)
imshow(bw_guass_scrabble)
title("HSV Processing")
subplot(1,2,2)
imshow(scrabble_gauss)
title("RGB Processing")
```

HSV Processing

RGB Processing

The letters on the pieces are obvious and distinct however so are the lines in between the tiles. To output just the letters we tired to smooth the space between the letters to be a consistent color that could then be filtered out. We tried modifying the different RGB layers and changing values based upon brightness however the lines were largely uneffected and were still present. After being unsuccessful, we tried processing the image in gray scale. We still wanted to to remove the lines in between the tiles so we tried using a gaussian filter to blur the lines in between hoping that they would then blend in with the surrounding pixel and then be filtered out entirely. After the blurring, we input the image into the edge detection code we developed in Worksheet 3 to see how well the letters were outlined. The letters were outlined well but the lines inbetween the tiles were still present. We then put the guassian filtered image into the built in edge function. The outlput did not produce anything that was more useful than the previous output.

After our first attempt at edge detection, we set out to have our main goal to be to separate the board from the tiles, through which we could eliminate a lot of the noise in the found edges. We tried numerous different ways to do this, but generally tried to focus on harnessing the fact that the tiles on the board are the only green things in the image.

Just filtering out these tiles proved to be ineffective, as the whiteish dividing lines between them still came up just as clearly when running edge detection. However, to counter this, our idea was to modify the greenish tiles to be white such that we could Gaussian blur them together and ultimately have a fairly uniform background

that wouldn’t show up in edge detection. Detecting and changing the green areas proved to be a challenge. We tried numerous filtering and masking strategies in the RGB space, but the fact that the tiles, despite being very green, was spread pretty evenly among the RGB spectrum proved it near impossible to effectively select them.

Our alternative solution was to move into the HSV color space. As color is mostly determined by the single hue value, this color space proved much more successfully in allowing us to select only the green tiles. Upon filtering and moving back to the RGB space, we were then able to more effectively use a Gaussian blur to create a more uniform background. This definitely improved on our original image in terms of weakening the edges between tiles, but did not completely eliminate them.

Overall, our final edge finding of the scrabble letters is improved from the original image. However, further filtering could be done to try and isolate the letters from the tiles that surround them. Further directions to take to refine this image could be deblurring, corner detection to try and separate letters from more square tiles, and further segmentation to remove the background.