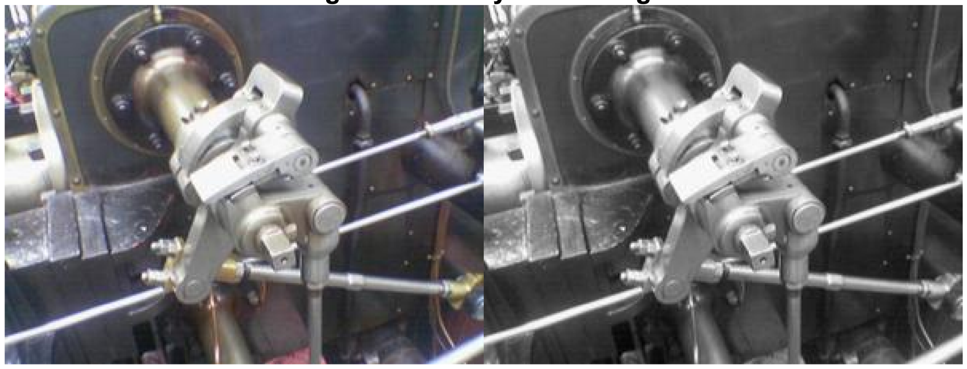
Image Processing

```
clc;
clear all;
close all;

%read the image
rgb_img = imread('image\Steamengine.jpg');
```

1. Convert RGB to Gray Level Image:

```
gray_img = rgb2gray(rgb_img);
imshowpair(rgb_img, gray_img, 'montage');
title('Original and Gray Level Images');
```

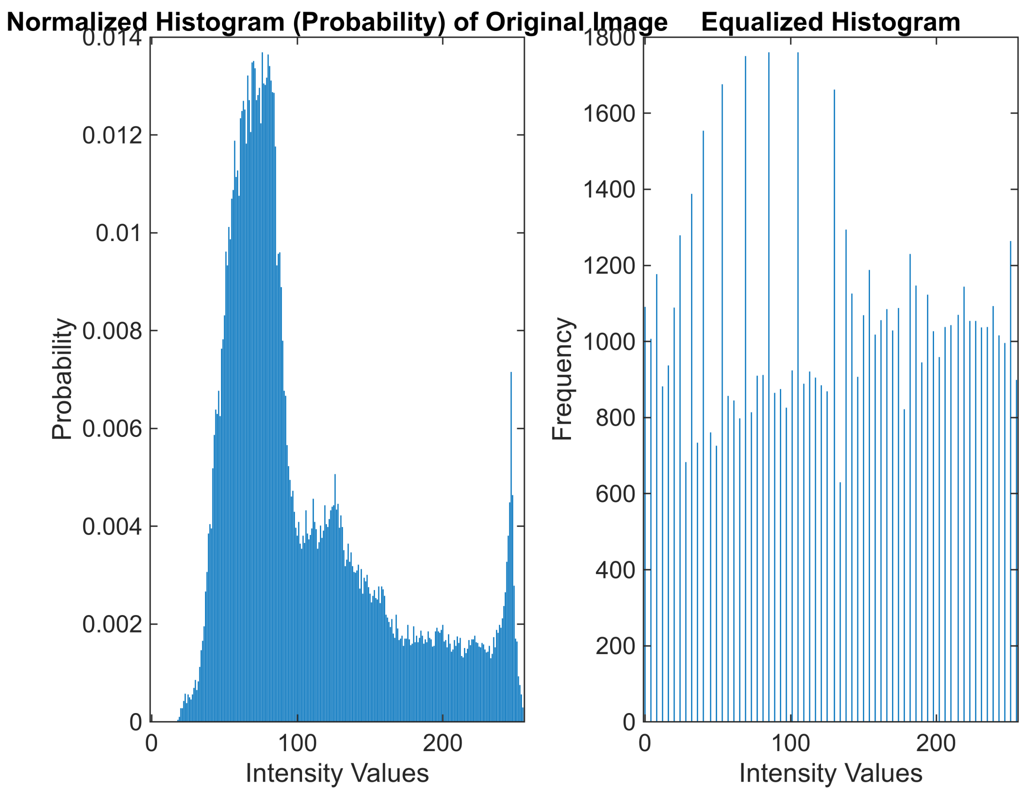
Original and Gray Level Images



2. Normalized and Equalized Histogram:

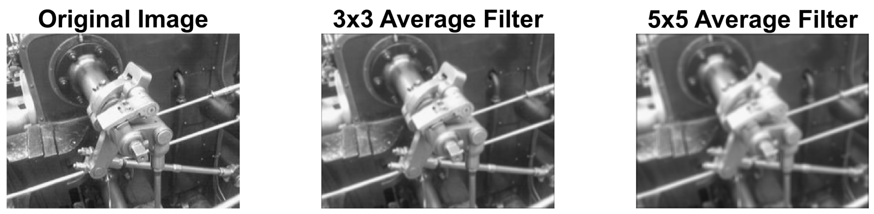
```
% Calculate the histogram of the grayscale image
[counts, binLocations] = imhist(gray_img);  % Get histogram data

% Normalize the histogram to convert frequencies to probabilities
normalized_counts = counts / numel(gray_img);  % Divide by the total number of
pixels

% Plot the normalized histogram as a probability distribution
figure;
subplot(1,2,1);
bar(binLocations, normalized_counts);  % Use bar to plot discrete probabilities
title('Normalized Histogram (Probability) of Original Image');
xlabel('Intensity Values');
ylabel('Probability');

% Apply histogram equalization
eq_img = histeq(gray_img);
```

```
% Plot equalized histogram

[counts_eq, binLocations_eq] = imhist(eq_img);  % Get histogram data for equalized
image
subplot(1, 2, 2);
bar(binLocations_eq, counts_eq);
title('Equalized Histogram');
xlabel('Intensity Values');
ylabel('Frequency');
```



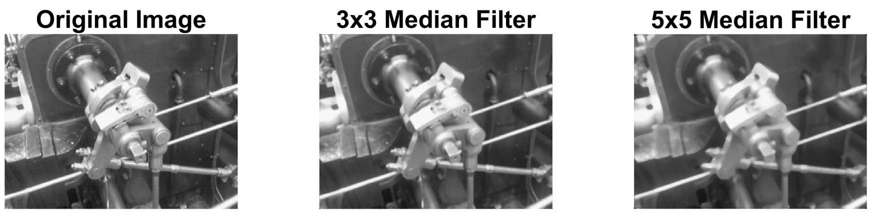
3. Apply average filters (3x3) and (5x5):

```
% create 3x3 and 5x5 average filters using fspecial

filter3x3 = fspecial('average', [3 3]);
filter5x5 = fspecial('average', [5 5]);

% apply filters

avg_img3x3 = imfilter(gray_img, filter3x3);
avg_img5x5 = imfilter(gray_img, filter5x5);

% Display the original and filtered images
figure;
subplot(1, 3, 1), imshow(gray_img), title('Original Image');
subplot(1, 3, 2), imshow(avg_img3x3), title('3x3 Average Filter');
subplot(1, 3, 3), imshow(avg_img5x5), title('5x5 Average Filter');
```



4. Apply median filters (3x3) and (5x5):

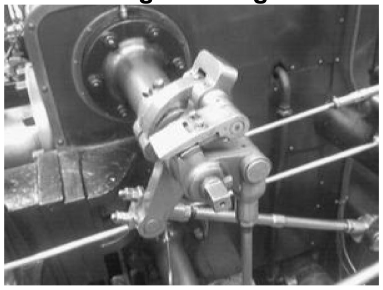
```
% Apply median filtering with 3x3 and 5x5 kernels
median_img3x3 = medfilt2(gray_img, [3 3]);
median_img5x5 = medfilt2(gray_img, [5 5]);

% Display the original and median filtered images
figure;
subplot(1, 3, 1), imshow(gray_img), title('Original Image');
subplot(1, 3, 2), imshow(median_img3x3), title('3x3 Median Filter');
subplot(1, 3, 3), imshow(median_img5x5), title('5x5 Median Filter');
```



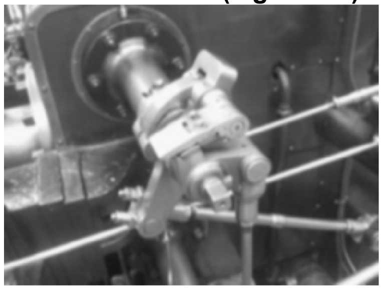
5. Apply Gaussian filter:

```
% Applying gaussian filter with standard deviation (sigma)

sigma = 1.0; % adjust
gaussian_img = imgaussfilt(gray_img, sigma);

figure;
subplot(1, 2, 1), imshow(gray_img), title('Original Image');
subplot(1, 2, 2), imshow(gaussian_img), title('Gaussian Filter (sigma = 1)');
```

Original Image

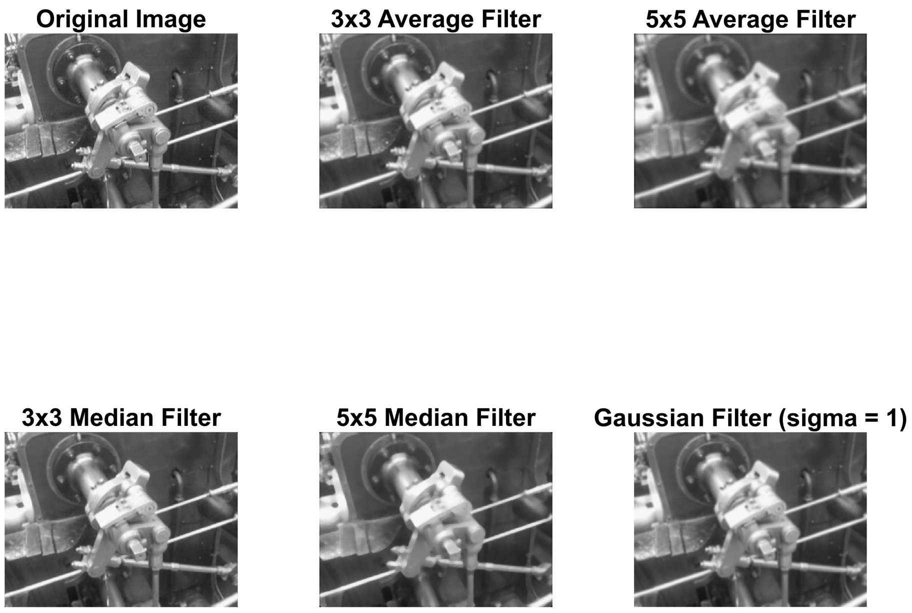


Gaussian Filter (sigma = 1)



6. Comments on 3, 4, and 5:

```
% plot all filters to compare:

figure;
subplot(2, 3, 1), imshow(gray_img), title('Original Image');

% Average Filtered Images
subplot(2, 3, 2), imshow(avg_img3x3), title('3x3 Average Filter');
subplot(2, 3, 3), imshow(avg_img5x5), title('5x5 Average Filter');

% Median Filtered Images
subplot(2, 3, 4), imshow(median_img3x3), title('3x3 Median Filter');
subplot(2, 3, 5), imshow(median_img5x5), title('5x5 Median Filter');

% Gaussian Filtered Image
subplot(2, 3, 6), imshow(gaussian_img), title('Gaussian Filter (sigma = 1)');
```



From the plot, we can see that the 3x3 average filter gives a moderate smoothing, while 5x5 filter offers stronger smoothing, which can cause a slight blurring effect.

The average filter is effective for reducing random noise but also can introduce blur at the edges. The median filter seems to be better at removing noise while preserving edges.

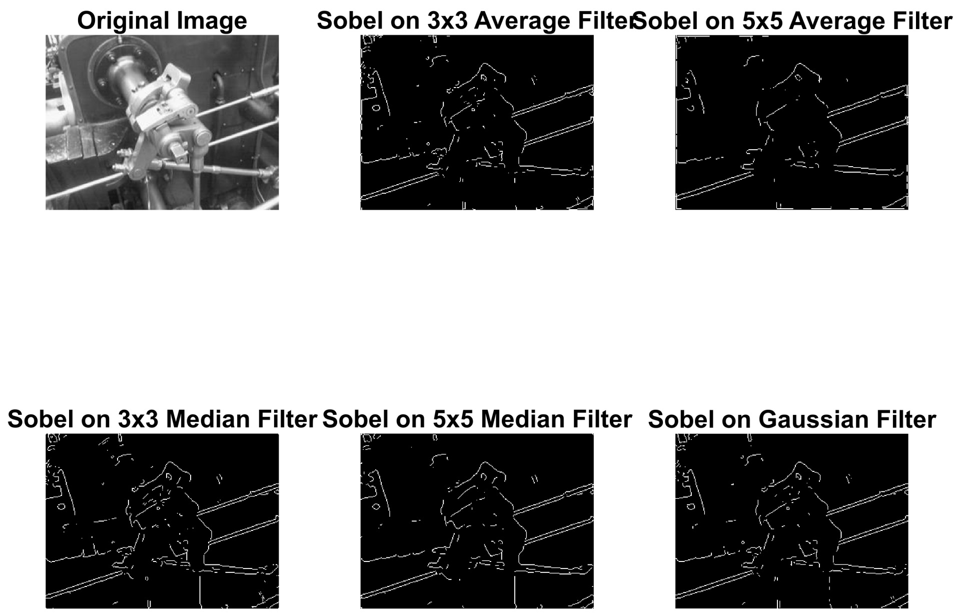
The 3x3 median filter preserves the image structure better than average filter. The 5x5 median filter provides stronger noise removal but also start to slightly blur fine details.

The Gaussian filter tends to smooth image naturally, blurring noise while preserving edges better than average filter.

7. Apply Sobel edge detection for steps 3,4, and 5:

```
% Sobel edge detection for Average Filtered Images
sobel_avg3x3 = edge(avg_img3x3, 'Sobel');
sobel_avg5x5 = edge(avg_img5x5, 'Sobel');

% Sobel edge detection for Median Filtered Images
sobel_median3x3 = edge(median_img3x3, 'Sobel');
sobel_median5x5 = edge(median_img5x5, 'Sobel');
```

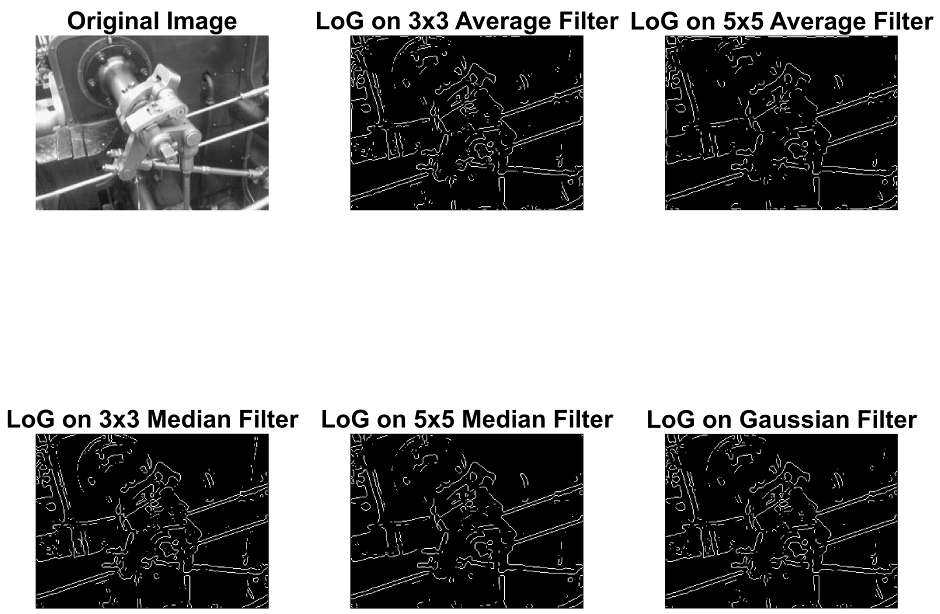
```
% Sobel edge detection for Gaussian Filtered Image
sobel_gaussian = edge(gaussian_img, 'Sobel');

% Plot the Sobel edge detection results:

figure;

subplot(2, 3, 1), imshow(gray_img), title('Original Image');

subplot(2, 3, 2), imshow(sobel_avg3x3), title('Sobel on 3x3 Average Filter');
subplot(2, 3, 3), imshow(sobel_avg5x5), title('Sobel on 5x5 Average Filter');

subplot(2, 3, 4), imshow(sobel_median3x3), title('Sobel on 3x3 Median Filter');
subplot(2, 3, 5), imshow(sobel_median5x5), title('Sobel on 5x5 Median Filter');

subplot(2, 3, 6), imshow(sobel_gaussian), title('Sobel on Gaussian Filter');
```


Original Image
Sobel on 3x3 Average Filter
Sobel on 5x5 Average Filter
Sobel on 3x3 Median Filter
Sobel on 5x5 Median Filter
Sobel on Gaussian Filter

8. Apply Laplacian for steps 3,4, and 5:

```
% using Laplacian of Gaussian (log) edge detection

% LoG edge detection on Average Filtered Images
log_avg3x3 = edge(avg_img3x3, 'log');
log_avg5x5 = edge(avg_img5x5, 'log');
```

```
% LoG edge detection on Median Filtered Images
log_median3x3 = edge(median_img3x3, 'log');
log_median5x5 = edge(median_img5x5, 'log');

% LoG edge detection on Gaussian Filtered Image
log_gaussian = edge(gaussian_img, 'log');

% Plot the LoG edge detection results:

figure;

subplot(2, 3, 1), imshow(gray_img), title('Original Image');

subplot(2, 3, 2), imshow(log_avg3x3), title('LoG on 3x3 Average Filter');
subplot(2, 3, 3), imshow(log_avg5x5), title('LoG on 5x5 Average Filter');

subplot(2, 3, 4), imshow(log_median3x3), title('LoG on 3x3 Median Filter');
subplot(2, 3, 5), imshow(log_median5x5), title('LoG on 5x5 Median Filter');

subplot(2, 3, 6), imshow(log_gaussian), title('LoG on Gaussian Filter');
```



9 Comments on step 7 and 8

From the plots in step 7 and 8, we can say that the sobel operator is ideal for fast and simple edge detection. It provides us with best result when we have high-contrast edges and low noise.

The laplacian operator more suited for noisy images as it handles noise better and provides cleaner and more accurate edge detection but with a slight loss in edge sharpness.

10 Find a suitable threshold and change results to binary form

```
% Check the Sobel and Laplacian outputs
class(sobel_avg3x3)
```

```
ans =
'logical'
```

```
unique(sobel_avg3x3)
```

```
ans = 2×1 logical array
   0
   1
```

```
class(log_avg3x3)
```

```
ans =
'logical'
```

```
unique(log_avg3x3)
```

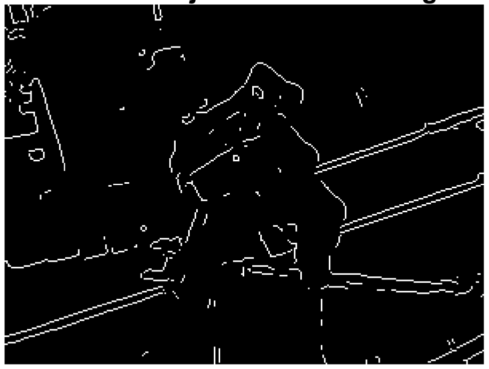
```
ans = 2×1 logical array
   0
   1
```

Since the edge function in MATLAB applies automatic thresholding, the images are already modified with an optimal threshold obtained by technique like Otsu's method and also the images are in binary form which can be observed from the above output.

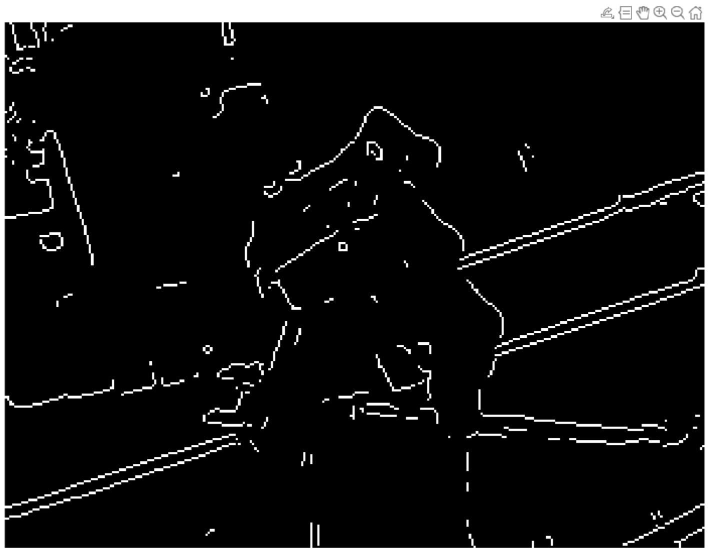
11 Select part of the object from step 7:

```
% Display the Sobel edge detection result to select a part of the object
figure, imshow(sobel_gaussian);
title('Select a Part of the Object from Sobel Edge Detection');
```

Select a Part of the Object from Sobel Edge Detection



```
% Use imcrop to manually select a region of interest (ROI)
selected_part = imcrop(sobel_gaussian);
```



```
% Display the selected part
figure, imshow(selected_part);
title('Selected Part of the Object from Sobel Edge Detection');
```

ted Part of the Object from Sobel Edge Dete

12 Find HSI model of the main image:

```
% Since the main image is already read in the first part of this program
% Normalize the image to [0 1] and also convert it to double format using
% im2double

rgb_img = im2double(rgb_img);

% Seperate R, G, and B channels

R = rgb_img(:, :, 1);
G = rgb_img(:, :, 2);
B = rgb_img(:, :, 3);

% Calculate the Intensity (I)

% Intensity is average of R,G, and B channels

I= (R+G+B)/3;

% Calculate Saturation (S)

% Find the minimum of R, G, and B at each pixel

min_RGB = min(cat(3, R, G, B), [], 3);

% Formula for Saturation

S = 1-(3./(R+G+B+eps)).* min_RGB; % eps added to avoid division by zero

% Calculate Hue (H)

% Numerator and Denominator for Hue calculation

num = 0.5 * ((R-G)+(R-B));
den = sqrt((R-G).^2 + (R-B).*(G-B)) + eps; % eps added to avoid division by zero

% Calculate theta in radians for Hue
```

```
theta = acos(num./den);

% Initialize H with theta values
H = theta;

% Adjust Hue based on condition: if B > G, then H = 2*pi - theta
H(B > G) = 2 * pi - H(B > G);

% Normalize H to range [0 1]

H = H/(2*pi);

% Combine H, S, and I to HSI image

HSI_img = cat(3, H, S, I); %concatenate along the third dimension

% Display the HSI components

figure;
subplot(1, 3, 1), imshow(H), title('Hue Channel');
subplot(1, 3, 2), imshow(S), title('Saturation Channel');
subplot(1, 3, 3), imshow(I), title('Intensity Channel');
sgtitle('HSI model of the main image');
```

HSI model of the main image