EXPERIMENT NO: 4

1. Aim: implementation of video noise filtering using spatial, temporal, and spatial-temporal domain filters with mean, median, and weighted average filters using 3-sizeand 5-sizekernels.

2. Software Used: MATLAB R2024b

3. Theory:

Noise in Video Sequences
Noise in video sequences can be caused by sensor imperfections, transmission errors, or low-light conditions. It manifests as unwanted variations in pixel intensity over time. Common types of noise include:

- **Gaussian noise:** Random variations in intensity due to thermal noise.
- **Salt & pepper noise:** Random occurrences of black and white pixels due to transmission errors.
- **Speckle noise:** Multiplicative noise common in coherent imaging systems.

To remove noise, **spatial filtering**, **temporal filtering**, and **spatial + temporal filtering** techniques are used.

Filters Used in the Spatial Domain:
Mean Filter: Computes the average of the neighbouring pixel values. It smooths the image but may blur edges.

$$I'(x,y) = \frac{1}{N} \sum_{(i,j)\in\text{kernel}} I(x+i, y+j)$$

Median Filter: Sorts the pixel values in the kernel and selects the median. It effectively removes salt-and-pepper noise while preserving edges.
Weighted Average Filter: Assigns higher weights to central pixels, reducing the impact of noise while maintaining structure.
Kernal_3*3 = [1, 1, 1; 1, 2, 1; 1, 1, 1]/10.
Kernal_5*5 = [1, 1, 1, 1, 1; 1, 2, 2, 2, 1; 1, 2, 4, 2, 1; 1, 2, 2, 2, 1; 1, 1, 1, 1, 1]/ 36.
Kernel Sizes Used:

- **3-sizeKernel:** Balances noise reduction and detail preservation.
- **5-sizeKernel:** Provides stronger noise reduction but can cause blurring.

Temporal Domain Filtering: Temporal filtering reduces noise by averaging or applying statistical operations across multiple frames at the same pixel location.
Filters Used in the Temporal Domain:

- **Temporal Mean Filter:** Averages pixel values across multiple frames to smooth variations.

$$I'(x,y,t) = \frac{1}{M} \sum_{k=0}^{M-1} I(x,y,t-k)$$

temporal Median Filter: Takes the median of pixel values across several frames. It is more effective against impulsive noise.
Temporal Weighted Average Filter: Assigns more weight to the current frame while considering previous frames.
Weight_Kernal_3*3 = [1, 2, 1]/4.
Weight_kernal_5*5 = [1, 2, 4, 2, 1]/10.

Spatial + Temporal Filtering: A combination of spatial and temporal filtering techniques provides superior noise reduction. The process involves:

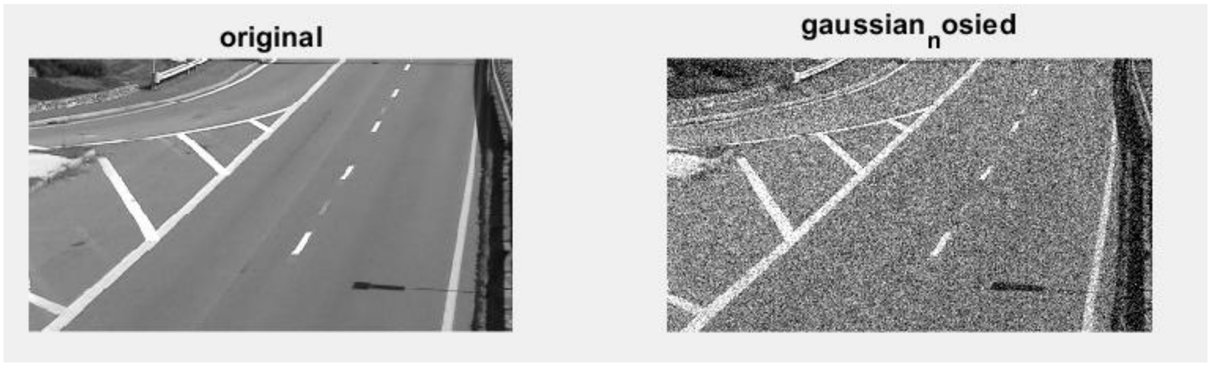
1. **Applying a spatial filter** (mean, median, or weighted average) to individual frames.
2. **Applying a temporal filter** (mean, median, or weighted average) across consecutive frames.
   Weight_kernal_for_3size = K1: K3: K1.
   Weight_kernal_for_3size = K1: K3: K5: K3: K1.
   K1 = 1. K3 = [1, 1, 1; 1, 2, 1; 1, 1, 1] / 10.
   Kernel K3 = [1, 1, 1, 1, 1; 1, 2, 2, 2, 1; 1, 2, 4, 2, 1; 1, 2, 2, 2, 1; 1, 1, 1, 1, 1] / 36.

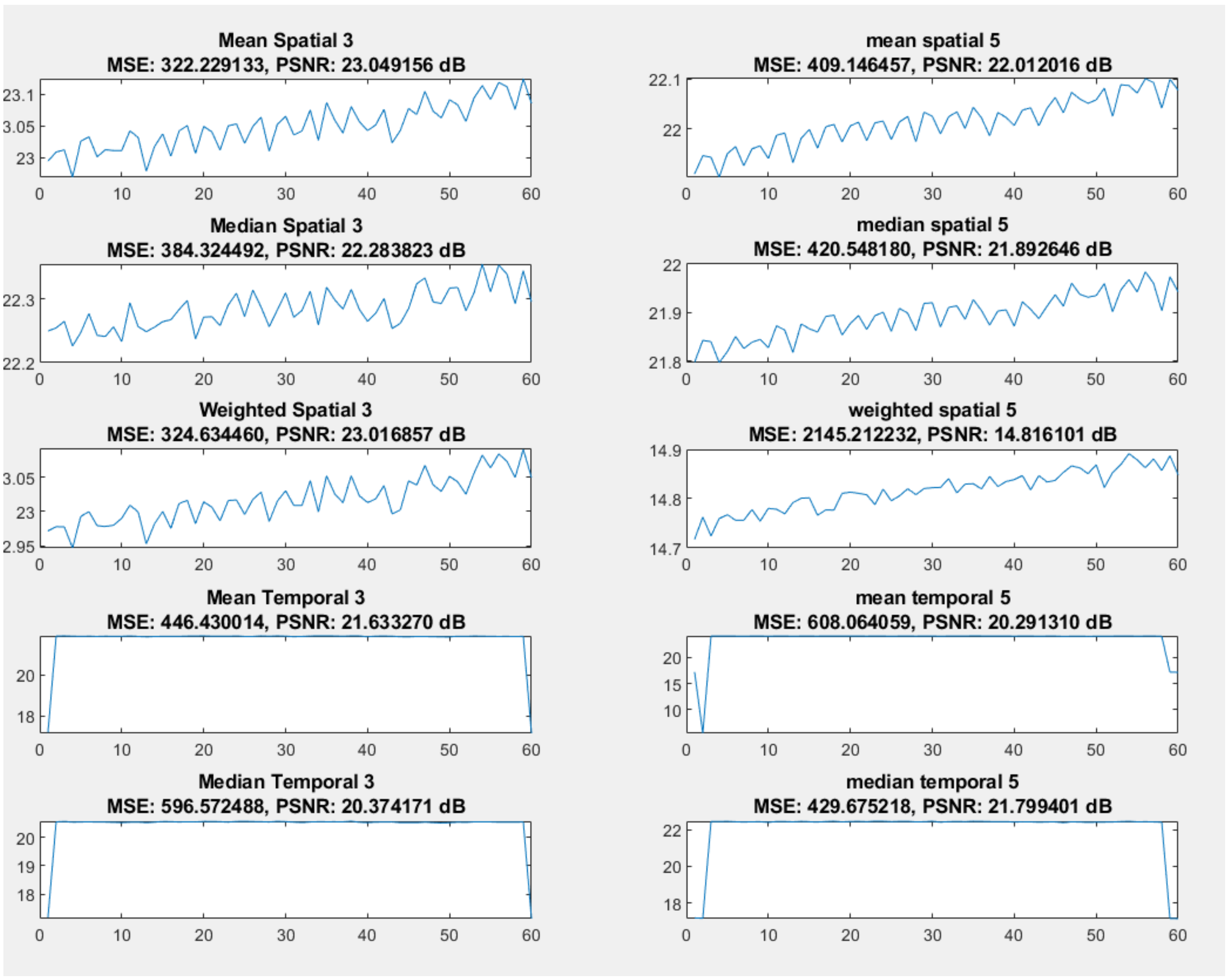
This hybrid approach enhances noise reduction while maintaining video quality.

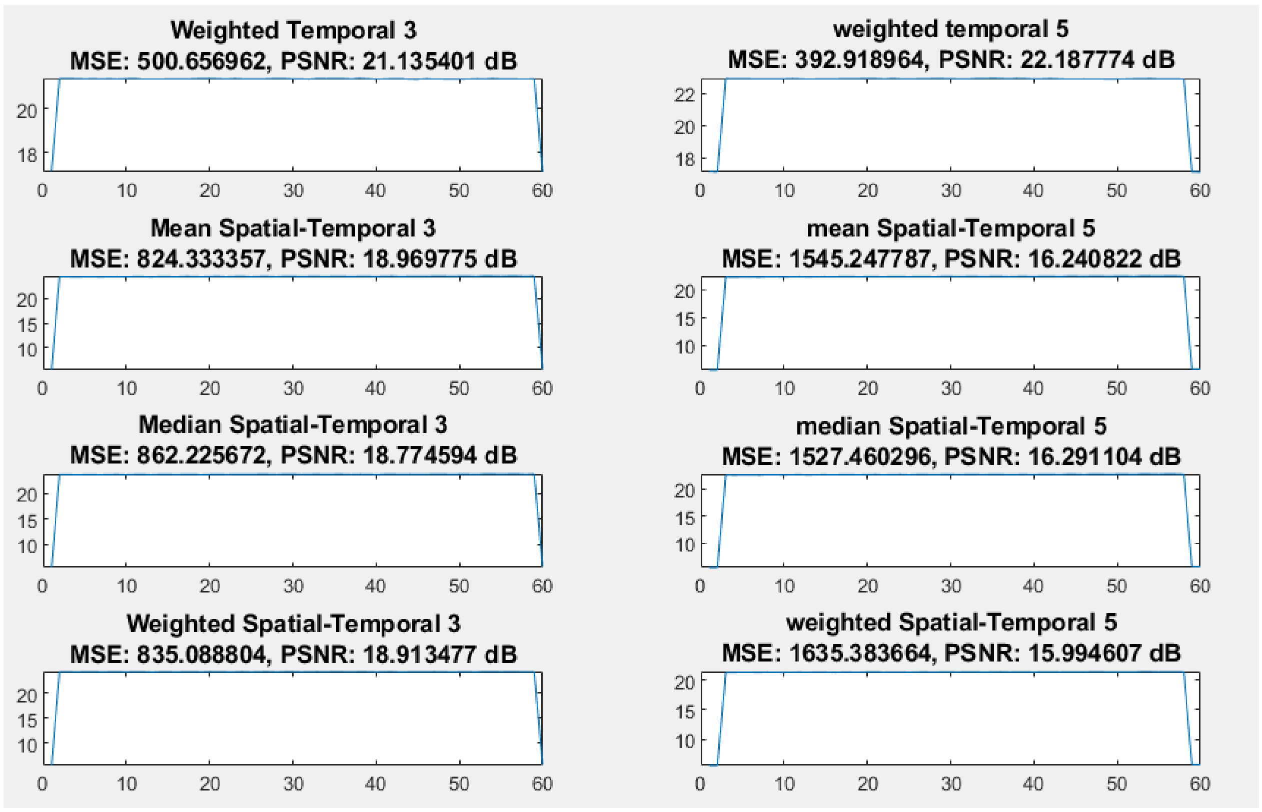
Video functions:

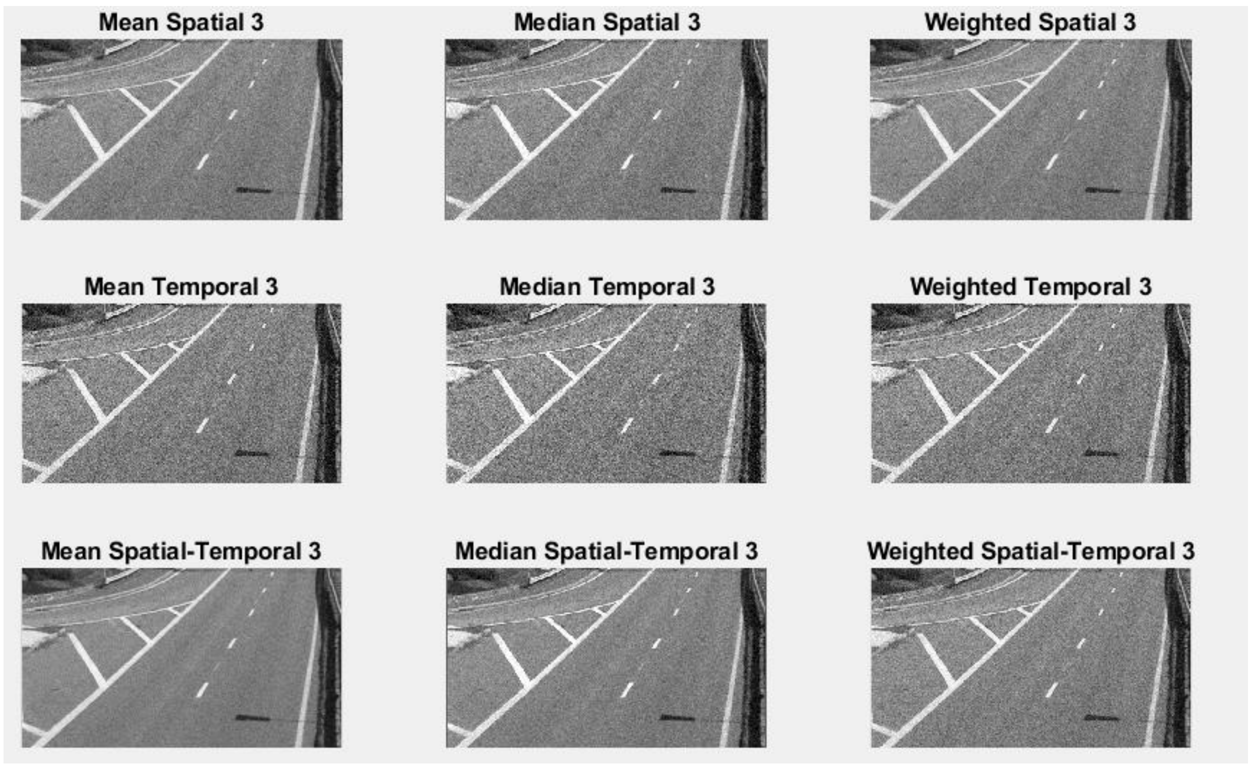
- **VideoReader():** Used to read video files frame-by-frame.
  ```
  vidObj = VideoReader('video.mp4');
  frame = read(vidObj, frameNumber); % Read a specific frame
  vidObj.FrameRate → Frame rate of video
  vidObj.Duration → Total video duration
  vidObj.Height, vidObj.Width → Frame size
  ```
- **VideoWriter():** Used to save processed video frames into a new video file.
  ```
  writerObj = VideoWriter('output.avi', 'Uncompressed AVI');
  open(writerObj);
  writeVideo(writerObj, frame); % Write frame to video
  close(writerObj);
  ```
- **getframe():** Captures frames from a figure window (useful for visualization).
  - frame = getframe(gca);
- **movie():** Plays a sequence of frames as a video in MATLAB.
  - movie(framesArray, numRepeats, frameRate);
- **read():** Reads video frames from a VideoReader object. Converts frames to image matrices for processing.
  - frame = read(vidObj, frameNumber);
- **imshow():** Displays a single frame.
  - imshow(frame);
- **implay():** Plays a video in MATLAB.
  - implay('video.mp4');

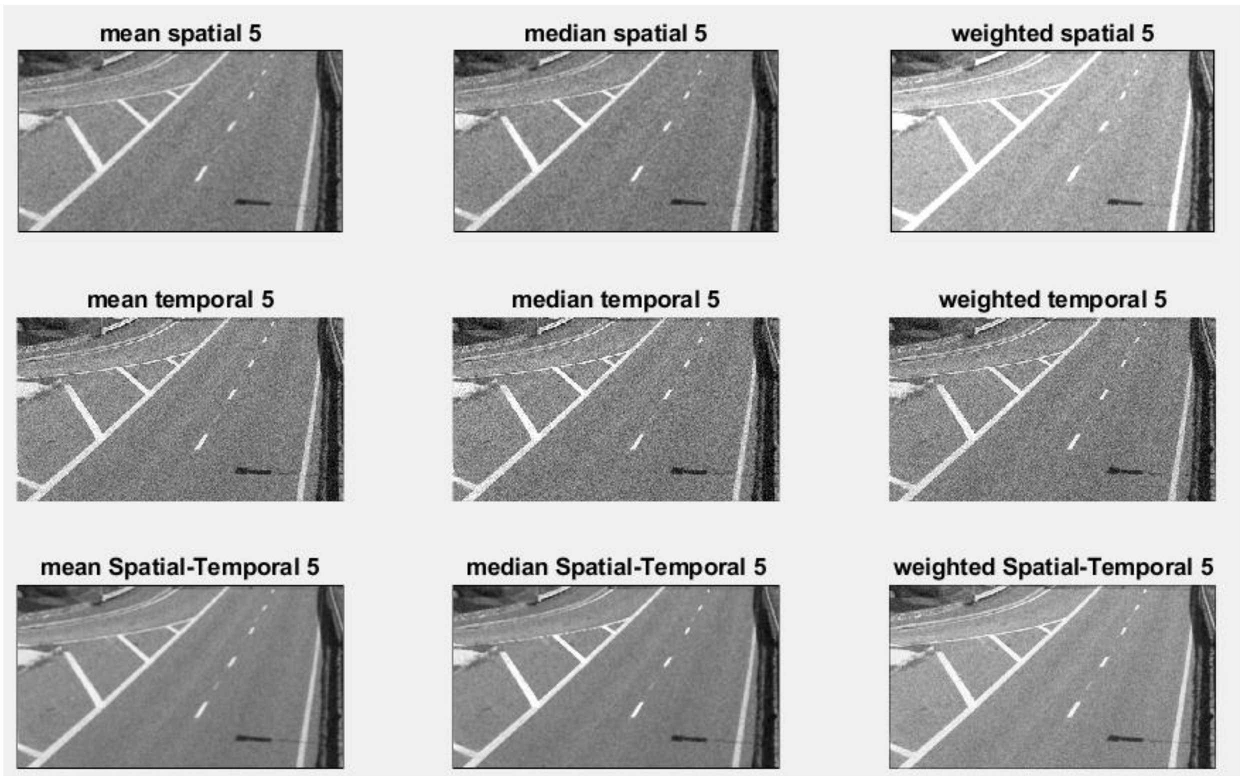
4. Result:

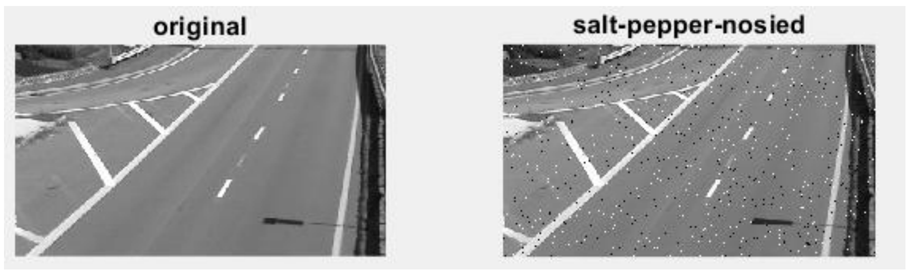


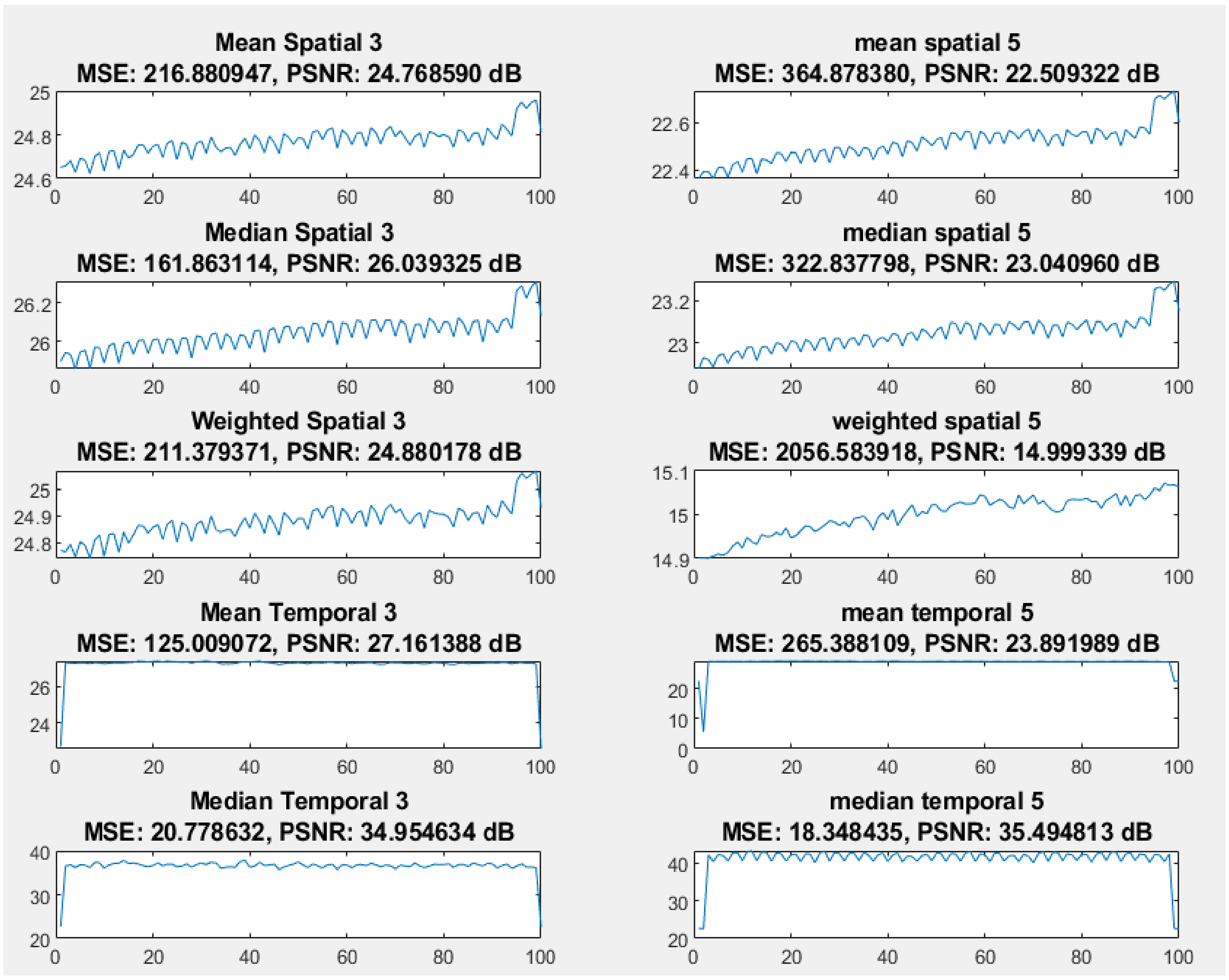

Mean Spatial 3
MSE: 322.229133, PSNR: 23.049156 dB
mean spatial 5
MSE: 409.146457, PSNR: 22.012016 dB
Median Spatial 3
MSE: 384.324492, PSNR: 22.283823 dB
median spatial 5
MSE: 420.548180, PSNR: 21.892646 dB
Weighted Spatial 3
MSE: 324.634460, PSNR: 23.016857 dB
weighted spatial 5
MSE: 2145.212232, PSNR: 14.816101 dB
Mean Temporal 3
MSE: 446.430014, PSNR: 21.633270 dB
mean temporal 5
MSE: 608.064059, PSNR: 20.291310 dB
Median Temporal 3
MSE: 596.572488, PSNR: 20.374171 dB
median temporal 5
MSE: 429.675218, PSNR: 21.799401 dB

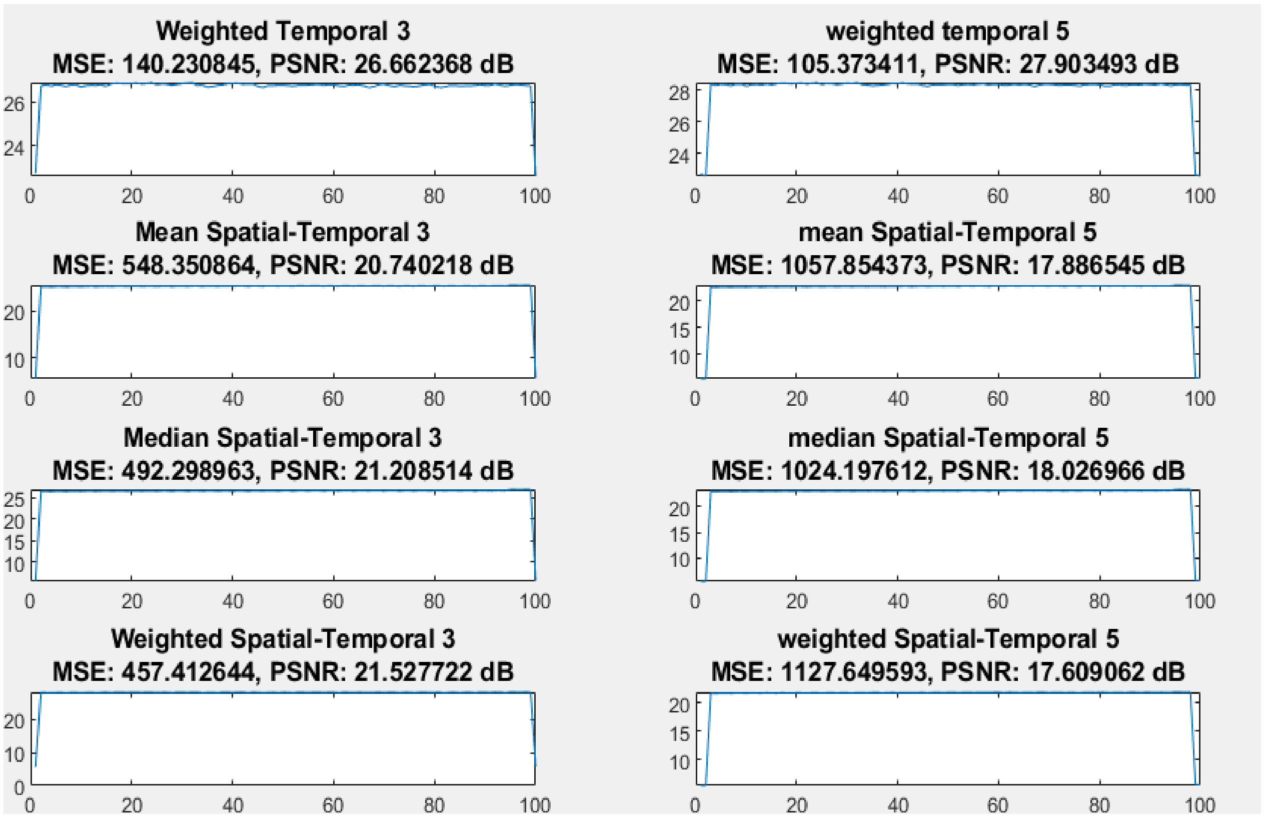

Weighted Temporal 3
MSE: 500.656962, PSNR: 21.135401 dB
weighted temporal 5
MSE: 392.918964, PSNR: 22.187774 dB
Mean Spatial-Temporal 3
MSE: 824.333357, PSNR: 18.969775 dB
mean Spatial-Temporal 5
MSE: 1545.247787, PSNR: 16.240822 dB
Median Spatial-Temporal 3
MSE: 862.225672, PSNR: 18.774594 dB
median Spatial-Temporal 5
MSE: 1527.460296, PSNR: 16.291104 dB
Weighted Spatial-Temporal 3
MSE: 835.088804, PSNR: 18.913477 dB
weighted Spatial-Temporal 5
MSE: 1635.383664, PSNR: 15.994607 dB

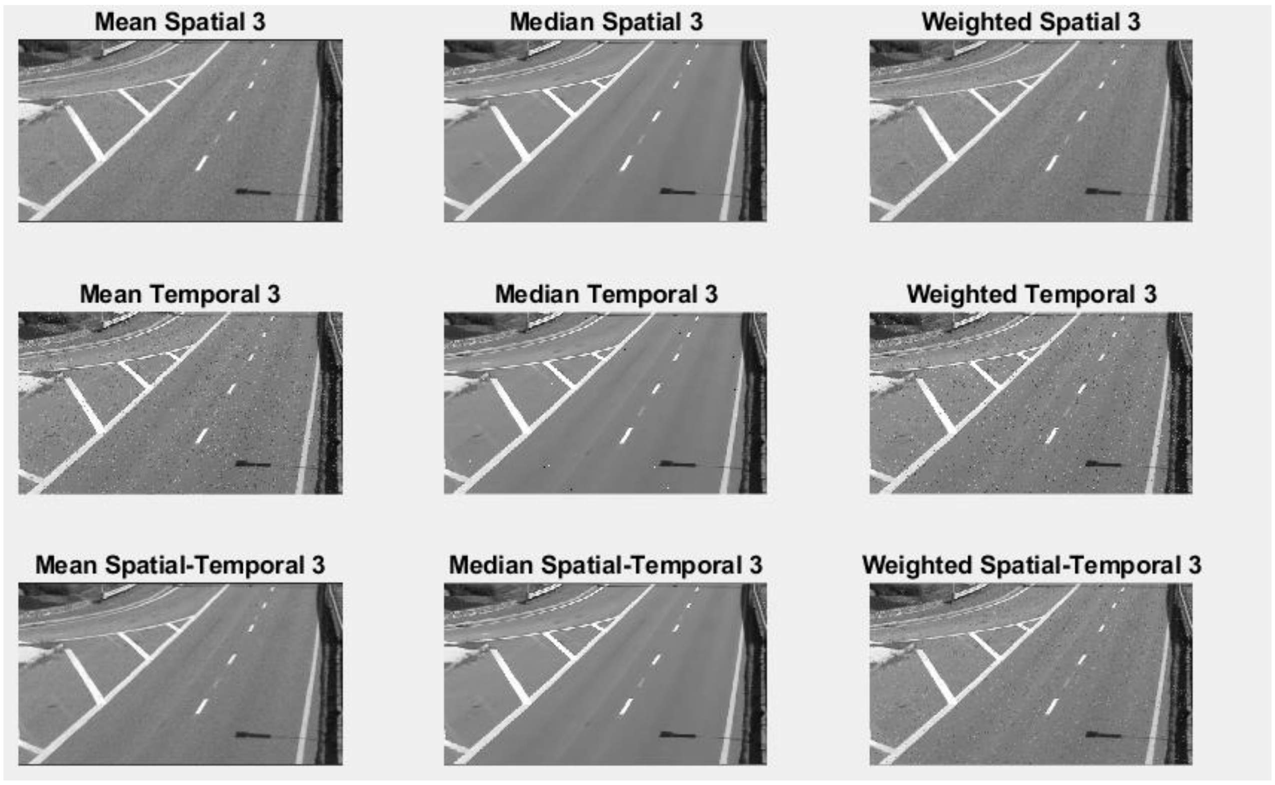

Mean Spatial 3
Median Spatial 3
Weighted Spatial 3
Mean Temporal 3
Median Temporal 3
Weighted Temporal 3
Mean Spatial-Temporal 3
Median Spatial-Temporal 3
Weighted Spatial-Temporal 3

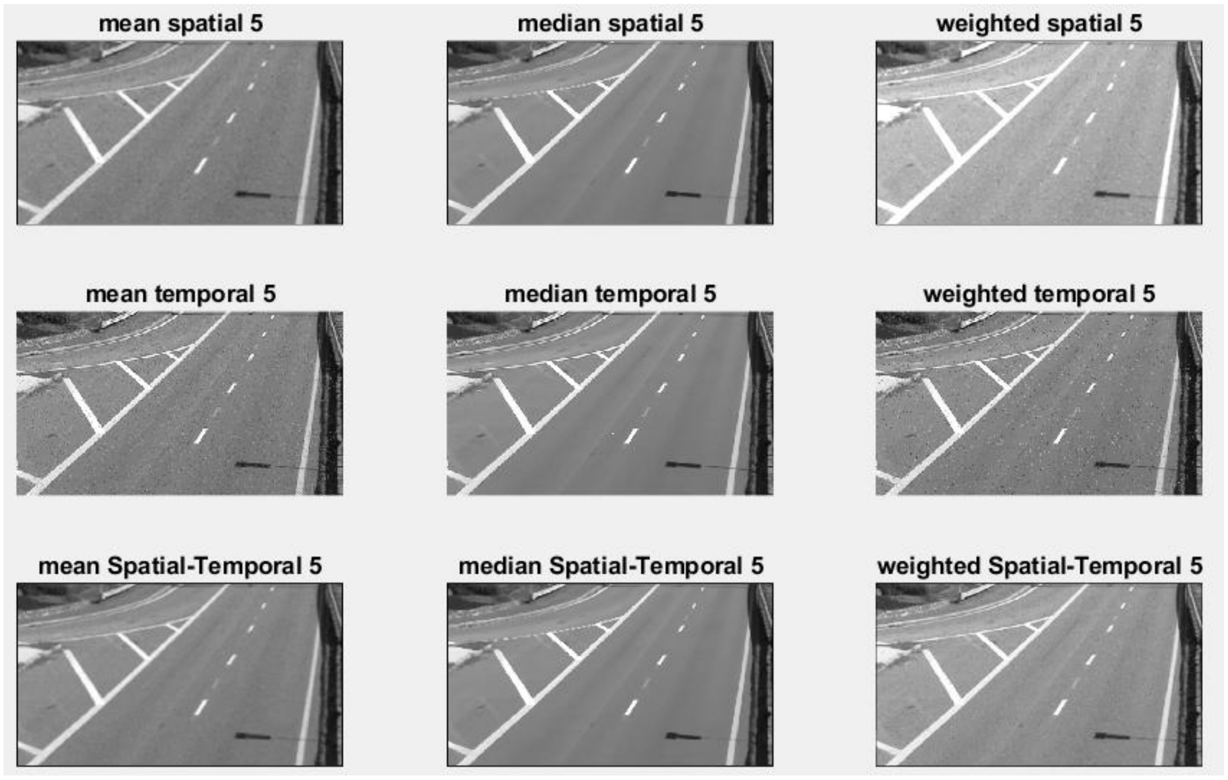

mean spatial 5
median spatial 5
weighted spatial 5
mean temporal 5
median temporal 5
weighted temporal 5
mean Spatial-Temporal 5
median Spatial-Temporal 5
weighted Spatial-Temporal 5


original
salt-pepper-nosied


Mean Spatial 3
MSE: 216.880947, PSNR: 24.768590 dB
mean spatial 5
MSE: 364.878380, PSNR: 22.509322 dB
Median Spatial 3
MSE: 161.863114, PSNR: 26.039325 dB
median spatial 5
MSE: 322.837798, PSNR: 23.040960 dB
Weighted Spatial 3
MSE: 211.379371, PSNR: 24.880178 dB
weighted spatial 5
MSE: 2056.583918, PSNR: 14.999339 dB
Mean Temporal 3
MSE: 125.009072, PSNR: 27.161388 dB
mean temporal 5
MSE: 265.388109, PSNR: 23.891989 dB
Median Temporal 3
MSE: 20.778632, PSNR: 34.954634 dB
median temporal 5
MSE: 18.348435, PSNR: 35.494813 dB


Weighted Temporal 3
MSE: 140.230845, PSNR: 26.662368 dB
weighted temporal 5
MSE: 105.373411, PSNR: 27.903493 dB
Mean Spatial-Temporal 3
MSE: 548.350864, PSNR: 20.740218 dB
mean Spatial-Temporal 5
MSE: 1057.854373, PSNR: 17.886545 dB
Median Spatial-Temporal 3
MSE: 492.298963, PSNR: 21.208514 dB
median Spatial-Temporal 5
MSE: 1024.197612, PSNR: 18.026966 dB
Weighted Spatial-Temporal 3
MSE: 457.412644, PSNR: 21.527722 dB
weighted Spatial-Temporal 5
MSE: 1127.649593, PSNR: 17.609062 dB





5. Discussion:

To evaluate the performance of the filters on **Gaussian noise** and **salt-and-pepper noise** with different kernel sizes (3x3 and 5x5), we need to consider how each filter behaves under these two types of noise. Below is a detailed comparison of the performance of the filters for both noise types and kernel sizes.

1. Gaussian Noise
Gaussian noise is characterized by random variations in pixel intensity, following a normal distribution. It affects all pixels in the image.
Spatial Filters

- **Mean Filter**:
    - **3x3 Kernel**: Reduces Gaussian noise effectively but introduces slight blurring.
    - **5x5 Kernel**: Reduces noise more aggressively but causes significant blurring and loss of fine details.
    - **Performance**: The 3x3 kernel is better for preserving details, while the 5x5 kernel is more effective at noise reduction but at the cost of blurring.
- **Median Filter**:
    - **3x3 Kernel**: Effectively reduces Gaussian noise while preserving edges and fine details.
    - **5x5 Kernel**: Reduces noise more effectively but may slightly blur edges.
    - **Performance**: The 3x3 kernel is better for preserving details, while the 5x5 kernel is more effective at noise reduction.
- **Weighted Filter**:
    - **3x3 Kernel**: Reduces noise while preserving more details than the mean filter.
    - **5x5 Kernel**: Reduces noise more effectively but introduces more blurring than the 3x3 kernel.
    - **Performance**: The 3x3 kernel is better for preserving details, while the 5x5 kernel is more effective at noise reduction.

Temporal Filters

- **Mean Temporal Filter**:
    - **3-Frame Kernel**: Reduces Gaussian noise but may introduce motion blur if there is significant motion between frames.
    - **5-Frame Kernel**: Reduces noise more effectively but introduces more motion blur.
    - **Performance**: The 3-frame kernel is better for preserving motion details.
- **Median Temporal Filter**:
    - **3-Frame Kernel**: Effectively reduces Gaussian noise while preserving motion details.
    - **5-Frame Kernel**: Reduces noise more effectively but may slightly blur motion details.
    - **Performance**: The 3-frame kernel is better for preserving motion details.
- **Weighted Temporal Filter**:
    - **3-Frame Kernel**: Reduces noise while preserving more motion details than the mean temporal filter.
    - **5-Frame Kernel**: Reduces noise more effectively but introduces more motion blur.
    - **Performance**: The 3-frame kernel is better for preserving motion details.

Spatial-Temporal Filters

- **Mean Spatial-Temporal Filter**:
    - **3x3 Kernel**: Reduces Gaussian noise effectively but introduces blurring.
    - **5x5 Kernel**: Reduces noise more effectively but causes significant blurring.
    - **Performance**: The 3x3 kernel is better for preserving details.

- **Median Spatial-Temporal Filter**:
    - **3x3 Kernel**: Effectively reduces Gaussian noise while preserving edges and motion details.
    - **5x5 Kernel**: Reduces noise more effectively but may slightly blur details.
    - **Performance**: The 3x3 kernel is better for preserving details.
- **Weighted Spatial-Temporal Filter**:
    - **3x3 Kernel**: Reduces noise while preserving more details than the mean Spatial-temporal filter.
    - **5x5 Kernel**: Reduces noise more effectively but introduces more blurring.
    - **Performance**: The 3x3 kernel is better for preserving details.

---

2. Salt-and-Pepper Noise

Salt-and-pepper noise is characterized by random occurrences of black (pepper) and white (salt) pixels in the image. It affects only a subset of pixels.

Spatial Filters

- **Mean Filter**:
    - **3x3 Kernel**: Less effective at removing salt-and-pepper noise, as it averages the noisy pixels with their neighbours, which can spread the noise.
    - **5x5 Kernel**: Slightly better at reducing noise but introduces significant blurring.
    - **Performance**: Poor for salt-and-pepper noise.
- **Median Filter**:
    - **3x3 Kernel**: Highly effective at removing salt-and-pepper noise while preserving edges and details.
    - **5x5 Kernel**: Even more effective at removing noise but may slightly blur edges.
    - **Performance**: The 3x3 kernel is better for preserving details, while the 5x5 kernel is more effective at noise reduction.
- **Weighted Filter**:
    - **3x3 Kernel**: Less effective than the median filter for salt-and-pepper noise.
    - **5x5 Kernel**: Slightly better at reducing noise but introduces more blurring.
    - **Performance**: Poor for salt-and-pepper noise compared to the median filter.

Temporal Filters

- **Mean Temporal Filter**:
    - **3-Frame Kernel**: Less effective at removing salt-and-pepper noise, as it averages the noisy pixels with their temporal neighbours.
    - **5-Frame Kernel**: Slightly better at reducing noise but introduces more motion blur.
    - **Performance**: Poor for salt-and-pepper noise.
- **Median Temporal Filter**:
    - **3-Frame Kernel**: Highly effective at removing salt-and-pepper noise while preserving motion details.
    - **5-Frame Kernel**: Even more effective at removing noise but may slightly blur motion details.
    - **Performance**: The 3-frame kernel is better for preserving motion details.
- **Weighted Temporal Filter**:
    - **3-Frame Kernel**: Less effective than the median temporal filter for salt-and-pepper noise.

    - **5-Frame Kernel**: Slightly better at reducing noise but introduces more motion blur.
    - **Performance**: Poor for salt-and-pepper noise compared to the median filter.

Spatial-Temporal Filters

- **Mean Spatial-Temporal Filter**:
    - **3x3 Kernel**: Less effective at removing salt-and-pepper noise.
    - **5x5 Kernel**: Slightly better at reducing noise but introduces significant blurring.
    - **Performance**: Poor for salt-and-pepper noise.
- **Median Spatial-Temporal Filter**:
    - **3x3 Kernel**: Highly effective at removing salt-and-pepper noise while preserving edges and motion details.
    - **5x5 Kernel**: Even more effective at removing noise but may slightly blur details.
    - **Performance**: The 3x3 kernel is better for preserving details.
- **Weighted Spatial-Temporal Filter**:
    - **3x3 Kernel**: Less effective than the median Spatial-temporal filter for salt-and-pepper noise.
    - **5x5 Kernel**: Slightly better at reducing noise but introduces more blurring.
    - **Performance**: Poor for salt-and-pepper noise compared to the median filter.

Comparison and Best Filters

For Gaussian Noise:

- Best Spatial Filter: Median Spatial Filter (3x3).
- Best Temporal Filter: Median Temporal Filter (3-Frame).
- Best Spatial-Temporal Filter: Median Spatial-Temporal Filter (3x3 -3frame**)**.

For Salt-and-Pepper Noise:

- Best Spatial Filter: Median Spatial Filter (3x3).
- Best Temporal Filter: Median Temporal Filter (3-Frame).
- Best Spatial-Temporal Filter: Median Spatial-Temporal Filter (3x3-3frame).

Kernel Size Comparison:

- **3x3 Kernels**: Better at preserving details and edges for both noise types.
- **5x5 Kernels**: More effective at noise reduction but introduce more blurring.

- **Median Filters**: Outperform mean and weighted filters for both Gaussian and salt-and-pepper noise.
- **3x3 Kernels**: Better for preserving details and edges.
- **5x5 Kernels**: More effective at noise reduction but at the cost of blurring.
- **Spatial-Temporal Filters**: Provide the best overall performance by leveraging both spatial and temporal redundancy.

6. Conclusion:

The experiment demonstrated that median filters consistently outperform mean and weighted filters for both Gaussian and salt-and-pepper noise, due to their ability to preserve edges and effectively remove outliers. The 3x3 kernel size proved superior to 5x5 for preserving fine details, despite slightly lower noise reduction. For Gaussian noise, the median spatial-temporal filter (3x3) delivered the best results by combining spatial and temporal filtering. For salt-and-pepper noise, the median spatial filter (3x3) was most effective. Overall, spatio-temporal filters provided the best balance of noise reduction and detail preservation, making them ideal for video denoising. The choice of filter and kernel size should align with the noise type and the need for detail preservation.

7. Code:

```
videoFile = 'visiontraffic.avi';
vidObj = VideoReader(videoFile);
frameRate = vidObj.FrameRate;
numFrames = 60;   rows = vidObj.Height;   cols = vidObj.Width;
originalFrames = zeros(rows, cols, numFrames, 'uint8');
noisyFrames = zeros(rows, cols, numFrames, 'uint8');
mean_spatial_3 = zeros(rows, cols, numFrames, 'uint8');
median_spatial_3= zeros(rows, cols, numFrames, 'uint8');
weighted_spatial_3 = zeros(rows, cols, numFrames, 'uint8');
mean_temporal_3 = zeros(rows, cols, numFrames, 'uint8');
median_temporal_3= zeros(rows, cols, numFrames, 'uint8');
weighted_temporal_3 = zeros(rows, cols, numFrames, 'uint8');
mean_s_t_3 = zeros(rows, cols, numFrames, 'uint8');
median_s_t_3= zeros(rows, cols, numFrames, 'uint8');
weighted_s_t_3 = zeros(rows, cols, numFrames, 'uint8');
mean_spatial_5 = zeros(rows, cols, numFrames, 'uint8');
median_spatial_5= zeros(rows, cols, numFrames, 'uint8');
weighted_spatial_5 = zeros(rows, cols, numFrames, 'uint8');
mean_temporal_5 = zeros(rows, cols, numFrames, 'uint8');
median_temporal_5= zeros(rows, cols, numFrames, 'uint8');
weighted_temporal_5 = zeros(rows, cols, numFrames, 'uint8');
mean_s_t_5 = zeros(rows, cols, numFrames, 'uint8');
median_s_t_5= zeros(rows, cols, numFrames, 'uint8');
weighted_s_t_5 = zeros(rows, cols, numFrames, 'uint8');
for i = 1:numFrames
   originalFrames(:, :,i ) = rgb2gray(read(vidObj, i));
   noisyFrames(:, :, i) = imnoise(originalFrames(:, :, i), 'gaussian', 0, 0.02);
end
for i = 1: numFrames
   frame = noisyFrames(:,:,i);
   [rows, cols] = size(frame);
   weighted_kernel = [1,1,1; 1,2,1; 1,1,1] / 10;
   for r = 2:rows-1
      for c= 2:cols-1
         window = frame(r-1:r+1, c-1:c+1);
         mean_spatial_3(r,c,i) = mean(window(:));
         median_spatial_3(r,c,i) = median(window(:));
         weighted_spatial_3(r, c, i) = sum(sum(double(window) .* double(weighted_kernel)));
      end
   end
```

```
end
for i = 1: numFrames
   frame = noisyFrames(:,:,i);
   weighted_kernel_5x5 = [1,1,1,1,1; 1,1,2,1,1; 1,2,4,2,1; 1,1,2,1,1; 1,1,1,1,1] / 24;
   for r = 3:rows-2
      for c= 3:cols-2
         window = frame(r-2:r+2, c-2:c+2);
         mean_spatial_5(r,c,i) = mean(window(:));
         median_spatial_5(r,c,i) = median(window(:));
         weighted_spatial_5 (r, c, i) = sum(sum(double(window) .*
double(weighted_kernel_5x5)));
      end
   end
end
mean_temporal_3(:,:,1)= noisyFrames(:,:,1);
median_temporal_3(:,:,1)= noisyFrames(:,:,1);
weighted_temporal_3(:,:,1)= noisyFrames(:,:,1);
for k = 2:numFrames-1
    mean_temporal_3(:,:, k) = 0.33*noisyFrames(:,:,k-1) + 0.33*noisyFrames(:,:,k)+
0.33*noisyFrames(:,:,k+1);
    median_temporal_3(:,:, k) = median(noisyFrames(:,:, k-1:k+1), 3);
    weighted_temporal_3(:,:, k) = 0.25*noisyFrames(:,:,k-1) + 0.5*noisyFrames(:,:,k)+
0.25*noisyFrames(:,:,k+1);

end
mean_temporal_3(:,:,numFrames)= noisyFrames(:,:,numFrames);
median_temporal_3(:,:,numFrames)= noisyFrames(:,:,numFrames);
weighted_temporal_3(:,:,numFrames)= noisyFrames(:,:,numFrames);
mean_temporal_5(:,:,1)= noisyFrames(:,:,1);median_temporal_5(:,:,1)=
noisyFrames(:,:,1);weighted_temporal_5(:,:,1)= noisyFrames(:,:,1);
mean_temporal_5(:,:,1)= noisyFrames(:,:,1);median_temporal_5(:,:,2)=
noisyFrames(:,:,2);weighted_temporal_5(:,:,2)= noisyFrames(:,:,1);
for k = 3:numFrames-2
    mean_temporal_5(:,:, k) = 0.2*noisyFrames(:,:,k-2) + 0.2*noisyFrames(:,:,k-1)+
0.2*noisyFrames(:,:,k)+0.2*noisyFrames(:,:,k+1)+ 0.2*noisyFrames(:,:,k+2);
    median_temporal_5(:,:, k) = median(noisyFrames(:,:, k-2:k+2), 3);
    weighted_temporal_5(:,:, k) = 0.1*noisyFrames(:,:,k-2) + 0.2*noisyFrames(:,:,k-1)+
0.4*noisyFrames(:,:,k)+0.2*noisyFrames(:,:,k+1)+0.1*noisyFrames(:,:,k+2);
end
mean_temporal_5(:,:,numFrames-1)= noisyFrames(:,:,numFrames-
1);median_temporal_5(:,:,numFrames-1)= noisyFrames(:,:,numFrames-
1);weighted_temporal_5(:,:,numFrames-1)= noisyFrames(:,:,numFrames-1);
mean_temporal_5(:,:,numFrames)=
noisyFrames(:,:,numFrames);median_temporal_5(:,:,numFrames)=
noisyFrames(:,:,numFrames);weighted_temporal_5(:,:,numFrames)=
noisyFrames(:,:,numFrames);
```

```
for i =  2: numFrames-1
  for r = 2 : rows-1
     for c = 2: cols-1
       median_s_t_3(r, c, i) = median(median(median(cat(3, ...
       noisyFrames(r-1:r+1, c-1:c+1, i-1), ...
       noisyFrames(r-1:r+1, c-1:c+1, i), ...
       noisyFrames(r-1:r+1, c-1:c+1, i+1)), 3)));
     end
  end
end
for i = 2: numFrames-1
    weighted_s_t_3(:,:,i) = 0.25* noisyFrames(:,:,i-1)+ 0.5*
weighted_spatial_3(:,:,i)+0.25*noisyFrames(:,:,i+1);
     mean_s_t_3(:,:,i) = 0.33*mean_spatial_3(:,:,i-1) + 0.33*mean_spatial_3(:,:,i)+
0.33*mean_spatial_3(:,:,i+1);
end
for i =  3: numFrames-2
   mean_s_t_5(:,:,i) = 0.2*mean_spatial_5(:,:,i-2)+0.2*mean_spatial_5(:,:,i-1) +
0.2*mean_spatial_5(:,:,i)+ 0.2*mean_spatial_5(:,:,i+1)+0.2*mean_spatial_5(:,:,i-2);
   weighted_s_t_5(:,:,i) = 0.1* noisyFrames(:,:,i-2)+ 0.2* weighted_spatial_3(:,:,i-1)+0.4*
weighted_spatial_5(:,:,i)+ 0.2* weighted_spatial_3(:,:,i+1)+0.1*noisyFrames(:,:,i+2);
end
for i =  3: numFrames-2
  for r = 3 : rows-2
     for c = 3: cols-2
      median_s_t_5(r, c, i) = median(median(median(cat(3, ...
      noisyFrames(r-2:r+2, c-2:c+2, i-2), ...
      noisyFrames(r-2:r+2, c-2:c+2, i-1), ...
      noisyFrames(r-2:r+2, c-2:c+2, i), ...
      noisyFrames(r-2:r+2, c-2:c+2, i+1), ...
      noisyFrames(r-2:r+2, c-2:c+2, i+2)), 3), 'all'));
     end
  end
end
filter_list_3 = {mean_spatial_3,median_spatial_3,
weighted_spatial_3,mean_temporal_3,median_temporal_3,
weighted_temporal_3,mean_s_t_3,median_s_t_3, weighted_s_t_3};
filter_list_5 = {mean_spatial_5, median_spatial_5, weighted_spatial_5,mean_temporal_5,
median_temporal_5, weighted_temporal_5,mean_s_t_5, median_s_t_5, weighted_s_t_5};
mse_values_3 = zeros(numFrames, 9);
psnr_values_3 = zeros(numFrames, 9);
mse_values_5 = zeros(numFrames, 9);
psnr_values_5 = zeros(numFrames, 9);

for i = 1:9
```

```
    for k = 1:numFrames
        mse_values_3(k,i) = mean((double(originalFrames(:,:,k)) -
double(filter_list_3{i}(:,:,k))).^2, 'all');
        mse_values_5(k,i) = mean((double(originalFrames(:,:,k)) -
double(filter_list_5{i}(:,:,k))).^2, 'all');
        if mse_values_3(k,i) == 0
            psnr_values_3(k,i) = Inf; % Avoid log(0)
        else
            psnr_values_3(k,i) = 10 * log10(255^2 / mse_values_3(k,i));
        end
        if mse_values_5(k,i) == 0
            psnr_values_5(k,i) = Inf;
        else
            psnr_values_5(k,i) = 10 * log10(255^2 / mse_values_5(k,i));
        end
    end
end

filter_name_3 = {'Mean Spatial 3', 'Median Spatial 3', 'Weighted Spatial 3', 'Mean Temporal
3', 'Median Temporal 3', 'Weighted Temporal 3', Spatial-Temporal 3', 'Median Spatial-
Temporal 3', 'Weighted Spatial-Temporal 3'};
filter_name_5 = {'mean_spatial_5','median_spatial_5',
'weighted_spatial_5','mean_temporal_5', 'median_temporal_5',
'weighted_temporal_5','mean_Spatial-Temporal_5','median_Spatial-
Temporal_5','weighted_Spatial-Temporal_5'};

figure();
subplot(1,2,1); imshow(originalFrames(:,:,5)); title("original");
subplot(1,2,2); imshow(noisyFrames(:,:,5)); title("gaussian_nosied");
figure();
for i = 1: 5
    avg_mse_3 = mean(mse_values_3(:,i));
    avg_mse_5 = mean(mse_values_5(:,i));
    avg_psnr_3 = 10 * log10(255^2 / avg_mse_3);
    avg_psnr_5 =  10 * log10(255^2 / avg_mse_5);
    subplot(5,2,2*i-1); plot(1:numFrames, psnr_values_3(:,i), '-');
    title(sprintf('%s\nMSE: %f, PSNR: %f dB', strrep(filter_name_3{i}, '_', ' '),avg_mse_3 ,
avg_psnr_3), 'FontSize', 10);
    subplot(5,2,2*i); plot(1:numFrames, psnr_values_5(:,i), '-');
    title(sprintf('%s\nMSE: %f, PSNR: %f dB', strrep(filter_name_5{i}, '_', ' '),avg_mse_5 ,
avg_psnr_5), 'FontSize', 10);
end
figure()
for i = 6 :9
    avg_mse_3 = mean(mse_values_3(:,i));
    avg_mse_5 = mean(mse_values_5(:,i));
    avg_psnr_3 = 10 * log10(255^2 / avg_mse_3);
    avg_psnr_5 =  10 * log10(255^2 / avg_mse_5);
    subplot(4,2,2*i-11); plot(1:numFrames, psnr_values_3(:,i), '-');
    title(sprintf('%s\nMSE: %f, PSNR: %f dB', strrep(filter_name_3{i}, '_', ' '),avg_mse_3 ,
avg_psnr_3), 'FontSize', 10);
    subplot(4,2,2*i-10); plot(1:numFrames, psnr_values_5(:,i), '-');
    title(sprintf('%s\nMSE: %f, PSNR: %f dB', strrep(filter_name_5{i}, '_', ' '),avg_mse_5 ,
avg_psnr_5), 'FontSize', 10);
end
figure();
for f = 1:numel(filter_list_3)
    subplot(3, 3, f);
    imshow(filter_list_3{f}(:,:,5));   % Convert string to variable
    title(strrep(filter_name_3{f}, '_', ' '), 'FontSize', 10);  % Replace underscores with spaces
for better readability
end
figure();
for f = 1:numel(filter_list_5)
    subplot(3, 3, f);
    imshow(filter_list_5{f}(:,:,5));   % Convert string to variable
    title(strrep(filter_name_5{f}, '_', ' '), 'FontSize', 10); % Replace underscores with spaces
for better readability
end
```