Table of Contents

Jeffrey Wong | ECE-435 | Project #1- Imaging Fundamentals

```
% Function definitions are at the bottom of the file. Check them out- they
% contain some important context for how functions are called and some
% derivations!
clear
close all
clc
```

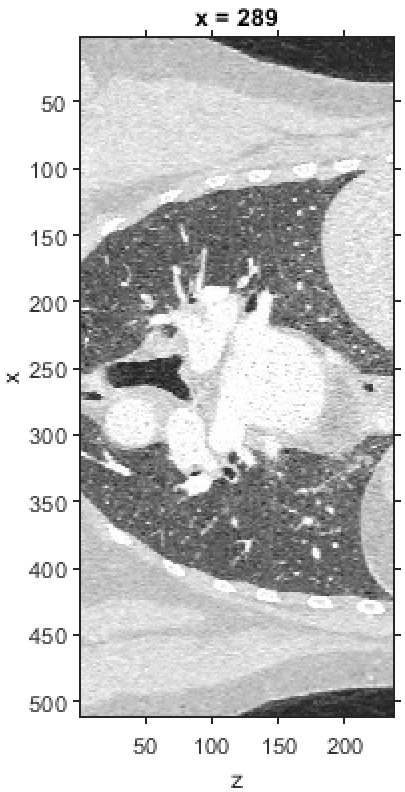
Notes on Scans

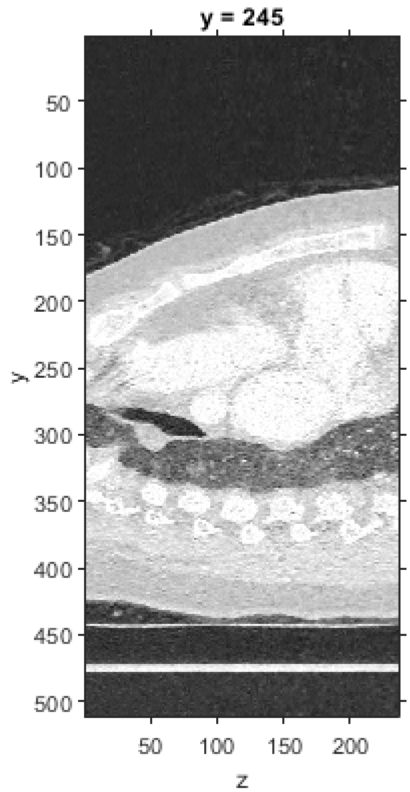
```
% CT Thorax
% Pixel Spacing 0.703 mm 0028,0030;
xyspacing = 0.703; % Spacings used in Gaussian Blur
% Slice Thickness 1.25 mm 0018,0050;
% Spacing Between Slides 0.625 mm 0018,0088;
zspacing = 1.25 + 0.625; % Spacing in z impacted by both slice thickness and
 spacing
% Number of slices 237
num_slices = 237;
```

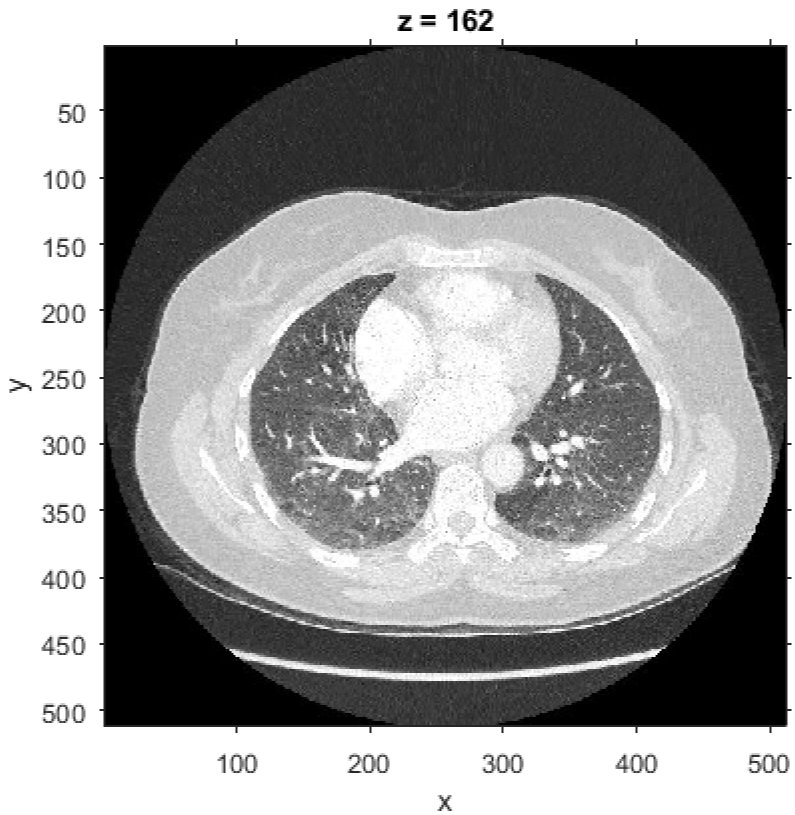
Part 1: Construction and Slicing

```
scan_size = size(rgb2gray(imread("thoraxCT\axial00001.jpg")));
volume = zeros([scan_size num_slices]); % Identifies size of scans and makes
 volume
for i = 1:num_slices
    % Hacky workaround to scan images, had to look up string formatting
    slice = rgb2gray(imread("thoraxCT\axial" + num2str(i, "%05d") + ".jpg"));
    % When passing in doubles value needs to be normalized between 0 and 1
    volume(:,:,i) = double(slice)/255;
end

% Configurable coordinates of slices
xloc = 289;
yloc = 245;
zloc = 162;
```

```
% Squeezing keeps stuff to two dimensions
xslice = squeeze(volume(xloc,:,:));
yslice = squeeze(volume(:,yloc,:));
zslice = squeeze(volume(:,:,zloc));

% x slice
figure
imshow(xslice)
axis on % Axis values have to be manually enabled for some reason
title("x = " + xloc)
xlabel("z")
ylabel("x")

% y slice
figure
imshow(yslice)
axis on
title("y = " + yloc)
xlabel("z")
ylabel("y")

% z slice
figure
imshow(zslice)
axis on
title("z = " + zloc)
xlabel("x")
ylabel("y")

% The units on the axes are just pixels. According to the scan
% specifications, each pixel in the x-y plane is 0.703mm x 0.703mm, giving
% the z-scan an aspect ratio of 1:1 and overall dimensions of 360mm*360mm,
% while each pixel in the x-z and y-z planes would be 0.703mm x 1.25mm,
% giving an overall aspect ratio of (512*0.703):(237*1.25) or about 6:5 and
% overall dimensions of 360mm*296mm.
```


x = 289
50
100
150
200
250
300
350
400
450
500
x
50
100
150
200
z

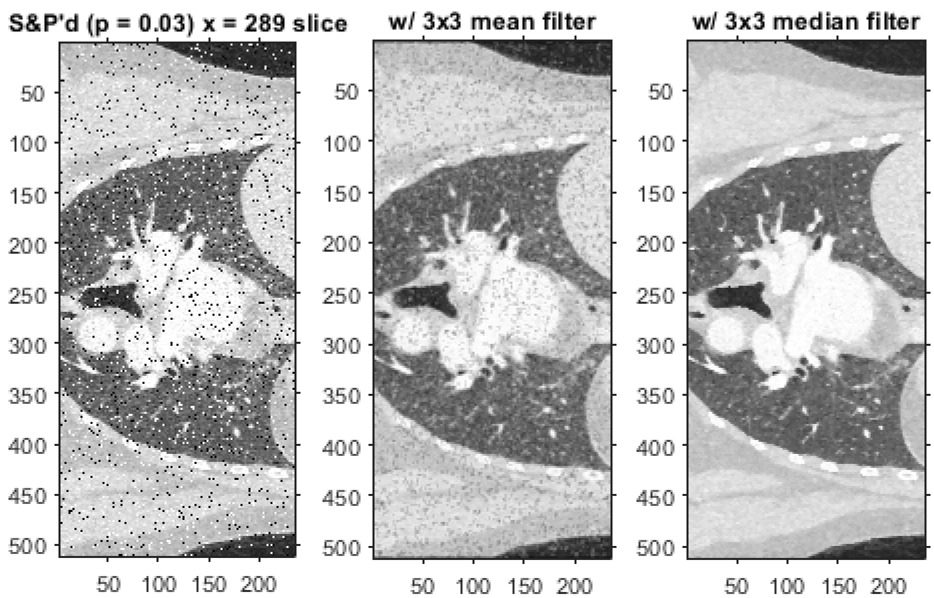

y = 245
y
z

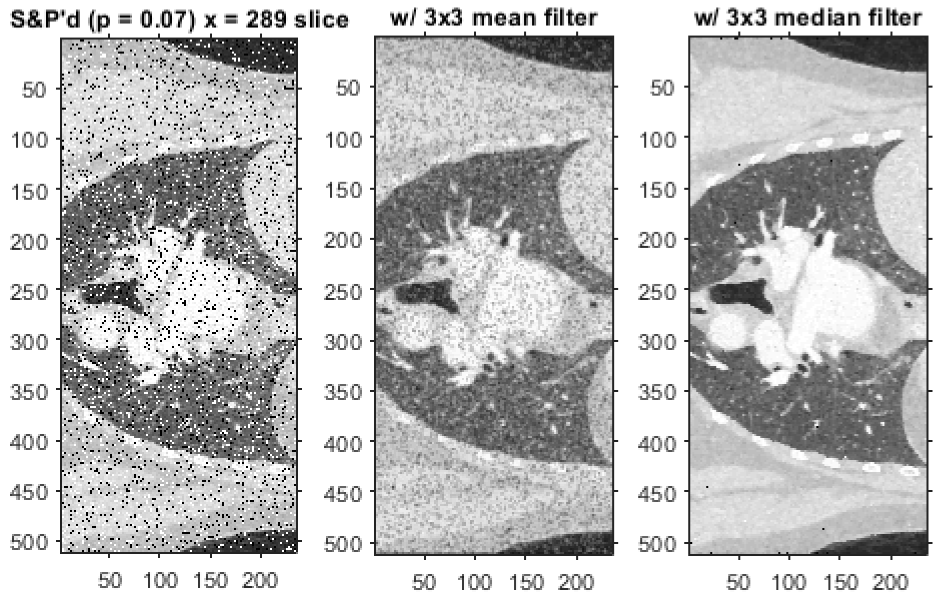


Part 2: Sal y Pimienta

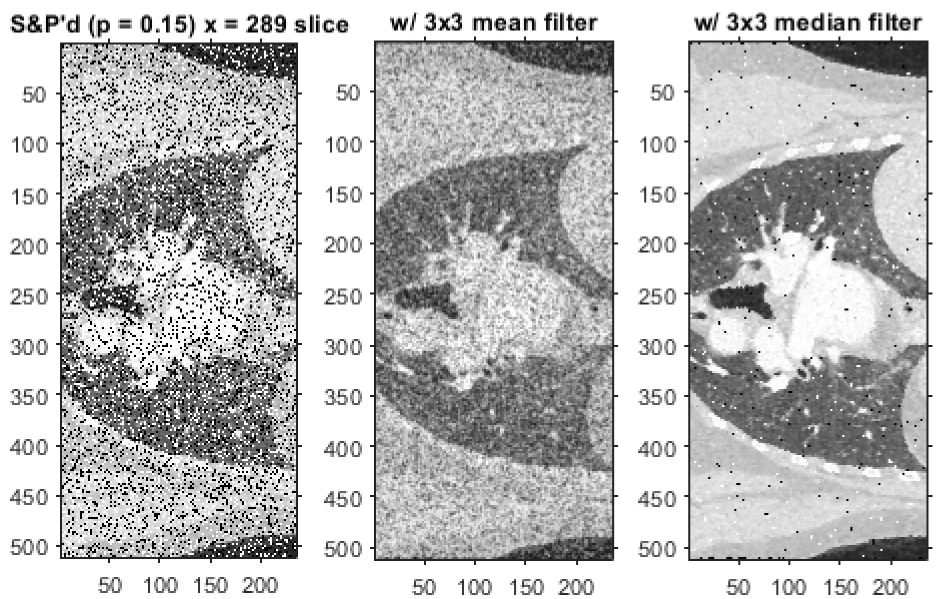
```
figure
subplot(1,2,1)
imshow(imfilter(xslice, ones(3)/9))
axis on
title("x = " + xloc + " slice w/ 3x3 mean filter")
subplot(1,2,2)
imshow(medfilt2(xslice, [3 3]))
axis on
title("x = " + xloc + " slice w/ 3x3 median filter")

figure
subplot(1,2,1)
imshow(imfilter(xslice, ones(5)/25))
axis on
title("x = " + xloc + " slice w/ 5x5 mean filter")
subplot(1,2,2)
imshow(medfilt2(xslice, [5 5]))
axis on
title("x = " + xloc + " slice w/ 5x5 median filter")

% The original slice did not exhibit too much noise to begin with, so the
% choice of a mean or median filter had little effect given the same
% neighborhood size. As expected, the 5x5 mean and median filters both
```

```
% eliminated more salt+pepper (especially on the right-hand light regions)
% and caused more blurring than the corresponding 3x3 filters.

salt_and_pepper_filtered(xslice, xloc, 0.03)
salt_and_pepper_filtered(xslice, xloc, 0.07)
salt_and_pepper_filtered(xslice, xloc, 0.15)

% The artificially introduced salt and pepper noise caused issues at all
% three probabilites shown above with mean filtering, with severe effects
% as early as p = 0.07. In contrast, the salt and pepper noise is almost
% entirely cleaned up by the median filters at p = 0.03 and 0.07, and only
% causes noticeable artifacts at p = 0.15, which appeared equivalent to the
% artifacts left by the mean filter at p = 0.03.
```





S&P'd (p = 0.07) x = 289 slice
w/ 3x3 mean filter
w/ 3x3 median filter

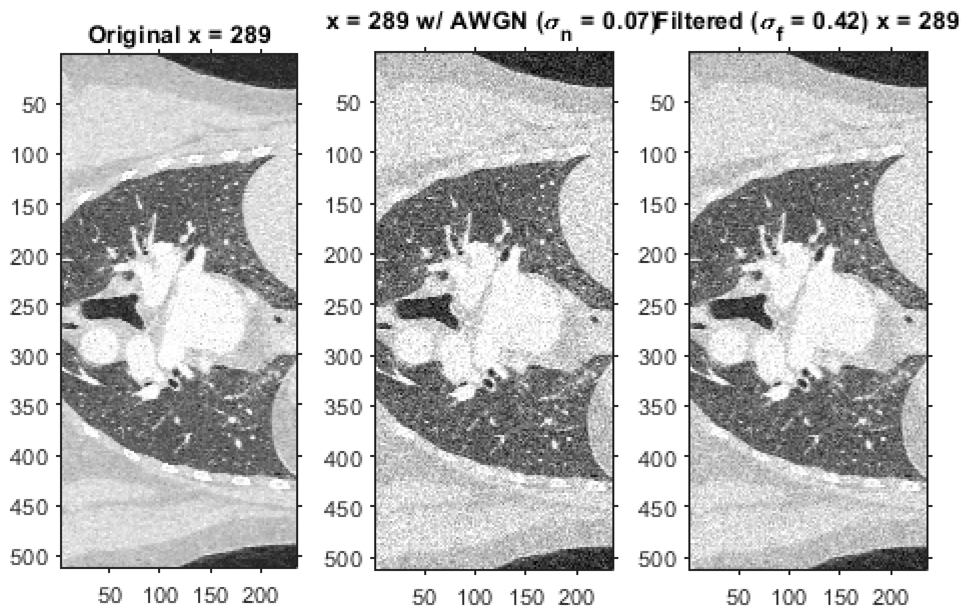


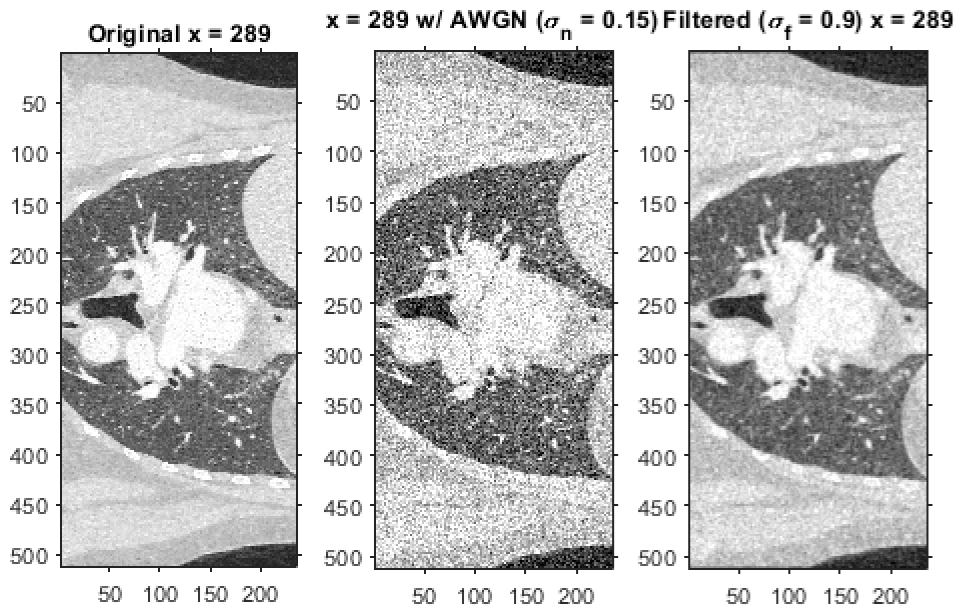
S&P'd (p = 0.15) x = 289 slice
w/ 3x3 mean filter
w/ 3x3 median filter

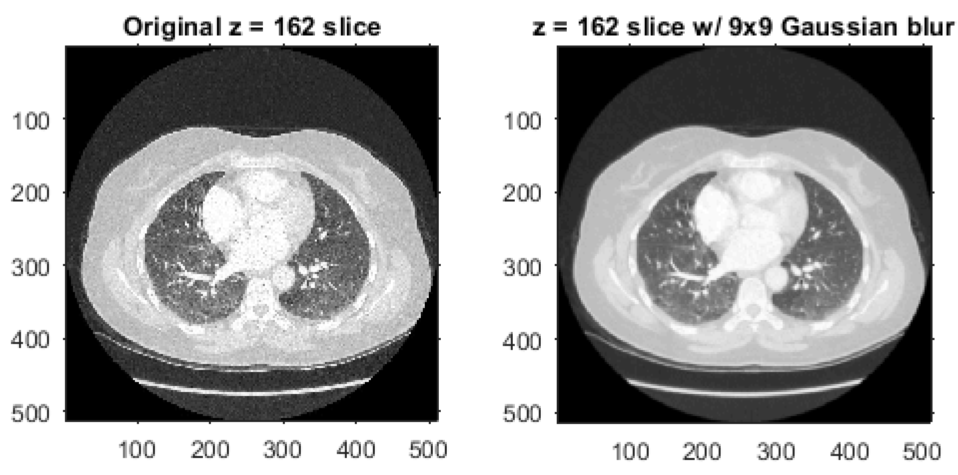
Part 3: Gauss' Wrath

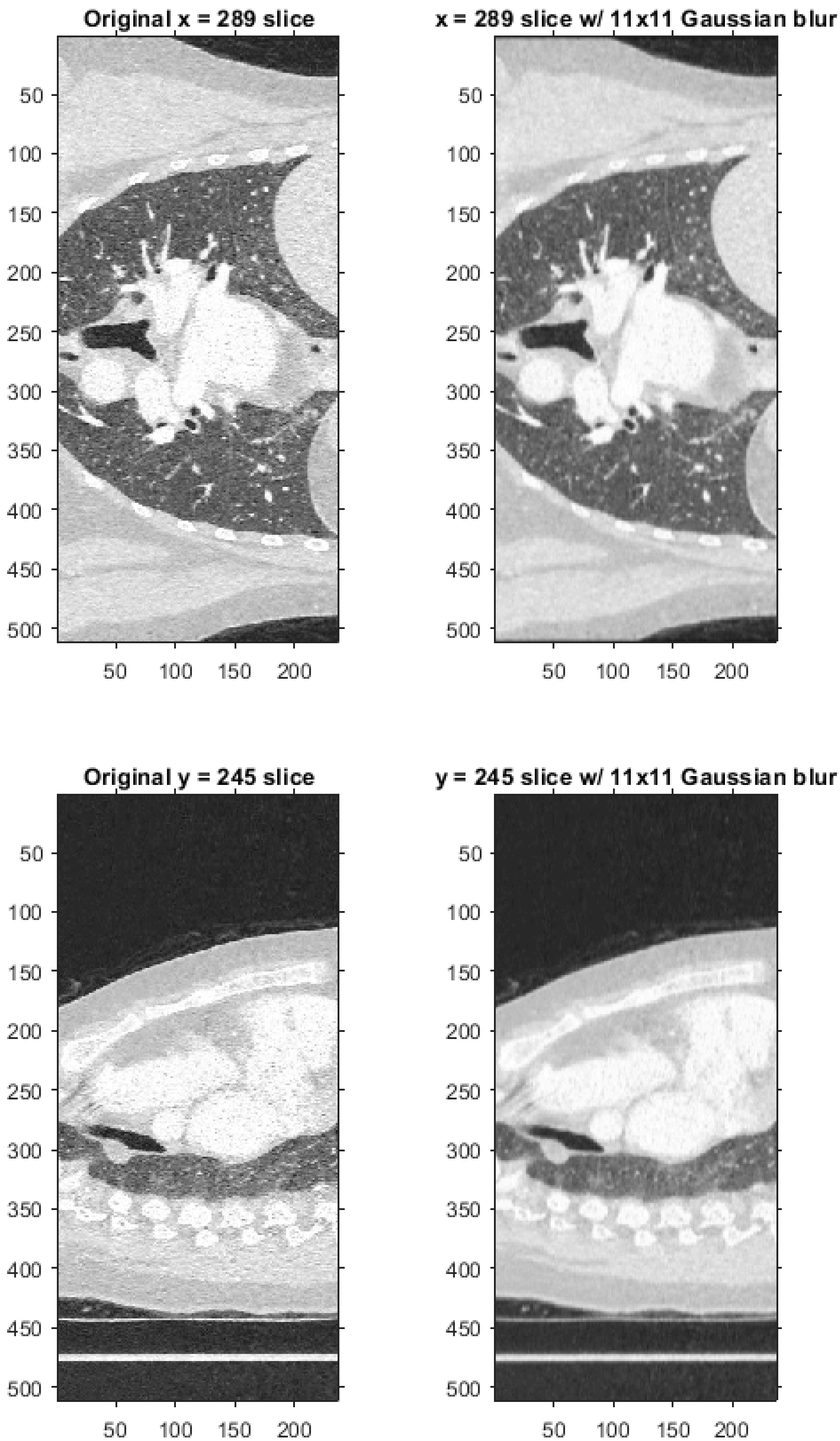
```
% Gaussian Noise
% sigma_n (Noise stdev) = 0.07, sigma_f (Filter stdev) = 0.42
gaussian_filtered(xslice, xloc, 0.07, 0.42)
% sigma_n = 0.15, sigma_f = 0.9
gaussian_filtered(xslice, xloc, 0.15, 0.9)

% It seems that beyond sigma_f = 1 the image becomes becomes very blurred,
% likely because the filter becomes too wide, which limits how much we can
% eliminate noise with a higher sigma_n such as the 0.15 used in the second
% case. The best values for sigma_f were 0.42 for sigma_n=0.07, and 0.9 for
% sigma_n=0.15 (both 6*sigma_n). These values for sigma_f tend to be much
% larger than the noise standard deviation,

% Generally, the filters did a good job blurring the dark regions and the
% extremely bright center but did poorly on the light gray regions towards
% the outside of the circle. Trying to get rid of the noise in those
% regions tends to cause more harm than good, however, as it tends to lead
% to excessive smoothing if you try to pump up sigma_f, especially if the
% initial noise power is already high.

% Gaussian Blur

% See the generate_Gaussian_kernel function for the derivation of how the
% kernel is generated

n = 5; % Affects kernel size
kersize = 2*n + 1;

% z image
% "Easy" because of 1:1 aspect ratio, the stdevs for x and y will be the
% same, resulting in isotropic Gaussian blur
% Now assume x/y/z are pixel offsets rather than distance offsets

dx = 2;
z_ker = generate_Gaussian_kernel(n, xyspacing, xyspacing, dx, dx);
figure
subplot(1,2,1)
imshow(zslice)
axis on
title("Original z = " + zloc + " slice")
subplot(1,2,2)
imshow(conv2(zslice, z_ker, "same"))
axis on
title("z = " + zloc + " slice w/ 9x9 Gaussian blur")

% x / y images
% Now our aspect ratio is not 1:1, so our Gaussian blur will be anisotripic

dz = 1;
xy_ker = generate_Gaussian_kernel(n, zspacing, xyspacing, dz, dx);
```

```
figure
subplot(1,2,1)
imshow(xslice)
axis on
title("Original x = " + xloc + " slice")
subplot(1,2,2)
imshow(conv2(xslice, xy_ker, "same"))
axis on
title("x = " + xloc + " slice w/ "+kersize+"x"+kersize+" Gaussian blur")

figure
subplot(1,2,1)
imshow(yslice)
axis on
title("Original y = " + yloc + " slice")
subplot(1,2,2)
imshow(conv2(yslice, xy_ker, "same"))
axis on
title("y = " + yloc + " slice w/ "+kersize+"x"+kersize+" Gaussian blur")

% Yup, those are definitely some blurred images. This is particularly
% prevalent for the z-slice, where the fine detail in the center-left and
% center-right gray regions are smoothed away, as the kernel is more spread
% out for the z slice because the xy dimensions have both a lower pixel
% spacing and a higher FWHM, but a bit of blurring can still be seen on the
% other slices, although it mostly happens vertically
```




Original x = 289
x = 289 w/ AWGN (σ_n = 0.15)
Filtered (σ_f = 0.9) x = 289

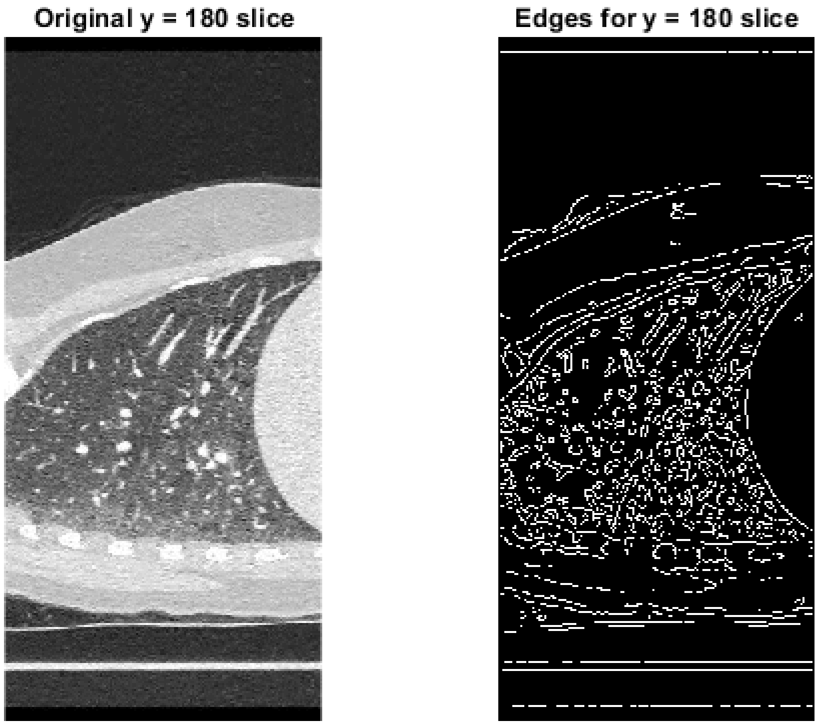

Original z = 162 slice
z = 162 slice w/ 9x9 Gaussian blur


Original x = 289 slice
x = 289 slice w/ 11x11 Gaussian blur
Original y = 245 slice
y = 245 slice w/ 11x11 Gaussian blur
50
100
150
200
250
300
350
400
450
500

Part 4: Foramina

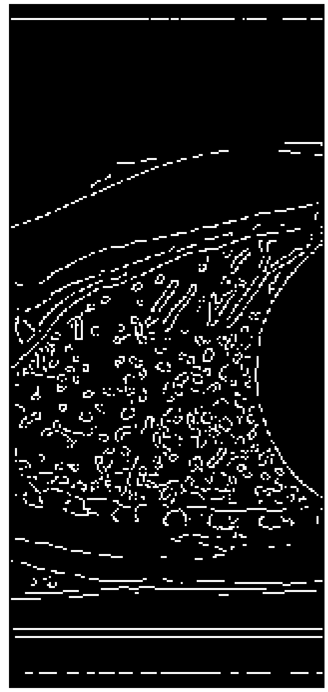
```
spine_slice = squeeze(volume(:,180,:));

% Edges of y-slice at y = 180
figure
subplot(1,2,1)
imshow(spine_slice)
title("Original y = 180 slice")
subplot(1,2,2)
imshow(edge(spine_slice, "canny"))
title("Edges for y = 180 slice")

% The vertebral disks can be seen in the lower third of the image as a
% series of large ovals, with the foramina above as a series of really
% small circles. Due to the noise in the center, there are quite a few
% spurious edges, with noise forming little "phantom" formamina and
% interfering with the disk, as well as adding some other extraneous edges
% above and below the sample that might be tied to other layers of the
% sample.

noisy_spine_slice = add_awgn(spine_slice, 0.07);
% Kernel for 3 pixel Gaussian filter
[P, Q] = meshgrid(-6:6, -6:6);
std3p_ker = exp(-P.^2/18 - Q.^2/18);
std3p_ker = std3p_ker/sum(std3p_ker, "all");

% Median + Part III Gaussian
figure
subplot(1,2,1)
imshow(edge(medfilt2(spine_slice, [3 3]), "canny"))
title("Edges for y = 180 slice w/ median filter")
subplot(1,2,2)
imshow(edge(conv2(spine_slice, z_ker, "same"), "canny"))
title("Edges for y = 180 slice w/ xz filter")

% AWGN + 3px Gaussian
figure
subplot(1,2,1)
imshow(edge(noisy_spine_slice, "canny"))
title("Edges for AWGN y = 180 slice")
subplot(1,2,2)
imshow(edge(conv2(spine_slice, std3p_ker, "same"), "canny"))
title("Edges for y = 180 slice w/ 3px filter")

% The median filter didn't have too much of an effect on the image, merely
% adding a few extra edges in the noisy rightmost section while removing
% some very small edges in the center-left. The xz Gaussian filter in part
% b also didn't caise too much of an effect, removing a few edges here and
% there but otherwise still showing the disks at the bottom of the middle
% third and most of the foramina in the center. Adding Gaussian noise, on
% the other hand, creates lots of high-gradient jumps and thus introduces a
% lot of spurous edges, while also breaking up the circular disks that were
```

```
% clear in the other pictures. The 3-pixel Gaussian filter in part d likely
% caused more smoothing than part b's xz Gaussian filter and has thus
% resulted in fewer overall edges being shown, with the rightmost discs
% being lost and the blobby regions in the center to bunch up and become
% larger.

% Bonus: Edges of z-slice at z = 42
figure
imshow(edge(zslice, "canny"))
title("Edges for z = 42 slice")
```

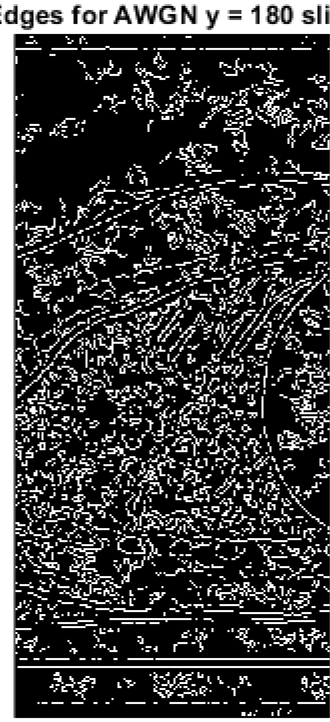


Edges for y = 180 slice w/ median filter

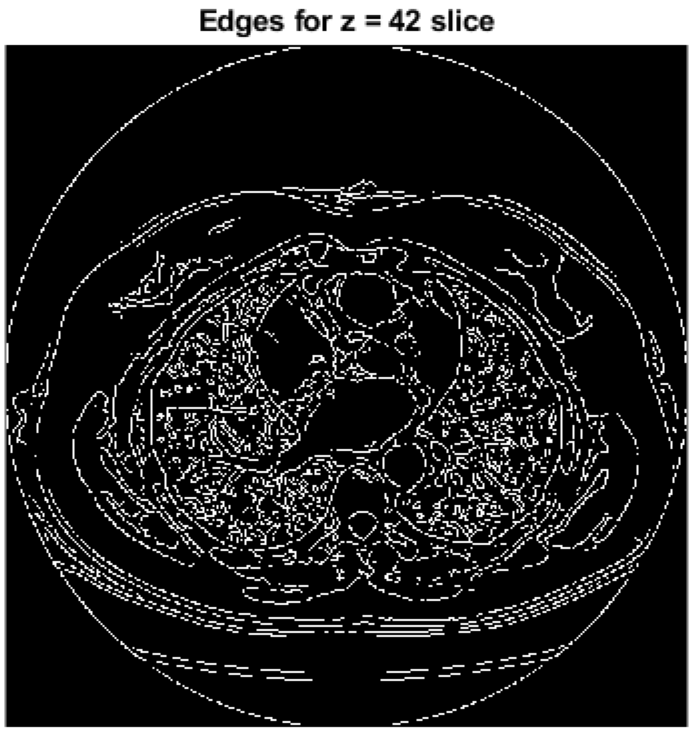
Edges for y = 180 slice w/ xz filter



Edges for AWGN y = 180 slice



Edges for y = 180 slice w/ 3px filter



Function Definitions

```
% Adds salt + pepper noise to an entire image
function output = add_sp_noise(x, p)
    mask = rand(size(x)); % Generates random numbers to map to salt+pepper
    output = x;
    output(mask < p) = 0;
    output(mask > 1-p) = 1;
end

% Salt + Peppers image, filters using 3x3 mean and median filter, and
% displays results!
function salt_and_pepper_filtered(slice, xloc, p)
    slice_sp = add_sp_noise(slice, p);
    figure
    subplot(1,3,1)
    imshow(slice_sp)
    axis on
    title("S&P'd (p = "+p+") x = " + xloc + " slice")
    subplot(1,3,2)
    imshow(imfilter(slice_sp, ones(3)/9))
    axis on
    title("w/ 3x3 mean filter")
    subplot(1,3,3)
    imshow(medfilt2(slice_sp, [3 3]))
```

```
    axis on
    title("w/ 3x3 median filter")
end

% Adds additive Gaussian noise to an image
function output = add_awgn(x, sigma)
    n = sigma * randn(size(x));
    output = x + n;
end

% Adds AWGN of specified sigma_n and applies Gaussian filter of specified
% sigma_f, then displays results!
function gaussian_filtered(slice, xloc, sigma_n, sigma_f)
    slice_awgn = add_awgn(slice, sigma_n);
    figure
    subplot(1,3,1)
    imshow(slice)
    axis on
    title("Original x = " + xloc)
    subplot(1,3,2)
    imshow(slice_awgn)
    axis on
    title("x = " + xloc + " w/ AWGN (\sigma_n = "+sigma_n+")")
    subplot(1,3,3)
    imshow(imgaussfilt(slice_awgn, sigma_f))
    axis on
    title("Filtered (\sigma_f = "+sigma_f+") x = " + xloc)
end

% Function to generate a 2n+1 by 2n+1 Gaussian blur kernel
function ker = generate_Gaussian_kernel(n, dim1_spacing, dim2_spacing,
 dim1_fwhm, dim2_fwhm)
    % Suppose we have a FWHM of dw mm in the w axis. Then our standard
    % deviation, sigma_w, is given by the following:

    % 0.5 = exp(-(dw/2)^2/(2*sigma_w)^2)
    % -ln(0.5) = ln(2) = (dw/2)^2/(2*sigma_w)^2
    % sigma_w^2 = dw^2/(8ln2)
    % sigma_w = dw/sqrt(8ln2)
    % The 1D Gaussian becomes P(w) = 1/sqrt((pi/(4ln2))*dw^2) * exp(-
(4w^2*ln2/dw^2))

    % The overall PSF is thus, for some constant alpha:
    % P(x,y,z) = alpha * exp(-(4x^2*ln2/dx^2)-(4y^2*ln2/dx^2)-(4z^2*ln2/dz^2))

    sigma_dim1 = dim1_fwhm/sqrt(8*log(2));
    sigma_dim2 = dim2_fwhm/sqrt(8*log(2));
    dim1_pts = (-n:n)*dim1_spacing;
    dim2_pts = (-n:n)*dim2_spacing;
    [dim1_grid, dim2_grid] = meshgrid(dim1_pts, dim2_pts);
    ker = exp(-(dim1_grid).^2/(2*sigma_dim1^2) - (dim2_grid).^2/
(2*sigma_dim2^2));
    ker = ker./sum(ker, "all"); % Normalize kernel to 1
end
```

Published with MATLAB® R2022b