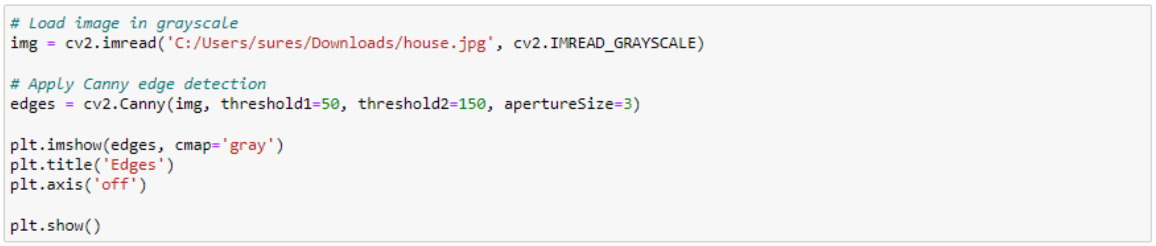
CSC 482
Assignment 4
Hithesh Shanmugam

Problem 1: Canny Edge Detector and Hough transform

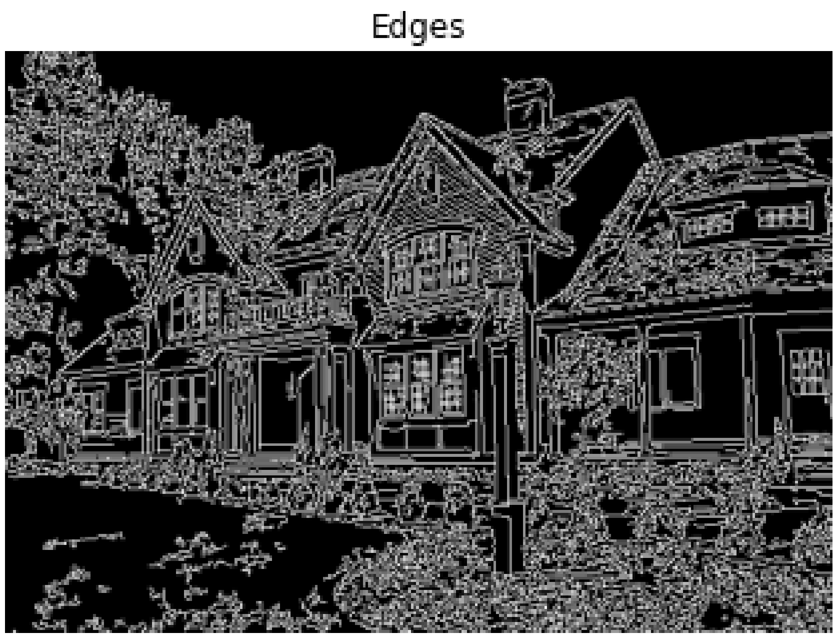
Detecting edges

Edges without blur:
Code:

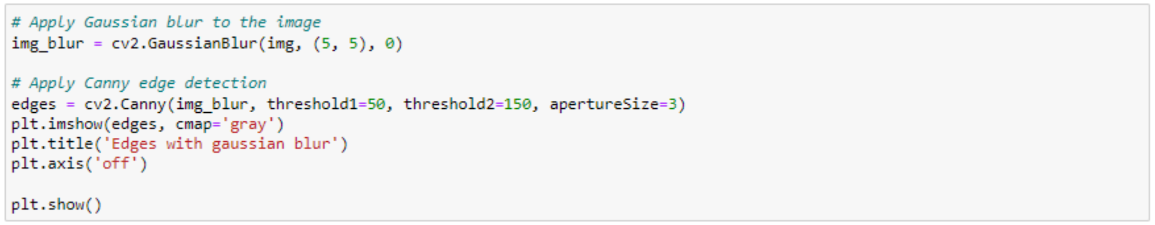


```
# Load image in grayscale
img = cv2.imread('C:/Users/sures/Downloads/house.jpg', cv2.IMREAD_GRAYSCALE)

# Apply Canny edge detection
edges = cv2.Canny(img, threshold1=50, threshold2=150, apertureSize=3)

plt.imshow(edges, cmap='gray')
plt.title('Edges')
plt.axis('off')

plt.show()
```

Output:

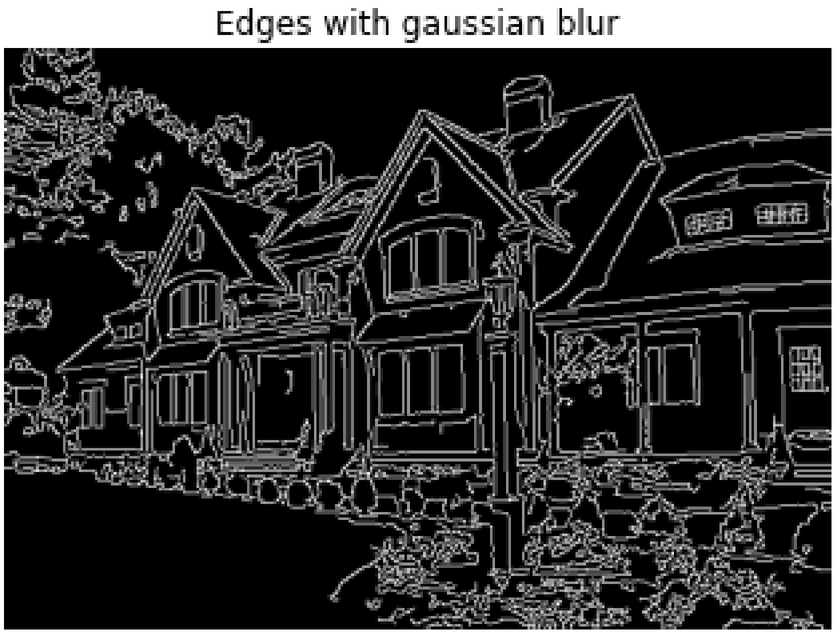


Edges with gaussian blur:
Code:



```
# Apply Gaussian blur to the image
img_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Apply Canny edge detection
edges = cv2.Canny(img_blur, threshold1=50, threshold2=150, apertureSize=3)
plt.imshow(edges, cmap='gray')
plt.title('Edges with gaussian blur')
plt.axis('off')

plt.show()
```

Output:



a) What is your opinion on what an edge is in this particular image?
Answer:

The resulting edges image is a binary image where the edges are represented by white pixels and the non-edges by black pixels. This image can be further processed or used for visualization as needed. In this case the edges detected were very noisy as we can see from the above image and which leads to the second approach by applying the Gaussian Blur.

Note that the Canny function in OpenCV does not include a Gaussian filtering step, so to apply a Gaussian filter to the image before edge detection, I used the Gaussian Blur function in OpenCV

b) Now your task is to adjust the Canny input parameters to detect the "best" set of edges that you can.
Answer:

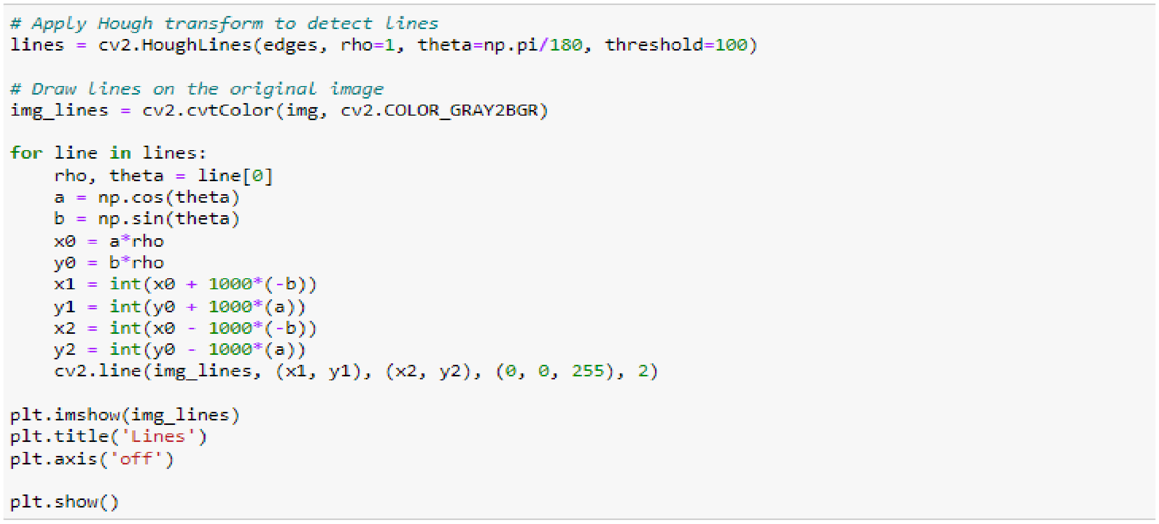
The image is read in grayscale and applies the Canny edge detection algorithm using OpenCV's Canny function. The function takes the following parameters:

img: The input image in grayscale.
threshold1: The lower threshold for hysteresis thresholding. Pixels with gradient values below this threshold will be discarded.
threshold2: The upper threshold for hysteresis thresholding. Pixels with gradient values above this threshold will be kept as strong edges if they are connected to pixels with gradient values above the lower threshold, and as weak edges otherwise.
aperture Size: The size of the Sobel kernel used for computing the gradient magnitude and direction.

In this above result, I used a Sobel kernel of size 3 and set the lower and upper threshold values to 50 and 150, respectively. These values gave the desired edges in adjusting the parameters various times.

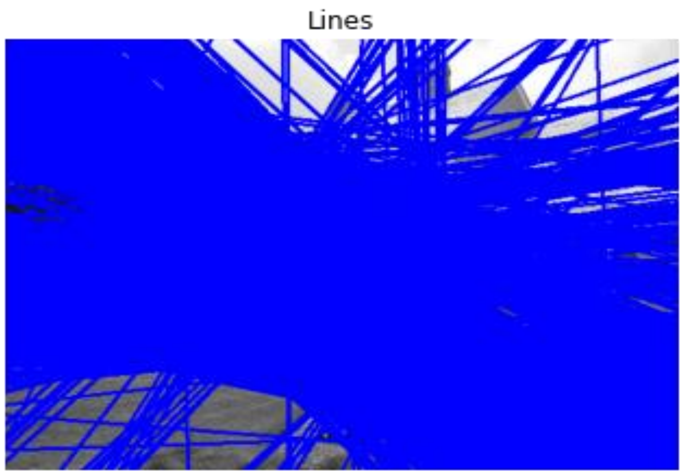
Detecting straight lines

Code:



```
# Apply Hough transform to detect lines
lines = cv2.HoughLines(edges, rho=1, theta=np.pi/180, threshold=100)

# Draw lines on the original image
img_lines = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img_lines, (x1, y1), (x2, y2), (0, 0, 255), 2)

plt.imshow(img_lines)
plt.title('Lines')
plt.axis('off')

plt.show()
```

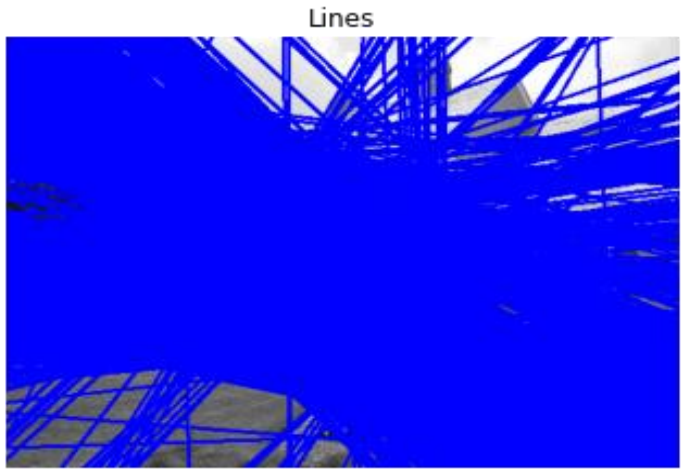
Output:



a) What does a specific position in the Hough space correspond to in the image?
Answer:

In the Hough transform, each point in the input image space maps to a sinusoidal curve in the Hough space. A line in the input image space maps to a single point at the intersection of the sinusoidal curves that correspond to the line's parameters in Hough space.

b) Which lines did you detect in the end? Show the image with detected lines plotted.
Answer:
Mostly all the edges from the house and it gets overlapped because of multiple edges are detected and the image is like this:



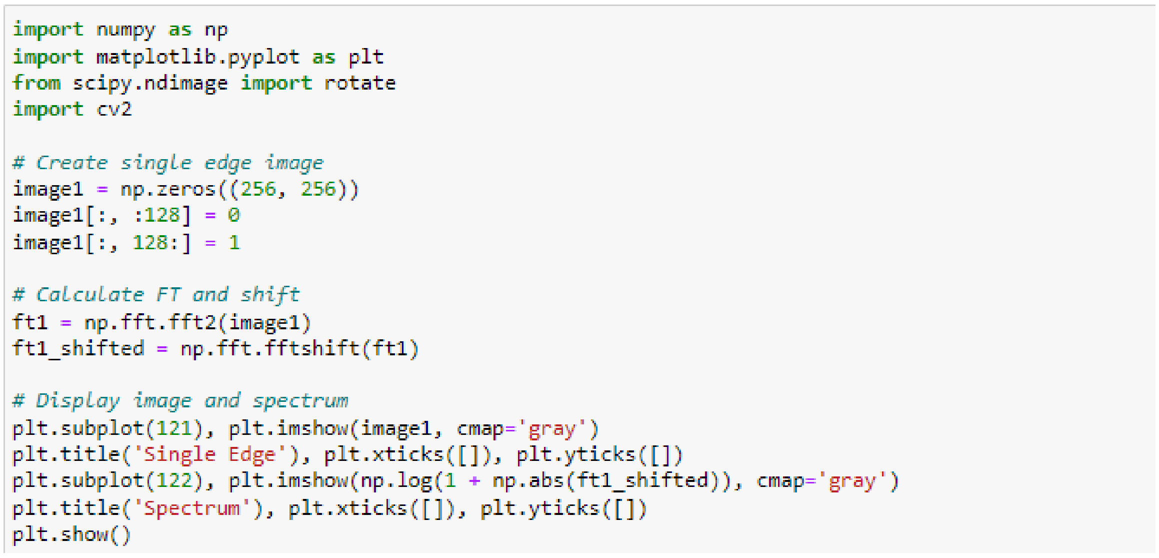
c) How noisy (i.e., broken) do the edges have to be before Hough is unable to detect the original geometric structure in the image?
Answer:
The performance of the Hough transforms in detecting geometric structures in an image depends on a number of factors, including the type of structures being detected, the quality of the edge detection, the presence of noise or other artifacts, and the parameters used for the Hough transform.

In general, the Hough transform is robust to noise and can often detect structures even in noisy images. However, as the level of noise or the complexity of the structures increases, the Hough transform may become less effective or require more careful parameter tuning.

The exact level of noise or complexity that causes the Hough transform to fail in detecting the original geometric structure in an image will depend on the specific image and the parameters used. In general, it is a good idea to preprocess the image to remove noise and enhance the edges before applying the Hough transform, and to experiment with different parameter values to find the settings that work best for the particular task at hand.

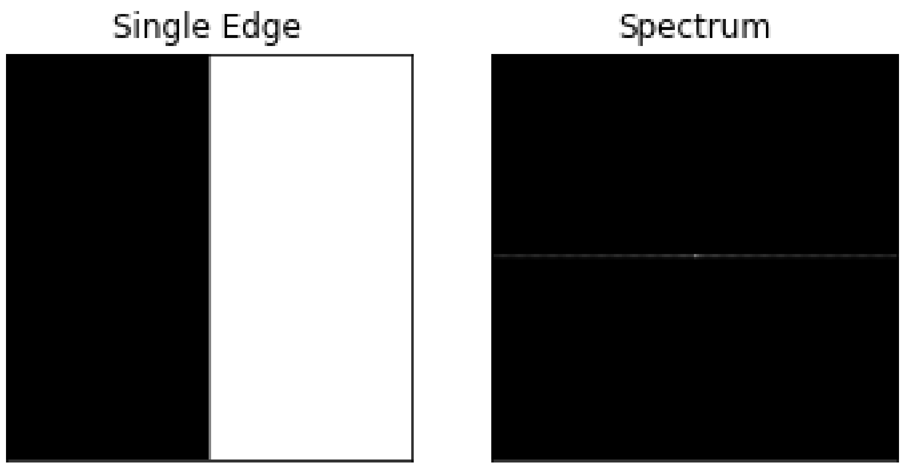
FFT Transform

Code:



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import rotate
import cv2

# Create single edge image
image1 = np.zeros((256, 256))
image1[:, :128] = 0
image1[:, 128:] = 1

# Calculate FT and shift
ft1 = np.fft.fft2(image1)
ft1_shifted = np.fft.fftshift(ft1)

# Display image and spectrum
plt.subplot(121), plt.imshow(image1, cmap='gray')
plt.title('Single Edge'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(np.log(1 + np.abs(ft1_shifted)), cmap='gray')
plt.title('Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output:



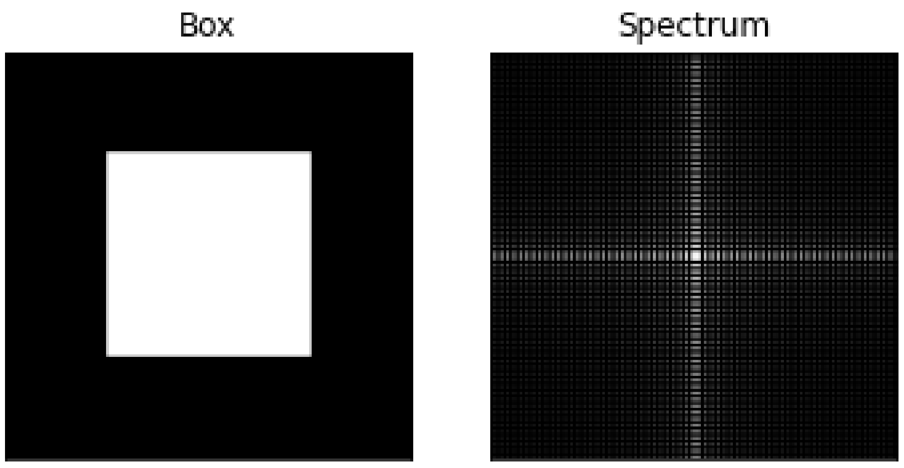
Explanation:

The Fourier Transform of the single edge image produces a spectrum with horizontal lines, indicating high frequencies along the edge. The DC coefficient is at the center of the spectrum.

Code:

```
# Create box image
image2 = np.zeros((128, 128))
image2[32:96, 32:96] = 1

# Calculate FT and shift
ft2 = np.fft.fft2(image2)
ft2_shifted = np.fft.fftshift(ft2)

# Display image and spectrum
plt.subplot(121), plt.imshow(image2, cmap='gray')
plt.title('Box'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(np.log(1 + np.abs(ft2_shifted)), cmap='gray')
plt.title('Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

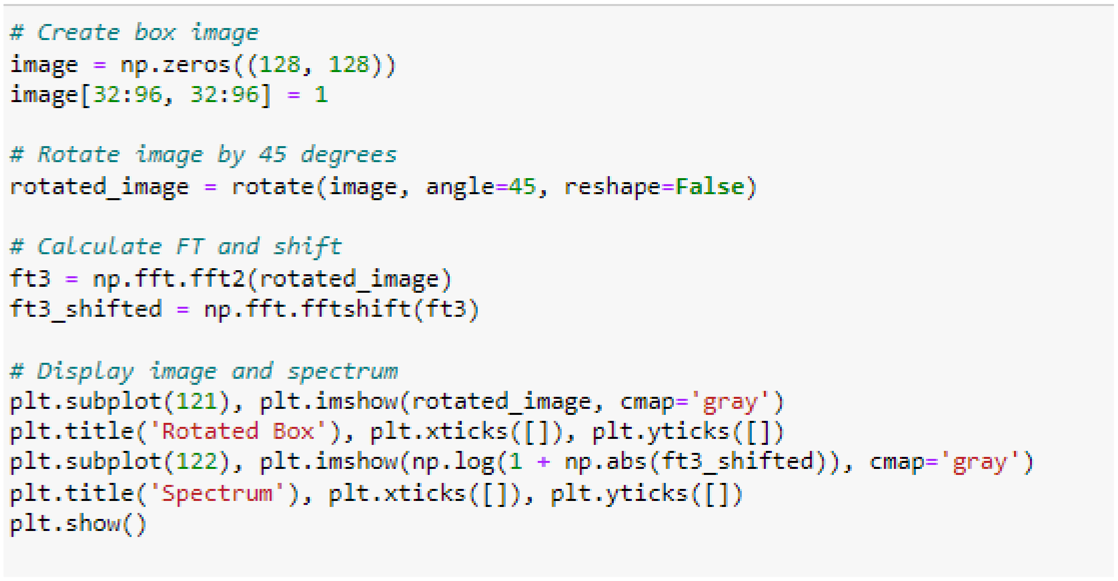
Output:



Explanation:

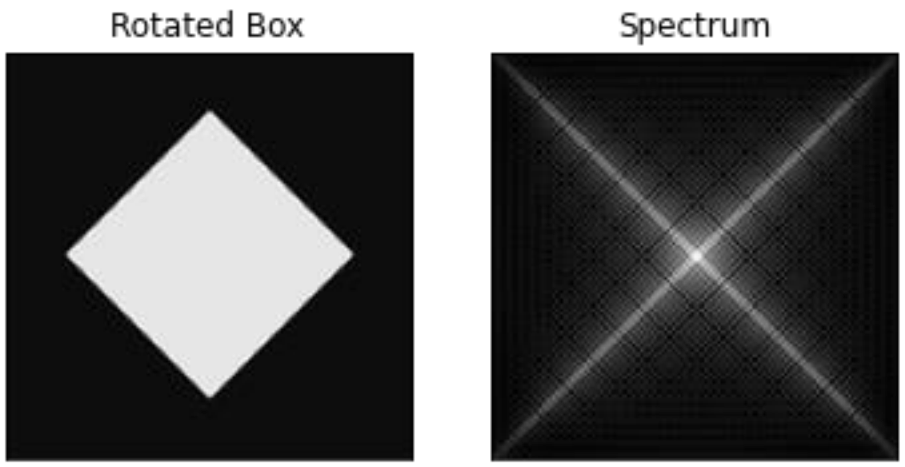
The Fourier Transform of the box image produces a spectrum with a centered peak, indicating low frequencies in the center of the box, and four corners, indicating high frequencies in the corners. The DC coefficient is at the center of the spectrum.

Code:



```python
# Create box image
image = np.zeros((128, 128))
image[32:96, 32:96] = 1

# Rotate image by 45 degrees
rotated_image = rotate(image, angle=45, reshape=False)

# Calculate FT and shift
ft3 = np.fft.fft2(rotated_image)
ft3_shifted = np.fft.fftshift(ft3)

# Display image and spectrum
plt.subplot(121), plt.imshow(rotated_image, cmap='gray')
plt.title('Rotated Box'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(np.log(1 + np.abs(ft3_shifted)), cmap='gray')
plt.title('Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output:



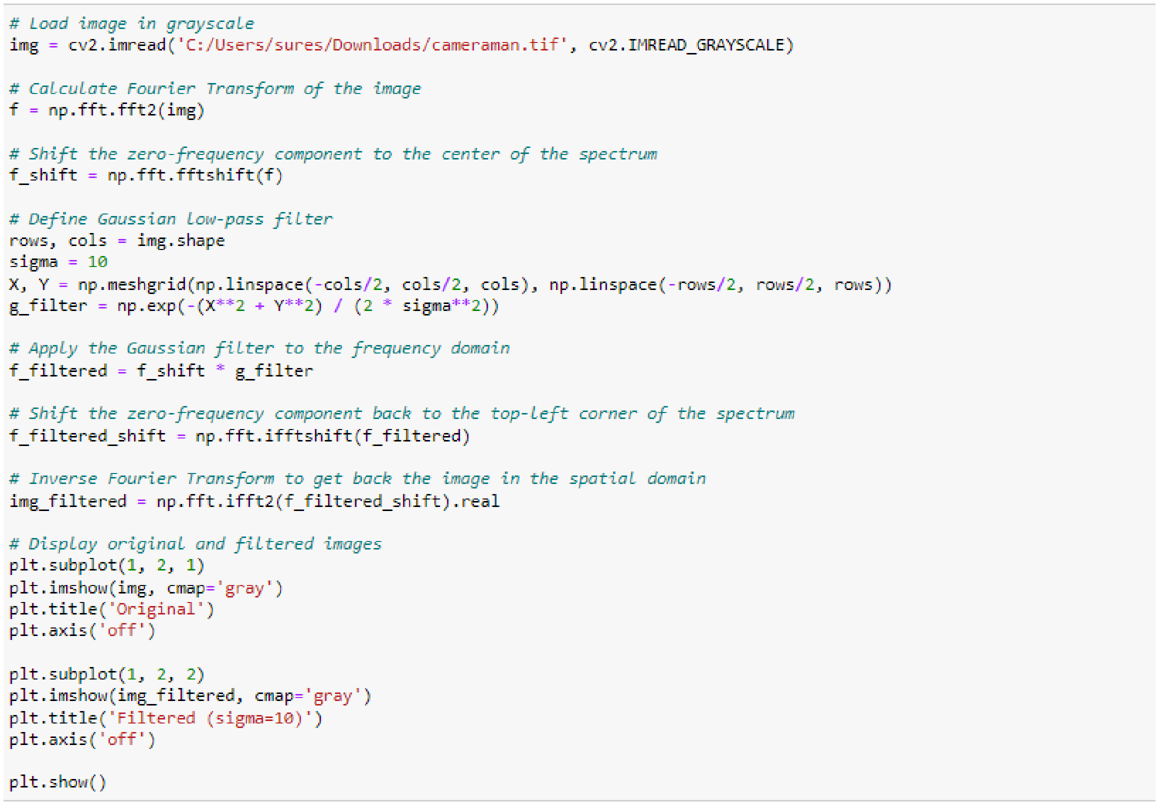
Explanation:

The Fourier Transform of the rotated box image produces a spectrum with two diagonal lines, indicating high frequencies along the diagonal of the rotated box. The DC coefficient is at the center of the spectrum.

The FT images show that the single edge has a simpler spectrum than the box and rotated box, with only one high frequency component along the edge. The box has a more complex spectrum with both low and high frequency components, while the rotated box has a diagonal high frequency component. The relationship among the FT images is that the box and rotated box have more complex spectra than the single edge, with the rotated box having a more specific directionality in its high frequency components.

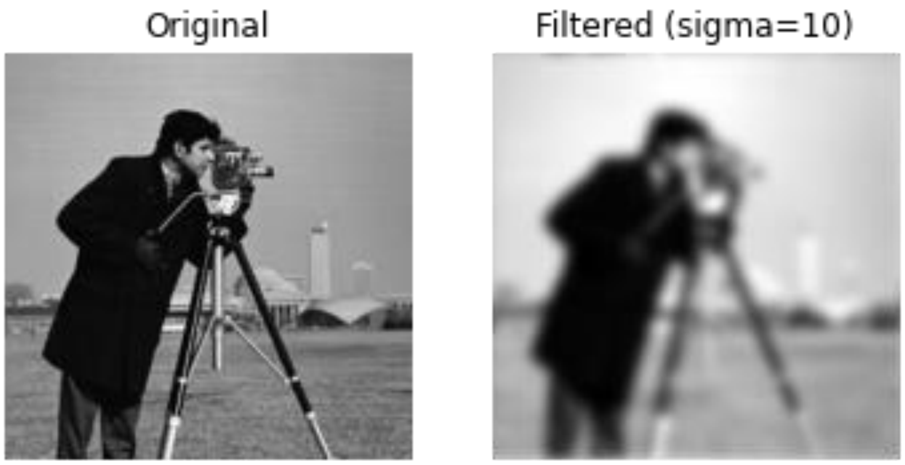
Gaussian Filtering in the frequency domain:

Code:

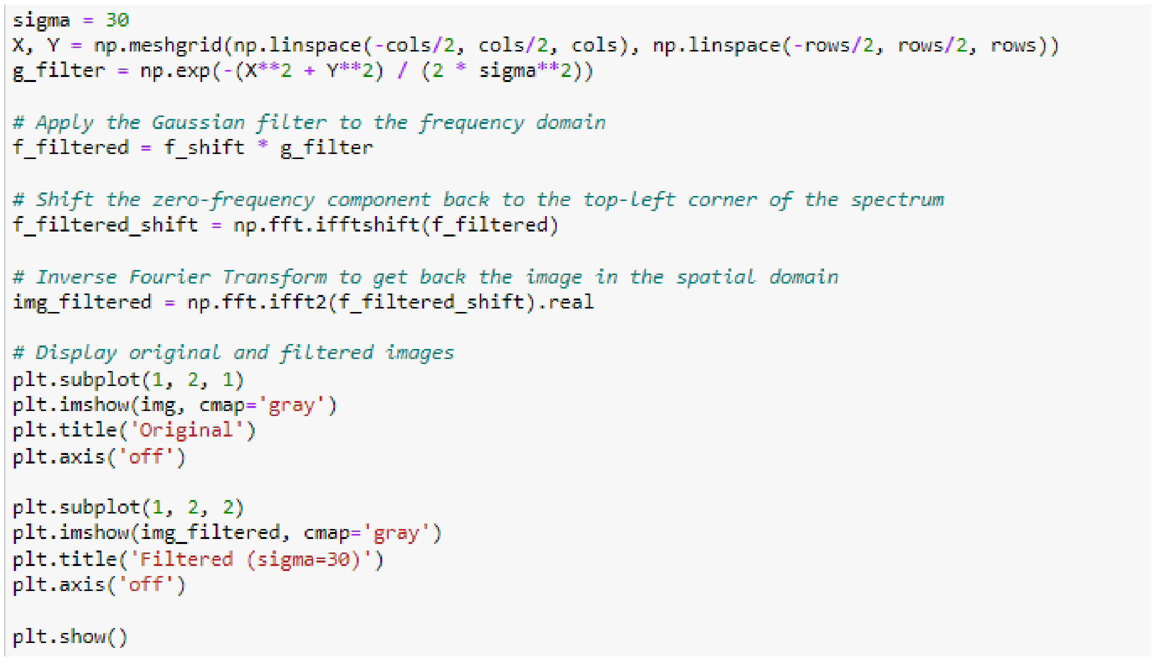


```
# Load image in grayscale
img = cv2.imread('C:/Users/sures/Downloads/cameraman.tif', cv2.IMREAD_GRAYSCALE)

# Calculate Fourier Transform of the image
f = np.fft.fft2(img)

# Shift the zero-frequency component to the center of the spectrum
f_shift = np.fft.fftshift(f)

# Define Gaussian low-pass filter
rows, cols = img.shape
sigma = 10
X, Y = np.meshgrid(np.linspace(-cols/2, cols/2, cols), np.linspace(-rows/2, rows/2, rows))
g_filter = np.exp(-(X**2 + Y**2) / (2 * sigma**2))

# Apply the Gaussian filter to the frequency domain
f_filtered = f_shift * g_filter

# Shift the zero-frequency component back to the top-left corner of the spectrum
f_filtered_shift = np.fft.ifftshift(f_filtered)

# Inverse Fourier Transform to get back the image in the spatial domain
img_filtered = np.fft.ifft2(f_filtered_shift).real

# Display original and filtered images
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_filtered, cmap='gray')
plt.title('Filtered (sigma=10)')
plt.axis('off')

plt.show()
```

Output:

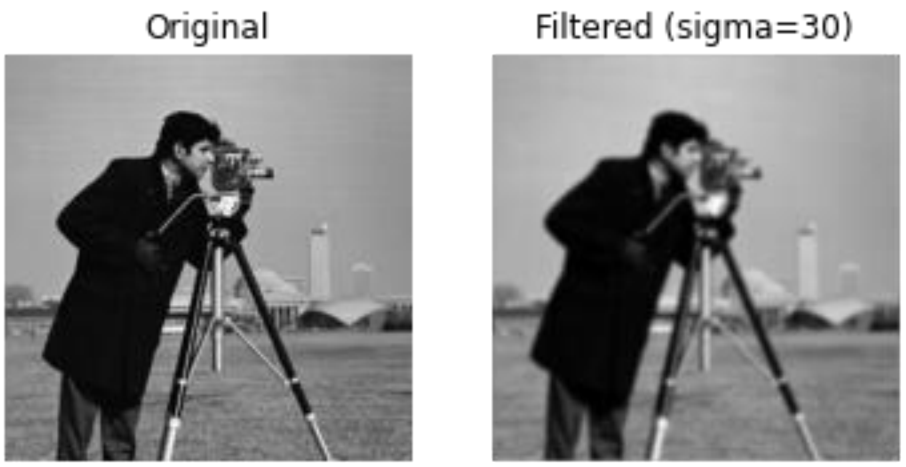


Code:



```
sigma = 30
X, Y = np.meshgrid(np.linspace(-cols/2, cols/2, cols), np.linspace(-rows/2, rows/2, rows))
g_filter = np.exp(-(X**2 + Y**2) / (2 * sigma**2))

# Apply the Gaussian filter to the frequency domain
f_filtered = f_shift * g_filter

# Shift the zero-frequency component back to the top-left corner of the spectrum
f_filtered_shift = np.fft.ifftshift(f_filtered)

# Inverse Fourier Transform to get back the image in the spatial domain
img_filtered = np.fft.ifft2(f_filtered_shift).real

# Display original and filtered images
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_filtered, cmap='gray')
plt.title('Filtered (sigma=30)')
plt.axis('off')

plt.show()
```

Output:



Explanation:
The image is converted in grayscale and calculates its Fourier Transform using NumPy's fft2 function. It then shifts the zero-frequency component to the center of the spectrum using fftshift, defines a Gaussian low-pass filter with a standard deviation of 10 and 30 (The difference in the images is that the lower the sigma value the blur is applied more than the higher sigma value), and applies the filter to the frequency domain by element-wise multiplication. The zero-frequency component is then shifted back to the top-left corner of the spectrum using ifftshift, and an inverse Fourier Transform is performed to obtain the filtered image in the spatial domain.

To generate a filtered image with a different standard deviation, you can simply change the value of sigma and repeat the filtering process.

When the value of sigma is small, the Gaussian filter has a small radius and therefore removes only high-frequency components from the image. This results in a slightly blurred image with less detail but sharper edges. As the value of sigma increases, the radius of the Gaussian filter also increases, and more low-frequency components are removed from the image. This leads to a more heavily blurred image with less detail and smoother edges.

In the frequency domain, the Gaussian filter is represented by a circular shape centered on the origin. As the value of sigma increases, the size of the circular shape also increases, effectively removing more and more high-frequency components from the image. This is evident in the frequency domain images, where the size of the circle in the center increases with increasing sigma.