Image Processing Filters - Fully Commented Code

```
import cv2 # Import OpenCV library for image processing
import numpy as np # Import NumPy for numerical operations

# ================= MEAN FILTERS =================

def arithmetic_mean(img, k=5):
# Apply arithmetic mean (average) filter
return cv2.blur(img, (k, k))

def geometric_mean(img, k=5):
# Convert image to float and add 1 to avoid log(0)
img = img.astype(np.float32) + 1
log_img = np.log(img) # Take logarithm
kernel = np.ones((k, k)) / (k * k) # Averaging kernel
filtered = cv2.filter2D(log_img, -1, kernel)
return np.uint8(np.exp(filtered)) # Exponential and cast

def harmonic_mean(img, k=5):
img = img.astype(np.float32)
kernel = np.ones((k, k))
inv = 1.0 / (img + 1) # Inverse values
filtered = cv2.filter2D(inv, -1, kernel)
hmean = (k * k) / filtered
return np.uint8(hmean)

def contraharmonic_mean(img, k=5, Q=1.5):
img = img.astype(np.float32)
kernel = np.ones((k, k))
num = cv2.filter2D(img ** (Q + 1), -1, kernel)
den = cv2.filter2D(img ** Q, -1, kernel)
return np.uint8(num / (den + 1e-5))

# ================= ORDER STATISTICS =================

def median_filter(img, k=5):
return cv2.medianBlur(img, k)

def min_filter(img, k=5):
return cv2.erode(img, np.ones((k, k), np.uint8))

def max_filter(img, k=5):
return cv2.dilate(img, np.ones((k, k), np.uint8))

def midpoint_filter(img, k=5):
return np.uint8((min_filter(img, k) + max_filter(img, k)) / 2)

def alpha_trimmed_mean(img, k=5, d=4):
pad = k // 2
padded = np.pad(img, pad, mode='edge')
out = np.zeros_like(img)
for i in range(img.shape[0]):
```

```
for j in range(img.shape[1]):
window = padded[i:i+k, j:j+k].flatten()
window.sort()
trimmed = window[d//2 : -d//2]
out[i, j] = np.mean(trimmed)
return np.uint8(out)

# ================= MANUAL CANNY =================

def manual_canny(image, low=50, high=150, kernel_size=5):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (kernel_size, kernel_size), 0)
grad_x = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=3)
grad_y = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=3)
grad_mag = np.sqrt(grad_x**2 + grad_y**2)
grad_mag = np.uint8(255 * grad_mag / np.max(grad_mag))
edges = np.zeros_like(grad_mag)
edges[grad_mag >= high] = 255
edges[(grad_mag >= low) & (grad_mag < high)] = 75
return edges
```