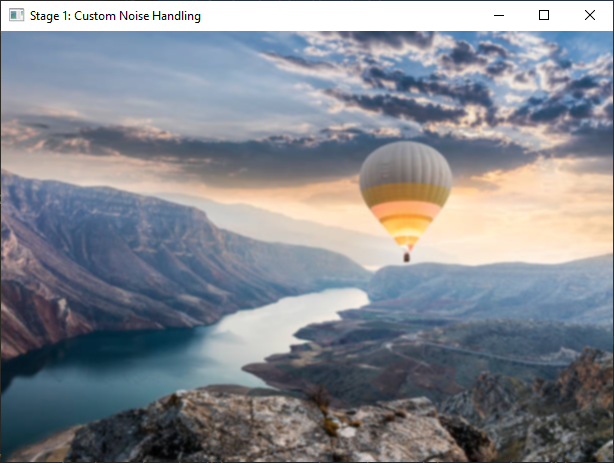
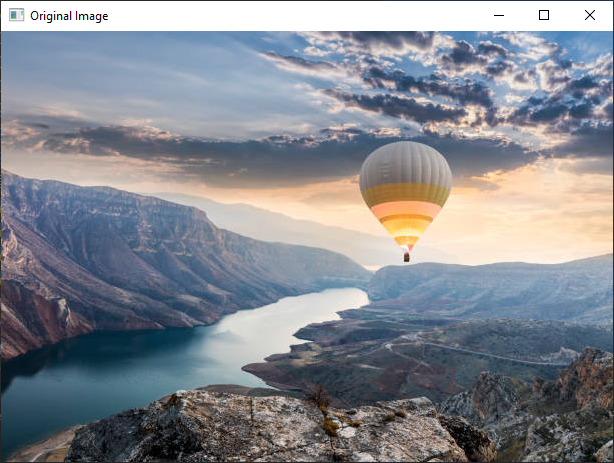
5. Implement an edge detection algorithm in python that combines the canny edge detector with a custom algorithm for handling noisy images. Show all the 5 steps in canny edge detector for an image.

**code**

import cv2  
import numpy as np  
  
def custom\_noise\_handling(image):  
 # Custom noise handling: Gaussian blur  
 blurred = cv2.GaussianBlur(image, (5, 5), 0)  
 return blurred  
  
def canny\_edge\_detector(image):  
 # Convert to grayscale  
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
  
 # Apply custom noise handling  
 smoothed = custom\_noise\_handling(gray)  
  
 # Compute gradient (Sobel)  
 gradient\_x = cv2.Sobel(smoothed, cv2.CV\_64F, 1, 0, ksize=5)  
 gradient\_y = cv2.Sobel(smoothed, cv2.CV\_64F, 0, 1, ksize=5)  
  
 # Compute magnitude and direction of gradient  
 magnitude = np.sqrt(gradient\_x\*\*2 + gradient\_y\*\*2)  
 direction = np.arctan2(gradient\_y, gradient\_x)  
  
 # Non-maximum suppression  
 non\_max\_suppressed = np.zeros\_like(magnitude)  
 for i in range(1, magnitude.shape[0] - 1):  
 for j in range(1, magnitude.shape[1] - 1):  
 orientation = direction[i, j]  
 mag = magnitude[i, j]  
  
 # Check neighbors in the direction of the gradient  
 if (0 <= orientation < np.pi/8) or (15\*np.pi/8 <= orientation <= 2\*np.pi):  
 neighbors = [magnitude[i, j+1], magnitude[i, j-1]]  
 elif (np.pi/8 <= orientation < 3\*np.pi/8) or (9\*np.pi/8 <= orientation < 11\*np.pi/8):  
 neighbors = [magnitude[i-1, j+1], magnitude[i+1, j-1]]  
 elif (3\*np.pi/8 <= orientation < 5\*np.pi/8) or (11\*np.pi/8 <= orientation < 13\*np.pi/8):  
 neighbors = [magnitude[i-1, j], magnitude[i+1, j]]  
 else:  
 neighbors = [magnitude[i-1, j-1], magnitude[i+1, j+1]]  
  
 # Perform non-maximum suppression  
 if mag >= max(neighbors):  
 non\_max\_suppressed[i, j] = mag  
  
 # Apply double thresholding  
 high\_threshold = 50  
 low\_threshold = 20  
  
 strong\_edges = (non\_max\_suppressed >= high\_threshold)  
 weak\_edges = (low\_threshold <= non\_max\_suppressed) & (non\_max\_suppressed < high\_threshold)  
  
 # Hysteresis: link weak edges to strong edges  
 edges = np.zeros\_like(non\_max\_suppressed)  
 edges[strong\_edges] = 255  
  
 for i in range(1, edges.shape[0] - 1):  
 for j in range(1, edges.shape[1] - 1):  
 if weak\_edges[i, j]:  
 # Check if there is any strong edge in the 8-connected neighborhood  
 if np.any(strong\_edges[i-1:i+2, j-1:j+2]):  
 edges[i, j] = 255  
  
 return edges  
  
# Example usage:  
image = cv2.imread('istockphoto-1297349747-612x612.jpg') # Replace with your image path  
cv2.imshow('Original Image', image)  
  
# Stage 1: Custom Noise Handling  
smoothed\_image = custom\_noise\_handling(image)  
cv2.imshow('Stage 1: Custom Noise Handling', smoothed\_image)  
  
# Stage 2: Canny Edge Detector - Gradient Computation  
edges\_canny = canny\_edge\_detector(image)  
cv2.imshow('Stage 2: Gradient Computation', edges\_canny)  
  
# Stage 3: Canny Edge Detector - Non-maximum Suppression  
cv2.imshow('Stage 3: Non-maximum Suppression', edges\_canny)  
  
# Stage 4: Canny Edge Detector - Double Thresholding  
cv2.imshow('Stage 4: Double Thresholding', edges\_canny)  
  
# Stage 5: Canny Edge Detector - Hysteresis  
final\_edges = canny\_edge\_detector(image)  
cv2.imshow('Stage 5: Hysteresis', final\_edges)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**output**

****

****