In [5]:

```python
"""
Usage example:
python harris_corner.py --window_size 3 --alpha value hw3_images/butterfly.jpg

"""
from pylab import *
from scipy import *
from scipy import signal
import cv2
import numpy as np
import sys
import getopt
import operator
import math
import matplotlib.pyplot as plt
import matplotlib.image as mpim

def gauss_derivative_kernels(size, sizey=None):
    """
    returns x and y derivatives of a 2D Gauss kernel array for convolution
   """
    size = int(size)
    if not sizey:
        sizey = size
    else:
        sizey = int(sizey)
    y, x = mgrid[-size:size+1, -sizey:sizey+1]
    #x and y derivatives of a 2D Gaussian with standard dev half of size
    # (ignore scale factor)
    gx = - x * np.exp(-(x**2/float((0.5*size)**2)+y**2/float((0.5*sizey)**2)))
    gy = - y * np.exp(-(x**2/float((0.5*size)**2)+y**2/float((0.5*sizey)**2)))
    return gx,gy

def gauss_kernel(size, sizey = None):
    """
    returns a normalized 2D Gauss kernel array for convolutions
    """
    size = int(size)
    if not sizey:
        sizey = size
    else:
        sizey = int(sizey)
    x, y = mgrid[-size:size+1, -sizey:sizey+1]
    g = np.exp(-(x**2/float(size)+y**2/float(sizey)))
    return g / g.sum()


def compute_harris_response(im, window_size, k):
    """
    compute the Harris corner detector response function for each pixel in the inpu
    :param im: input image
    :param window_size: size of the Gaussian window
    :param k: Harris corner constant (usually 0.04 - 0.06)
    :return r: Harris responses of the input image
```

```
    YOUR CODE GOES IN HERE
    Hint: First compute compute partial derivatives at each pixel, and then compute
    matrix in a Gaussian window around each pixel.
    Compute the Harris response r using the determinant and trace of the second mom
    """

    #derivatives
    gx,gy = gauss_derivative_kernels(3)
    imx = signal.convolve(im,gx, mode='same')
    imy = signal.convolve(im,gy, mode='same')
    #kernel for blurring
    gauss = gauss_kernel(3)
    #compute components of the structure tensor
    Wxx = signal.convolve(imx*imx,gauss, mode='same')
    Wxy = signal.convolve(imx*imy,gauss, mode='same')
    Wyy = signal.convolve(imy*imy,gauss, mode='same')
    #determinant and trace
    Wdet = Wxx*Wyy - Wxy**2
    Wtr = Wxx + Wyy

    return Wdet / Wtr

def get_harris_points(harrisim, min_distance=10, threshold=0.1):
    """
    find local maxima of the Harris response to filter the corner points
    :param harrisim: Harris response of the input
    :param min_distance (optional): minimum number of pixels for distinguishing cor
    :param threshold (optional): threshold for Harris reponse
    :return filtered_coords: coordinates of filtered corner points
    """
    #find top corner candidates above a threshold
    corner_threshold = max(harrisim.ravel()) * threshold
    harrisim_t = (harrisim > corner_threshold) * 1

    #get coordinates of candidates after non-max supression
    #alternatively sort the candidates to get top 1000 corners
    filtered_coords = np.argwhere(harrisim_t)
    filtered_coords[:, [0, 1]] = filtered_coords[:, [1, 0]]  # Swap x, y coordinate

    return filtered_coords


def plot_harris_points(image, points):
    """
    plots corners found in image
    """
    """Plot corners found in image."""
    plt.imshow(image, cmap='gray')
    plt.plot([p[0] for p in points],[p[1] for p in points],'r*')
    plt.axis('off')
    plt.show()

def main():
    """
    parses argument list, calls compute_harris_response and get_harris_points to fi
```

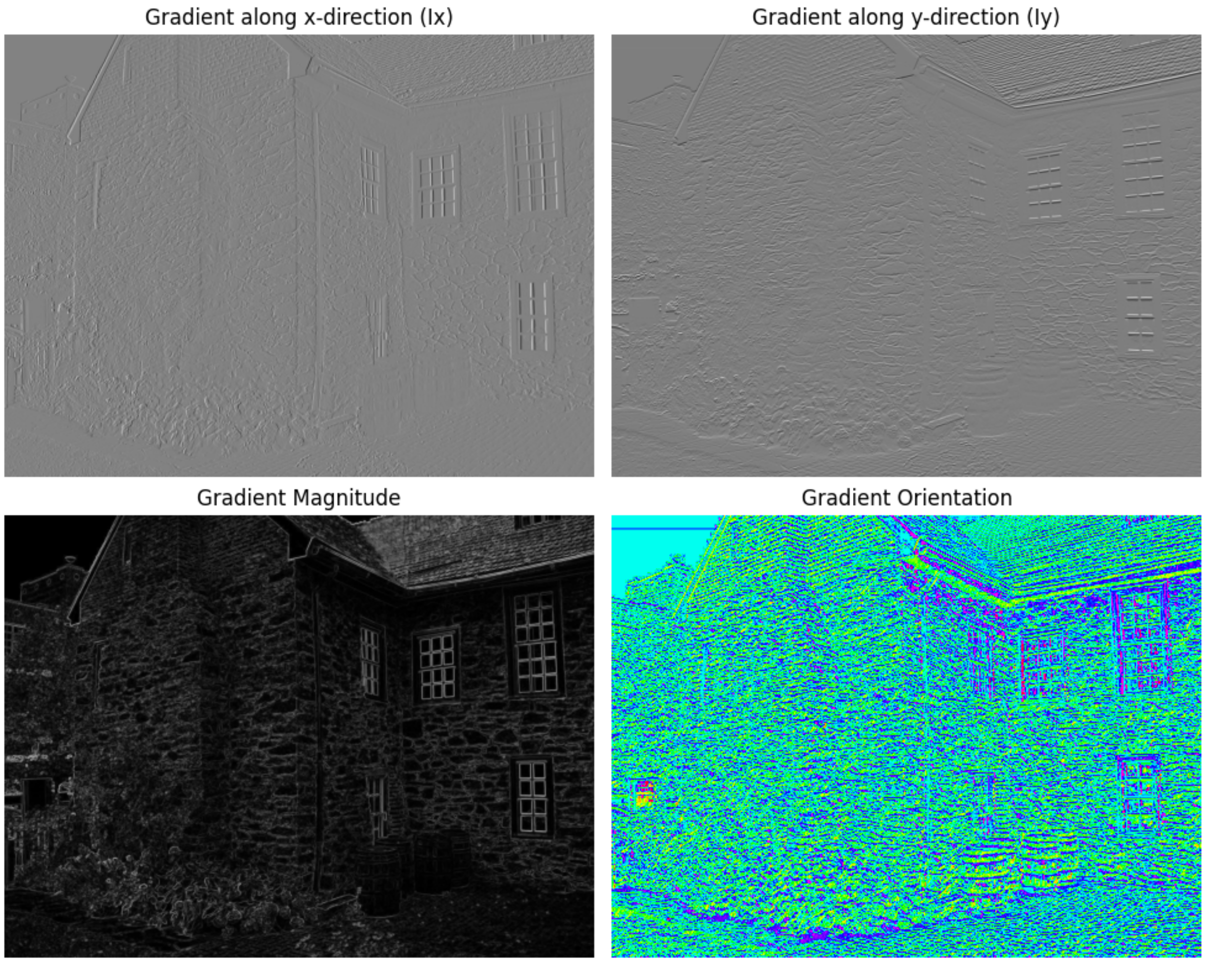
```
    :return: none
    """

    # Define the command-line arguments
    sys.argv = ['program.py', '--window_size', '100', '--alpha', '0.5', '--image_na

    args, img_name = getopt.getopt(sys.argv[1:], '', ['window_size=', 'alpha=', 'im
    args = dict(args)
    print(args)
    print(img_name)
    window_size = int(args.get('--window_size'))
    k = float(args.get('--alpha'))

    print("Image Name: " + str(img_name[0]))
    print("Window Size: " + str(window_size))
    print("Window Size type:" , type(window_size))
    print("K alpha: " + str(k))
    print("K type:", type(k))


    img = cv2.imread('hw3_images/'+str(img_name[0]))
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    #print(img)

    #compute the harris response function (you need to fill in our own code in this
    harrisim = compute_harris_response(img, window_size,k)

    #find local maxima to filter the detected corner points
    filtered_coords = get_harris_points(harrisim, min_distance = 10, threshold = 0.

    #visualize detected Harris corner points on the input image
    plot_harris_points(img, filtered_coords)


if __name__ == "__main__":
    main()
```

```
{'--window_size': '100', '--alpha': '0.5', '--image_name': ''}
['house1-4down.jpg']
Image Name: house1-4down.jpg
Window Size: 100
Window Size type: <class 'int'>
K alpha: 0.5
K type: <class 'float'>
program.py:76: RuntimeWarning: divide by zero encountered in true_divide
```

In [2]:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def compute_gradients(im):
    """
    Compute image gradients Ix and Iy using Sobel operators.

    :param im: input image (grayscale)
    :return Ix: gradient along x-direction
    :return Iy: gradient along y-direction
    """
    dx = cv2.Sobel(im, cv2.CV_64F, 1, 0, ksize=3)  # Compute gradient along x-direc
    dy = cv2.Sobel(im, cv2.CV_64F, 0, 1, ksize=3)  # Compute gradient along y-direc
    return dx, dy

def compute_gradient_magnitude(Ix, Iy):
    """
    Compute gradient magnitude from gradients Ix and Iy.

    :param Ix: gradient along x-direction
    :param Iy: gradient along y-direction
    :return gradient_magnitude: gradient magnitude
    """
    gradient_magnitude = np.sqrt(Ix**2 + Iy**2)
    return gradient_magnitude

def compute_gradient_orientation(Ix, Iy):
    """
    Compute gradient orientation from gradients Ix and Iy.

    :param Ix: gradient along x-direction
```

```python
    :param Iy: gradient along y-direction
    :return gradient_orientation: gradient orientation
    """
    gradient_orientation = np.arctan2(Iy, Ix)
    return gradient_orientation

# Load the image
image = cv2.imread('hw3_images/house1.jpg', cv2.IMREAD_GRAYSCALE)

# Compute gradients
Ix, Iy = compute_gradients(image)

# Compute gradient magnitude and orientation
gradient_magnitude = compute_gradient_magnitude(Ix, Iy)
gradient_orientation = compute_gradient_orientation(Ix, Iy)

# Display images
plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.imshow(Ix, cmap='gray')
plt.title('Gradient along x-direction (Ix)')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(Iy, cmap='gray')
plt.title('Gradient along y-direction (Iy)')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(gradient_magnitude, cmap='gray')
plt.title('Gradient Magnitude')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(gradient_orientation, cmap='hsv')
plt.title('Gradient Orientation')
plt.axis('off')

plt.tight_layout()
plt.show()
```


Gradient along x-direction (Ix)
Gradient along y-direction (Iy)
Gradient Magnitude
Gradient Orientation

In [ ]: