Programming Assignment 1

Luis Obed Vega Maisonet

CAP 5415 Computer Vision

Dr. Yogesh Singh Rawat

Section 0V62, Fall 2025

September 30, 2025

I. Introduction

This document serves as a write-up for Programming Assignment 1 (PA1) of the Computer Vision course CAP5415 of the University of Central Florida (UCF). PA1's main goal is the implementation of the Canny Edge Detection algorithm, which extracts clean and well-localized edges by applying a sequence of operations. The operations are the following: Gaussian smoothing, gradient computation, non-maximum suppression, and hysteresis thresholding. Three grayscale images from the BSDS dataset described in the PA1 instructions were tested with multiple σ values to examine the trade-off between noise suppression and edge localization. To implement the algorithm at hand, a solution was developed in the Python programming language, which is complementary to this documentation.

II. System Requirements

The Canny Detection system required is retrieved from PA1. This includes a concise, detailed description of how to implement such an algorithm.

1. Read a gray-scale image you can find from Berkeley Segmentation Dataset, Training images, and store it as a matrix named I.
2. Create a one-dimensional Gaussian mask G to convolve with I. The standard deviation(σ) of this Gaussian is a parameter to the edge detector (call it $\sigma > 0$).
3. Create a one-dimensional mask for the first derivative of the Gaussian in the x and y directions; call these Gx and Gy. The same $\sigma > 0$ value is used as in step 2.
4. Convolve Ix with Gx to give I′x, the x component of I convolved with the derivative of the Gaussian and convolve Iy with Gy to give I′y, the y component of I convolved with the derivative of the Gaussian.
5. Compute the magnitude of the edge response by combining the x and y components. The magnitude of the result can be computed at each pixel (x, y) as: $M(x, y) = sqr(I'x\ (x, y)^2 + I'y\ (x, y)^2)$.
6. Implement the non-maximum suppression algorithm that we discussed in the lecture. Pixels that are not local maxima should be removed with this method. In other words, not all the pixels indicating strong magnitude are edges in fact. We need to remove false-positive edge locations from the image.

7. Apply Hysteresis thresholding to obtain the final edge map. You may use any existing library function to compute connected components if you want.

III. Implementation

Python 3.13 programming language was used in conjunction with the Integrated Development Environment (IDE) PyCharm 2025.2.1.1. The delivered codebase is divided into a main script, which contains a main function and module imports. Python modules were developed locally to achieve proper functionality. These are named Utility, GaussianKernels, PaddingConvolutionHelpers, GradientComputation, NMS, and Hysteresis. Every Python file includes a code header containing the information of authorship, course context like UCF, CAP5415, term/section, instructor, and a description. All scripts contain a section named Imports and Global, where library dependencies are imported. Libraries used include sys for system processes, OpenCV (cv2) for image I/O and connected components, NumPy for numerical arrays and vectorized math, and Matplotlib for plotting images and text. Additionally, Optional is used for implementing optional parameters. The modules developed for this project can be described as follows:

1. **Utility Module:** A module that provides helper functions to read user input, parse command-line style flags, and load images. Includes error handling for missing inputs and invalid parameter combinations. Normalizes grayscale images into [0,1] and provides utility conversions (to_u8) and plotting helpers (prep_image). Ensures flexibility with default parameters (sigma, radius, thresholds) and user overrides. The helper functions are divided into the following:
    a. **parse_flag:** A function that parses/searches a token list for a flag and returns the cast value following it. It returns None if the flag is absent and terminates on a flag without a value.
    b. **read_input:** A function that gathers the image path and optional parameters -s (sigma), -r (radius), -l/-h (hysteresis thresholds). It enforces that low/high are specified together, loads the image, and prints the resolved parameters to the user.
    c. **read_image(path):** A function that loads an image, converts it to grayscale, and returns a normalized image in [0,1] and the raw grayscale.

d. **to_u8(a):** A function that normalizes any numeric array to [0,255] and casts it to uint8.
e. **prep_image(ax, img, title):** A function that normalizes an array to [0,1] for display and renders it with a gray colormap, hiding axes and setting a title.

2. **GaussianKernels Module:** A module that implements the mathematical foundation for image smoothing and derivative filtering. Functions validate/compute sigma and kernel radius. Provides get_gaussian_kernel for Gaussian smoothing and get_gaussian_derivative for first-order derivative-of-Gaussian filters.
    a. **sigma_radius_check(sigma, radius):** A function that validates sigma is greater than zero, checks if radius is None, and sets radius to the ceiling of 3σ to capture approximately 99% of the Gaussian energy.
    b. **get_gaussian_kernel(sigma, radius):** A function that computes and returns a Gaussian kernel of length 2*radius+1.
    c. **get_gaussian_derivative(sigma, radius):** A function that computes and returns the first derivative of the Gaussian using analytic form.
3. **Padding Convolution Helpers Module:** A Module that handles border conditions with reflecting padding, ensuring accurate convolution results near image edges. Provides dot_product to apply kernels and convolution_correlation for flexible filtering.
    a. **apply_reflecting_padding(I, radius, axis):** A function that reflects pads rows or columns depending on axis (“x” left/right, “y” top/bottom).
    b. **dot_product(height, width, kernel, padded_image, axis):** A function that slides the inner product along one axis for each output pixel.
    c. **convolution_correlation(I, kernel, axis, convolution):** A function that applies 1-D convolution or correlation along axis with reflect padding.
    d. **get_image_subcomponents_by_axis(I, gaussian_kernel_x, gaussian_kernel_y):** A function that applies a two-pass separable filtering, first along x with gaussian_kernel_x, then along y with gaussian_kernel_y. For example, if the inputs were dG, G, then the output would be Ix, and if the input is G, dG, then the output would be Iy.

4. **Gradient Computation Module:** A module that focuses on computing the gradient magnitude and gradient orientation to measure edge direction in degrees [0,180). These results drive the non-maximum suppression step.
    a. **gradient_magnitude(Ix, Iy):** A function that computes the square root of the sum of the image subcomponents in x and y ($Ix^2 + Iy^2$).
    b. **gradient_orientation(Ix, Iy):** A function that uses the formula atan2(Iy, Ix) and converts the results of the calculation to degrees in the range of [0,180).
5. **Non-Maximum Suppression Module:** A module that maps continuous gradient orientations into four canonical bins and applies the Non-Maximum Suppression algorithm to ensure precision in edge localization by suppressing weaker, redundant pixels.
    a. **quantize_orientation_4(theta_deg):** A function that maps continuous orientations into four bins {0°, 45°, 90°, 135°} using ±22.5° thresholds matching the "in-class" 4-bin NMS variant.
    b. **non_maximum_suppression_4dir(M, theta_q):** A function that takes each interior pixel M[y,x], and compares it against the two neighbors along its quantized direction, and keeps it only if it is ≥ both. This thins broad ridges into one-pixel-wide edges and suppresses non-maxima.
6. **Hysteresis Module**: A module that implements a double-threshold edge linking. The module classifies edges as strong, weak, or suppressed. Uses OpenCV's connectedComponents function call to ensure weak edges are retained only if connected to strong ones. Finally, it produces a clean binary edge map that goes from 0 to 255.
    a. **hysteresis(nms, low, high, relative=True, connectivity=8**): A function that enforces double thresholding through the usage of high and low input parameters to keep weak pixels only if connected to any strong pixel.

The main function in main.py executes a pipeline in which all previous modules are called. This main pipeline initiates the reading of parameters and the image, it builds G and dG, it computes smoothed images Igx/Igy, gradients Ix/Iy, magnitude M, orientation θ, quantized directions, it applies non-maximum suppression and hysteresis for the final edge map. It then lays out plots corresponding to the assignment's (a)–(f) panels and displays them. An additional (g) panel is added, containing the final image with the hysteresis process applied.

IV. Execution Steps

To be able to execute the Python project associated with this documentation, download the Python archive file named PA1_VegaMaisonetLuisO.zip and extract it into your directory of choice. Open the Python Project with your IDE of choice or with your command line prompt of choice and go to the project path where the main.py file is located. Be advised that you need Python 3 installed in your environment to be able to run this project. Note: These instructions were developed by using Python 3.13, which shortened the Python command to the acronym of py since I am a Windows User. Run the following command in the command line to install project dependency libraries:

If you are using Windows command prompt:

py -m pip install numpy opencv-python matplotlib

Or if you are using a Linux-based system command line terminal:

pip install numpy opencv-python matplotlib

Now run the command to execute the main.py. If you are using Windows command prompt:

py -m main

Or if you are using a Linux-based system command line terminal:

python -m main

The following output should be generated on the screen:

Welcome to CAP5415 PA1: Canny Edge Detection

Enter the image path, and optionally specify parameters:

-s <sigma> (default = 1.0)

-r <radius> (default = ceil(3*sigma))

-l <low> -h <high> (defaults = 0.1, 0.2)

Example: images/chessboard.png -s 2.0 -r 7 -l 0.05 -h 0.15

>

This same output is achieved by opening the project in the IDE of choice, importing the libraries manually by hovering over the import code lines, clicking on the import library message option, and clicking the run icon option. The displayed output illustrates input instructions for the user. The first parameter is the image path, this should be the complete full path with the image extension, and this is the only non-optional parameter. The second parameter is the Gaussian sigma, denoted by the -s flag. If not specified, it will be defaulted to 1. The third parameter is the radius or the domain of the Gaussian distribution function; if not specified, it will default to the ceiling of 3 times sigma. The fourth parameter is the low hysteresis threshold, denoted by the -l flag; if not specified, it will default to the common value of 0.1. The fifth and the last parameter is the high hysteresis threshold, denoted by -h; if not specified, it will default to a common value of 0.2.

V. Results

Continuing the execution steps from the past section, we can enter one of the images of the given BSDS dataset by specifying the image path with the image name and its designated extension. Let's say we select the image named 37073.jpg, which is shown in Figure 1 below.

Figure 1. Image named 37073.jpg from the BSDS test dataset

The input image should be similar to what is illustrated in Figure 2, with the respective unique path to the image. For this run, the image will be the only input parameter, and all other optional parameters will keep their default values.

```
C:\Users\luisv\PycharmProjects\CAP5415_PA1>py -m main
Welcome to CAP5415 PA1: Canny Edge Detection
Enter the image path, and optionally specify parameters:
  -s <sigma>          (default = 1.0)
  -r <radius>         (default = ceil(3*sigma))
  -l <low> -h <high>  (defaults = 0.1, 0.2)
Example: images/chessboard.png -s 2.0 -r 7 -l 0.05 -h 0.15
> C:\Users\luisv\OneDrive\Desktop\University\UCF\CAP_5415\BSDS300-images\BSDS300\images\test\37073.jpg
```

Figure 2. Running Project Assignment 1 in Windows Command Prompt

After entering the command and pressing the Enter key, the project will take approximately a couple of seconds to complete. The result is displayed in Figure 3 below.

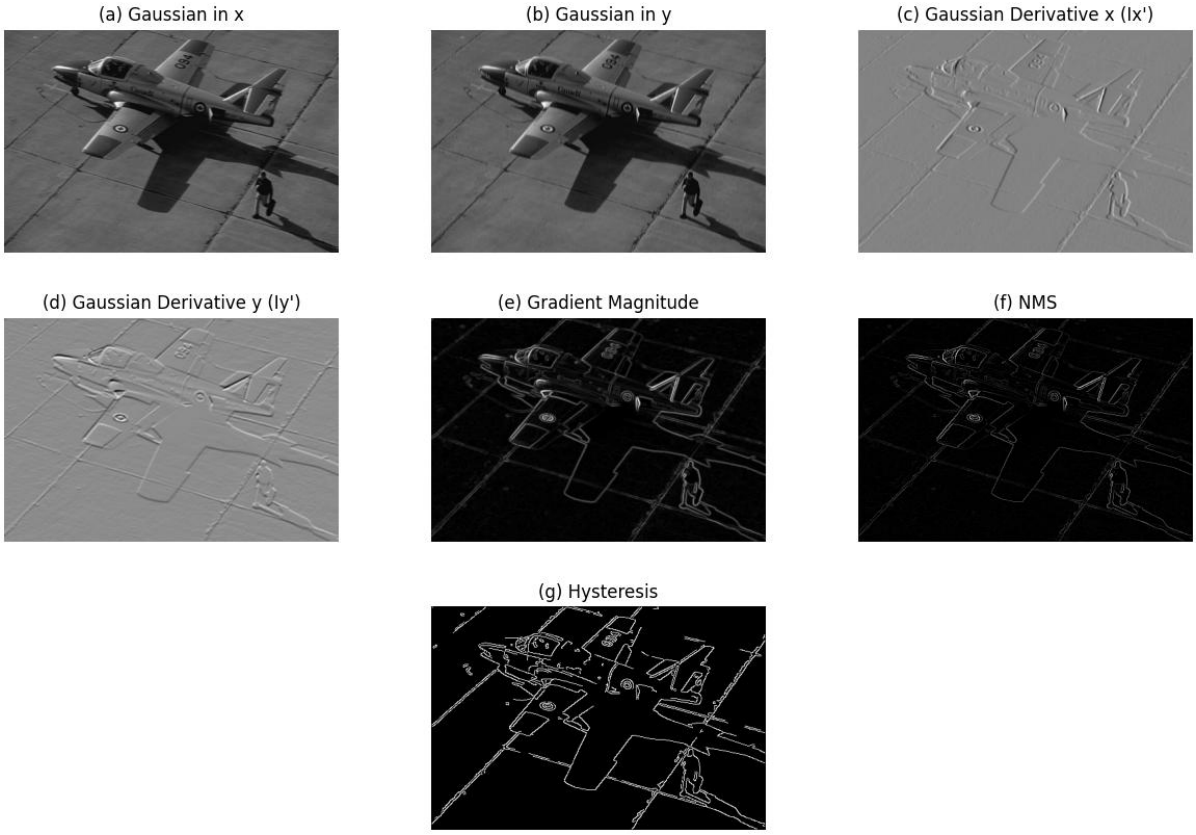

Figure 3. Output Images for 37073.jpg with sigma = 1

As requested by PA1 requirements, before each of these steps, it is important to remember that the image is turned into a greyscale image. The Python project outputs a total of 6

images (a) to (f), with an additional output image (g) covering the hysteresis step. The image (a) represents the Gaussian filter applied along the x-axis of the image (convoluted component I*G(x)), the image (b) represents the Gaussian filter applied along the y-axis of the image (convoluted component I*G(y)), the image (c) represents the image x subcomponent which is a result of correlating I*dG(x) with G(y), the image (d) represents the image y subcomponent which is a result of correlating I*dG(y) with G(x), the image (e) represents gradient magnitude of the original grayscale image, the image (f) illustrates the non-maximum suppression algorithm applied to the original grayscale image and lastly the image (g) illustrates the hysteresis thresholding algorithm applied to the original grayscale image, which shows only edges with sigma =1, radius = 3, low = 0.1 and high = 0.2. The hysteresis thresholding image, image (g) illustrates proper division of the air vehicle parts, but since the floor has similar pixel values to the air vehicle, the algorithm has trouble detecting some edges. Figure 4 displays the second image selected as an input to this project.

Figure 4. Image named 302003.jpg from the BSDS train dataset

Figure 4, also known as 302003.jpg in the train dataset of BSDS, illustrates a woman with Asian race-like features. Inputting it into the algorithm and maintaining the same default parameters will produce the images in Figure 5.

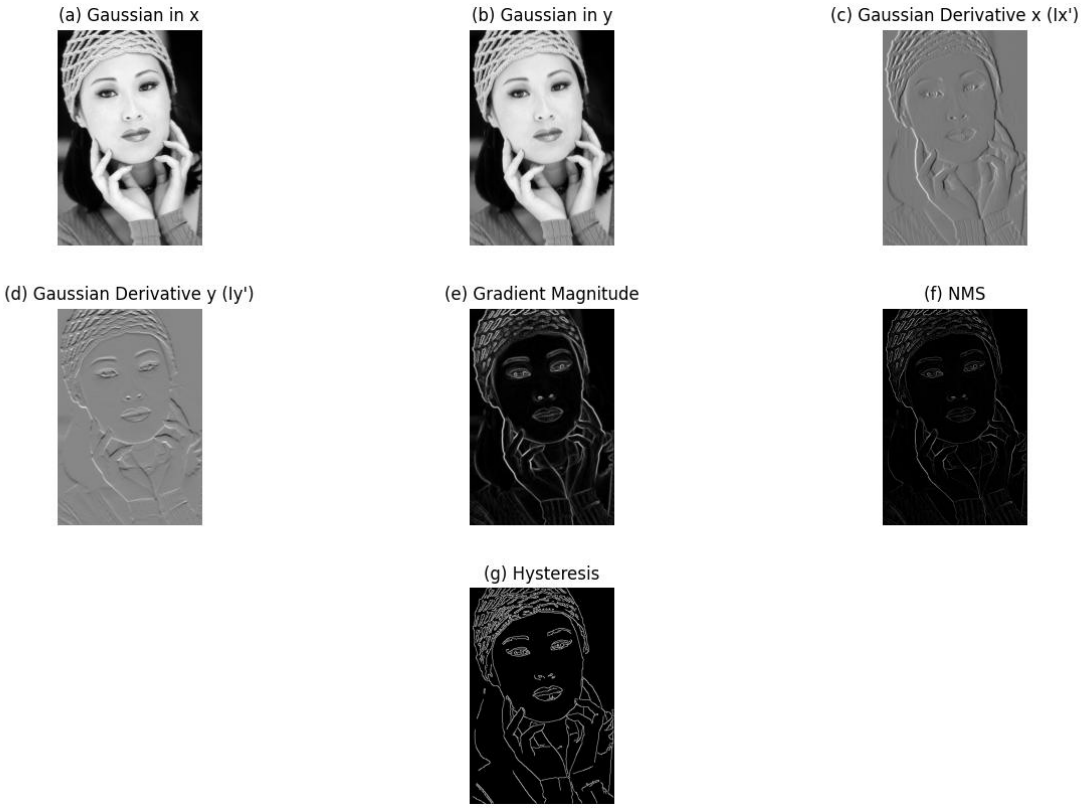

Figure 5. Output Images for 302003.jpg with sigma = 1

The final hysteresis threshold result, image (g), shows proper detection of edges regarding facial features. The only edges that seem off are the ones that are between the background and the woman's hair, since these are of similar pixel value. Another important factor is that the algorithm is putting too much edges in the woman's eyes. The final image selected for this analysis can be seen in Figure 6, which illustrates a mountain view, and it's also part of the BSDS dataset as 20008.jpg.

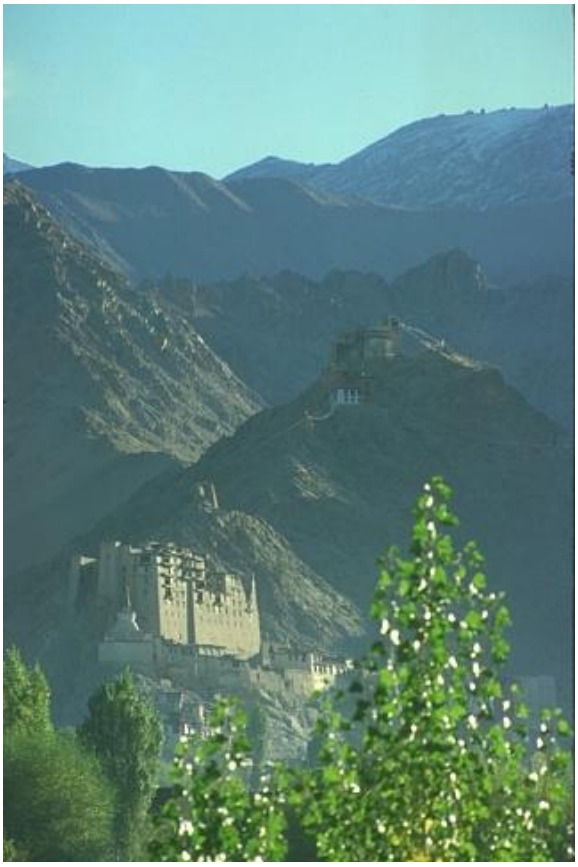

Image named 20008.jpg from the BSDS train dataset

As with the other previous 2 images, the image path of 20008.jpg served as input to the Canny Edge Detection algorithm. The output can be seen in Figure 7 below.

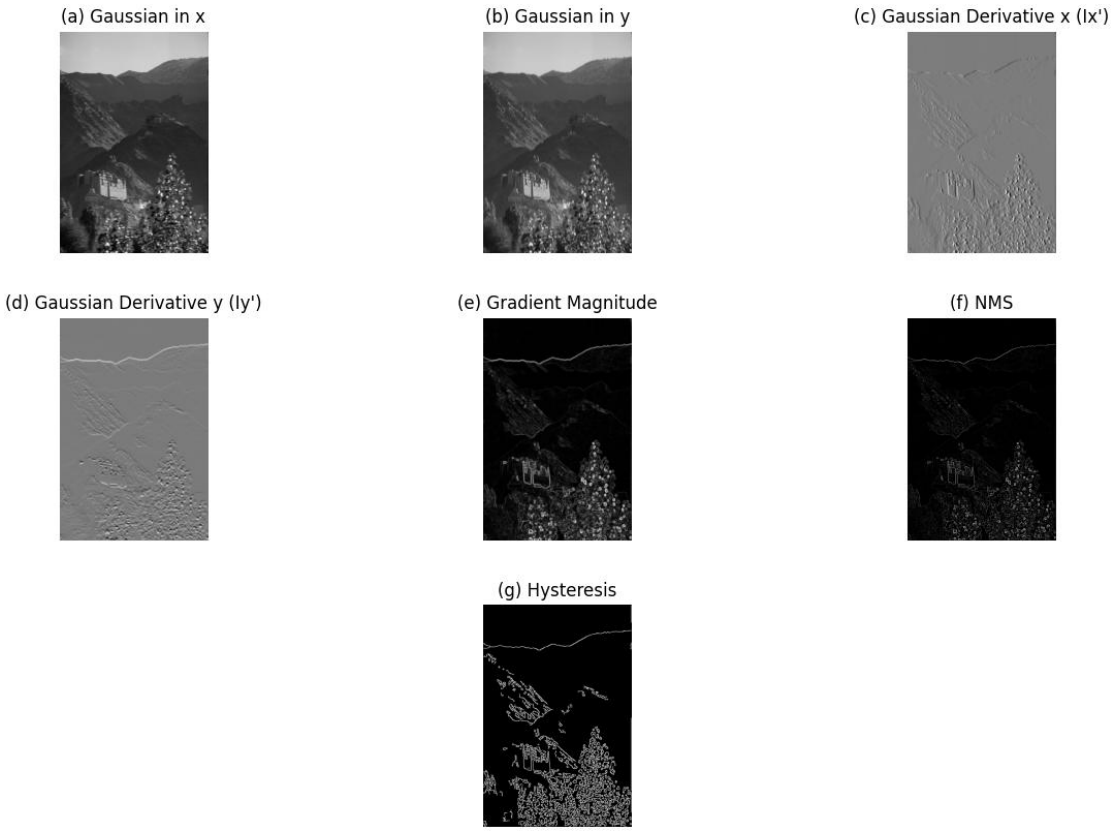

Figure 7. Output Images for 20008.jpg with sigma = 1

The final hysteresis threshold result, image (g), shows proper detection of edges, but in this case, since the castle on the mountain is of a similar pixel value to the mountain itself, our algorithm blended them, unable to detect the proper edges of the castle and part of another mountain.

VI. Sigma Results

As a final requirement of PA1, three different values of sigma were selected, and the output images of these iterations are illustrated below in Figure 8, Figure 9, and Figure 10 for the air vehicle image, Figure 11, Figure 12 and Figure 13 for the woman's face image and Figure 14, Figure 15 and Figure 16 for the mountain view image. The sigma values chosen for these figures are 2, 4, and 8. Taking the first image as an input candidate (the air vehicle image) for the sigma variation process, we obtain the following figures:

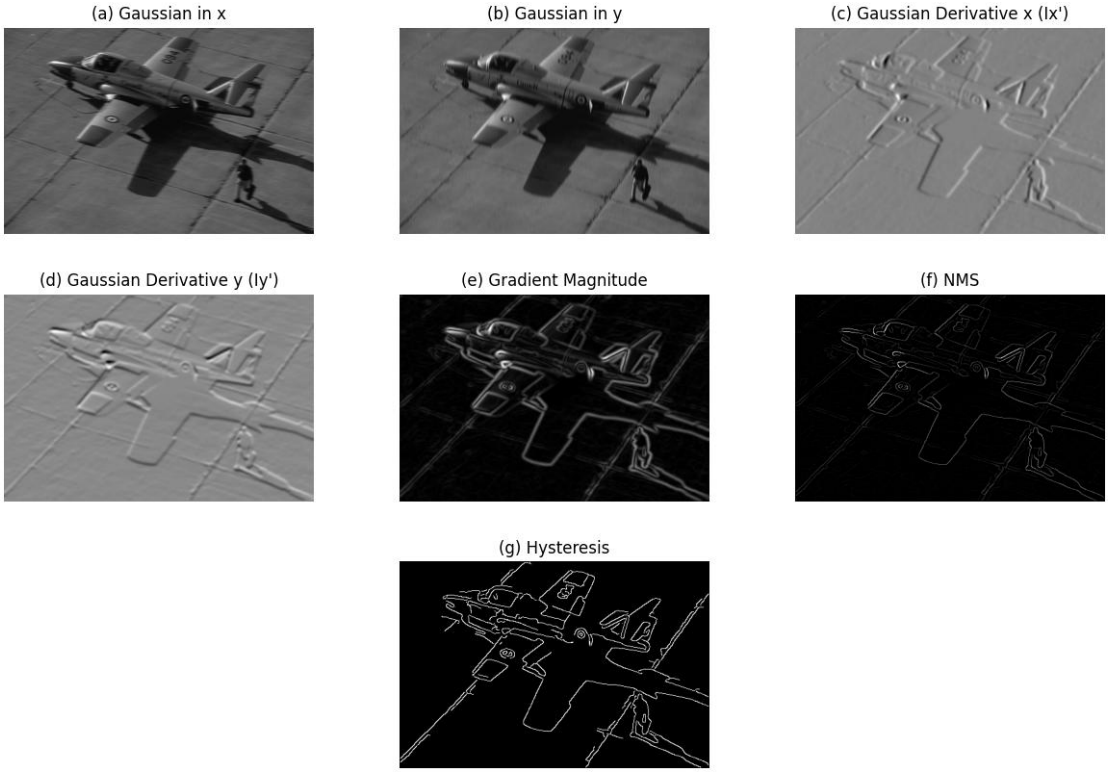

Figure 8. Output Images for 37073.jpg with sigma = 2

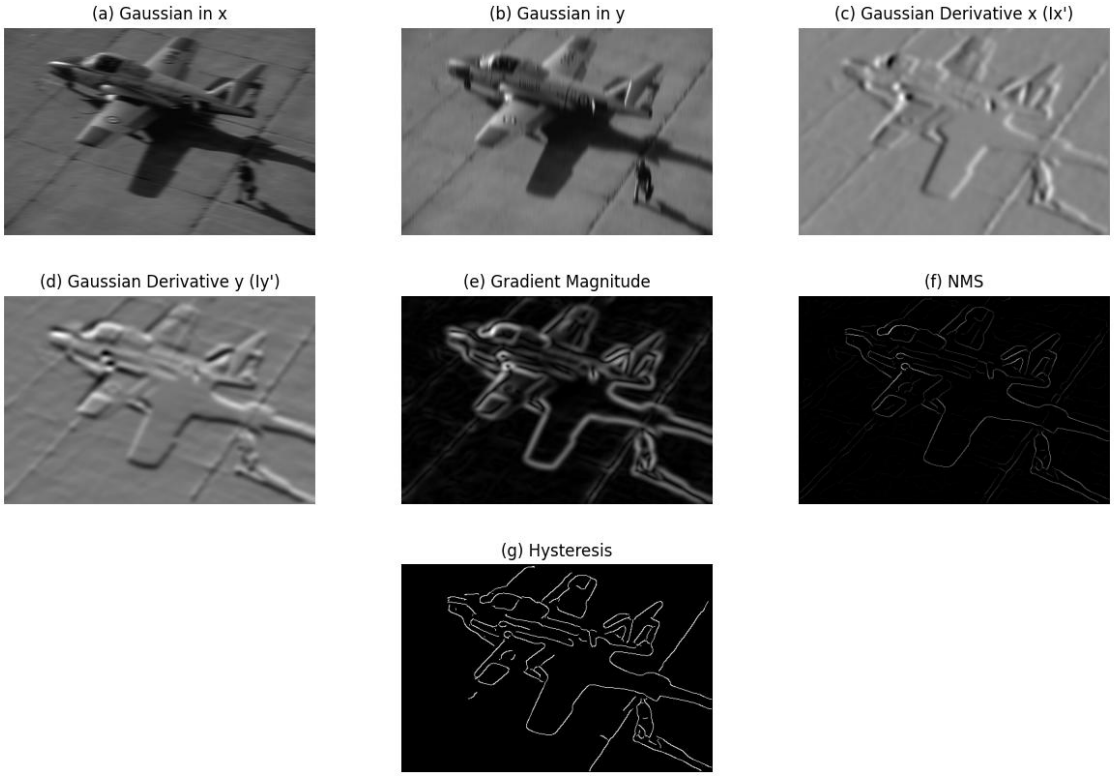

Figure 9. Output Images for 37073.jpg with sigma = 4

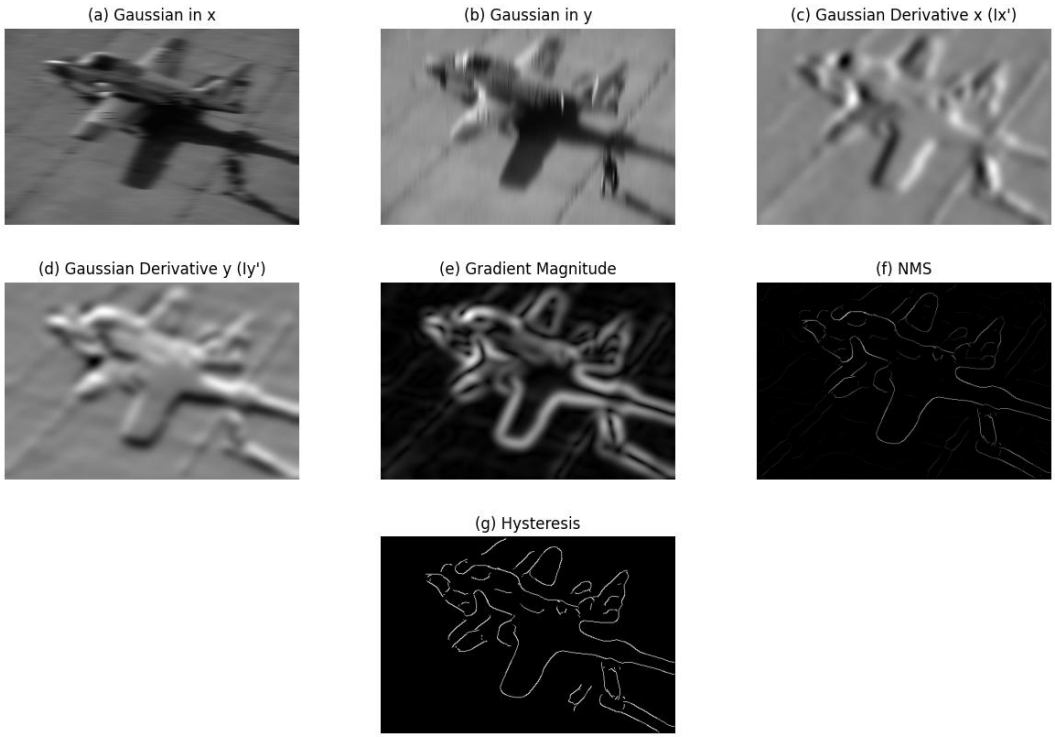

Figure 10. Output Images for 37073.jpg with sigma = 8

Depending on the type of application, there are sigma values that are better than others. Possible applications of images shown on Figures 8 to 10 could be detecting edges of the pilot, edges of air vehicle shadows, and/or edges of air vehicles. For any of these cases, the value of sigma of 2 would suffice. Beyond sigma equal to 2, edge detection becomes challenging. Varying sigma for the second image would result in the following figures:

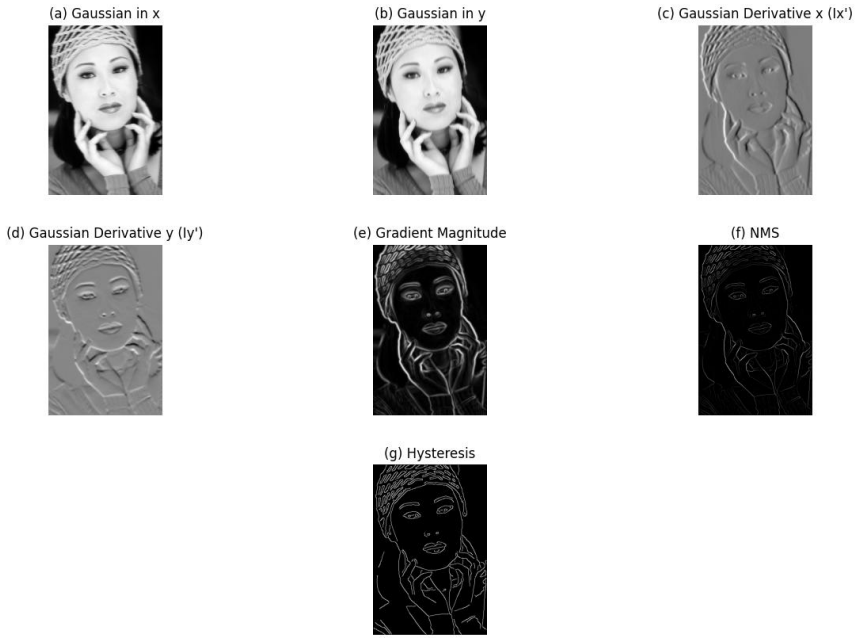

Figure 11. Output Images for 302003.jpg with sigma = 2

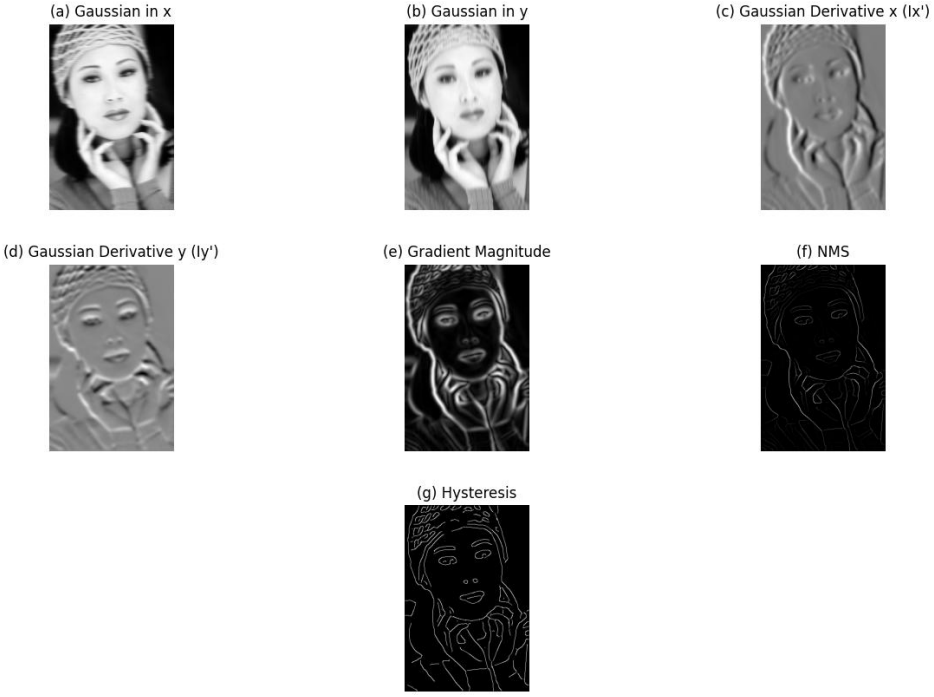

Figure 12. Output Images for 302003.jpg with sigma = 4

Figure 13. Output Images for 302003.jpg with sigma = 8

As the first sigma determination, there are different applications that come to mind regarding the images from Figures 11 to 13. If we just want to detect that there is a human face in the image, a sigma value of 4 would be best suited since human face edges are still present, but if we desire to use the images for detecting feminine woman features like lips and eyes, then a sigma of 2 would be best suited. Continuing with our last image, the mountain view image, we get the following figures:

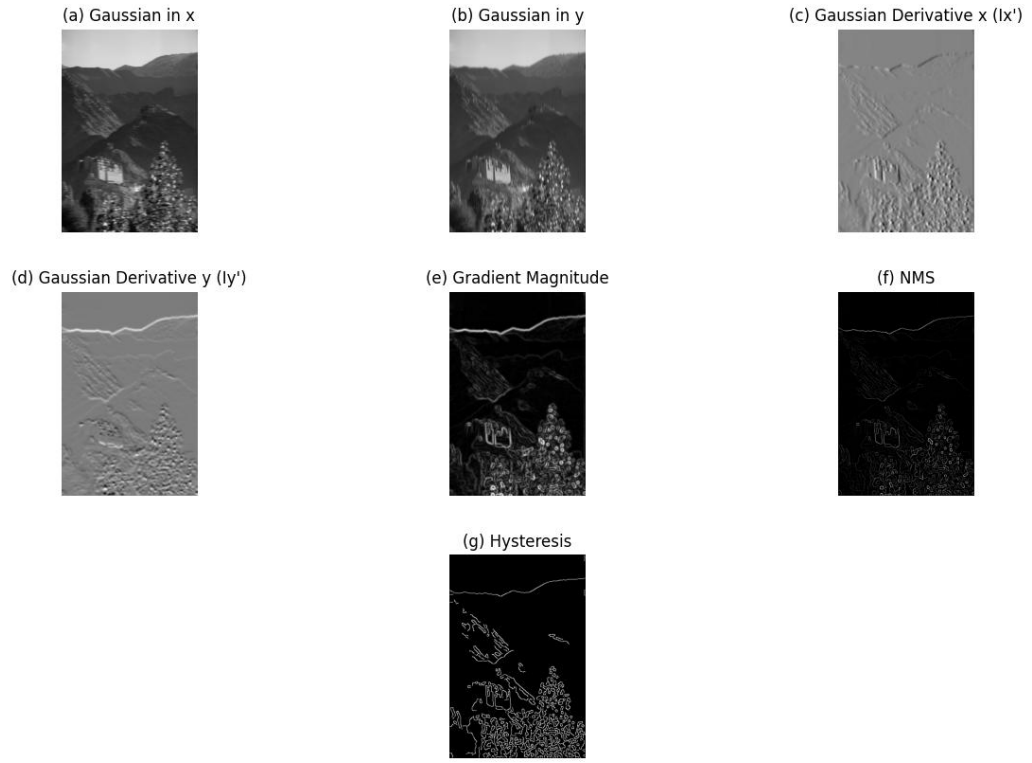

Figure 14. Output Images for 20008.jpg with sigma = 2

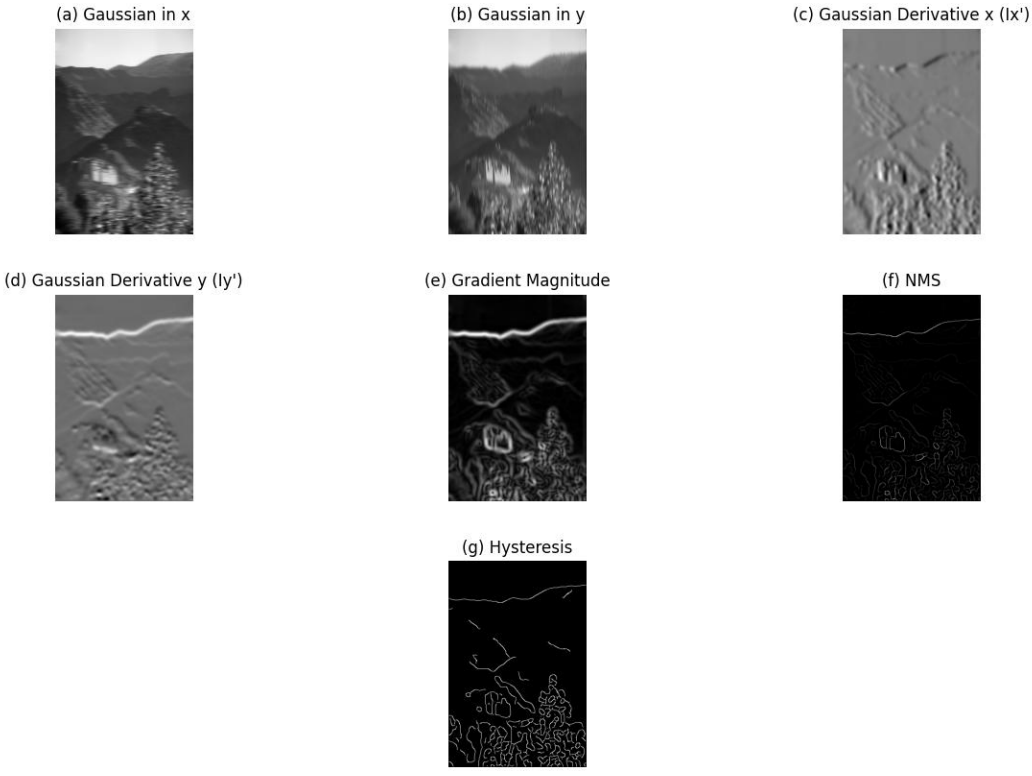

Figure 15. Output Images for 20008.jpg with sigma = 4

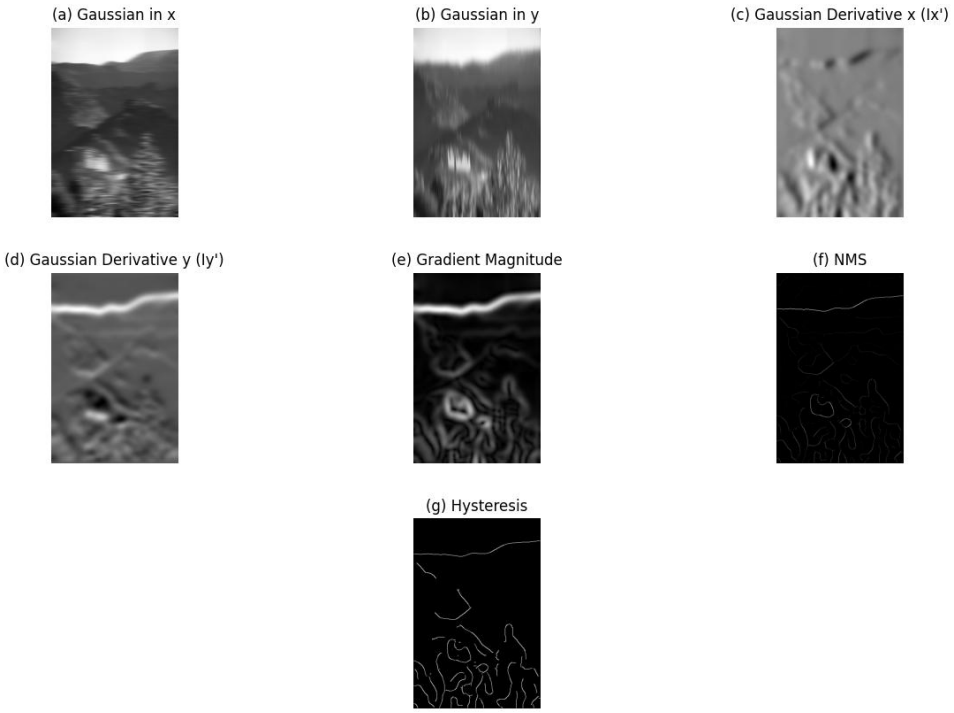

Figure 16. Output Images for 20008.jpg with sigma = 8

For the mountain view image, a sigma of 2 is best suited since selecting a greater sigma doesn't provide any readable objects in the output. I may see a good application by utilizing a sigma with a value of 8 in conjunction with the gradient magnitude, since the castle is visible and looks brighter in Figure 16 compared to Figure 15 and Figure 14. In conclusion, based on the three analyzed images with the three different sigma values, a sigma value of 2 would suffice for most of the proposed applications if this version of the Canny Edge Detection algorithm is used. Other possible algorithm improvements to selecting a better sigma would be to vary the low and high input values of the hysteresis threshold without varying sigma, like a design of experiment process, to detect edges more accurately.