ECEN 468/719 Advanced Logic Design

Department of Electrical and Computer Engineering

Texas A&M University

Lab 4: Design of Canny Edge Detector

Objectives

In this lab, we will implement a canny edge detector and use Vista for simulation and verification.

Introduction

Edge detection refers to identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity that characterize the boundaries of objects in a scene. Classical edge detection methods involve convoluting the image with an operator (2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions.

There is a vast number of edge detectors available, each designed to be sensitive to certain types of edges. Variables involved in the selection of an edge detection operator include edge orientation and noise environment. The geometry of the operator determines a characteristic direction in which is most sensitive to edges. Operators are optimized to look for horizontal, vertical, or diagonal edges. Also, edge detection is difficult in noisy images since the high-frequency components exist in both the noise and the edges. Attempts to reduce the noise result in blurred and distorted edges.

There are many ways to perform edge detection. The majority of different methods may be grouped into two categories: the gradient method and the Laplacian method. In general, the Laplacian method results in higher quality and complexity. We will use the gradient method to detect edges. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image.

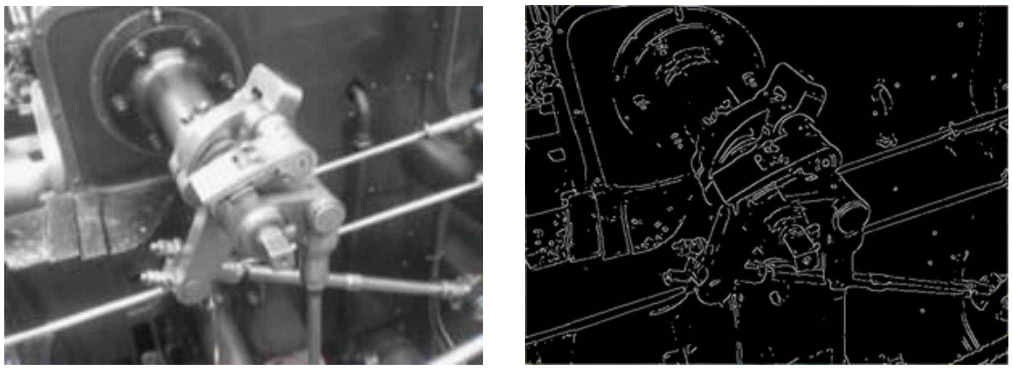


Figure 1. An example of Canny Edge Detection

Edge Detection Flow

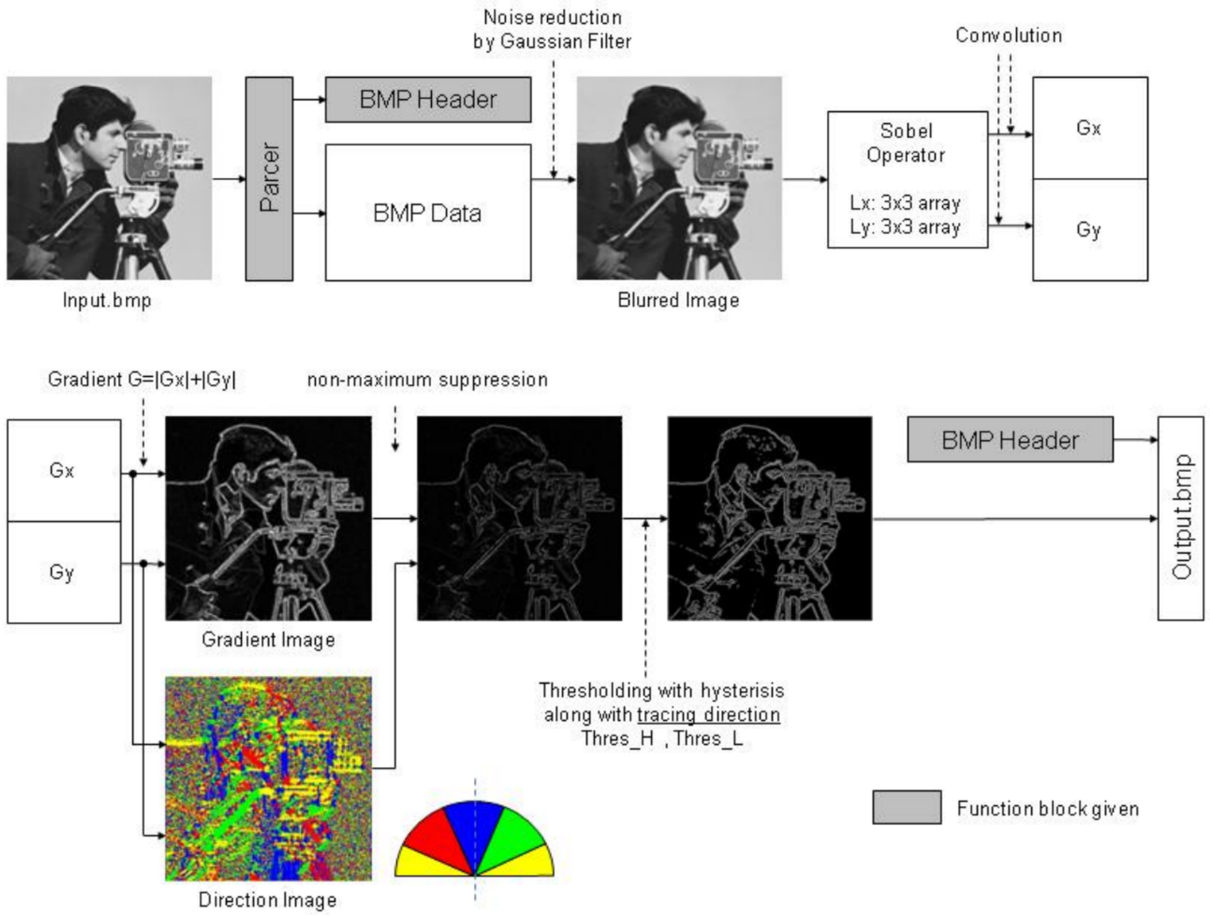


Figure 2. Data flow of Edge Detection

Figure 2 shows the data flow of a Canny Edge Detector. The input image will go through five operations: **Noise Reduction**, **Gradient Calculation**, **Non-maximum Suppression**, and **Thresholding**. The following sections introduce each part in detail.

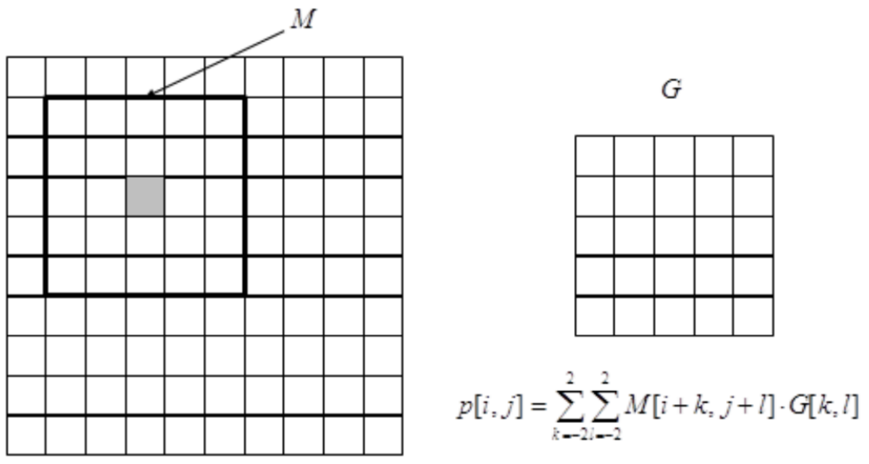
Noise Reduction

The first step is to filter out noises in the original image with the Gaussian filter. The Gaussian filter should be well defined since the filtering step not only eliminates the noise components but also results in the degradation of the image quality.

Once a suitable mask has been decided, the Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask slides over the image, casting a square of pixels at a time. The larger the size of the Gaussian mask, the less the detector's sensitivity to noise. The localization error in the detected edges also increases slightly as the mask size increases. The Gaussian mask shown in Figure 3 will be used in this lab.

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 3. Gaussian Mask G



$$p[i,j]=\sum_{k=-2}^{2}\sum_{l=-2}^{2}M[i+k,j+l]\cdot G[k,l]$$

Figure 4. Convolution using the Gaussian mask

Figure 4 shows the process of convolution. For the grey pixel p[i,j] in the left image, the window M will be selected from the original image and used in the convolution. The value of the corresponding pixel in the smoothed image is calculated using the equation shown in the figure.

Gradient Calculation

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. We use the Sobel operators in Figure 5 to perform a 2-D gradient measurement on an image. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (Vertical edge) and the other estimating the gradient in the y-direction (Horizontal edge). The mathematical equations in Figure 6 give the gradient value Gx and Gy in the x and y directions.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

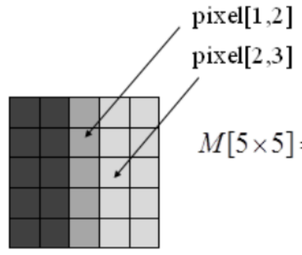
(a) (b)

Figure 5. Sobel operators (a) Sobel X, (b) Sobel Y

$$Gx[i,j] = \sum_{k=-1}^{1}\sum_{l=-1}^{1} M[i+k, j+l]\cdot SobelX[k,l]$$

$$Gy[i,j] = \sum_{k=-1}^{1}\sum_{l=-1}^{1} M[i+k, j+l]\cdot SobelY[k,l]$$

Figure 6. Equations of gradient calculation



(a)

$$M[5\times 5] = \begin{bmatrix} 0 & 0 & 30 & 100 & 100 \\ 0 & 0 & 30 & 100 & 100 \\ 0 & 0 & 30 & 100 & 100 \\ 0 & 0 & 30 & 100 & 100 \\ 0 & 0 & 30 & 100 & 100 \end{bmatrix} \quad Gx = \begin{bmatrix} . & . & . & . & . \\ . & 120 & 400 & 280 & . \\ . & 120 & 400 & 280 & . \\ . & 120 & 400 & 280 & . \\ . & . & . & . & . \end{bmatrix} \quad Gy = \begin{bmatrix} . & . & . & . & . \\ . & 0 & 0 & 0 & . \\ . & 0 & 0 & 0 & . \\ . & 0 & 0 & 0 & . \\ . & . & . & . & . \end{bmatrix}$$

(b) (c) (d)

Figure 7. An example of gradient calculation

Figure 7 shows an example of gradient calculation. The 5x5 image in Figure 7 (a) has pixel values in Figure 7 (b). It contains vertical edges (i.e., a large gradient in the x direction). Figures 7 (c) and (d) show the gradient after applying the convolutional masks SobelX and Sobel Y. For pixel [1,2], Gx = 400, Gy = 0. The large gradient in the x direction corresponds to the vertical edge at pixel [1,2]. For pixel [2,3], Gx = 280, Gy = 0. It also indicates a vertical edge at that location.

Based on Gx and Gy, we can calculate the **magnitude** and the **direction** of the gradient.
The **magnitude**, or edge strength, is approximated using the equation shown in Figure 8. It has a good performance with low computational complexity.

$$|G| = (|Gx|+|Gy|)/\alpha \quad (\alpha\text{: constant for matching full pixel level})$$

Figure 8. The magnitude of the gradient

In this equation, we use the constant value α for normalization. In our lab, each pixel is represented by eight bits, which gives a maximum value of 255. So the maximum value of Gx and Gy is 255*4. Then, the maximum value of |Gx| + |Gy| is 255*8. Therefore, setting $\alpha = 8$ makes the value of |G| between 0 and 255.

However, for real images, if we use a constant 8 to normalize the gradient, the magnitudes of most gradients will be very small. Therefore, we can adjust the constant value according to our used images. Make sure that the magnitude should not exceed 255 in our system. **In this lab, please use α = 2.**

The **direction** of the gradient can be calculated using the equation in Figure 9. However, implementing the arctan function is expensive in hardware. Instead, we will use an approximation method to determine the direction of the gradient.

$$\text{Theta} = \tan^{-1}(Gy/Gx)$$

Figure 9. The direction of the gradient

The purpose of calculating the direction of the gradient is to determine the direction of the edge. We categorize the edges into horizontal (0 degree), vertical (90 degree), and diagonal (45, 135 degree), as shown in Figure 10. Using the approximation method in Figure 11, we can quickly determine the direction of the edge by comparing the values of Gx and Gy.

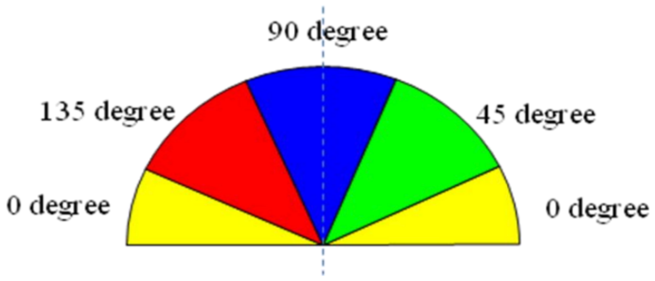


Figure 10. Directions of the edges

1st step: If($Gy<0$) { $Gx = Gx * (-1)$ $Gy = Gy * (-1)$ }

2nd step: $Gx \geq 0$, $Gy \leq 0.4*Gx \rightarrow$ degree 0
$0.4*Gx < Gy \leq 2.4*Gx \rightarrow$ degree 45
$2.4*Gx < Gy \rightarrow$ degree 90

$Gx < 0$, $Gy \leq -0.4*Gx \rightarrow$ degree 0
$-0.4*Gx < Gy \leq -2.4*Gx \rightarrow$ degree 135
$-2.4*Gx < Gy \rightarrow$ degree 90

Figure 11. An approximation method for gradient directions

Non-maximum Suppression

Due to multiple responses, edge magnitude M(x,y) may contain wide ridges around the local maxima. Non-maxima suppression removes the non-maxima pixels preserving the connectivity of the contours. Figure 12 shows an example of non-maximum suppression. Suppose we want to do a non-maximum suppression for pixel C. Following the two directions perpendicular to the edge, we can find the two neighboring pixels, A and B.

(1) If $M(C) \geq M(A)$ and $M(C) \geq M(B)$: discard pixels A and B by setting $M(A) = M(B) = 0$;
(2) Otherwise ($M(C) < M(A)$ or $M(C) < M(B)$): discard pixel C by setting $M(C) = 0$.

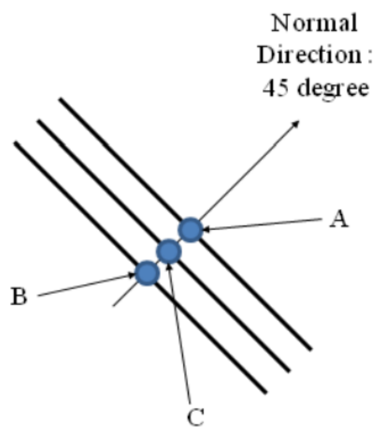


Figure 12. Non-maximum suppression

Hysteresis Thresholding

The output of the non-maxima suppression still contains noisy local maximum. In this lab, we use the hysteresis thresholding method to remove the noise further and improve the image quality.

The thresholds need to be set carefully to remove the weak edges while preserving the connectivity of the contours. This algorithm uses two thresholds, T_high and T_low.

(1) A pixel (x,y) is called **strong** if M(x,y) ≥ T_high;
(2) A pixel (x,y) is called **weak** if M(x,y) ≤ T_low;
(3) A pixel (x,y) is a candidate pixel otherwise (i.e., T_low < M(x,y) < T_high)) .

In each position of (x,y), we
(4) discard the pixel (x,y) if it is weak;
(5) keep the pixel if it is strong;
(6) If the pixel is a candidate, we should check its two neighbor pixels on its edge directions.

(6a) If the candidate pixel (x,y) is connected to a strong neighbor, keep the pixel;
(6b) If the candidate pixel (x,y) is connected to another candidate pixel that has already been regarded as a strong pixel, keep this candidate pixel;
(6c) Otherwise, discard the candidate pixel.

In our lab, please use T_high = 20 and T_low = 5.

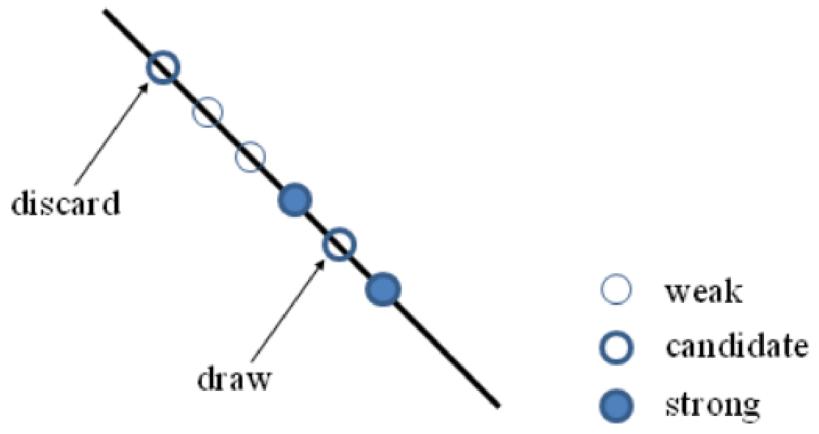


Figure 13. Hysteresis thresholding

Implementation & Simulation

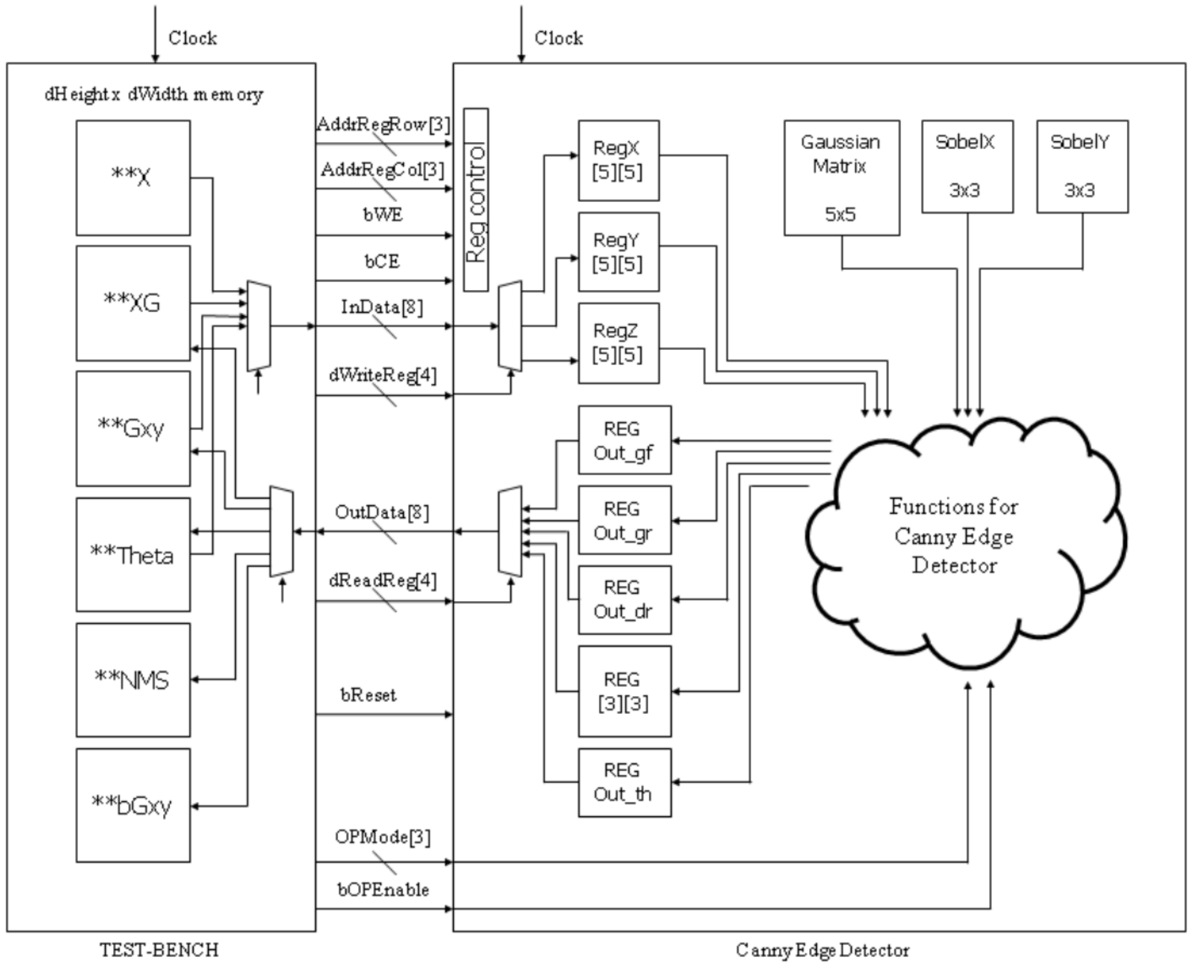


Figure 14. Data flow of the top module and test bench.

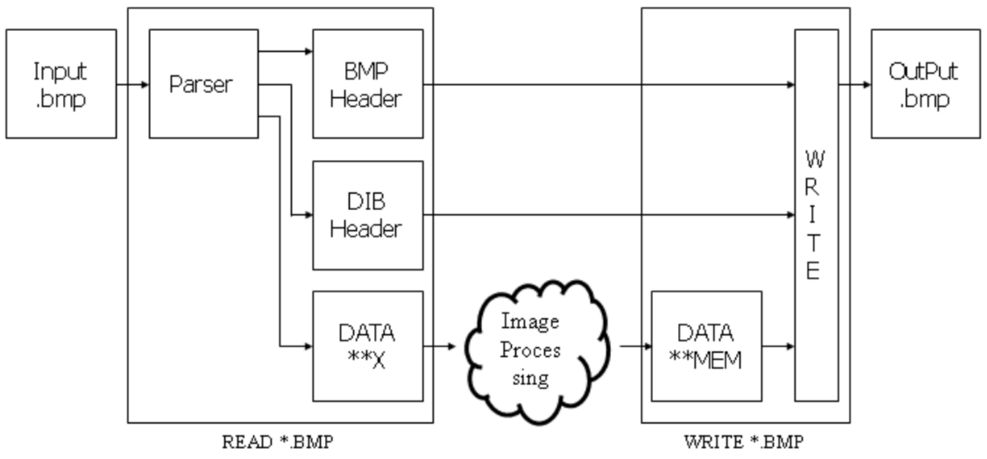


Figure 15. File in/out process

Figure 14 and 15 show the structure of the design.

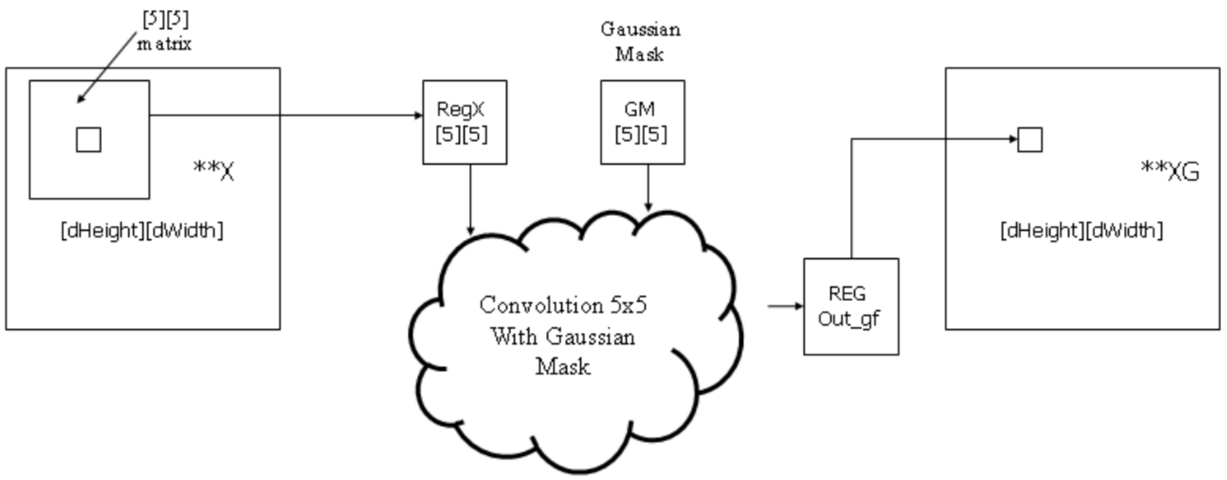


Figure 16. Data flow of noise reduction

Figure 16 shows the data flow of the noise reduction process. In the first step, the top module receives 5x5 data of the original image(**X) from the test bench. And the array of data is convoluted with a Gaussian mask for noise reduction at the noise reduction phase. Finally, the data is loaded into register Out_gf, and it goes out to another memory area (**XG) on the test bench.

Please login to the Olympus server and create a working directory for this lab using the following commands.

```
## Create and navigate to the working directory.
mkdir -p $HOME/ECEN468/Lab4/src
cd $HOME/ECEN468/Lab4/src
```

Download the zip file (lab4_code.zip) from Piazza and extract it. In the extracted folders, you will find files **"Canny_Edge.cpp", "Canny_Edge.h", "test.cpp", "test.h", "main.cpp", and "cman_200.bmp"**. Copy them to the working directory.

Once you complete the implementation, please verify the correctness of your design by simulation. You will see the output BMP files in your working directory and the output message in the Vista command window. Please include the screenshots of the simulation output message and the output files in the report.

Description of the BMP files:

1. OutputOrigin.bmp: The original image saved in local memory.
2. OutputGauss.bmp: The image after Gaussian filtering.
3. OutputGradient.bmp: The gradient components of the OutputGauss.bmp.
4. OutputAngle.bmp: The direction component based on the gradient Gx and Gy.
5. OutputNMS.bmp: The image after Non-Maximum Suppression.
6. OutputThres.bmp: The image after Hysteresis Threshold.

The code given only generates the first two images. To generate the rest of the images, please complete the design in "**Canny_Edge.cpp**".

For your convenience, the simulation will show the matching ratio between the images generated by your implementation, with the reference images.

Commands for reference:

```bash
load-ecen-468
source /opt/coe/mentorgraphics/vista312/setup.vista312.linux.bash
vista &
source /opt/coe/synopsys/wv/O-2018.09/setup.wv.sh
wv &
```

Submission

Please only submit one PDF file containing the following items:

1. Screenshots of the images generated by the simulation.
2. Screenshots of the simulation output in Vista.
3. Screenshots of your code in this design with reasonable comments.
4. Q1: You will see a lot of double lines for edges in the final output image. What are the possible reasons and solutions for this?