Overview

Canny Edge Detection, developed by John F. Canny in the late 80s, is an algorithm optimized to have single response edge detection and good localization. The algorithm is optimized to only mark an edge once and to not create false edges from image noise whenever possible. The algorithm also aims for low probability of failing to mark real edges and low probability of falsely marking non edges. Good localization requires the center of the true edge to be as close as possible to the detected edges. Canny Edge Detection consists of the following sequence of steps:

1) Conversion to grayscale image. The Canny Edge Detection algorithm outputs a black and white image to outline edges and it relies on grayscale intensity. As such, it is standard procedure to first convert a color image to a grayscale image.

2) Image Smoothing. A two dimensional Gaussian kernel is convolved with the image to reduce high frequency noise. This preprocessing step ensures better localization and single response edge detection.

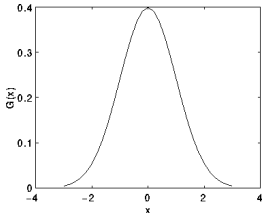
3) Calculating image gradient. It is common practice to use the Sobel operator to calculate gradient components, gradient magnitude, and gradient direction.

4) Non-maximum suppression. The Sobel operator gives thick edges, so non-maximum suppression is used to thin edges to single pixel width and reduce noise.

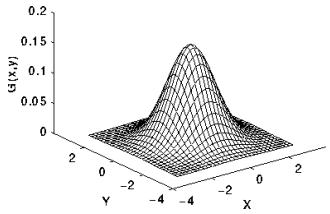
5) Hysteresis based thresholding. Double thresholding is used to further reduce noise and discard false edges.

Gaussian Smoothing

For preprocessing purposes it's important to first reduce the noise of an image by convolving it with a 2d Gaussian kernel ($n_\sigma * I$). The Gaussian function gives a symmetric distribution centered at the mean, known as the normal distribution. This bell-shaped distribution is desirable because its frequency is highest at the center and the frequency exponentially decreases as you stray further from the center. A kernel representation of a two dimensional Gaussian function can be convolved with an image to produce a weighted average of pixels. Given the bell shaped nature of the two dimensional Gaussian distribution, the pixels near the center of a Gaussian kernel are weighted more than those further away. This convolution eliminates pixels that inaccurately represent their surroundings. It does so by replacing the intensity level of each pixel in the original image with a weighted average of intensity values of pixels in its surrounding neighborhood. By placing more weight on pixels closer to the center, the Gaussian kernel reduces the noise created by distant pixels. This convolution results in a blurred image. The graph below (left) of the one dimensional Gaussian function illustrates a unimodal frequency peaking at the center. The equation below gives the two dimensional Gaussian function with $\boldsymbol{\mu} = 0$. The 2 dimensional Gaussian distribution is represented by the graph on the right with $\boldsymbol{\mu} = 0$ and $\boldsymbol{\sigma} = 1$.



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

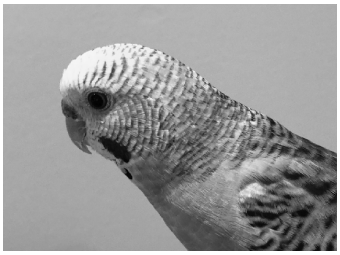


Technically, it is impossible to map a two dimensional Gaussian distribution to a finite kernel, as the function is non zero across the entire real domain (infinitely non zero). Practically, we only need to consider the Gaussian distribution within 3 standard deviations from the mean, as 99.7% of the distribution lies in this range. Since the two dimensional Gaussian distribution is a continuous function and images are discrete matrices of pixels, in order to construct a kernel representation we need to discreetly approximate the 2d Gaussian function. One method is to "fit" the kernel on top of the Gausssian distribution (the center pixel of the kernel is centered at the peak of the distribution and the boundaries of the kernel extend to 3 standard deviations from the mean). The intensity value of each pixel in the kernel is calculated by integrating the portion of the Gaussian distribution that lies in the corresponding pixel. Each intensity value in the kernel is divided by the sum of all pixel intensity values to create a weighted average Gaussian kernel. It's important to note that the standard deviation determines the width of the Gaussian distribution. Thus, a higher standard deviation requires a larger kernel. The kernel below demonstrates one such discrete approximation of the 2d Gaussian function convolved with an image to smooth its noise.

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

*

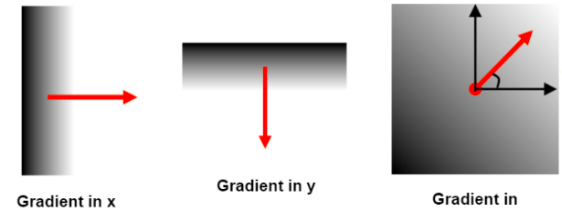


=

This is my budgie Pooki.

Intensity Gradient

Next, we must take our blurred image and find edges by determining the intensity gradient at each pixel ($\nabla n_\sigma * I = G$). An edge is defined to be a rapid change of brightness in a small region. When given a one dimensional continuous intensity function f(x), we can simply compute f'(x) to determine edge strength and use local extrema to determine precise edge location. In a two dimensional image, the gradient of its intensity function will give us the maximum rate of change in intensity (magnitude of gradient vector) and the direction of this maximum rate of change. In a continuous intensity function f(x,y), the gradient G is the following vector: [∂f/∂x, ∂f/∂y]. ∂f/∂x gives the rate of intensity change in the x-direction. Similarly, ∂f/∂y gives the rate of intensity change in the y-direction. The pictures below illustrate the gradients [∂f/∂x, 0], [0, ∂f/∂y] , and [∂f/∂x, ∂f/∂y] respectively.



Gradient in x

Gradient in y

Gradient in both x and y

In order to indicate edges in a grayscale image, we set the gradient magnitude of each pixel to be the new intensity level of the pixel because gradient magnitude equals edge strength. For discrete images, we need to use a discrete gradient operator that approximates the gradient of a continuous function. We can utilize many operators for this approximation such as the Robert Cross and Prewitt operators, but for Canny edge detection it is most common to use the Sobel operator. This discrete differentiation operator first computes approximate gradient components G_x and G_y. The gradient components are calculated using x-direction and y-direction Sobel kernels. The three by three Sobel kernels shown below are weighted to emphasize intensity change in their respective direction. These kernels are separately convolved with the image to calculate the gradient components G_x and G_y. To demonstrate this process, consider the red pixel in image I. By convolving image I with the x-direction kernel, the red pixel's intensity becomes 40. When convolving image I with the y-direction kernel, the red pixel's intensity becomes 0. These intensity values are rough indicators of rate of intensity change in the x and y direction for the red pixel.

$I =$

10	10	20	20
10	10	20	20
10	10	20	20
10	10	20	20
10	10	20	20

X – Direction Kernel

$G_x =$

-1	0	1
-2	0	2
-1	0	1

$* I$

Y – Direction Kernel

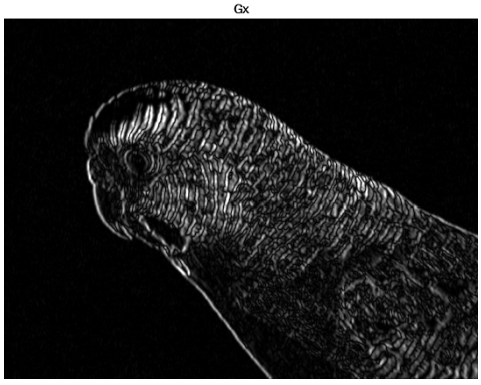
$G_y =$

-1	-2	-1
0	0	0
1	2	1

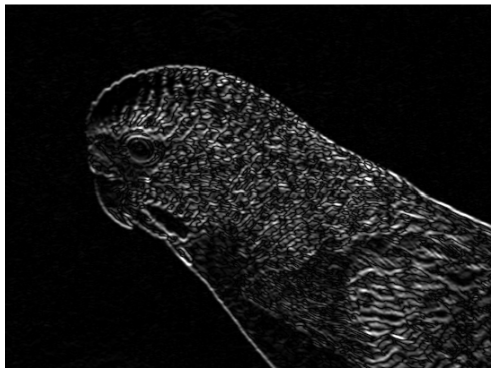
$* I$

Once we obtain Gx and Gy, we can calculate the approximate gradient magnitude of the image. This is simply done by the applying the following formula to each corresponding pair of pixels in Gx and Gy: $||G|| = \sqrt{Gx^2 + Gy^2}$ and setting this magnitude equal to the intensity value of the pixel. The output image ||G|| will highlight strong edges (large gradient). We can also determine the orientation of each pixel's gradient vector using the following formula: $\theta = arctan(\frac{Gy}{Gx})$. The gradient vector is perpendicular to the edge, so we can also determine the edge direction. The images below show G_x, G_y, and ||G|| applied to our blurred picture. In G_x, the intensity value represents the rate of change in the x direction (which is why you get vertical edges). In G_y, the intensity value represents rate of change in the y direction (which is why you get horizontal edges).

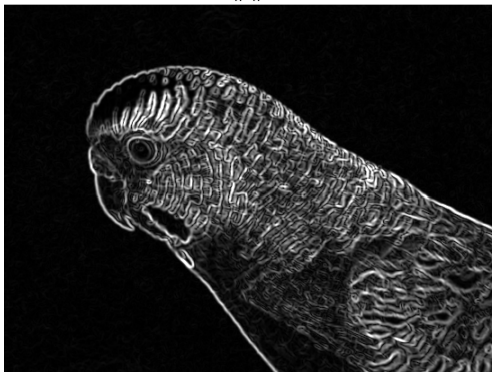
Gx



Gy



||G||

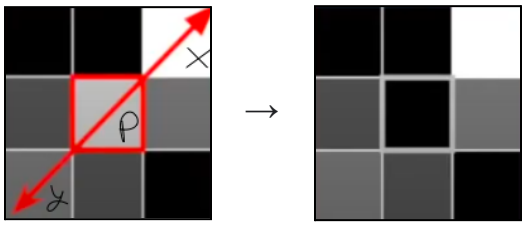


Non-maximum Suppression

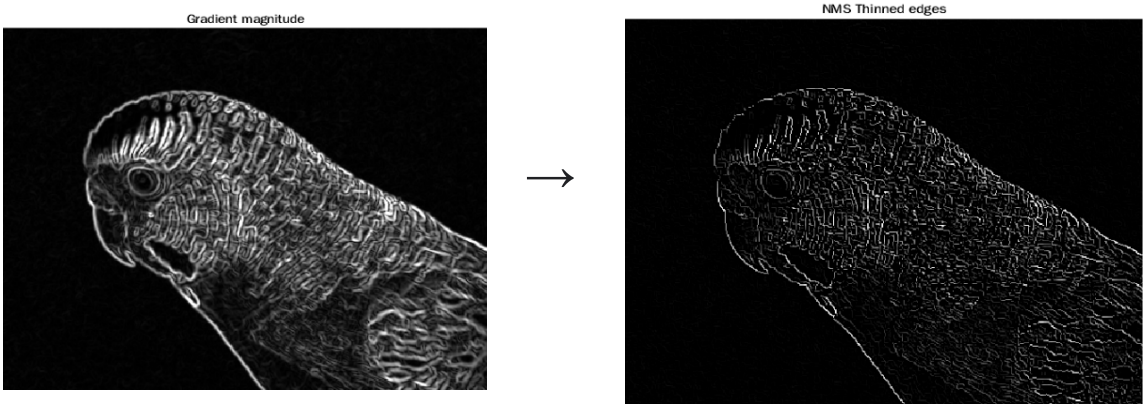
From the pictures above it seems that approximating the image's gradient magnitude is enough to find edges. However, these gradient magnitude intensity edges are too thick to precisely locate the real edges. To fix this, the Canny algorithm performs an edge thinning procedure known as non-maximum suppression. For all pixels p in ||G||, the following steps are taken. First calculate the gradient vector's orientation of pixel p using the formula: $\theta = arctan(\frac{Gy}{Gx})$. Next, we need to relate the gradient vector's orientation to one of the following four directions: east west, north south, positive diagonal, or negative diagonal. This is done by rounding θ to 0 (east west), 45(positive diagonal), 90 (north south), or 135 degrees (negative diagonal) using the following bounds:

[0, 22.5) ∨ [157.5, 202.5) ∨ [337.5, 360] → θ = 0
[22.5, 67.5) ∨ [202.5, 247.5) → θ = 45
[67.5, 112.5) ∨ [247.5, 292.5) → θ = 90
[112.5, 157.5) ∨ [292.5, 337.5) → θ = 135

Next, we use the rounded θ to find the two neighboring pixels of p that go along one of these four directions. If p is not a local maxima of this set of three pixels (at least one neighboring pixel of p has a larger gradient magnitude, represented by intensity value), then we suppress p by setting its intensity value to 0 (making it black). Consider the image below. The pixel p's gradient vector points in the positive diagonal (θ is rounded to 45) direction. The two neighboring pixels that lie on the positive diagonal are labeled x and y. Since x has a higher intensity value than p, we suppress p's intensity to 0.



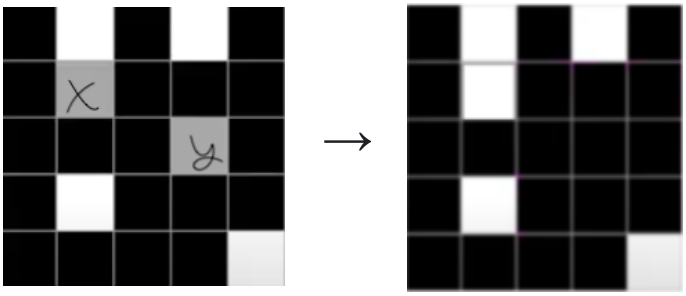
As shown below, the gradient magnitude image (left) has thick, noisy edges. After applying non-maximum suppression, the image (right) has much thinner (single pixel width), less noisy edges.



Hysteresis-Based Thresholding

Even after thinning out edges, there is still noise and false edges that we need to eliminate. The Canny algorithm performs hysteresis based thresholding to eliminate non edges. Single thresholding is the process of letting some intensity value t be the threshold. We compare each pixel's intensity value with the threshold. The pixel's intensity is set to 0 (black) if it's below the threshold, or 1 (white) if it's above the threshold. The issue with single thresholding for edge detection is streaking. Streaking occurs when an edge has intensity values that oscillate above and below the threshold. This leads to dashed lines as edges.

Hysteresis based thresholding is used to resolve this issue. A high threshold T_1 and a low threshold T_2 are empirically determined. Any pixel with intensity above T_1 is labeled as strong and immediately determined to be an edge pixel (set to 1). Any pixel with intensity below T_2 is labeled as irrelevant and immediately determined to be a non edge pixel (set to 0). Pixels with intensity values between T_1 and T_2 are labeled as weak pixels. For each weak pixel, the algorithm checks if one of its neighbors is a strong pixel (if it is directly touching a strong pixel). If at least one neighboring pixel is a strong pixel, the weak pixel is determined to be an edge pixel (set to 1). If none of the weak pixel's neighbors are strong pixels, then the weak pixel is determined to be a non edge pixel (set to 0). In the image below, the white pixels represent strong pixels, the black pixels represent non edge pixels, and pixels x and y represent weak pixels. Since pixel x has a strong pixel neighbor, it becomes an edge pixel. Since y has no strong pixel neighbors, it becomes a non edge pixel.

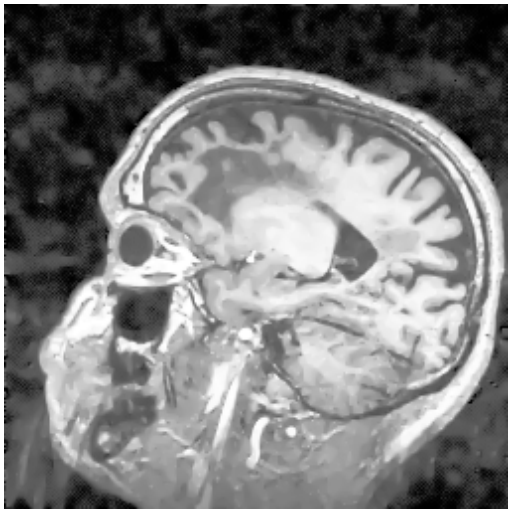


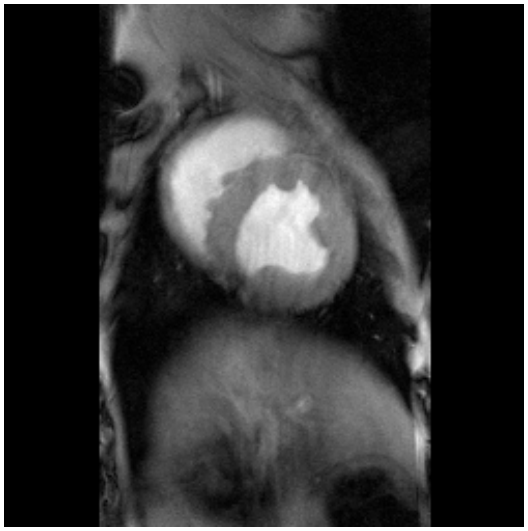
Weaknesses of Canny's Algorithm

The first issue with Canny edge detection is use of the Gaussian smoothing to reduce noise. Gaussian smoothing reduces all high frequency signals. This means both noise and some weak edges will be smoothed out of the picture. Ideally, we need a smoothing method that only targets noise and has little effect on smoothing edges. Another issue is using the three by three sobel kernels as they make the gradient magnitude calculation sensitive to noise. Using a larger kernel or other operators such as Prewitt and Scharr can reduce noise sensitivity. The hysteresis based thresholding step also has room for improvement. Empirically deciding the values of two global thresholds is a slow process. This limits the algorithm's adaptability to detect edges of

many different images. Also by using fixed global values for the upper and lower thresholds, it becomes difficult to accurately find real edges for more complex images. We need an adaptive thresholding method that determines local thresholds for smaller regions of complex images.

Final Results of Canny Edge Detection:





Sources

[1] A, Robert (n.d.). *Gaussian filtering - auckland*. Auckland University Computer Vision Lectures. Retrieved May 3, 2022, from https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf

[2] G. Jie and L. Ning, "An Improved Adaptive Threshold Canny Edge Detection Algorithm," 2012 International Conference on Computer Science and Electronics Engineering, 2012, pp. 164-168, doi: 10.1109/ICCSEE.2012.154.

[3] Himanshu, A. (2009). (rep.). *Study and Comparison of Various Image Edge Detection Techniques* (pp. 1–12). Patiala, Punjab: IJIP.

[4] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[5] Vincent, R. (2009). (tech.). *A Descriptive Algorithm for Sobel Image Edge Detection* (pp. 99–105). Abeokuta, Ogun: InSite.

[6] Zhang Jin-Yu, Chen Yan and Huang Xian-Xiang, "Edge detection of images based on improved Sobel operator and genetic algorithms," 2009 International Conference on Image Analysis and Signal Processing, 2009, pp. 31-35, doi: 10.1109/IASP.2009.5054605.