Canny edge detection algorithm

1. **PROBLEM DISCRIPTION:**
   Detect the edges of the input image using canny edge detection algorithm.

2. **ALGORITHM:**
   The Canny edge detection is a multi-stage algorithm to detect a wide range of edges in images.
   This algorithm is implemented in five different steps:
   - **Smoothing:** Blurring of the image to remove noise.
   - **Finding gradients:** The edges should be marked where the gradients of the image have large magnitudes.
   - **Non-maximum suppression:** Only local maxima should be marked as edges.
   - **Double thresholding:** Potential edges are determined by thresholding.
   - **Edge tracking by hysteresis:** Final edges are determined by suppressing all edges that are not connected to a very strong edge.

   To detect edges first Image to be processed.
   Step 1: convert the Image to gray Scale to detect the edges accurately.

   Step 2: And then after converting to gray scale. Smoothing process should be done by applying a Guassian_filter. The kernel of a Gaussian filter with a standard deviation of $\sigma = 1.4$.

   Step 3: IF GUASSION_FILTER IS DONE THEN go to calculating the gradient of image intensity at each pixel within the image. Such as (gx, gy) by applying sobelFilter.

   Step 4: IF CALCULATING OF GRADIENT IS DONE THEN. Apply the Non-Maximum-Suppression to convert the blurred edges in the image of the gradient magnitude to sharp edges.

   Step 5: Next apply double thresholding and find all the weak edges connected through strong edges. And to find all the strong and weak edges.

Step 6: Finally String edges are considered as certain edges and included in the final edge image. And weak edges are included if and only iff they are connected to strong edges.

CODE OF THE ABOVE ALGORITHM:

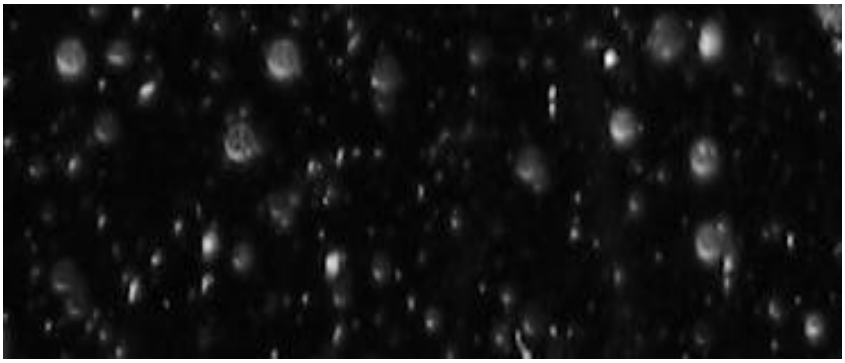
```
from scipy import misc
        import imageio
        from scipy import ndimage
        import numpy as np
        import matplotlib.pyplot as plt


        img = imageio.imread('snowflakes.png')
        img = img.astype('int32')

        # Blur the grayscale image so that only important edges are extracted and the noisy ones
        ignored
        img_guassian_filter = ndimage.gaussian_filter(img, sigma=1.4)
        #cv2.imshow("new",img_guassian_filter);


        def SobelFilter(img, direction):
           if(direction == 'x'):
              Gx = np.array([[-1,0,+1], [-2,0,+2],  [-1,0,+1]])
              Res = ndimage.convolve(img, Gx)
           if(direction == 'y'):
              Gy = np.array([[-1,-2,-1], [0,0,0], [+1,+2,+1]])
              Res = ndimage.convolve(img, Gy)

           return Res
        def Normalize(img):
           img = np.multiply(img, 255 / np.max(img))
           return img



        gx = SobelFilter(img_guassian_filter, 'x')
        gx = Normalize(gx)

        #cv2.imshow("new",gx)
        gy = SobelFilter(img_guassian_filter, 'y')
```

```
gy = Normalize(gy)

#cv2.imshow("new",gy)
type(gx)
gx.shape

dx = ndimage.sobel(img_guassian_filter, axis=1) # horizontal derivative
dy = ndimage.sobel(img_guassian_filter, axis=0) # vertical derivative

type(dx)
dx.shape
dy.shape


Mag = np.hypot(gx,gy)
Mag.shape

mag = np.hypot(dx,dy)
mag.shape

#cv2.imshow("new", Mag)

Gradient = np.degrees(np.arctan2(gy,gx))
gradient = np.degrees(np.arctan2(dy,dx))

def NonMaxSup(Gmag, Grad, Gx, Gy):
   NMS = np.zeros(Gmag.shape)

   for i in range(1, int(Gmag.shape[0]) - 1):
      for j in range(1, int(Gmag.shape[1]) - 1):
         if((Grad[i,j] >= 0 and Grad[i,j] <= 45) or (Grad[i,j] < -135 and Grad[i,j] >= -180)):
            yBot = np.array([Gmag[i,j+1], Gmag[i+1,j+1]])
            yTop = np.array([Gmag[i,j-1], Gmag[i-1,j-1]])
            x_est = np.absolute(Gy[i,j]/Gmag[i,j])
            if (Gmag[i,j] >= ((yBot[1]-yBot[0])*x_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-
yTop[0])*x_est+yTop[0])):
                NMS[i,j] = Gmag[i,j]

            else:
               NMS[i,j] = 0
         if((Grad[i,j] > 45 and Grad[i,j] <= 90) or (Grad[i,j] < -90 and Grad[i,j] >= -135)):
            yBot = np.array([Gmag[i+1,j] ,Gmag[i+1,j+1]])
            yTop = np.array([Gmag[i-1,j] ,Gmag[i-1,j-1]])
            x_est = np.absolute(Gx[i,j]/Gmag[i,j])
```

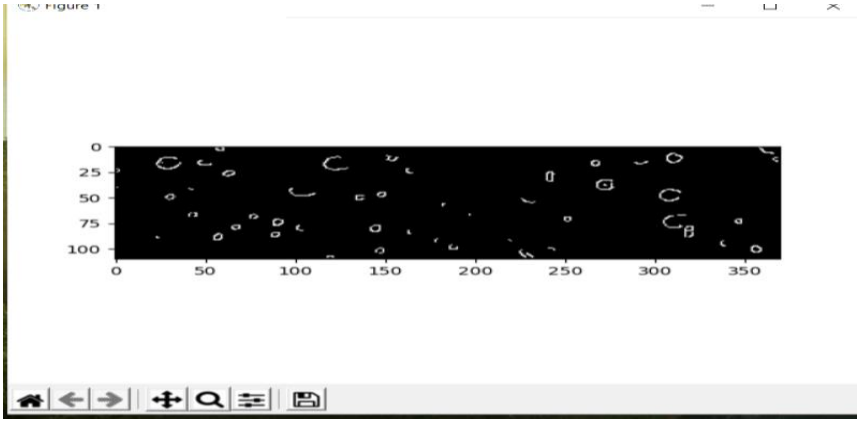
```
            if (Gmag[i,j] >= ((yBot[1]-yBot[0])*x_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])*x_est+yTop[0])):
                NMS[i,j] =Gmag[i,j]
            else:
                NMS[i,j] = 0
        if((Grad[i,j] > 90 and Grad[i,j] <= 135) or (Grad[i,j] < -45 and Grad[i,j] >= -90)):
            yBot = np.array([Gmag[i+1,j] ,Gmag[i+1,j-1]])
            yTop = np.array([Gmag[i-1,j] ,Gmag[i-1,j+1]])
            x_est = np.absolute(Gx[i,j]/Gmag[i,j])
            if (Gmag[i,j] >= ((yBot[1]-yBot[0])*x_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])*x_est+yTop[0])):
                NMS[i,j] =Gmag[i,j]
            else:
                NMS[i,j] = 0
        if((Grad[i,j] > 135 and Grad[i,j] <= 180) or (Grad[i,j] < 0 and Grad[i,j] >= -45)):
            yBot = np.array([Gmag[i,j-1] ,Gmag[i+1,j-1]])
            yTop = np.array([Gmag[i,j+1] ,Gmag[i-1,j+1]])
            x_est = np.absolute(Gy[i,j]/Gmag[i,j])
            if (Gmag[i,j] >= ((yBot[1]-yBot[0])*x_est+yBot[0]) and Gmag[i,j] >= ((yTop[1]-yTop[0])*x_est+yTop[0])):
                NMS[i,j] =Gmag[i,j]
            else:
                NMS[i,j] = 0

    return NMS


NMS = NonMaxSup(mag, gradient, dx, dy)
NMS = Normalize(NMS)

nms = NonMaxSup(mag, gradient, dx, dy)
nms = Normalize(nms)


def DoThreshHyst(img):
    highThresholdRatio =0.32


    lowThresholdRatio = 0.30
    GSup = np.copy(img)
    h = int(GSup.shape[0])
    w = int(GSup.shape[1])
    highThreshold = np.max(GSup) * highThresholdRatio
    lowThreshold = highThreshold * lowThresholdRatio
```

```
    x = 0.1
    oldx=0

    for i in range(1,h-1):
        for j in range(1,w-1):
            if(GSup[i,j] > highThreshold):
                GSup[i,j] = 1
            elif(GSup[i,j] < lowThreshold):
                GSup[i,j] = 0
            else:
                if((GSup[i-1,j-1] > highThreshold) or
                    (GSup[i-1,j] > highThreshold) or
                    (GSup[i-1,j+1] > highThreshold) or
                    (GSup[i,j-1] > highThreshold) or
                    (GSup[i,j+1] > highThreshold) or
                    (GSup[i+1,j-1] > highThreshold) or
                    (GSup[i+1,j] > highThreshold) or
                    (GSup[i+1,j+1] > highThreshold)):
                    GSup[i,j] = 1


    GSup = (GSup == 1) * GSup # This is done to remove/clean all the weak edges which are
not connected to strong edges

    return GSup

Final_Image = DoThreshHyst(NMS)
plt.imshow(Final_Image, cmap = plt.get_cmap('gray'))
plt.show()
```

3. INPUT IMAGE:



4. OUTPUT IMAGE:



5. DISCUSSION:

Canny edge detection algorithm detects the edges of the image. By following the five steps:

Smoothing: all the images taken from the camera will contain some noise. so, noise has to be reduced in-order to detect the accurate edges. First image smoothed by applying a Gaussian filter. The kernel of a Gaussian filter with a standard deviation of $\sigma = 1.4$.

Finding gradients: The Canny algorithm basically finds edges where the grayscale intensity of the image changes the most. These areas are found by determining gradients of the image. Gradients at each pixel in the smoothed image is determined by applying the Sobel-operator. First step is to approximate the gradient in the x- and y-direction respectively by applying the kernels.

The gradient magnitudes can be determined as a Euclidean distance measure by applying the law of Pythagoras. To reduce the computational complexity. The Euclidean distance measure has been applied to the test image. The computed edge strengths are compared to the smoothed.

|G| = squareRoot(G2 x + G2y)Gx and Gy are the gradients in the x- and y-directions respectively.

Non-maximum suppression: To convert the blurred edges in the image of the gradient magnitudes to sharp edges. Basically, this is done by preserving all local maxima in the gradient image, and deleting everything else. The algorithm is for each pixel in the gradient image.

Double thresholding: The edge-pixels remaining after the non-maximum suppression step is marked with their strength pixel-by-pixel. Many of these will be true edges in the image, but some may be caused by noise due to rough surfaces. The simplest way to discern between these would be to use a threshold, so that only edges stronger that a certain value would be preserved. The Canny edge detection algorithm uses double thresholding. Edge pixels stronger than the high threshold is marked as strong. weaker than the low threshold is suppressed and between the two thresholds are marked as weak.

Edge tracking by hysteresis: Strong edges are interpreted as certain edges, and can be included in the final edge image. Weak edges are included if and only if they are connected to strong edges.
strong edges will only be due to true edges in the original image. The weak edges can either be due to true edges or noise variations. The latter type will probably be distributed independently of edges on the entire image, and thus only a small amount will be located adjacent to strong edges. Weak edges due to true edges are much more likely to be connected directly to strong edges.