Canny Edge Detector

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
import matplotlib.image as mpimg
```

In [3]:

```
def convolution(image, kernel, average=False):
    image_row, image_col = image.shape
    kernel_row, kernel_col = kernel.shape

    output = np.zeros(image.shape)

    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)

    padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 * pad_

    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded

    for row in range(image_row):
        for col in range(image_col):
            output[row, col] = np.sum(kernel * padded_image[row:row + kernel_row
            if average:
                output[row, col] /= kernel.shape[0] * kernel.shape[1]

    print("Output size of image: {}".format(output.shape))

    return output
```

In [4]:

```
def normal_dist(x, mu, sd):
    return 1 / (np.sqrt(2 * np.pi) * sd) * np.e ** (-np.power((x - mu) / sd, 2)

def gaussian_kernel(size, sigma=1):
    kernel_1D = np.linspace(-(size // 2), size // 2, size)
    for i in range(size):
        kernel_1D[i] = normal_dist(kernel_1D[i], 0, sigma)
    kernel_2D = np.outer(kernel_1D.T, kernel_1D.T)

    kernel_2D *= 1.0 / kernel_2D.max()

    return kernel_2D

def gaussian_blur(image, kernel_size):
    kernel = gaussian_kernel(kernel_size, sigma=1)
    return convolution(image, kernel, average=True)
```

In [5]:

```
def sobel_edge_detection(image, filter_x, filter_y):
    new_image_x = convolution(image, filter_x)

    new_image_y = convolution(image, filter_y)
    gradient_magnitude = np.sqrt(np.square(new_image_x) + np.square(new_image_y)

    gradient_magnitude *= 255.0 / gradient_magnitude.max()

    gradient_direction = np.arctan2(new_image_y, new_image_x)

    return gradient_magnitude, gradient_direction
```

In [6]:

```
def non_max_suppression(gradient_magnitude, gradient_direction):
    image_row, image_col = gradient_magnitude.shape

    output = np.zeros(gradient_magnitude.shape)

    PI = 180

    for row in range(1, image_row - 1):
        for col in range(1, image_col - 1):
            direction = gradient_direction[row, col]

            if (0 <= direction < PI / 8) or (15 * PI / 8 <= direction <= 2 * PI)
                before_pixel = gradient_magnitude[row, col - 1]
                after_pixel = gradient_magnitude[row, col + 1]

            elif (PI / 8 <= direction < 3 * PI / 8) or (9 * PI / 8 <= direction
                before_pixel = gradient_magnitude[row + 1, col - 1]
                after_pixel = gradient_magnitude[row - 1, col + 1]

            elif (3 * PI / 8 <= direction < 5 * PI / 8) or (11 * PI / 8 <= direc
                before_pixel = gradient_magnitude[row - 1, col]
                after_pixel = gradient_magnitude[row + 1, col]

            else:
                before_pixel = gradient_magnitude[row - 1, col - 1]
                after_pixel = gradient_magnitude[row + 1, col + 1]

            if gradient_magnitude[row, col] >= before_pixel and gradient_magnitu
                output[row, col] = gradient_magnitude[row, col]
    return output
```

In [7]:

```
def threshold(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):

    highThreshold = img.max() * highThresholdRatio;
    lowThreshold = highThreshold * lowThresholdRatio;

    M, N = img.shape
    res = np.zeros((M,N), dtype=np.int32)

    weak = np.int32(25)
    strong = np.int32(255)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak
    return (res, weak, strong)
```

In [8]:

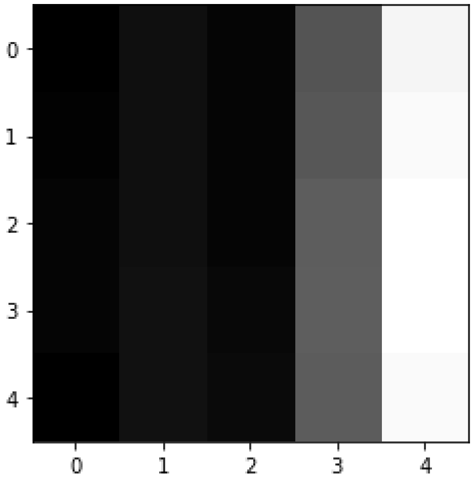
```
def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or
                        or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] == strong)
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

In [9]:

```
#Read input image patch 5*5
img = mpimg.imread('edge.jpg')
print(img)
plt.imshow(img, 'gray')
```

```
[[ 28  39  32  87 199]
 [ 30  39  32  89 202]
 [ 32  39  32  93 206]
 [ 32  40  34  94 206]
 [ 28  40  35  92 202]]
```

Out[9]: `<matplotlib.image.AxesImage at 0x7fdbd88c8790>`



In [10]:

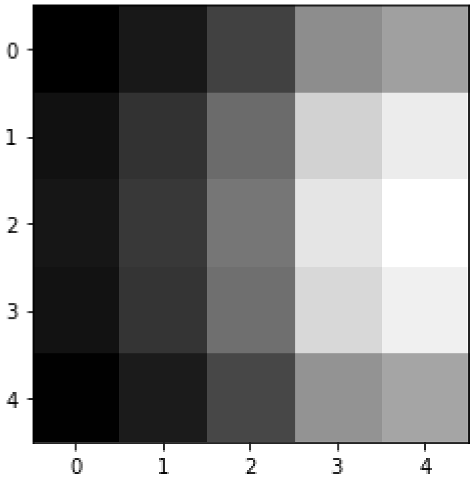
```
#Define Gaussian Kernel
kernel = gaussian_kernel(5, sigma=1)
kernel
```

Out[10]:

```
array([[0.01831564, 0.082085  , 0.13533528, 0.082085  , 0.01831564],
       [0.082085  , 0.36787944, 0.60653066, 0.36787944, 0.082085  ],
       [0.13533528, 0.60653066, 1.        , 0.60653066, 0.13533528],
       [0.082085  , 0.36787944, 0.60653066, 0.36787944, 0.082085  ],
       [0.01831564, 0.082085  , 0.13533528, 0.082085  , 0.01831564]])
```

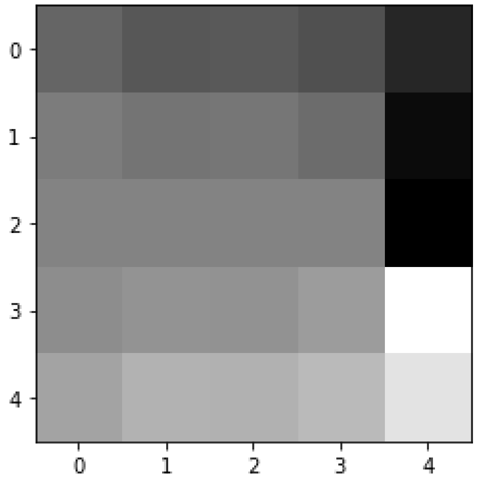
```
In [11]: #Step 1: Apply Gaussian Filtering for noise reduction
         blurred_patch = gaussian_blur(img, 5)
         plt.imshow(blurred_patch, 'gray')
         blurred_patch
```

```
         Output size of image: (5, 5)
Out[11]: array([[ 3.97094029,  6.12678662,  9.7684125 , 16.33958849, 18.00340614],
                [ 5.46248646,  8.35616697, 13.31576016, 22.29897595, 24.54585617],
                [ 5.894442  ,  8.97986522, 14.31531486, 23.94299137, 26.28527793],
                [ 5.58656441,  8.58289256, 13.70980996, 22.81533724, 24.95466299],
                [ 4.08124618,  6.36617629, 10.20673402, 16.90344986, 18.4381401 ]])
```



```
In [12]: #Step 2 compute horizontal and vertical changes with 1st derivatives
         sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
         sobel_y = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
         gradient_magnitude, gradient_direction = sobel_edge_detection(blurred_patch, sob
         print(gradient_magnitude)
         print(gradient_direction)
         plt.imshow(gradient_magnitude, 'gray')
         plt.imshow(gradient_direction, 'gray')
```

```
         Output size of image: (5, 5)
         Output size of image: (5, 5)
         [[ 77.37676024 110.95414556 183.1564292  238.489607   247.04071619]
          [ 89.14937328  88.57540208 155.03610336 139.96649945 241.96091448]
          [ 95.68894147  90.0154443  159.34899424 127.35257011 255.        ]
          [ 90.76283016  89.78638995 155.85480713 137.09152526 245.14532546]
          [ 79.67957603 114.51407213 188.0800883  243.27191058 252.69212218]]
         [[-0.7521046  -1.06950336 -1.03042285 -1.24671669 -2.22703944]
          [-0.2075368  -0.38644398 -0.35301189 -0.58135534 -2.8642129 ]
          [-0.01360655 -0.02959618 -0.02634755 -0.03952662 -3.12724987]
          [ 0.18962658  0.34738649  0.31978071  0.5476247   2.88452114]
          [ 0.74745253  1.06123941  1.03017331  1.25322316  2.23234145]]
Out[12]: <matplotlib.image.AxesImage at 0x7fdc3910ac70>
```



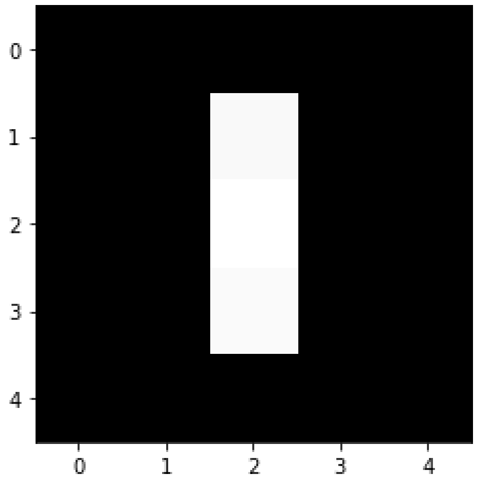
In [13]:

```
#Step 3 Non max supression
non_max_img = non_max_suppression(gradient_magnitude, gradient_direction)
print(non_max_img)
plt.imshow(non_max_img, 'gray')
```

```
[[  0.           0.           0.           0.           0.        ]
 [  0.           0.         155.03610336   0.           0.        ]
 [  0.           0.         159.34899424   0.           0.        ]
 [  0.           0.         155.85480713   0.           0.        ]
 [  0.           0.           0.           0.           0.        ]]
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x7fdbd89c2ee0>
```



In [14]:

```
#Step 4 Double threshold
res, weak, strong = threshold(non_max_img)
print(res, weak, strong)
```

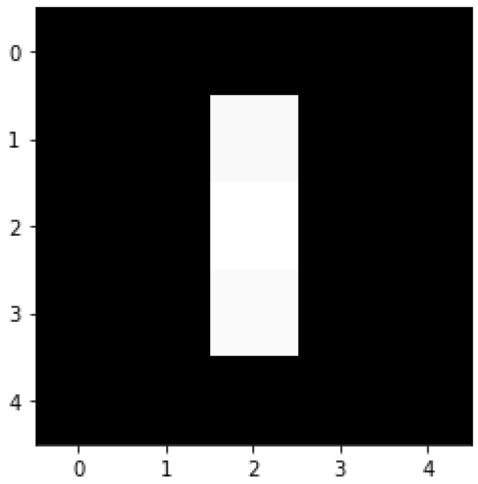
```
[[  0   0   0   0   0]
 [  0   0 255   0   0]
 [  0   0 255   0   0]
 [  0   0 255   0   0]
 [  0   0   0   0   0]] 25 255
```

In [15]:

```
# Step 5: Hysterisis
final = hysteresis(non_max_img, weak, strong)
print(final)
plt.imshow(final, 'gray')
```

```
[[   0.            0.            0.            0.            0.        ]
 [   0.            0.          155.03610336   0.            0.        ]
 [   0.            0.          159.34899424   0.            0.        ]
 [   0.            0.          155.85480713   0.            0.        ]
 [   0.            0.            0.            0.            0.        ]]
```

```
Out[15]: <matplotlib.image.AxesImage at 0x7fdbf83488b0>
```


0
1
2
3
4
0
1
2
3
4