Week 2: Lab 1: Edge Detection

Task 1:

This task involves two key concepts: **edge strength** and **threshold**.

Edge strength refers to the quantified level of how distinct the edge is at each pixel point in an image. It is usually determined by the **gradient magnitude of the pixel**, where the gradient represents **the rate of change in brightness** at that point.

In Task 1, edge strength is calculated by applying a **Sobel** filter. The Sobel filter detects edges in both **horizontal and vertical directions** by calculating the Pythagorean sum **(the square root of the sum of squares)** of the gradients in these two directions, thus integrating this edge information.

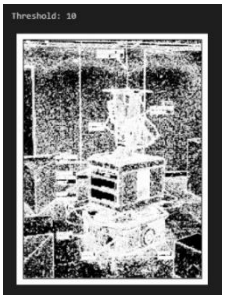
```
def magnitude(x,y):
    return np.sqrt(x**2 + y**2)
```

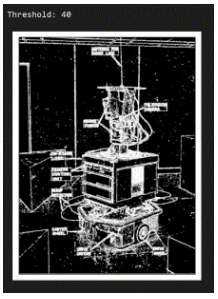
The threshold is a predefined limit used to determine whether the edge strength of a pixel (**the size of the pixel's gradient** or **the rate of brightness change at that point**) is high enough to be considered as part of an edge. The choice of threshold determines which areas the algorithm will consider as edges. A lower threshold will capture more subtle brightness changes, while a higher threshold focuses only on more **significant changes**.

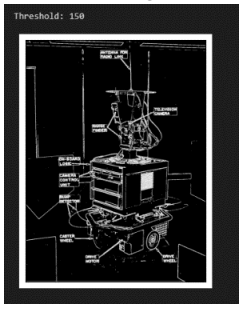
In Task 1:

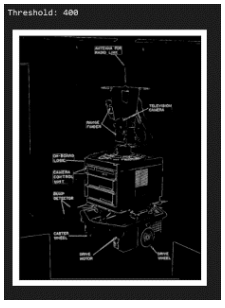
- When the (**threshold >10**), very weak brightness change rates are detected. This may lead to a lot of noise in the overall image, as changes that are not significantly marked as edges are also tagged.
- When the (**threshold >40**), only more noticeable brightness change rates can pass the threshold check. This reduces some overall image noise, but the real image edge strength may not be sufficiently prominent.
- When the (**threshold > 150**), there is no apparent noise in the overall image, and only strong edges (brightness change rates) are detected, allowing for good capture of image edge strength.
- When the (**threshold > 400**), only very strong edges (brightness change rates) are detected. This might result in many real edges being overlooked, especially those not particularly prominent in terms of brightness change rates, making it difficult to observe true edges.

Therefore, choosing a threshold requires balancing between avoiding noise and capturing real edges. A threshold that is too low might lead to a lot of noise in the image (incorrectly identifying non-edge regions as edges), while one that is too high might result in the loss of important details and weak edges.









Task 2:

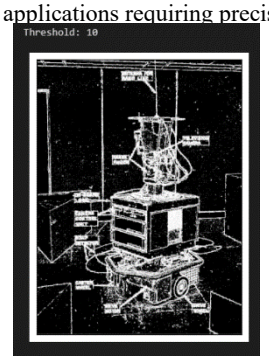
```
roberts_x = np.array(
    [[1,0],
     [0,-1]])

roberts_y = np.array(
    [[0,1],
     [-1,0]])
```

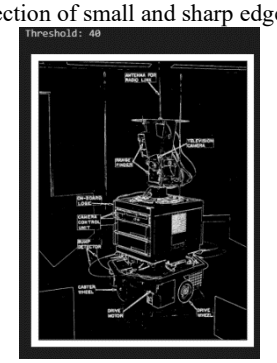
In Task2: The Roberts operator excels in edge detection, especially in identifying sharp edges and fine details along **diagonal directions**, mainly due to its unique design and working principles. Utilizing a **2x2 small kernel**, the Roberts operator can precisely locate local changes in an image, particularly those along **diagonals**.

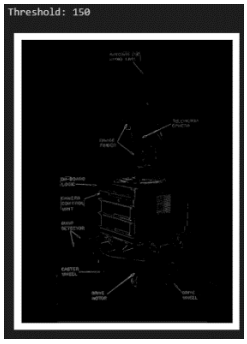
In such a small area, the pixel changes on the diagonals most effectively represent local features. Additionally, its high responsiveness to **high-frequency information** (such as sharp edges or fine details) enables the Roberts operator to effectively capture areas of rapid change, typically corresponding to edges in the image. Moreover, its diagonal difference method focuses directly on luminance changes in the diagonal direction, allowing the Roberts operator to better capture **actual edges** (high-frequency features) with **less noise** at the same threshold compared to the Task1 Sobel operator (Wide and smooth edges are detected).

Therefore, the Roberts operator, with its core design focusing on capturing **local intensity changes in this direction**

and its response to **high-frequency features**, proves to be more effective and suitable than the Sobel operator in applications requiring precise detection of small and sharp edges.







Task 3:

Magnitude:

```
def magnitude(x,y):
    return np.sqrt(x**2 + y**2)
```

Calculation Method: The magnitude is computed as the square root of the sum of the squares of the gradients in the x and y directions: $\sqrt{}$ **(Gx² + Gy²).**

Sensitivity: This method is sensitive to both the size and direction of the gradient, providing a more accurate representation of the gradient's true strength.

Edge Representation: Edges detected using magnitude offer a more accurate representation of the true edge strength. This method is better at distinguishing between strong and weak edges.

Computational Complexity: More complex computationally due to the square root operation.

Absolute:

```
def approximate_magnitude(Gx, Gy):
    return np.abs(Gx) + np.abs(Gy)
```

Calculation Method: When using absolute values, this method sums the absolute values of the gradients in the x and y directions: **|Gx| + |Gy|.**

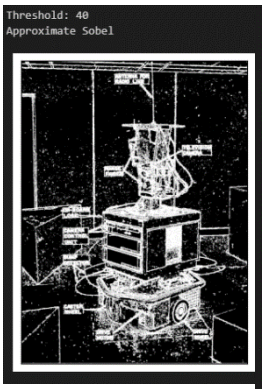
Sensitivity: Although this method can also identify high-gradient areas, it might not represent the true strength of the gradient as accurately as the magnitude method. It tends to uniformly emphasize edges.

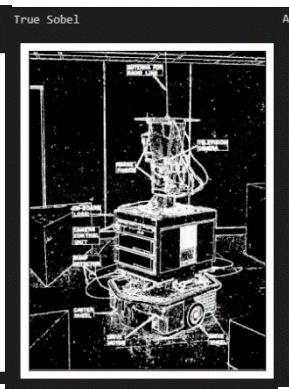
Edge Representation: Edges detected using the sum of absolute values might appear more pronounced but coarser in terms of gradient strength.

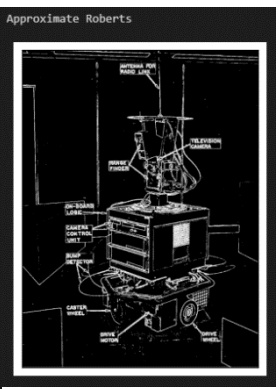
Computational Complexity: Simpler and faster to compute, involving only absolute values and addition. Suitable for quick processing but less accurate in representing edge strength.

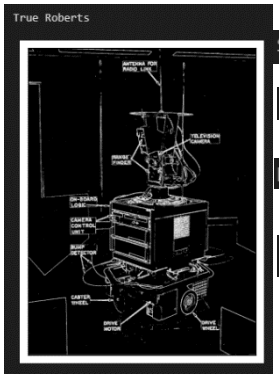
In practice, at the same (**Threshold >40**), both the Sobel and Roberts operators provide good edge strength representation. However, when using absolute values, some **noise** is introduced compared to using gradient calculations. The reason is that the absolute value method lacks the **detailed representation of edge strength and sensitivity to the gradient direction found in the magnitude method**. Consequently, this method might sometimes **overemphasize weaker edges**. In terms of computational complexity, the absolute value method is **faster** because it involves only addition, making it suitable for rapid processing but less accurate in edge strength representation. The gradient method, involving square root operations, is computationally more **complex** but more **accurate in terms of edge strength**.

Overall, although the absolute value method has advantages in computational speed and is suitable for scenarios requiring quick processing, the gradient magnitude method performs better in accurately representing edge strength and sensitivity to gradient direction. Therefore, the choice between these two methods should be based on **specific application requirements and priorities**, such as computational efficiency versus accuracy in edge representation.










```
Sobel Approx Time taken:  0.13650870323181152
Sobel Time taken:  0.141038179397583
Roberts Approx Time taken:  0.12699913978576
Roberts Time taken:  0.14599990844726562
```