A screenshot of a computer generated image

Description automatically generated**Task 1: Edge Detection**

Figure Original Images

**AIM:** To programmatically apply five edge detectors to the images of fluorescing cells. Discuss the use of noise removal, and evaluate the efficacy of the edge detectors, comparing the edge detections to the ground truth.

**METHOD:** Step 1: Load the three fluorescing cell images and the ground truths using `io.imread()`.

A screenshot of a computer screen

Description automatically generatedStep 2: Convert to greyscale by extracting the green channel of the cells (this ensures maximal pixel contrast given the cells are green) and using `rgb2grey` for the ground truths. Converting the images to greyscale enables the functions to work effectively. This process is required by some edge detection algorithms (which need the image in greyscale rather than RGB) as it focusses on changes in intensity of a single value, removes variations in chromatic components and emphasises structural features. It also reduces the computational load and complexity of the calculations, making the detection process faster.

Figure Noise Filtering

Step 3: Apply three different types of noise reduction (Gaussian, Mean, and Cone filtering). Having noise in an image can make it harder to distinguish edges so it is important to remove as much as possible without discarding vital information.

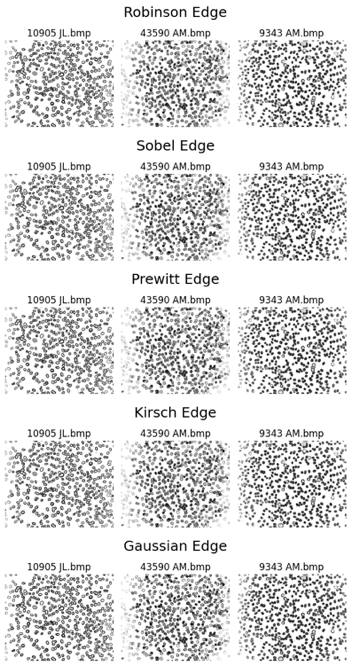
Visual inspection did not give any discernible difference, so the Peak Signal-To-Noise-Ratio was calculated (giving a quantitative measure - the higher the PSNR, the higher the quality of the extracted image, i.e. less noise). The “mean-filter” achieved the highest PSNR across all images, which implies that it retains more of the original images’ detail while still performing noise reduction. This was surprising as, typically, Gaussian is known to suppress high-frequency noise best; however, given the limited noise, the Gaussian filter seemed to over-smooth the images, leading to a loss of detail and a lower PSNR.

Figure Edge Detection

Step 4: Define the five edge detectors: code the kernels (as given), convolute the image using `scipy.signal.convolve2d`, and calculate the magnitude of the x and y components.

Step 5: Apply thresholding (mean and mean + standard thresholding).

Step 6: Invert the images so that they have the same colour scheme as the ground truths.

Given the limited differences between the three noise reduction algorithms, tests were run to compare the edge detection with: three types of Noise Reduction, no noise reduction, mean thresholding, and mean + standard thresholding

**RESULTS:** The Precision, Recall, F1-Score and Edge Detection Accuracy value for these combinations were then calculated, and the results and impact discussed.

A graph of different colored bars

Description automatically generatedThe F1-Score was calculated to quantify the edge detectors’ performance in relation to each noise reduction algorithm. F1-Score is the best measurement as it gives a balanced measure of the overall edge detection performance. No noise reduction performed considerably better than the three filters. This is likely due to the filters “over smoothing” and therefore removing detail from the intricate images required for the edge detectors to perform well. Given the obvious benefit of not filtering the images, noise reduction is removed from further discussion.

A graph of different colored lines

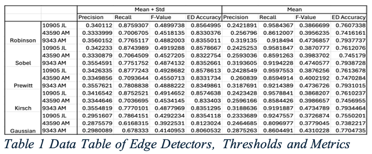
Description automatically generatedComparing the five edge detectors with the four metrics and two thresholds, Gaussian edge detection consistently performed worst.

In this application, misdetection of the cell boundaries could lead to misdiagnosis; therefore, the comprehensive evaluation of F1-Score and ED Accuracy is too vague. Instead, Precision and Recall are considered:

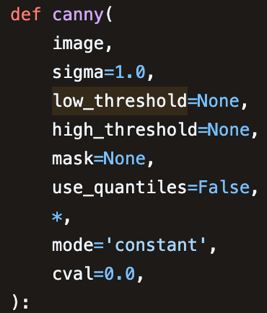
* Precision prioritises limited false positives (high cost of false positives) – i.e. few incorrect edges detected
* Recall prioritises limited false negatives (high cost of false negatives) – i.e. few missing edges

A graph of a bar chart

Description automatically generated with medium confidenceThe relative importance of these depends on the application of the edge detectors - missing true edges could lead to misdiagnosis. Therefore, Recall is prioritised over Precision (extra edges are acceptable if critical edges aren’t missed).

Next, the discussion moves to thresholding. The Recall metric is evaluated with respect to both thresholding techniques and all edge detection methods. Mean thresholding consistently produced better recall values than mean + standard.

**CONCLUSION:** The edge detection methods can be discussed given the streamlined pre-processing method (no noise reduction, mean thresholding, evaluating with respect to Recall value). Prewitt, Robinson, Kirsch, and Sobel all perform very similarly; however, for the purpose of medical imaging, Prewitt is the best edge detector, as there was a very slight improvement in Prewitt’s Recall value. It is interesting how the application/use case impacts which metrics, thresholding, and edge detection method works best.

**Task 2: Advanced Edge Detection**

**AIM:** Implement the Canny Edge Detection Algorithm on the images of fluorescing cells. Evaluate and discuss the efficacy of Canny in comparison to the five above.

**METHOD:** Implement Canny with `skimage.feature.canny`, an imported function. The application is eased by the image loading completed in task 1. No additional preprocessing is required as this is part of the Canny algorithm.

The Canny Edge Detection Algorithm works as follows:

* The image is smoothed with a Gaussian Filter, the filters kernel size is impacted by the σ parameter.
* The Sobel operators are applied and the gradient magnitude and direction calculated.
* A close-up of a logo

  Description automatically generatedNon-maximum suppression (NMS) is used to calculate a single response for each edge; this thins potential edges to 1-pixel wide.
* Hysteresis thresholding is performed with respect to a low and high threshold.

The impact of modifying the Gaussian filter within the built-in function will be discussed and evaluated.

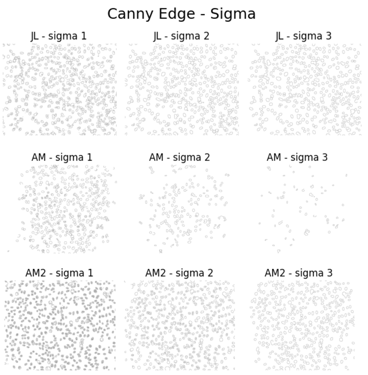
Figure Canny Edge Detection

A graph with different colored lines

Description automatically generated**RESULTS:** Comparing the metrics of the output of the standard Canny algorithm versus the previous “ideals” from task 1 leads to the following conclusion: Canny performs much better at Precision and considerably worse at Recall than the other detectors.

This conclusion can also be drawn when using Canny’s proposed criteria to measure the efficacy of an edge detector (detection, localisation and single response) visually.

A screenshot of a graph

Description automatically generatedChanging the σ parameter impacts the size of the Gaussian kernel, a larger value of σ results in a wider kernel and a smoother output, while a smaller value of σ results in a narrower kernel and a more localised estimate. Three tests were completed on each of the three images (σ = 1, σ = 2 and σ = 3). These tests revealed several things:

* In 43590 AM, where there is little gradient differentiation on the image, smoothing the image leads to very low edge detection rates and poor performance. This is particularly noticeable as σ is increased.

Figure 5 Changing Sigma within Canny Edge Detection

* 9343 AM precision rating increased with an increase in σ, this is due to the few incorrect edges missed.

This test also gave valuable insight on the accuracy of the metrics, specifically, how poor Edge Detection Accuracy is. EDA consistently returns very high scores even when the image is of a low quality.

**CONCLUSIONS:** Here, Canny is the best detector when optimising Precision; however, given the application defined in task 1, the focus is on Recall therefore Canny is the worst overall edge detector. When comparing edge detectors, the application, and therefore the optimised metric, plays a massive role.