NPL
National Physical Laboratory

NPL REPORT MS 49

THE UNCERTAINTY-AWARE CANNY OPERATOR EDGE DETECTION METHOD

CLARKE, J.
WRIGHT, L. R.

DATA SCIENCE

MARCH 2023

Understanding Complex Systems: Uncertainty-Aware Canny Operator Edge Detection Algorithm

Clarke, J., Wright, L. R.
Data Science

ISSN 1754-2960

https://doi.org/10.47120/npl.MS49

National Physical Laboratory
Hampton Road, Teddington, Middlesex, TW11 0LW

This work was funded by the UK Government's Department for Science, Innovation & Technology through the UK's National Measurement System programmes.

Approved on behalf of NPLML by
Louise Wright – Head of Science for Data Science

Glossary

Keyword	Definition
Canny operator	Edge detection algorithm used to find the edges in input images
Edge Image	Image consisting of only the edges of the shapes featured in the original image
Layer	A bounded region found in the edge image
Max Pixel	A pixel with maximum intensity value (255)
Min / Max image	An image with pixel intensities precisely 0 or 255.
Neighbouring Matrix	A 3x3 square of pixels taken around any edge pixel P, which can then be analysed to determine which layers pixel P belongs to (Section 2.2.4)
Zero background	An image where the background has intensity value of zero
Zero Pixel	A pixel with an intensity value of zero

Abbreviations

The following abbreviations are used in this document.

CDF – Cumulative Distribution Function

LHS – Latin Hypercube Sampling (Type of MCM)

MCM – Monte Carlo Method

N (μ, σ) – Normal distribution, with mean μ and standard deviation σ

U(x,y) – Uniform distribution between x and y

ROI – Region Of Interest

RS – Random Sampling (Type of MCM)

CONTENTS

1 INTRODUCTION

Uncertainty quantification in measurements is incredibly important in any scientific field and allows for reliable comparisons and conclusions to be made about the acquired data. In many areas of modern scientific research, images are typically the primary method of scientific measurement; from nanoscale electron microscopy to astronomical images captured by the James Webb Space Telescope. Uncertainties associated with images, however, are commonly ignored or incorrectly calculated due to their highly complex nature.

Measurements in images are often used to make informed decisions about the experiment but without associated uncertainties there can be no reliable comparison between measurements, or comparisons made to a standard. This work was aimed at quantifying some uncertainties associated with image processing and considered what effect this will have on the conclusions of the image analysis. This report focuses on one aspect of image processing commonly used in analysis: the Canny operator edge detection method [11].

An example which highlights the importance of robust Uncertainty Quantification (UQ) is in medical imaging. When imaging tumours, the measured area of the tumour can be crucial to determine treatment and patient lifespan. Quantifying the uncertainty of the area of this tumour would improve the likelihood of using the correct treatment (e.g., use of lumpectomy versus mastectomy [1]) and save money and lives in doing so [1].

The chosen method of UQ in this work is the propagation of uncertainties through a Monte Carlo approach. Typically, image analysis can be computationally expensive due to high resolution (large number of pixels) of modern images. Additional computational cost comes from the Monte Carlo (MC) approach adopted here, which requires repeated calculations for a given measurement model (in this work, the measurement model is the Canny operator [11]). A solution to this is using a pseudo-random sampling method (rather than a rudimentary random sampling method) such as Latin hypercube which allows for much fewer calculations without the loss of output accuracy [3]. Fewer iterations means that computation time is reduced, and hence larger images can be processed more easily.

This report is structured as follows: the methodology for quantifying uncertainties of areas of features within images is given in Section 2 – describing the edge detection method and uncertainty propagation process. Section 3 describes the uncertainty propagation process and makes a comparison between the two sampling methods used here – Random Sampling (RS) and Latin Hypercube Sampling (LHS). Section 4 provides some examples of the LHS sampling method, which is the preferred sampling method due to its speed over larger samples [3]. Section 5 outlines the limitations of the algorithm and methods of improvements and Section 6 provides concluding remarks.

.

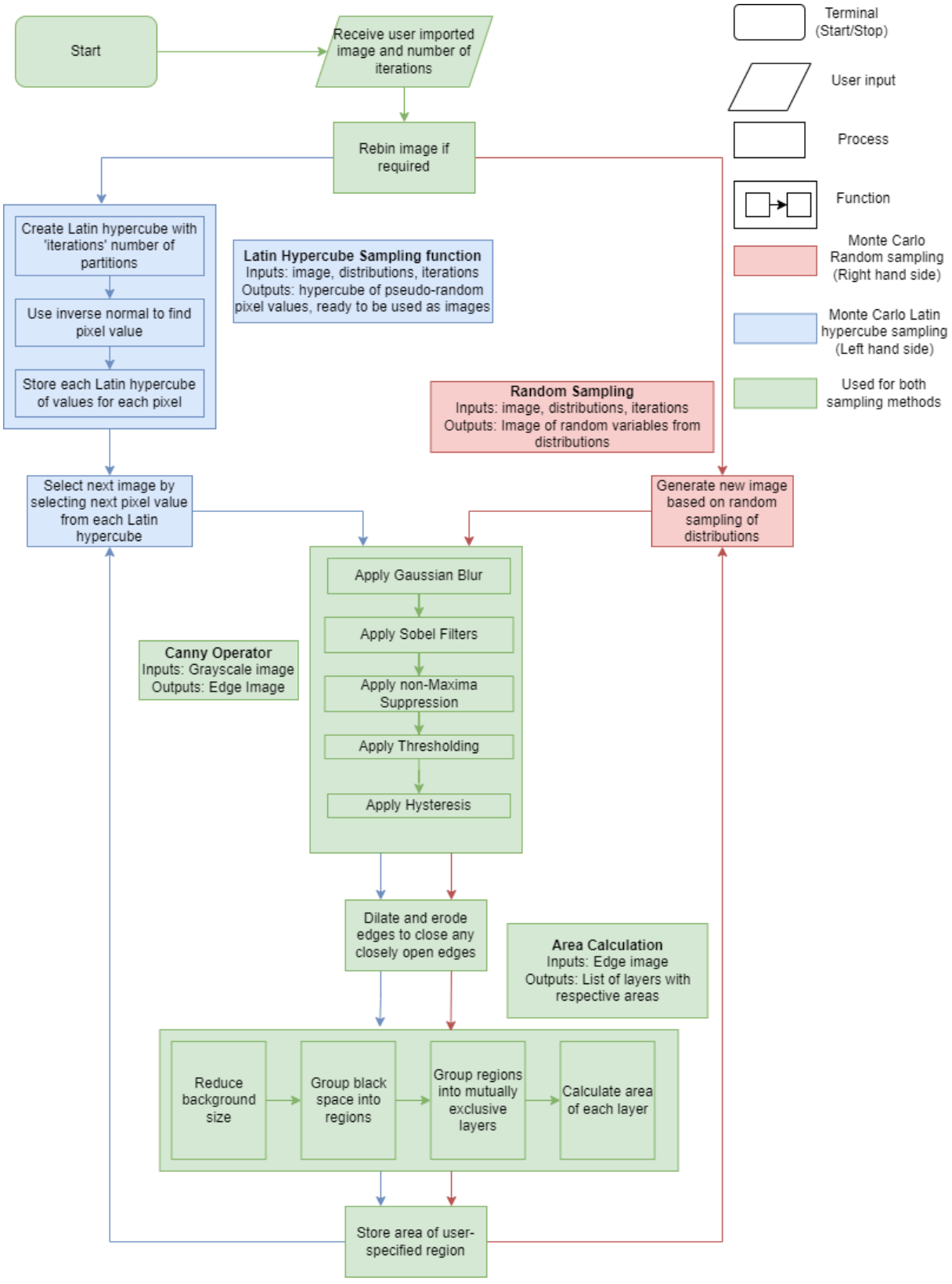

Figure 1: Flow chart demonstrating the method discussed in the report.

2 METHODOLOGY

This work is centred on creating and testing an uncertainty-aware Canny operator edge detection method. The proposed method is summarised in Figure 1, which shows the workflow for the tool, which outlining the edge-detection method, metric calculation, and uncertainty propagation approach employed here. In this section, a summary of each step in the uncertainty aware Canny operator algorithm is given.

2.1 CANNY OPERATOR EDGE DETECTION

This work uses a Canny operator to detect edges in grayscale images due to its speed, simplicity to implement, and commonality for general use. There are five steps in the Canny operator edge detection algorithm: Gaussian blur, Sobel filters, non-maxima suppression, thresholding, and hysteresis. The following sections describe each step in the Canny operator.

2.1.1 Gaussian blur

This first step in the operator transforms the original image by introducing a Gaussian blur, making the edges in the image less sharp, which is necessary to control detail and suppress noise. This is implemented by first creating a transformation matrix $\mathrm{G}(\mathrm{x},\mathrm{y})$ calculated by

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

where σ_{G} is the standard deviation of the Gaussian distribution, and x and y are matrices of distances from the origin of the kernel. The transformation matrix is applied to the original image, I, using a simple convolution

$$\mathrm{I}_{\mathrm{g}}(\mathrm{x},\mathrm{y}) = \mathrm{I}(\mathrm{x},\mathrm{y}) * \mathrm{G}(\mathrm{x},\mathrm{y}), \tag{2}$$

where $\mathrm{I}_{\mathrm{g}}(\mathrm{x},\mathrm{y})$ is the new Gaussian-blurred image. An example of this function can be seen in Figure 2.

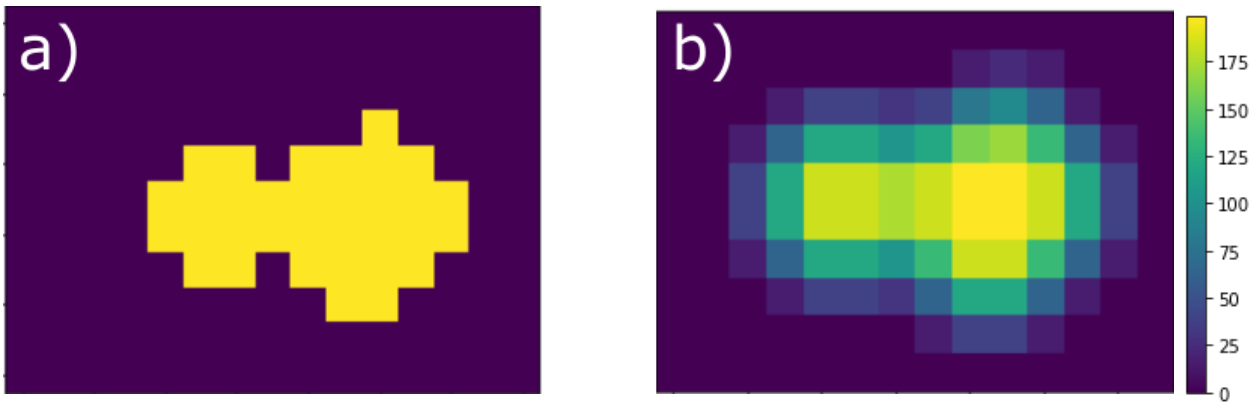

Figure 2. Example of Gaussian Blur on an image (a) original image, (b) Gaussian blurred image with kernel size 11 and standard deviation, $\sigma_G = 3$.

2.1.2 Sobel filters

After applying the G matrix transformation to the image, the tool then uses Sobel filters [9] to track the changes in pixel intensity across the image. Sobel filters are commonly used in edge detection, and they calculate the image gradient in the horizontal and vertical axes through an image convolution. The approximate derivatives used in the Sobel filter are:

$$\mathrm{H_x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ and } \mathrm{H_y} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \quad (3)$$

In the uncertainty aware Canny operator algorithm proposed here, the matrices in equation (3) are applied to the blurred imaged $\mathrm{I_g(x,y)}$ by

$$\mathrm{I_x(x,y)} = \mathrm{I_g(x,y)} * \mathrm{H_x}, \quad (4)$$

$$\mathrm{I_y(x,y)} = \mathrm{I_g(x,y)} * \mathrm{H_y}. \quad (5)$$

Equations 4 and 5 produce two gradient images and applying Sobel filters to Figure 2 results in Figure 3.

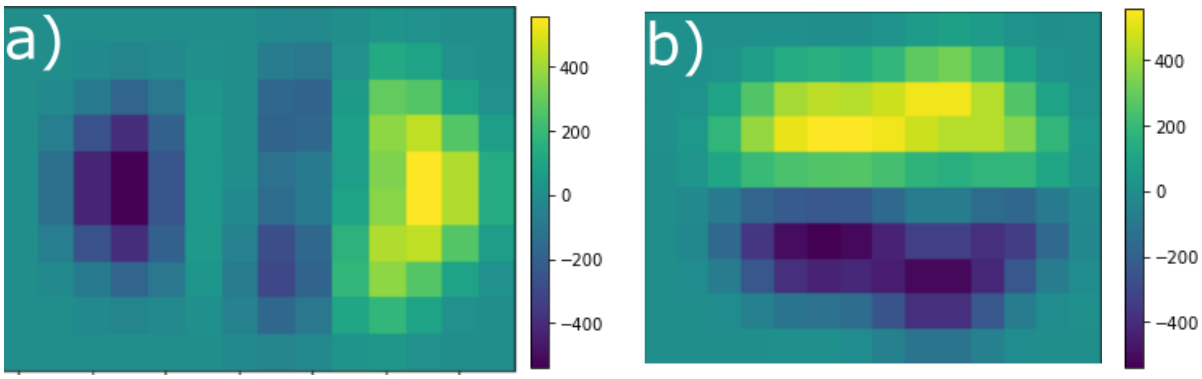

Figure 3: Sobel Filters applied to Figure 2 (a) $\mathrm{I_x}$ – changes in intensity horizontally, (b) $\mathrm{I_y}$ – changes in intensity vertically.

The next step in the algorithm is to calculate the magnitude of the intensity change with a hypotenuse calculation and edge direction calculation as follows

$$|I_G|(\mathrm{x,y}) = \sqrt{\left(\mathrm{I_x^2} + \mathrm{I_y}^2\right)} \quad (6)$$

$$\theta(\mathrm{x,y}) = \arctan\left(\frac{\mathrm{I_x}}{\mathrm{I_y}}\right). \quad (7)$$

Equations 6 and 7 return two output matrices: one of magnitudes ($|I_G|(\mathrm{x,y})$) and the other being the matrix of direction to edges ($\theta(\mathrm{x,y})$) at each given point. $\theta(\mathrm{x,y})$ is a gradient map, highlighting the directions of greatest pixel intensity change. It calculates the elementwise inverse tangent of $\mathrm{I_x}$ and $\mathrm{I_y}$ for each pixel, thereby finding the direction of intensity change in two separate directions. $\theta \in [-\pi, \pi)$. These are shown in Figure 4.

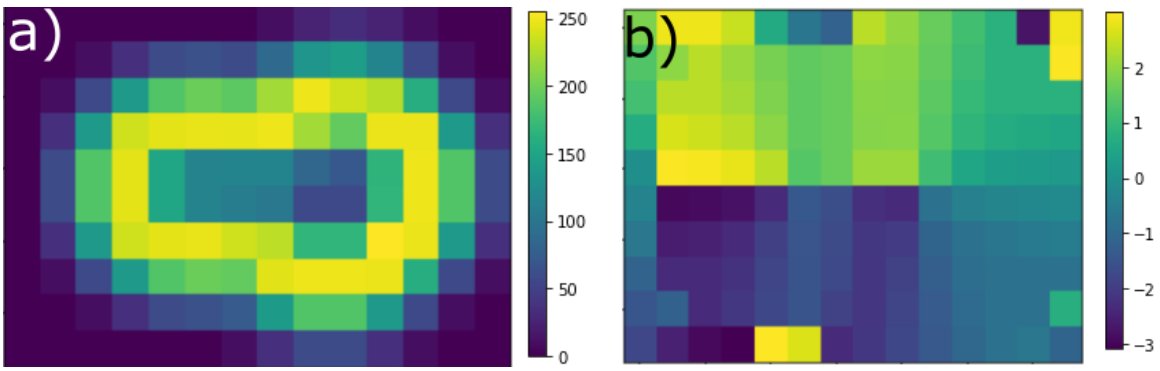

Figure 4: Magnitude and edge direction matrices of Figure 2, after applying Sobel filters. (a) is the magnitude image ($|I_G|(\mathrm{x},\mathrm{y})$) and (b) is the convolution of Figure 3 for overall edge direction matrix ($\theta(\mathrm{x},\mathrm{y})$).

2.1.3 Non-maxima suppression

The next step in the algorithm is to carry out a non-maximum suppression. Examining the results in Figure 4 (a), the edges of the shape can be seen in yellow, with faded colour to blue towards the edges. Non-maxima suppression is a method to suppress this faded region and highlight only the most intense pixels. The algorithm assesses each pixel in $|I_G|(\mathrm{x},\mathrm{y})$, detecting whether there is a more intense pixel in the direction of the edge given by $\theta(\mathrm{x},\mathrm{y})$. If there is a more intense pixel in the direction of the edge, only the most intense pixel is kept and the other pixels in this row are set to zero as shown in Figure 5.

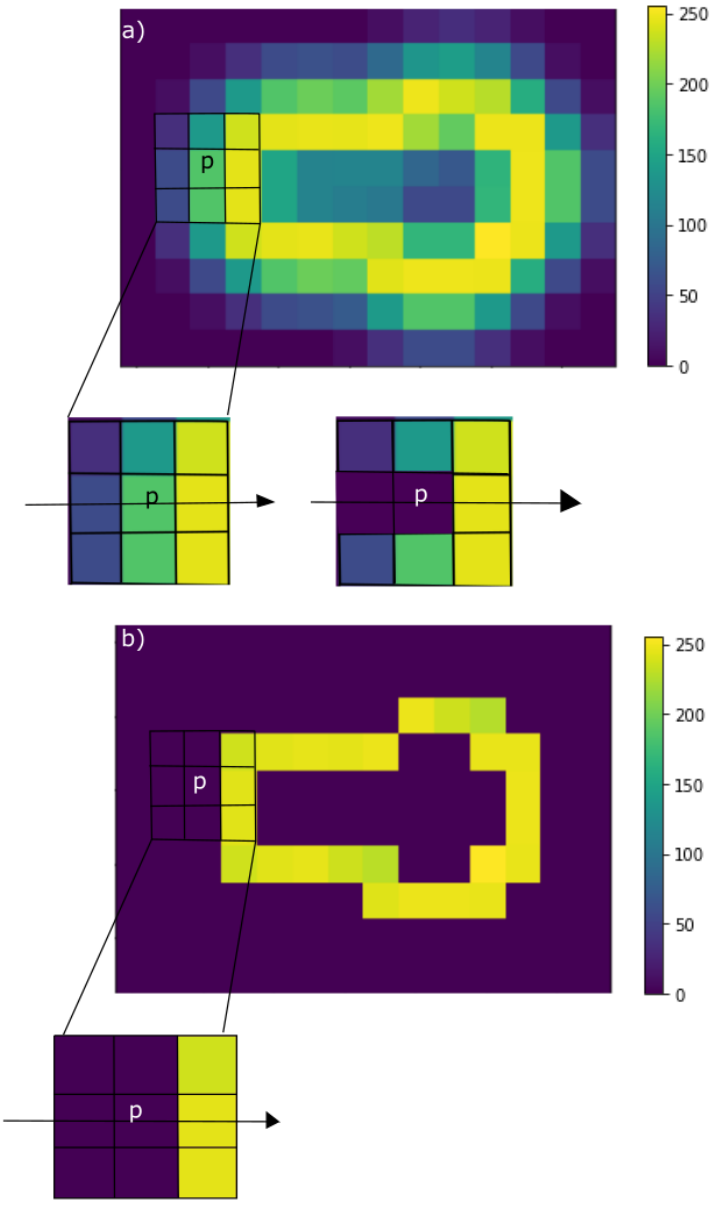

Figure 5: (a) A demonstration of how non-maxima suppression works for a give pixel. A 3×3 region is selected around a pixel of interest (pixel p in this case). The pixel to the right of p is the maximum in the horizontal axis (indicated by the arrow) and therefore the other pixels on

the axis are set to zero. This step is repeated for each pixel in the image, resulting in (b), where all other non-maximum pixels have been suppressed to zero.

Figure 5 demonstrates the non-maxima suppression process using the images shown in Figures 2-4. Using pixel p in Figure 5 (a) as an example, the pixels surrounding p are selected and the direction of the edge is shown by the arrow inset. The pixel to the right side of pixel p on the arrow has maximal intensity in the arrow line, meaning that the left-hand pixel and pixel p are both suppressed to zero. Upon using this process for the entire image, the method returns Figure 5 (b), and as shown in the highlight of pixel p, the non-edge values are given a value of zero.

2.1.4 Thresholding

Following non-maxima suppression, the algorithm then takes user inputs for thresholds to define the intervals for ‘strong’ and ‘weak’ pixels. These thresholds are proportions of the maximum intensity in the image. Maximum pixel intensity for greyscale images is set at 255 (standard maximum intensity for a computer-read image), hence a lower threshold of 0.1 would be the equivalent of a pixel of 25.5 intensity. For thresholding in the Canny operator, two values are required: an upper, t_u, and a lower, t_l, value. There are three outcomes to the thresholding of each pixel with respect to the upper and lower threshold values: pixel intensities above the upper threshold are strong, i.e. $I(x,y) > t_u$, pixel intensities between thresholds are weak, $t_l < I(x,y) < t_u$, and pixels below the lower threshold are determined to be non-edge pixels and are set to zero, $I(x,y) < t_l \rightarrow I(x,y) = 0$.
.

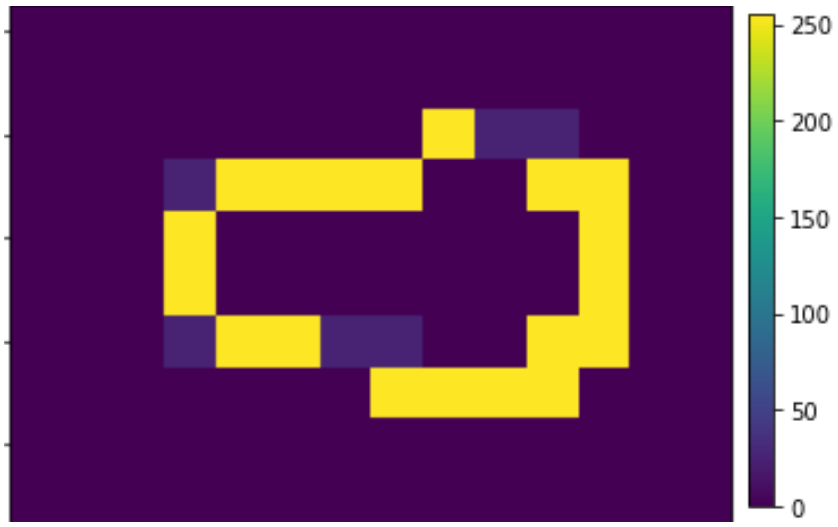

Figure 6: Thresholding applied to Figure 5.

Figure 6 shows thresholding applied to Figure 5, with a lower threshold of 0.4 and upper threshold of 0.95. The thresholds were chosen such that some pixels would be weak for demonstration purposes (the darker pixels in the image). These pixels would be turned to strong pixels in the hysteresis process.

2.1.5 Hysteresis

After thresholding has been applied to the image, the resulting image is then passed into a hysteresis step. This is necessary to examine weak pixels and assess whether they should be turned to strong pixels or whether they should be removed entirely. This is carried out by examining neighbouring pixels to find strong pixels. If the weak pixel directly adjoins one strong pixel, then it will be turned into a strong pixel, otherwise it will be removed. From Figure 6, all weak pixels have a neighbour of a strong pixel, and hence all pixels will be converted to strong pixels – resulting in the final edge image shown in Figure 7.

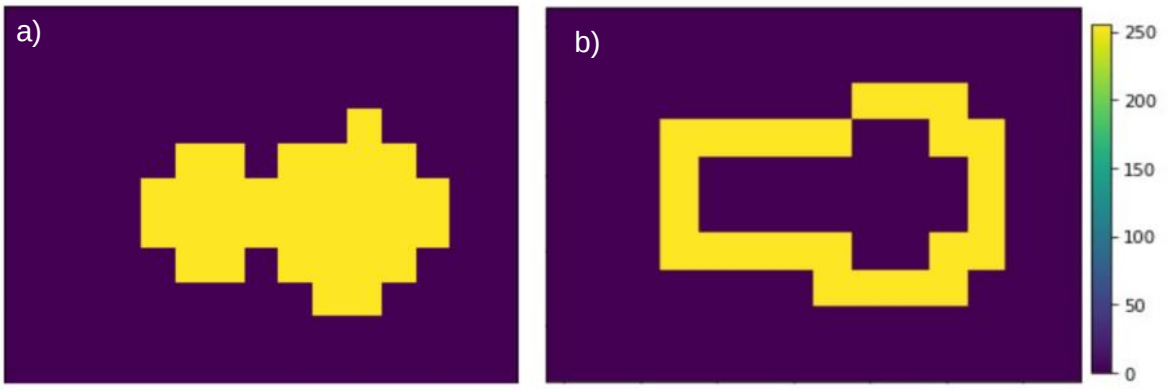

Figure 7: (a) initial image (Figure 2) (b) edge image returned after applying the canny operator algorithm to (a)

Note the loss of detail in Figure 7 in comparison to Figure 2. This is accentuated due to the small size of the image and is extreme for demonstration purposes. The loss of detail in Figure 7 when compared to Figure 2 is predominantly due to the Gaussian blur, however this loss of accuracy is also a disadvantage of the Canny operator [12]. This is discussed further in Section 5.

2.1.6 Erosion and dilation

It is possible for hysteresis outputs to be mistakenly unbounded, which can occur if there are spaces with two or more neighbouring weak edge pixels. To counteract this, the hysteresis output is dilated and then eroded. Dilation and erosion are simple mathematical morphology processes typically used in binary images, although they have successfully been applied to greyscale images [10] (as is the case in this report). Dilation is the gradual enlargement of the boundary image, offsetting the mistaken unbounded regions. Dilation is implemented by iterating through each pixel, selecting a 3 x 3 grid around that pixel, and changing the central pixel's value to the maximum pixel intensity found in the 3 x 3 grid. Repeating this process for N iterations, any previously unbounded edges with gaps at most 2N weak pixels across will be bounded.

Following this dilation process, erosion is used to return the image to its original size. Erosion is the gradual process of removing the edges of the boundary image – opposing the dilation process without unbounding the features. Similar to the dilation process, erosion considers each pixel, creating a 3 x 3 surrounding grid, and sets the pixel value to the minimum pixel intensity in the grid. This is the inverse action of the dilation, but due to the wider lines from the dilation process, any small gaps in the original edges are filled, provided the number of iterations of erosion are less than or equal to the number of iterations of dilation. Further details on these steps are beyond the scope of this report but can be found in reference [10].

2.2 AREA CALCULATION

After the edges have been found within an image, the tool then carries out a series of processing steps to reduce the background, group non-edge pixels, unite regions, and calculate the area for any found region. The area of a given shape was chosen as the metric to examine the proposed uncertainty-aware Canny operator since the uncertainties in pixel intensities will change the size of detected regions, allowing for easy quantification. The area was chosen as it allows a Monte Carlo Method (MCM) algorithm to produce a single numeric

value (the “measurand”) from the image processing, and the result of the MCM is a distribution of that measurand [2].

2.2.1 Background reduction

After the edges in the image are found using the process presented in Section 2.1, the algorithm then calculates the area contained in each region bounded by edges. The first step in the area calculation algorithm is to reduce the image such that there is only one layer of background pixels around the initial shape. This reduces unnecessary processing and thus speeds up computation. Furthermore, an additional edge is artificially added to the right and bottom of the image, ensuring the method in Section 2.2.2 selects all points.

2.2.2 Grouping non-edge pixels

The next step for the algorithm is identifying each bounded region. The algorithm iterates through each zero pixel and takes 2 x 2 squares around these, placing the selected zero pixel in the top left corner. It determines the intensity of pixels within the squares and tracks to see whether the pixel in question is in a previous region. If so, it appends all new pixels to the region and, if not, creates a new region for this pixel and its surrounding selected pixels. This method is demonstrated with a simple example shown in Figure 8.

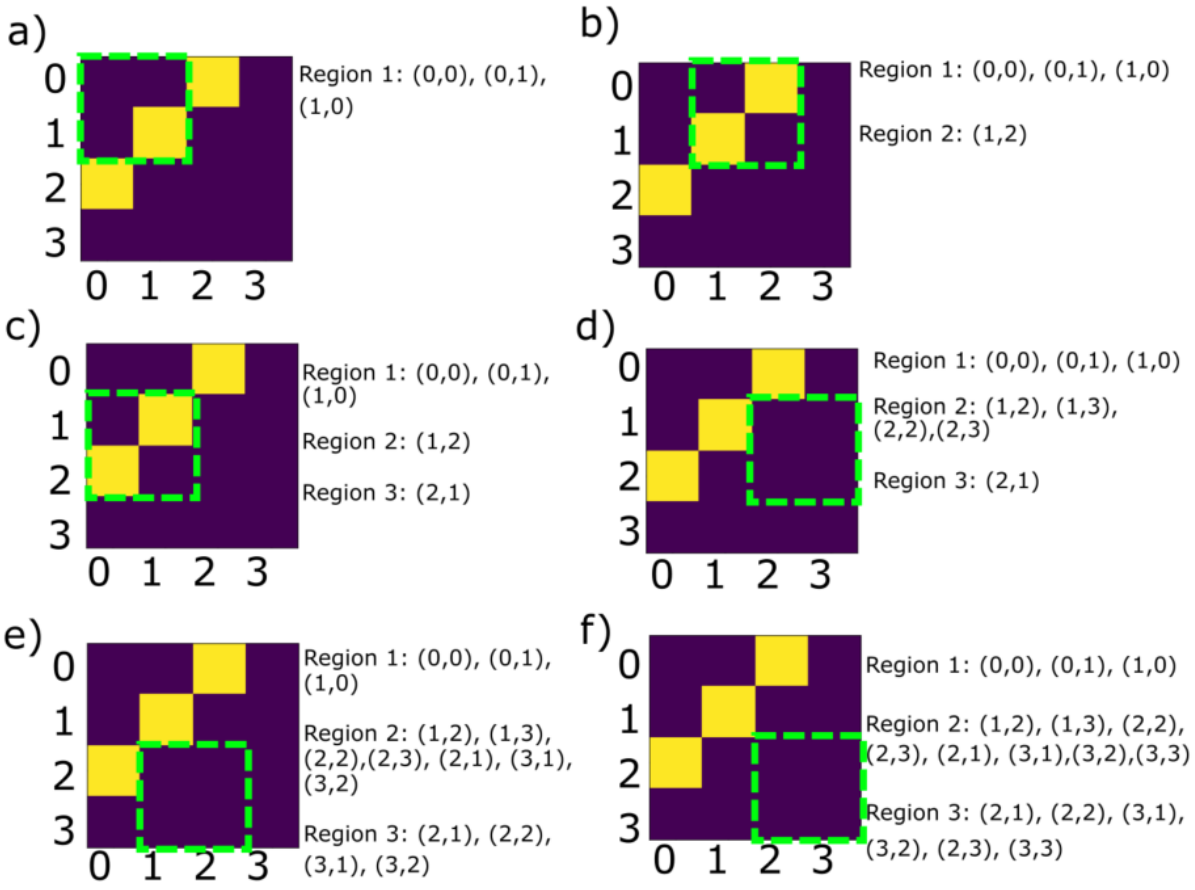

Figure 8: Demonstration of grouping edges method describer in Section 2.2.2. Pixels are referenced in (row, column) format.

Below is a step-by-step breakdown of method 2.2.2 using Figure 8:

- (a) demonstrates the first 2x2 square of pixels highlighted by the algorithm in 2.2.2. (0,0) is the first zero pixel identified and examined. The pixels in the 2 x 2 square are not separated by any strong pixel lines, and hence are in the same region.

- (b) (0,1) is the next zero pixel horizontally. The 2 x 2 square drawn around that finds a line separating (0,1) and (1,2), meaning these pixels are in separate regions. (0,1) is already in region 1, meaning (1,2) must be in region 2.
    - The next zero pixel is (0,3). A 2 x 2 square cannot be drawn around this pixel because it is the right-most pixel on row 0. Hence, the next zero pixel is (1,0).
- (c) (1,0) is the next zero pixel. The 2 x 2 square contains a line. (1,0) is in region 1, and (2,1) is not in region 2, hence it is in region 3.
- (d) (1,2) is the next zero pixel. There are no lines in this square, hence the pixels all belong to the same region. (1,2) is in region 2, meaning all pixels are added to region 2.
    - The next zero pixel is (1,3), which cannot be used since it is the right-most pixel in row 1. The next zero pixel to examine is thus (2,1).
- (e) (2,1) is the next zero pixel. The 2 x 2 square contains no lines, meaning all pixels belong to the same region. (2,1) is in region 3, meaning all pixels are added to region 3. However, (2,2), in the square, is already in region 2, meaning all pixels are also in region 2, and hence added to region 2.
- (f) (2,2) is the next zero pixel. The 2 x 2 square contains no lines. (2,2) belongs to both regions 2 and 3, and hence the pixels are added to both regions.
    - The next zero pixel is (2,3), which cannot be used since it is the right-most pixel of row 2.
    - The next zero pixel is (3,0), which cannot be used since it is the bottom-most pixel of column 0, meaning a 2 x 2 square cannot be drawn around it. This is the same for all pixels in row 3, meaning all suitable pixels have been examined.

Examining the output of Figure 8, the grouping method ignores some pixels (pixels (0,3) and (3,0)) since 2 x 2 squares cannot be created on the outermost points. The algorithm accounts for this issue by adding a border, with a pixel value of zero, to the right-hand side and bottom of the image. These added pixels will not be assessed themselves but act as placeholders so that all outermost pixels can be examined. In the case of the image in Figure 8, by applying this process, Figure 9 is returned.

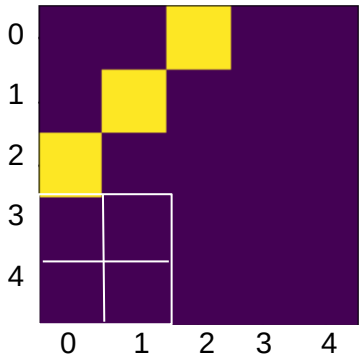

Figure 9: Example shape from Figure 8 with additional border width around the outside. The white square represents the 2x2 matrix which can now be made so that (3,0) is assessed.

This addition means that all pixels in the original image can be assigned regions. Applying the method to Figure 9 returns the following regions:

- Region 1: (0,0), (0,1), (1,0)
- Region 2: (0,3), (1,2), (1,3), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)
- Region 3: (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3)

Other methods for feature extraction exist, such as the ‘bwboundaries’ function in MATLAB for example, but a custom python area calculator was the preferred option in this work as it allowed for the control over all parameters.

2.2.3 Uniting regions

Just as in the example shown in Section 2.2.2, there are cases where regions are not mutually exclusive. In some cases, it is only clear to see that specific regions should be combined once all pixels have been analysed. To resolve this issue, the algorithm loops through regions and unites them if they have shared pixels. It will continue to iterate through all regions until no new changes are made. The output from this is a refined list of ‘layers’, which are defined as the closed bounded regions found in an edge image. For the example in Section 2.2.2, this would result in regions 2 and 3 combining. Figure 10 shows the inputs and outputs of the uniting process for the simple image example used in Section 2.1.

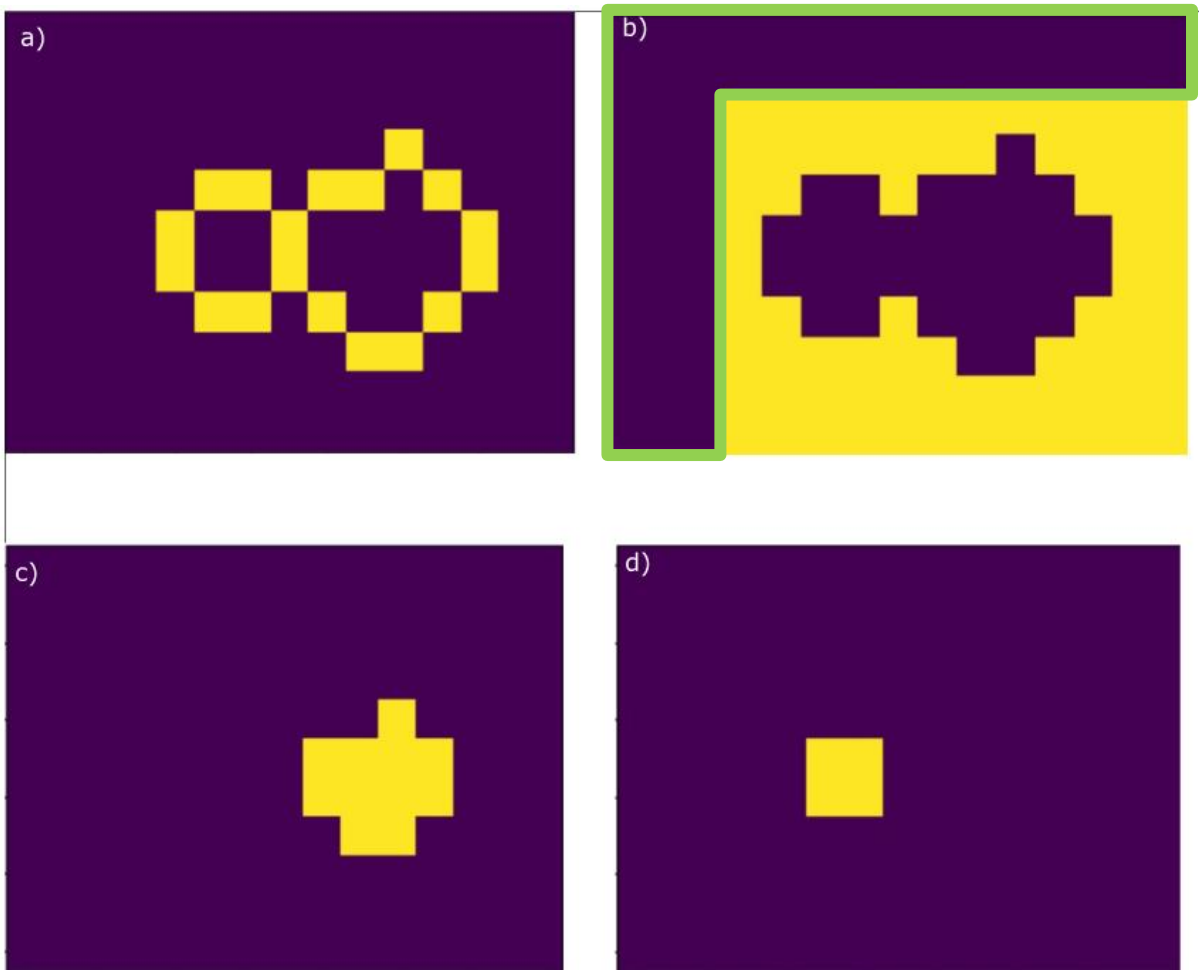

Figure 10: (a) shows an example edge image in the format obtained by the Canny operator. (b), (c), and (d) show the separate “layers”, which are the separate closed, bounded regions detected within this image by the algorithm.

Background cropping is a feature used within the tool to aid the processing speed of images. In the background layer, there are many max pixels processed unnecessarily since the background layer is not useful for further calculations. The only use for the background image is to show where the other layers are within the edge-image, which means that the a large portion of the background layer can be set to a pixel value of zero without affecting the layer at all. Setting the unused background pixels to zero decreases the computational cost

of the uniting regions step which is important in the usability of the uncertainty-aware Canny operator tool.

The tool could crop the background from all directions, leaving only the outline of the shape. However, due to the direction of pixel searching employed by Python (sweeping from left to right along the horizontal direction), accessing the bottom and right-hand sides of the background image is more difficult and requires additional processing time. Because of this, the software only crops the top and left-hand sides of the background image. An example of this is shown in Figure 10 (b), where the highlighted region is cropped out of the background image, without affecting the outline of the shape.

2.2.4 Assigning edges to regions

Examining Figures 10 (c) and (d), the inside regions are mapped correctly but the edges of the regions are not included in these layers. The next step is to add the edges to their respective layers. In this stage of the tool, three images are considered: the edge-image, the layer, and the combined image of the layer and edge-image. Every edge-pixel in the edge image is examined, and for each pixel a 3 x 3 region (known as the ‘neighbouring matrix’) is selected. The same region, centred on the same pixel, is selected in the combined image. For demonstrative purposes, an example of this step is shown in Figure 11 where pixel P is the pixel of interest – neighbouring matrices are given inset (the layer shown in Figure 10(d) was used). If the neighbouring matrices differ between the images, it indicates that the process of combining the edge image and the layer has caused the edge pixel to gain some neighbouring pixels. This means that the edge pixel must be neighbouring the layer, hence the pixel must be in the layer’s edge. In Figure 11, the neighbouring matrices surrounding pixel P are different for the edge where the edge image (Figure 11(a)) and combined image (Figure 11(b)) are given.

The result of finding this difference in the neighbouring matrices is that pixel P from the edge image is added to the original layer image. If there are no differences in the neighbouring matrices of the respective images, however, the centred pixel is not an edge pixel of this layer and is ignored. This process is repeated for every non-zero value on the edge image and is repeated for every layer extracted from the grouping process. Issues arise, however, when the edge line is more than one pixel thick, then the algorithm will only take the first pixel of that edge. In this iteration of the tool, this has not been addressed and will form part of future development of the method.

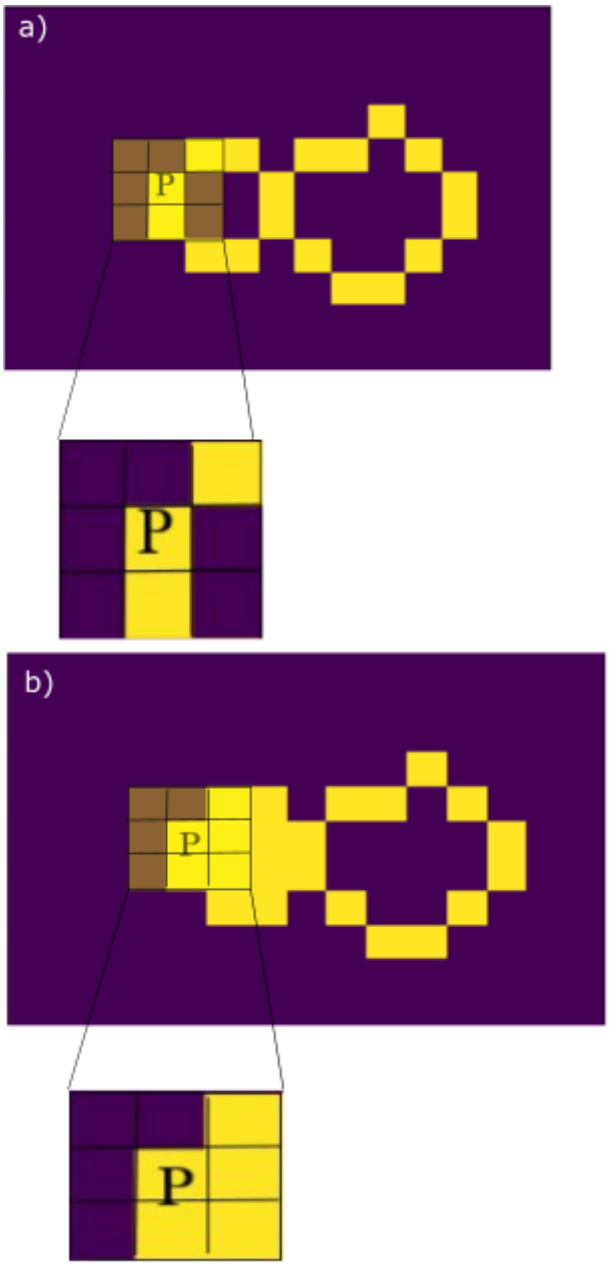

Figure 11: (a) The edge image initially used. (b) The edge image combined with layer (Figure 10 (d)).

Figure 12: Layer from Figure 10(d) with edges applied through process described in Section 2.2.4.

2.2.5 Calculating area

Finally, the algorithm counts the number of max pixels in each layer and stores the area of the selected region. In Figure 12, the area for the selected layer is 12 pixels. It is important to note that edge lines are considered in both the area of the inside shapes and the area of the background. This, combined with the background cropping for efficiency, means that the background area is not likely to be accurate. The results of the area calculation are determined in pixels; however this can easily be extended to a measurement of area in SI units (typically m^2) if the pixel size is known.

2.3 UNCERTAINTY PROPAGATION IN IMAGE PROCESSING

Propagating uncertainties through a measurement model can be implemented through several different methods [2] depending on the system being quantified. One such numerical method is a Monte Carlo method (MCM), which involves calculating estimates of the model outputs using an iterative process. MCM uses random sampling of input parameters from known distributions, which are then passed through a measurement model, and results in an output distribution for the measurand. The measurement model in this work is the image processing detailed in Sections 2.1 and 2.2, and the output measurand is the area of a layer. In this section, the sampling method was examined, where rudimentary random sampling is compared to a more sophisticated method: Latin hypercube sampling.

2.3.1 Monte Carlo method with random sampling

Simple random sampling (RS) selects data from a defined distribution function of each input parameter on each iteration. For RS considered here, a number of input parameters could be sampled. Using an image as a base, a new image can be generated pixel-wise, based on the distributions of each pixel, which can be specified by the user. The Gaussian blur σ_G, and Canny thresholds, t_l and t_u, values can also be sampled – in this work, both were sampled from a uniform distribution which is explored in section 4.1 After the relevant input parameters have been sampled, methods described in Section 2.1 and 2.2 are applied to the new image to return an area value for the iteration, as shown in the workflow shown in Figure 1. The MCM using RS process iterates 10^6 times as this gives a reliable estimate for 95 % coverage intervals (to two significant figures) [2] and returns the calculated areas and the calculated area distribution can be used to estimate an uncertainty on the measurand.

2.3.2 Monte Carlo method with Latin hypercube sampling

Latin hypercube sampling (LHS) uses pseudo-random sampling on the joint distribution of the input parameters to yield a more reliable estimate of the cumulative distribution function (CDF) of an output quantity for a smaller number of iterations than random sampling. The approach partitions the CDF of each input into M intervals of equal probability, where M is the number of iterations set by the user. The method samples a value of an input once within each subdivision. For a model with n inputs, M groups of input values are created by randomly assigning each sampled value of each input to exactly one of the groups. The hypercube is the full set of M groups of points in n-dimensional space; the hypercube is Latin because for each input there is exactly one value within the groups in each subdivision. LHS results in a more evenly distributed sampling of the input space than random sampling.

To demonstrate the LHS method, a uniform (0,1) distribution for two inputs and five iterations is considered. The range of each input is split into five even intervals ([0-0.2), [0.2-0.4), [0.4-0.6), [0.6-0.8) and [0.8-1.0]). The LHS method then selects a random value within each interval and carries out this sampling for both inputs. The hypercube points are then created

by pairing up the sampled values randomly. For instance, if the sampled values of the first input were [0.97, 0.16, 0.28, 0.65, 0.52] and those for the second input were [0.43, 0.89, 0.12, 0.76, 0.25] then one possible hypercube is plotted in figure 13. Note that there is exactly one cross in each row and column: this is why the sample is known as “Latin”. The same approach can be used in more dimensions and with other distributions, including combining samples from different distributions.

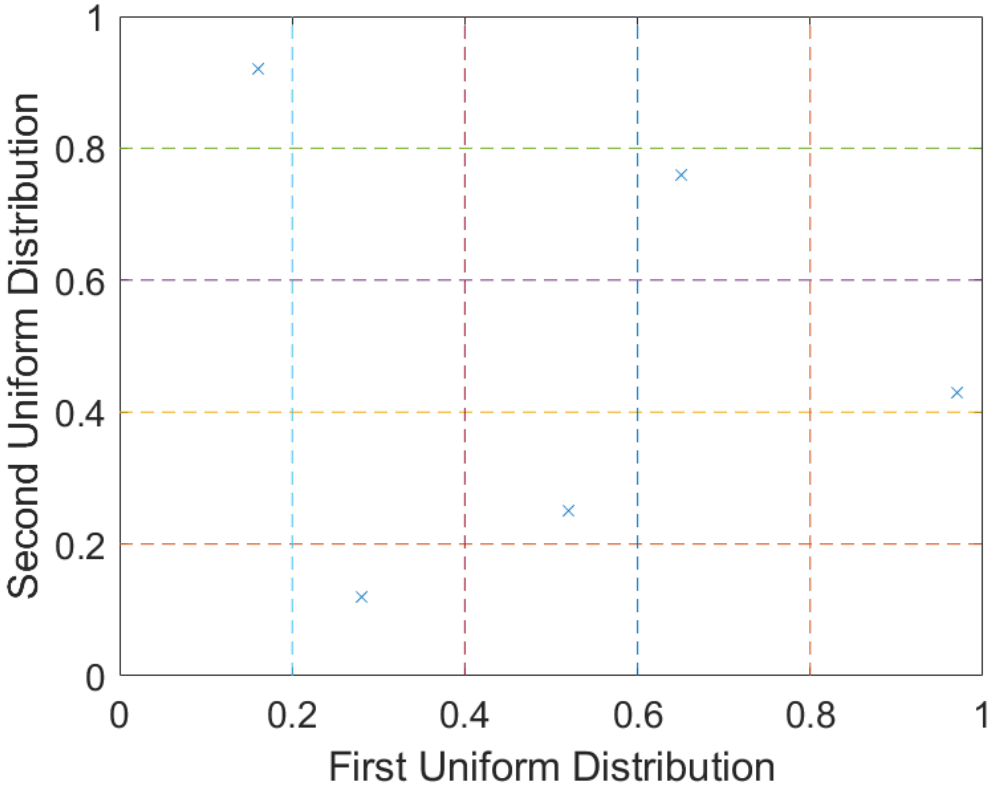

Figure 13: an example of Latin hypercube sampling with two distributions (uniform distributions between 0 and 1) and five samples. Axes are partitioned into five intervals and there is one point in each interval for each distribution respectively.

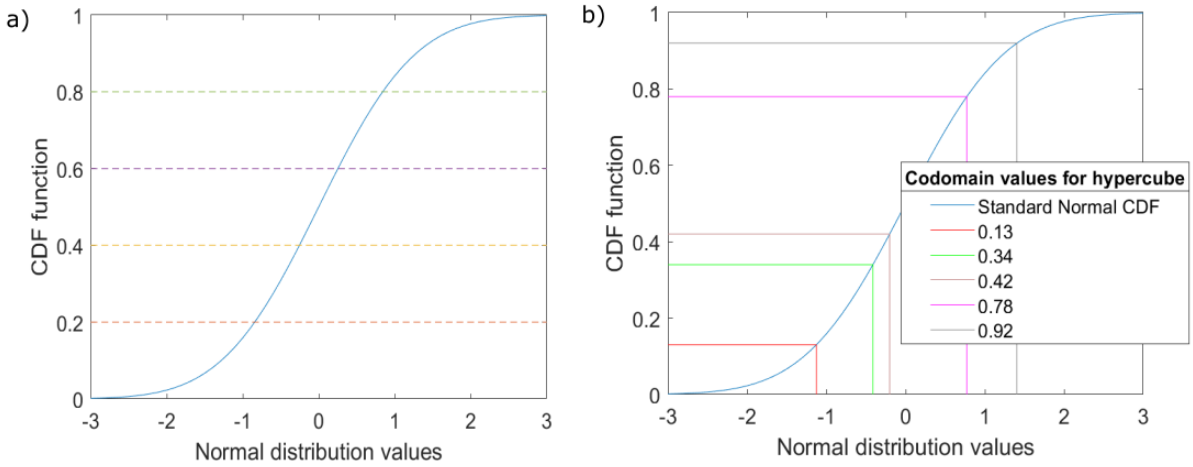

Figure 14. Example of normal CDF graph with 5 partitions. (a) Demonstrates the partition of the codomain and (b) demonstrates a random variable sampled from each of the intervals.

Figure 14 is an example using a Normal N(0,1) distribution with five samples. The samples of probability [0.42, 0.34, 0.92, 0.13, 0.78] lead to sampled values of [-0.20, -0.41, 1.41, -1.13, 0.78]. These values could be combined with samples of other inputs to generate a hypercube.
Figures 13 and 14 show this simple, yet highly effective, method of sampling for the Monte Carlo approach used here. In the case of the uncertainty-aware Canny operator, the algorithm generates a hypercube that treats each pixel in the image as an input, leading to a randomly sampled version of the image that is then processed using the Canny operator. The tool can also generate a Latin hypercube for the Gaussian blur σ_G the Canny threshold limits t_l And t_u.

3 COMPARING MONTE CARLO METHOD AND LATIN HYPERCUBE SAMPLING

Within the MCM used in this work, images are iteratively regenerated to measure the output of the proposed uncertainty-aware Canny operator algorithm using area as the measurand. When using RS, the typical number of trials (and the number of trials used here) is 10^6 iterations, which can often be expected to reliably deliver a 95 % coverage interval (to two significant figures) for the output quantity. [2], resulting in a quantitative response for area uncertainty. However, the large number of images generated means that the RS method has a significant computational time and cost. To ensure that the RS MCM approach computes results in a reasonable time, image size has to be reduced (hence using a 57 x 57 pixel image in the comparison method below), which is unrealistic in real-world scenarios where smaller images result in a loss of detail and information due to the low resolution. LHS provides a suitable alternative, yielding more reliable results for smaller sample sizes (discussed in Section 3.3), so long as estimation of the extremes of the tails of the input distributions are not required [3]. Fewer iterations are required for LHS to obtain more stable results than RS [5]. Through testing and qualitative comparison, the LHS was set to 1000 iterations, which is shown in the following method to be comparable to 10^6 iterations of RS. This takes significantly less time (see Section 3.3 for a quantitative comparison).

3.1 INPUT DISTRIBUTIONS

Before the sampling methods are implemented, the input distributions must be defined. In real-world applications of this method, the user will be able to assign appropriate, traceable uncertainties to the input image. However, the work presented here is based on idealised test images and the uncertainties are assigned for testing purposes only. The input distributions used within the RS and LHS comparison were:

- Individual pixel intensities:
  - N(μ, σ_p) where μ is the pixel intensity from the original image, and σ_p is user specified. For the following method, σ_p is set to five since this yields some variation around the edges of images without changing the main body of shapes. This normal distribution is discretised and truncated to take integer values between 0 and 255.
- Canny thresholds t_l and t_u:
  - U(0,1) because thresholds are always between 0 and 1. For some images, as demonstrated in Section 4, certain thresholds allow different areas of the image to be selected, which can cause issues for analysis. It is required that $t_l < t_u$, and so the uniform distribution is sampled twice from [0,1], with t_l and t_u being set to the lower and upper of these values respectively.
- The Gaussian blur standard deviation σ_G:
  - fixed to a value of $\sigma_G = 3$ for Section 3.2, however it can be attributed a uniform distribution, as demonstrated in Section 4.
  - The kernel size is set to a size of 11.

3.2 CANNY OPERATOR ON SIMPLE IMAGE

To test the proposed method, a simple 57x57 pixel image of a zero background with a high intensity (pixel values = 255) 38x30 pixel rectangle and expected area of 1140 pixels was created, shown in Figure 15. A gradient was added to the top and bottom boundary of the rectangle to allow for some uncertainty in the edges. The test image and the resulting Canny operator output is shown in Figure 15(b). In Figure 15(b), the corners have not been identified accurately by the Canny operator. This is likely due to the Gaussian blur and deficiencies in the Canny operator which are further described in Section 5.2.

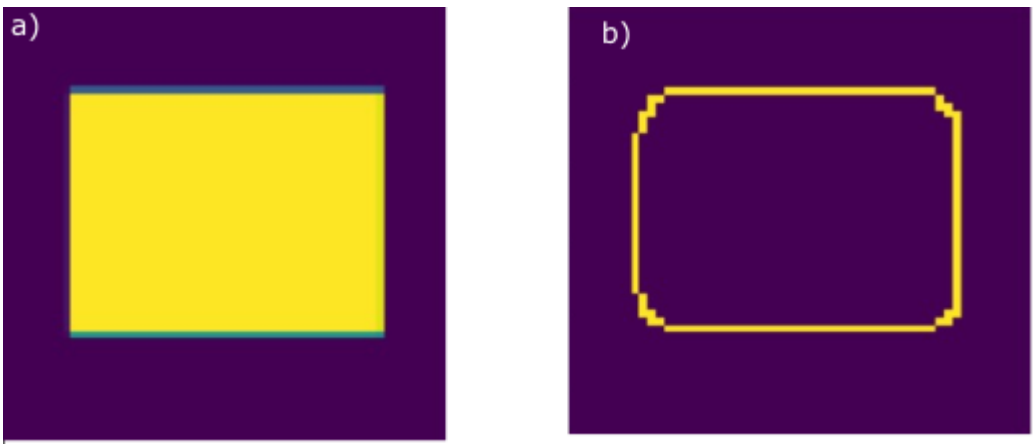

Figure 15: (a) Simple rectangular geometry with gradient to boundary on the top and bottom. (b) Shape returned after applying Canny operator to it with distributions as described in Section 3.1.

The region within the shape is then filled with maximum pixels and the area is calculated and stored, resulting in Figure 16. Following this, both RS and LHS are applied to the image,

resulting in Figures 17 and 18 respectively (which are discussed in Section 3.3). The expected and measured areas (for both RS and LHS) for the simple rectangle are not an exact match for several reasons. Firstly, the Gaussian blurred top and bottom edges have increased the size of the shape, which will inevitably increase the measured area. Secondly, the corners of the edges found by the Canny operator are chamfered which, again, will alter the measured area of the rectangle with respect to the expected area. However, these issues are common in most edge-detection image processing method – this tool aims at addressing the need for uncertainty quantification in image processing rather than perfecting the Canny operator. Other edge-detection methods could solve this problem and including them into the tool is the basis of future work.

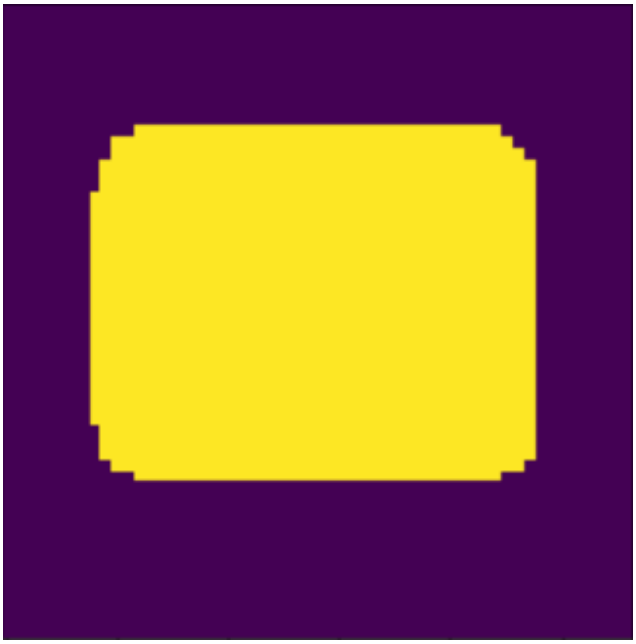

Figure 16: Filled image of Figure 15 (b).

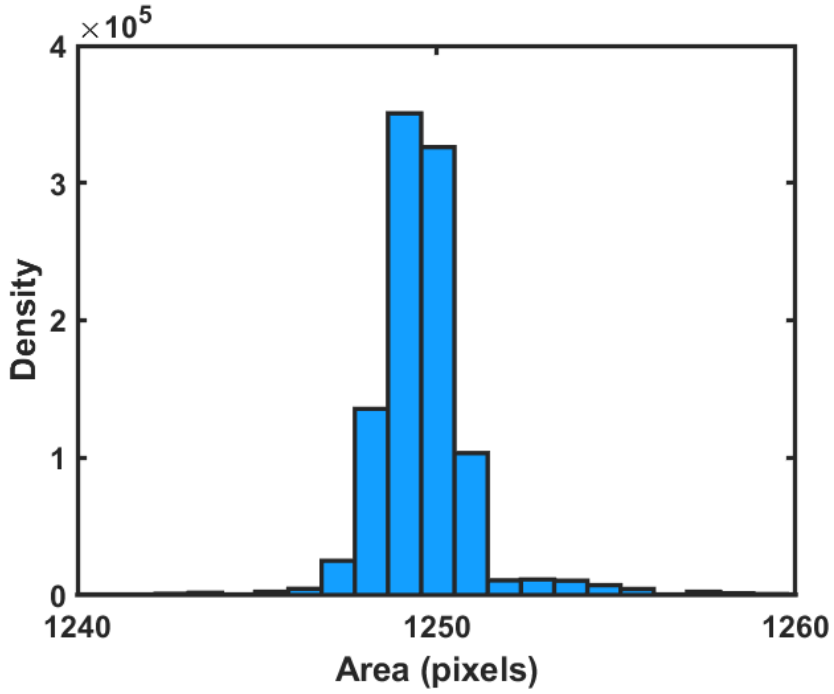

Figure 17: Histogram of areas generated from 10^6 iterations of RS, based on Figure 15.

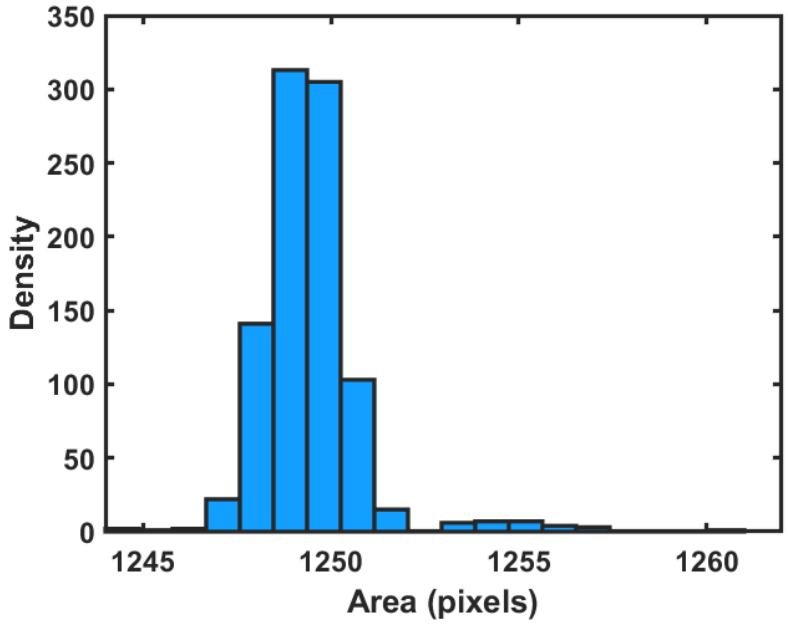

Figure 18: Histogram of areas generated by 10^3 iterations of LHS, based on Figure 15.

3.3 COMPARISONS BETWEEN RS AND LHS RESULTS

The distributions of the calculated area for the simple rectangle image, using the MCM with RS and MCM with LHS methods, are given in Figures 17 and 18 respectively. The general shape is shared across RS and LHS with a wider range in the RS graph. Both distributions have the same mean of 1249, with RS $\sigma = 1.525$ and LHS $\sigma = 1.471$. Using a two-sampled z test on the data, the z-score value is -0.09 with p value of 0.93, meaning we accept the null hypothesis. The null hypothesis of a two-sample z-test is that both datasets come from the same normal distribution, meaning there is not enough evidence to suggest the data comes from different distributions. This, therefore, provides evidence that LHS with 10^3 samples could be an accurate replacement for RS with 10^6 samples.

The difference in range is to be expected because LHS returns a more symmetric distribution and is less reliable to assess the extreme points of the input distributions [3]. Other sampling methods, such as importance sampling [3], could provide a better alternative for reliably sampling from the extrema, however this was beyond the scope of this project and could form part of future work. It is important to note that some images lose ‘closedness’ when running the uncertainty methods, meaning when generating random variables for pixels from the distributions, multiple consecutive edge pixels are generated as non-edge pixels, leaving a gap in the detected edge. This gap means that the image would be layered differently to expected, since if an edge is not closed, it is not shown as a layer. For example, suppose that Figure 15 was generated such that the edge was left open, the only layer found would be the background, which would consist of the whole image. This background would be the zeroth layer, meaning if the algorithm was measuring the 1st layer, a ‘zero area’ would be returned. This is because the 1st layer would not exist on this iteration. These zero areas are not included in the histograms but is important to note as a comparison. In LHS, 68 out of 1000 images had zero areas and in RS, 73253 results were zero and hence omitted (1073253 iterations were carried out so that the software returned 10^6 values). In both cases, approximately 7% of results are zero areas and hence discounted. This issue can be minimised by specifying Canny threshold ranges and standard deviations such that edges are more easily found. However, this issue could be more severe if the algorithm was run on images where the number of closed regions is unknown. Note that the threshold and pixel intensity were sampled together, as it was assumed that both uncertainties would be present

in this edge detection algorithm. Separating them and carrying out a sensitivity analysis was beyond the scope of this project, but could form part of future studies.

4 RESULTS

In the following section, two example images will be evaluated to demonstrate the breadth of use for the algorithm. In Section 4.1, a small 116 x 58 pixel image with overlapping layers is used to demonstrate the layering functionality. Section 4.1 is further used to demonstrate boundaries on thresholding – since specific boundaries mean specific regions are selected due to edges not closing (as mentioned in Section 3.3.1). In Section 4.2, a realistic 1130 x 806 pixel image was used. Due to the size of this image, a rebinning process was applied which scaled the image down to make it smaller and faster to process. This rebinning process is described in more detail in Section 4.2.

4.1 SMALL LAYERED IMAGE

In this section, a 116 x 58 pixel image is used with multiple layers of overlapping simple shapes; the raw image is shown in Figure 19. This image is made up of four simple geometries (circles and ellipses), with some overlapping shapes to increase the complexity – done so to test the tool's ability to distinguish them. It is expected that the tool will be able to extract five layers from the image, including the background, which are labelled in Figure 19. The expected area for the shapes corresponding to (b) to (e) were: 1533 pixels, 1610 pixels, 531 pixels and 126 pixels respectively (calculated by knowing the radii and major and minor semi-axes of the respective shapes). Differences in measured and expected areas were anticipated as the image must go through numerous processing steps (Gaussian blurring, non-maximum suppression, dilation, and erosion, etc.) before the area is calculated which can slightly change where the edges are and, therefore, how big the shapes are.

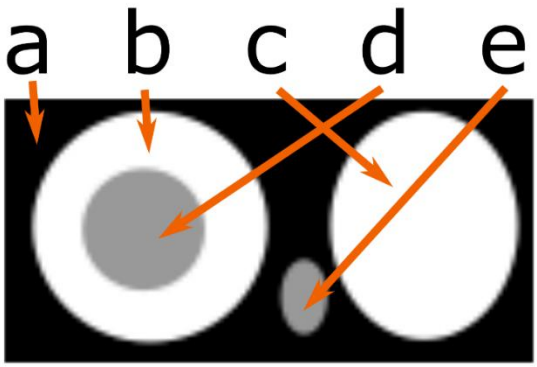

Figure 19: Small layered image with 5 expected layers (including the background layer).

After the image has been processed in the steps defined in Sections 2.1 and 2.2 (i.e., the Canny operator and Area calculation but before the MCM), the resulting extracted layers were generated and shown in Figure 20 with each calculated area is given for the corresponding shapes. Qualitatively, one can see that the number of extracted layers matches the number of layers in Figure 19 but as mentioned above, there are differences in the measures and expected areas. The results presented in Figure 20 were generated using fixed Canny lower and upper threshold values of $t_l = 0.1$ and $t_u = 0.3$ respectively. However, when these threshold values are changed, the Canny operator closes certain shapes and leaves others open.

The lower and upper Canny threshold limits were set to t_l = 0.3 and t_u = 0.6 respectively, and the generated images of the extracted shapes is given in Figure 21. As can be seen in Figure 21, the Canny operator produced only three shapes, including the background, by combining the shapes of b and d in Figure 19, and completely discarding shape e. This is due to the lack of strong pixels during thresholding, which causes hysteresis to reject too many edge pixels, thus leaving edges open and not bounding layers (see Section 2.1). The three extracted shapes do not reflect the expected number of extracted shapes in the original image (Figure 19) and this outcome stresses the need for care to be taken when setting the threshold limits – as is the case for any application of the Canny operator.

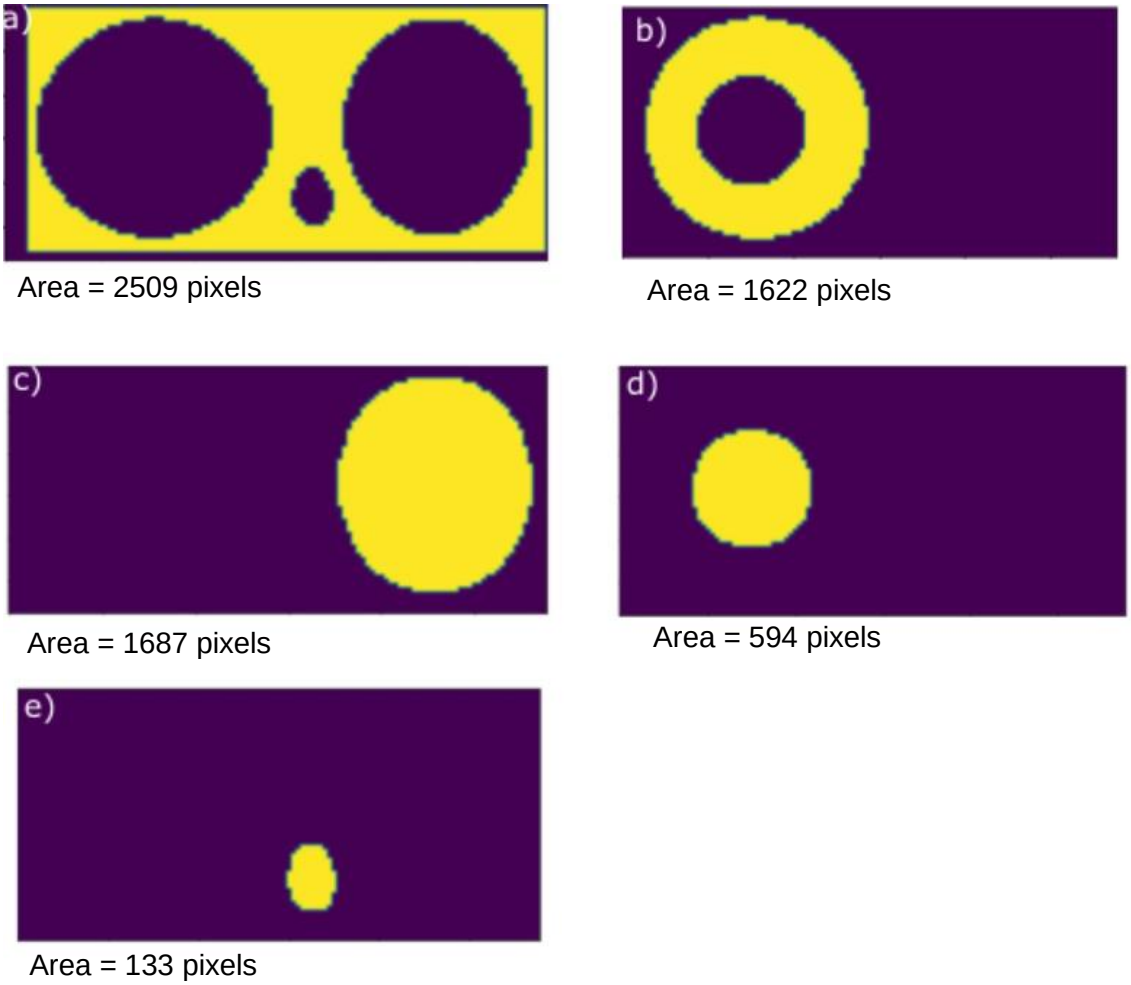

Figure 20: Layers of Figure 19, calculated by the method described in Section 2. Layers are highlighted in yellow with their measured areas given below

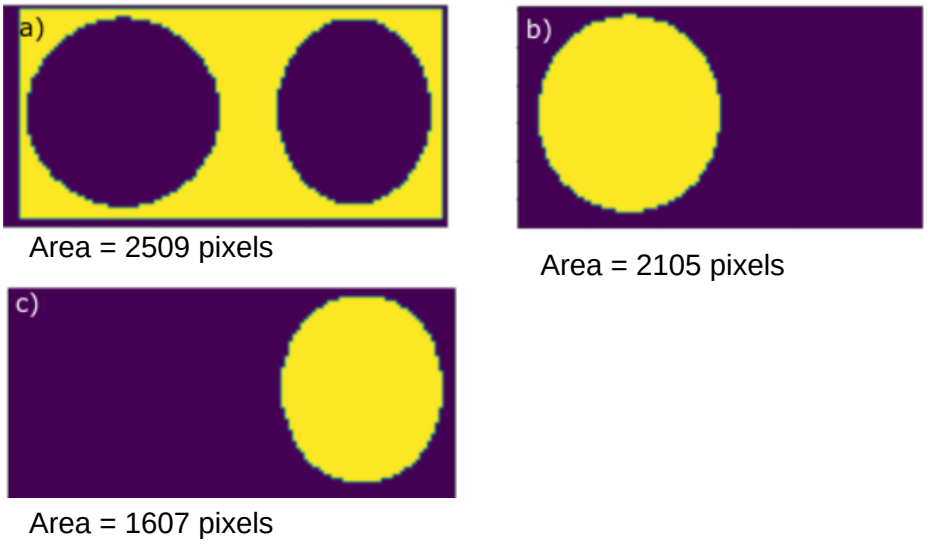

Figure 21: Example of processing Figure 19 with non-optimal thresholds (0.3 and 0.6), causing a loss of layers.

To explore the Canny thresholding sensitivity argument more, the input Canny threshold values were assigned a distribution and the image was processed through the full uncertainty-aware Canny operator tool, as described above. The Gaussian blur and pixel intensity inputs were fixed in order to examine the output of the tool given a varying set of threshold values. The target shape is (b) in Figure 19. The Canny threshold values were drawn from a uniform distribution U(0,1), and the resulting area distribution is given in Figure 22.

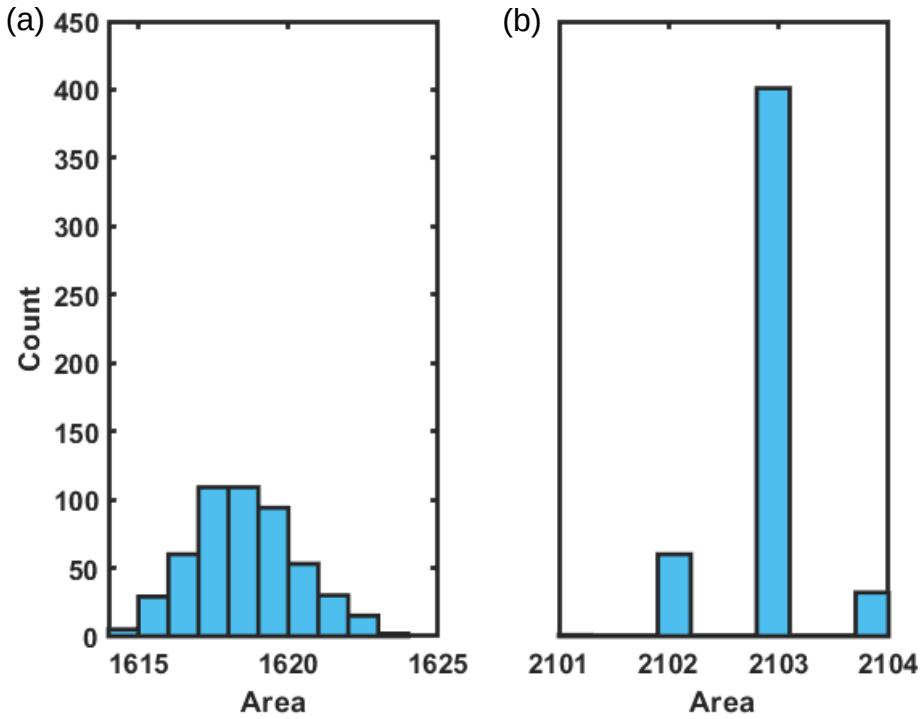

Figure 22: Histogram of the area distribution acquired from sampling Figure 19 (b), with thresholds selected using a U(0,1). Figures (a) and (b) are subplots from the single area distribution carried out with the LHS MCM. Subplot (a) shows the resulting area distribution corresponding to the layer in Figure 20 (b). (b) shows the resulting area distribution corresponding to the layer in Figure 21 (b).

Figures 22 (a) and (b) depict two sub-histograms from the same source area distribution plot. Each subplot has a distinct peak in its respective area domain. The use of a U(0,1) on the threshold distribution means that both layer breakdowns (Figures 20 and 21) can occur in different iterations of the same LHS method. This results in the tool analysing area distributions for both Figure 20 (b) and Figure 21 (b) in the same overall area distribution (for Figure 19 (b)), creating two distinct area distributions in the output plot (shown in Figure 22).

The tool's inability to extract the expected number of shapes in this simple image stresses the importance of selecting appropriate threshold values, prior to the application of the LHS MCM process. From here on in, suitable threshold values were chosen and set as fixed values – in the case of Figure 19, they were $t_l = 0.1$ and $t_u = 0.3$. These thresholds enable a consistent layer breakdown in the format of Figure 20.

To further test the tool, the Gaussian blur σ_G was assigned a uniform distribution U(1,2) and the LHS MCM was applied. Figure 23 shows the result of considering both the threshold and Gaussian blur σ_G uncertainties for the extracted shape shown in Figure 20 (b). As one can see, the calculated area distribution becomes slightly skewed and narrower in comparison to Figure 22(a), and the mean area calculated of Figure 20 (b) is 1619 pixels, with a standard uncertainty of 1.76 pixels.

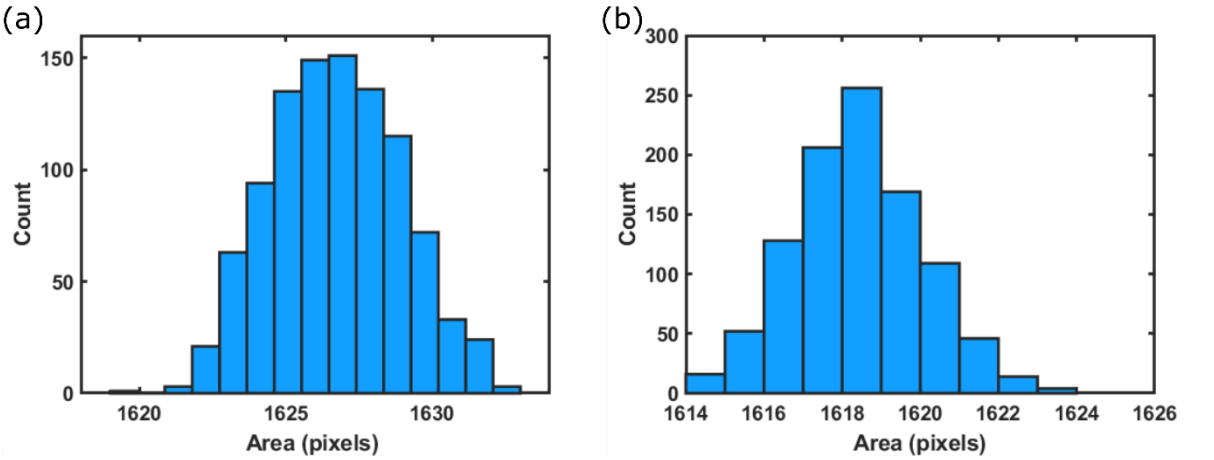

Figure 23: LHS of layer depicted in Figure 20 (b), with the following parameters for each plot. (a) Thresholds U(0.1,0.3), Gaussian blur kernel size 23, Gaussian blur σ_G = 2, pixel distribution σ_p = 5. (b) Thresholds U(0.1,0.3), Gaussian blur kernel size 23, Gaussian blur σ_G = U(1,2), pixel σ_p = 5. σ_G was set with uniform distribution in b) to test the effects of changing the Gaussian blur standard deviation on the areas calculated.

4.2 LARGE LAYERED IMAGE

Image processing using the uncertainty-aware Canny operator tool takes a significant amount of time due to the vast number of calculations, image generation, and process steps undertaken by the algorithm. When the image size is increased, the computational time also increases, however, to alleviate this issue, the image can be pre-processed by rebinning. Rebinning is a method which involves finding means of clusters of pixels and using those in a new, smaller image. This reduces processing time but also reduces precision in the image.

This technique is often used in fields that generate a large amount of data which can be reduced without loss of information, for example in positron-emission tomography [13].

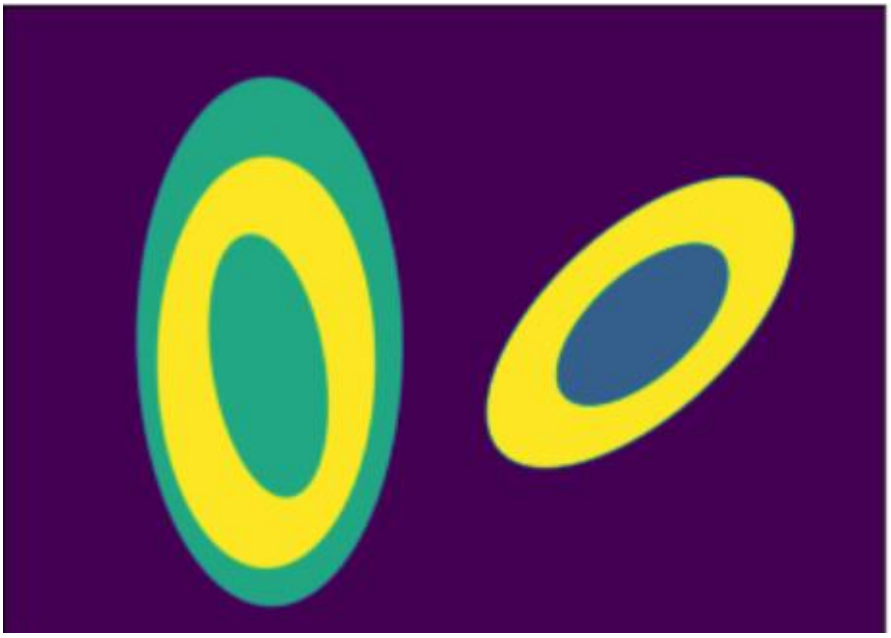

Figure 24: Layered image 1130 x 806 pixels.

Figure 24 is an example of a large image which takes inconveniently long to process with a LHS (approximately 51 minutes for one iteration) but, by rebinning the image, it can be processed into a useful output significantly faster. Figure 24 was reduced to a 1104 x 800 pixel image by cropping the outer edges of the image which meant that image can be rebinned by a factor of eight. The resulting rebinned image is shown in Figure 25.

.

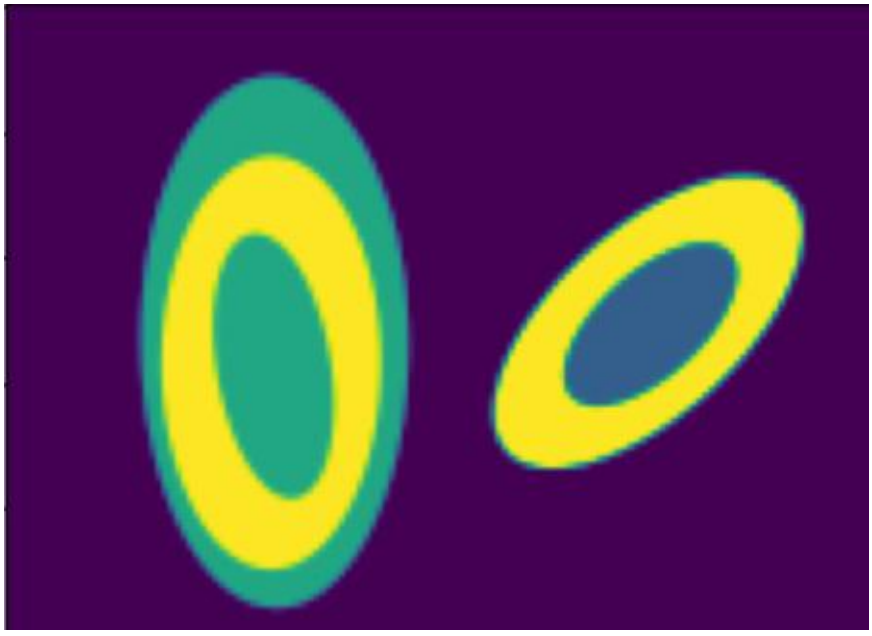

Figure 25: Rebinned Figure 25 to 138 x 100 pixels.

Comparing Figure 24 and Figure 25, the edge lines are more blurred in Figure 25, as expected. However, the general features are still distinguishable, and the edge detection process time is significantly faster (LHS with 1000 iterations takes 11 minutes). If an image contained small features, rebinning could potentially remove any detail of the feature or completely remove it from the rebinned image (this depends on the size of the image relative to the rebinned pixel size).

A specific region of interest was selected (chosen for its size and consistent detection through thresholding) for demonstration and is shown in Figure 26. The MCM using LHS on this resulted in the distribution given in Figure 27.

Figure 26: Region of interest from Figure 25. This is a combination of the outer green and middle yellow layers. The reason this region is used is because the parameters to separate these layers could not be found. This will be elaborated on in Section 5.2.

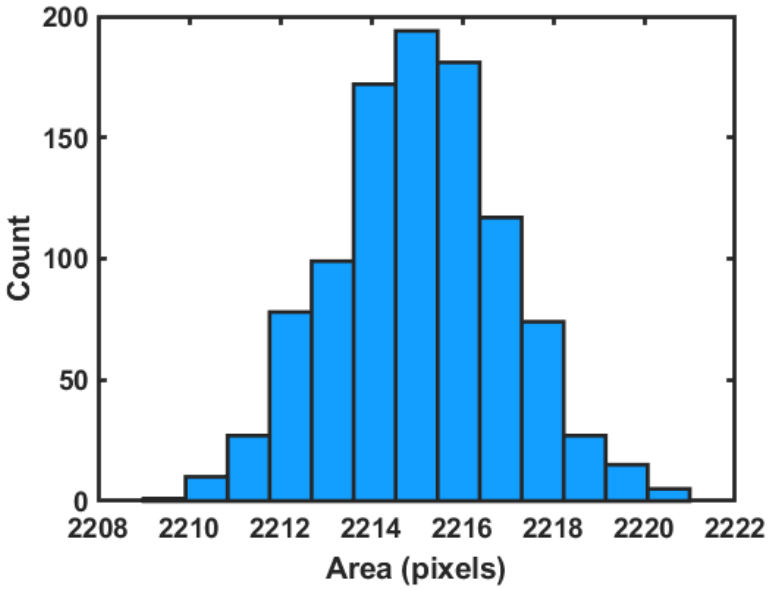

Figure 27: Area results from applying the LHS MCM to the image shown in Figure 26, with parameters: kernel size 23, σ_G = 2, lower and upper thresholds of 0.05 and 0.1 respectively.

Following the rebinning process, the area and uncertainty must be scaled by a factor of the rebinning process as the image was scaled down in order to process it in a timely manner. If the pixel size is known (for instance, in SI units of m), then rescaling is not required as the new pixel size is defined by the rebinning – for example, suppose the pixel size of Figure 24 is 1 mm x 1 mm. The rebinned image, Figure 25, has a pixel size of 8 mm x 8 mm. The rescaling of the final area would therefore not be required. However, since the pixel size is not known in Figures 25, 26, and 27, the measured area must be rescaled in order to accurately describe the area of the original image. From Figure 27, a suitable area estimate might be mean of 2311 pixels with standard deviation of 2.659 pixels, which translates to 18488 pixels with standard deviation 21.27 pixels in terms of the original image.

5 LIMITATIONS OF THE ALGORITHM

5.1 IMAGE SIZE AND PROCESSING TIME

One major limit of this work is the time to run iterations, which is caused by the size of images. Real-world applications of imaging as a scientific measurement typically result in large images, or large datasets consisting of a lot of images. However, calculation time using these larger image datasets can be inconvenient and unusable. One simple solution is the rebinning approach shown in Section 4.3. This process decreases the image size by finding the mean of adjacent pixel intensities and using those in a new smaller image. This decreases resolution of the image and changes shapes however, if only used on large features (with respect to the image size), these differences are not likely to impact the edges detected or areas calculated. Further improvements to the algorithm could be made which would enable the image processing to be carried out on a High-Performance Computing (HPC) hardware, or to be processed in parallel on multiple CPU (or possibly GPU) threads. Another method to reduce this problem is to segment the original image into regions of interest or sub-images. The onus, however, is on the user to select the right sub-image (automatic segmentation is an area which will be explored in future projects).

5.2 LIMITATIONS OF THE CANNY OPERATOR

A further limit of our study is the use of the Canny operator. Despite its widespread use, the Canny operator can fall short on edge detecting without appropriate image pre-processing, especially for real-life grayscale images. This is due to the natural changes in intensity not at edges and due to lighting/shadows. An example of this is shown in Figure 28.

Figure 28: Canny operator applied to a colour photograph of an apple. Complexity in the image results in an edge-image which would be unusable for the uncertainty-aware Canny operator.

When applying the Canny operator to a colour photograph of an apple, textures in the image are shown as edges and regions in the image are removed which shouldn't be. For example, the bottom left corner of Figure 28 is lost due to the angle of lighting on the apple, which creates an unwanted gradient to the background.

This is similar to the issue in Section 4.2, where the Canny operator parameters could not be found such that the area layers could be separated. There is no guarantee that the Canny operator is able find all closed edges accurately, which could set-back the tool's general utility. Furthermore, in applying Gaussian blur to the image, precision and accuracy can be lost, as demonstrated in Section 2.1. In larger images, this is usually covered by the uncertainty bounds, yet in smaller images, this could be a crucial element.

In future iterations of the software, using a more sophisticated edge detection method could alleviate these issues and improve the tool's real-world application. The choices of edge detection algorithm would largely depend on the specific use-case for the software. For example, a wavelet transformation method could be used as an alternative method [6] e.g., if medical images were the chosen application. Furthermore, the wavelet transform could be combined with the Canny operator to aid the processing of images [7]. Alternatively, an Adaptive Neuro-Fuzzy Inference System (ANFIS) could be a useful edge detection method for medical images or facial recognition [8], especially when paired with an algorithm that could join the edges detected together when appropriate. Other edge detection methods available can be found within the literature review [4] and comparison of these methods in real-world applications could be a useful future area of study.

6 CONCLUDING REMARKS

This report presents an overview for the software tool designed to account for uncertainty in image edge detection using a Monte Carlo approach. The Canny edge detection operator was used to detect edges on several example figures, and two different sampling methods were examined for the propagation of uncertainties through the model – LHS is the recommended method due to it being much faster than the rudimentary RS method. The areas for a number of input images were calculated using a custom area calculation process, and the tool was able to successfully extract multiple shapes (overlapping and separate) in images when the input parameters were correctly set. The outcome of the uncertainty-aware Canny operator presented here is the area distribution – which can be used to appropriately assign uncertainties to the measured area which would previously be assigned through some educated guesswork.

This tool was created to be a generalised framework to quantify uncertainties associated with edge detection, however a user can tailor the tool to their needs. If the uncertainties for a specific image are known (acquisition uncertainties leading to pixel-by-pixel uncertainty in the image, for instance), the tool can be changed to accommodate them. For future work, different, more sophisticated edge-detection methods can be implemented, allowing the user to have multiple choices of methods to apply.

7 ACKNOWLEDGEMENTS

The work described in this report was part of the National Metrology Systems project titled 'Understanding Complex Systems' with project code 127098. This work was funded by the UK Government's Department for Science, Innovation & Technology through the UK's National Measurement System programmes

8 REFERENCES

1) Hamza A, Ramen S et al., ‘Tumor Size in Breast Carcinoma: Gross Measurement Is Important!’, International Journal of Surgical Pathology, 2018.
2) BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, and OIML, ‘Evaluation of measurement data — Supplement 1 to the “Guide to the expression of uncertainty in measurement” — Propagation of distributions using a Monte Carlo method’, Joint Committee for Guides in Metrology, JCGM 101:2008. URL: https://www.bipm.org/documents/20126/2071204/JCGM_101_2008_E.pdf/325dcaad-c15a-407c-1105-8b7f322d651c
3) Saltelli A, Chan K, Scott E M, ‘Sensitivity Analysis’, Wiley Series in Probability and Statistics, 2000.
4) Ghosh C, Majumder S, Ray S, Datta S, Mandal S. ‘Different EDGE Detection Techniques: A Review’, Electronic Systems and Intelligent Computing, 2020.
5) Helton JC, Davis FJ., ‘Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems’, 2003.
6) Wan-She L, et al., ‘Application of Wavelet Transform in edge detection’, 4th International Congress on Image and Signal Processing, 2011.
7) Xue L, et al. ‘Edge Detection combining Wavelet Transform and Canny Operator based on fusion rules’, International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR), 2009.
8) Anwar S, et al. ‘A Neural Network approach to edge detection using Adaptive Neuro-Fuzzy Inference System’, International Conference on Advances in Computing, Communications and Informatics (ICACCI); 2014.
9) Joseph K. A, ‘Technical report on Edge detection using Sobel Operator in GDAL’, 2015.
10) Dougherty E. R., ‘An Introduction to Morphological Image Processing’, SPIE Optical Engineering Press, An Introduction to Morphological Image Processing, 1992.
11) Canny, J., A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8, 1986, doi:10.1109/TPAMI.1986.4767851.
12) *Y. Li and B. Liu, ‘*Improved edge detection algorithm for canny operator’*,* 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, 2022, doi: 10.1109/ITAIC54216.2022.9836608.*
13) M. Defrise, P. E. Kinahan, D. W. Townsend, C. Michel, M. Sibomana and D. F. Newport, "Exact and approximate rebinning algorithms for 3-D PET data," in *IEEE Transactions on Medical Imaging*, 16, 2, 1997, doi: 10.1109/42.563660.