Cell Nuclei Detection

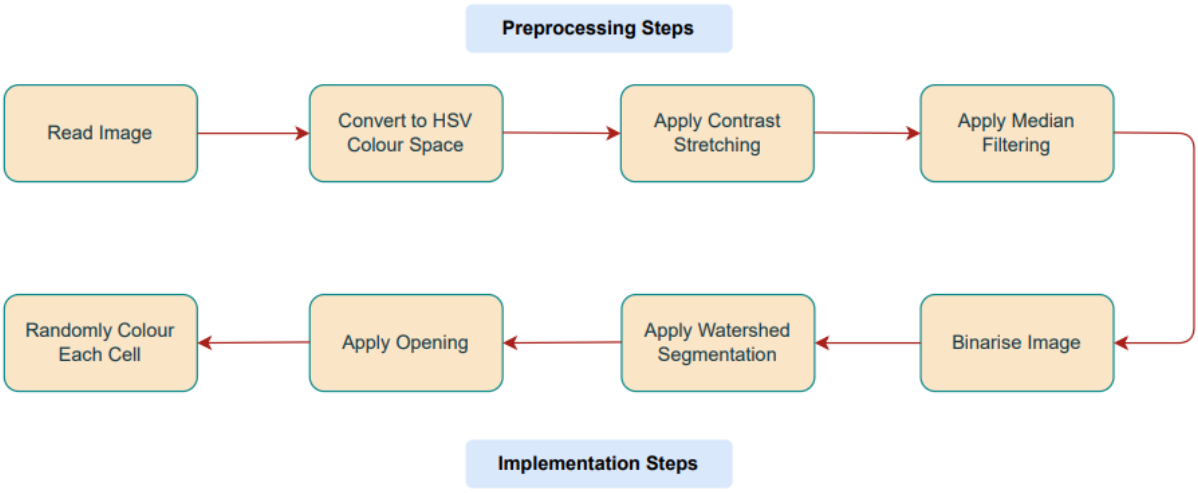

Figure 1.1.1 : Pipeline of the entire process

1 Pre-processing

1.1 Reading the Image

First and foremost, use the *imread* function to read and display the original image as shown in Figure 1.1.2. The default example image for the following diagrams will be "StackNinja1.bmp" if no other image is specified.

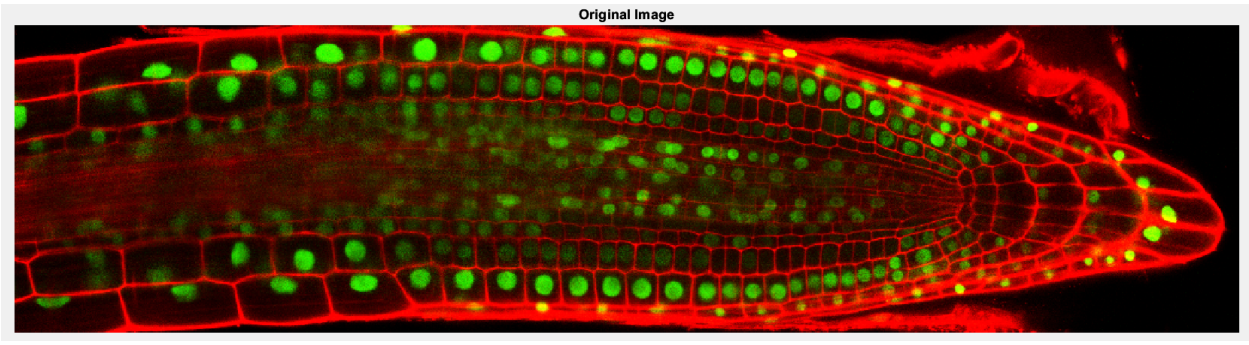

Figure 1.1.2 : StackNinja1 Image

1.2 Contrast Enhancement

The image shown in Figure 1.1.2 shows many green nuclei visible to the human eye. However, Matlab may have difficulty picking up the darker green nuclei during the subsequent processes. To address this issue, I have chosen to use *imlocalbrighten* which is a local contrast enhancement that brightens the overall image. Bressan (2007) states that contrast enhancement helps bring out the important features of the image that may have been hidden due to poor contrast, in this case, the darker shades of green nuclei. Figure 1.2.1 shows the brightened image.

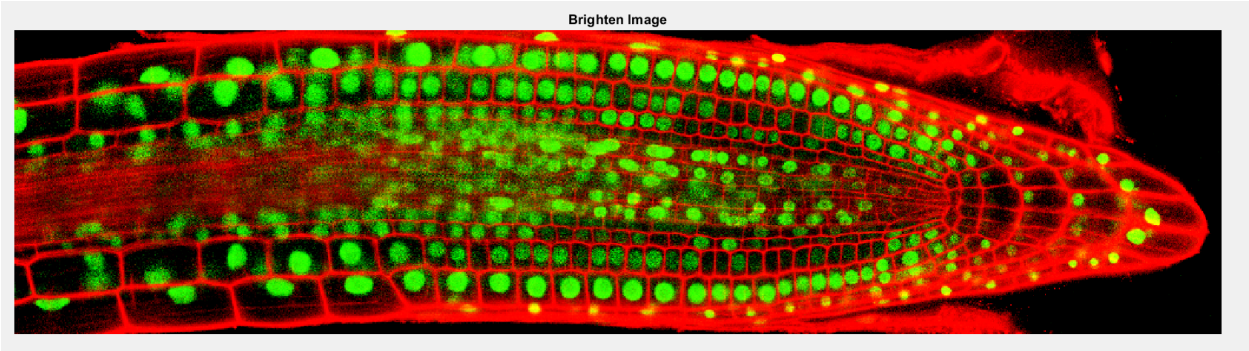

Figure 1.2.1 : Brighten Image

One Issue with contrast enhancement was that the red cell walls also became brighter along with the green cell nuclei. This caused some of the isolated cells to not be detected due to being surrounded by bright walls. Another method I had considered was intensity scaling. The process involves multiplying the green pixels by a multiplication factor to increase the intensity of the pixels, making them brighter. This allowed me to extract the harder-to-reach pixels that were surrounded by red cell walls. However, I decided against using this technique as intensity scaling caused a large amount of noise to appear. A majority of the noise created cannot be removed in future steps as they formed large chunks of objects that were incorrectly depicted as cells and therefore not removed during preprocessing.

1.3 RGB

For the colour space, I have decided to work in the Red, Green, Blue (RGB) colour space. I first separated each colour channel, and performed the formula *green - (blue + red)/2* which helps with colour correction and balancing.

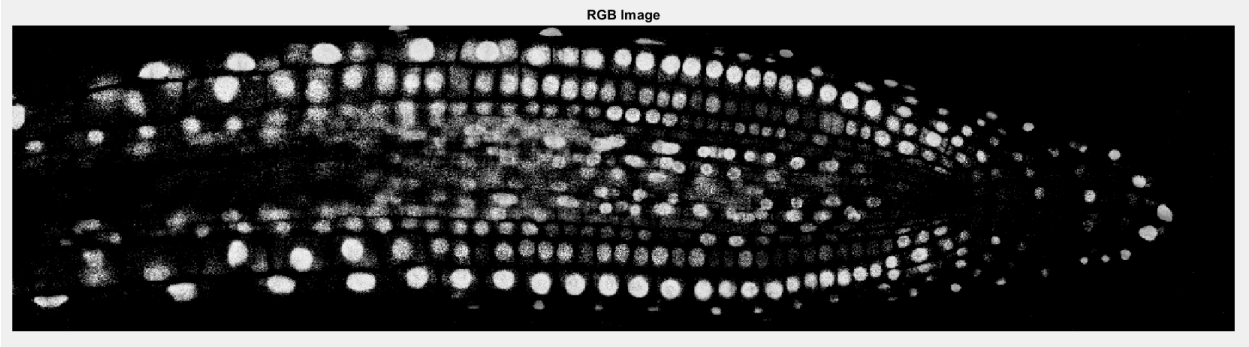

Figure 1.3.1 : Extracted Green Pixels from RGB

When choosing a colour space to work with, it is important to consider the properties of the image to be processed. According to Palus (1998), RGB is one of the most common colour spaces to work with for digital images such as the ones provided. However, a disadvantage of RGB is that it isn't perceptually uniform, and cannot fully represent all the colours perceived by the human eye. When compared to other popular colour spaces such as HSV, the amount of green cell nuclei extracted were similar, with a slight edge to RGB as it managed to extract more of the outskirt pixels. The primary advantages of HSV are the straightforward ways to manipulate the colour values or saturation, but since we have already applied contrast enhancement, there is no need to manipulate these values further.

1.4 Median Filtering

After extracting the green pixels, noise will appear within the structure and outside.

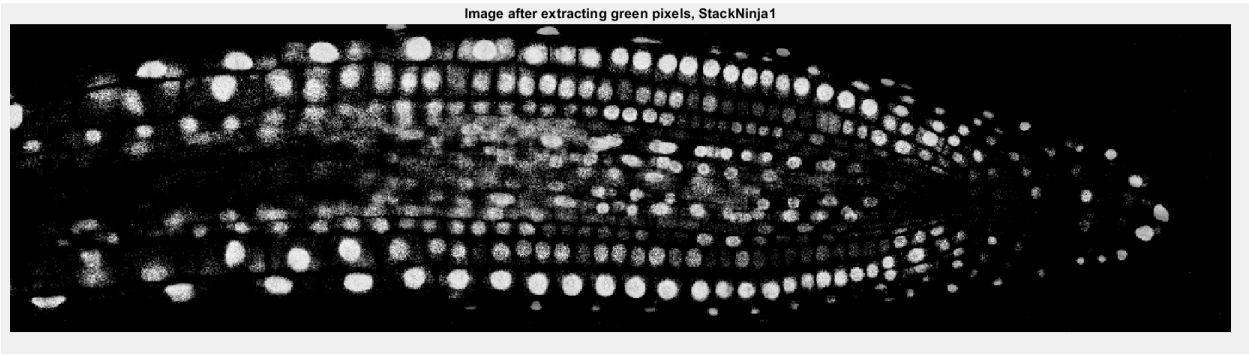

Figure 1.4.1 : StackNinja1 image after extracting green pixels

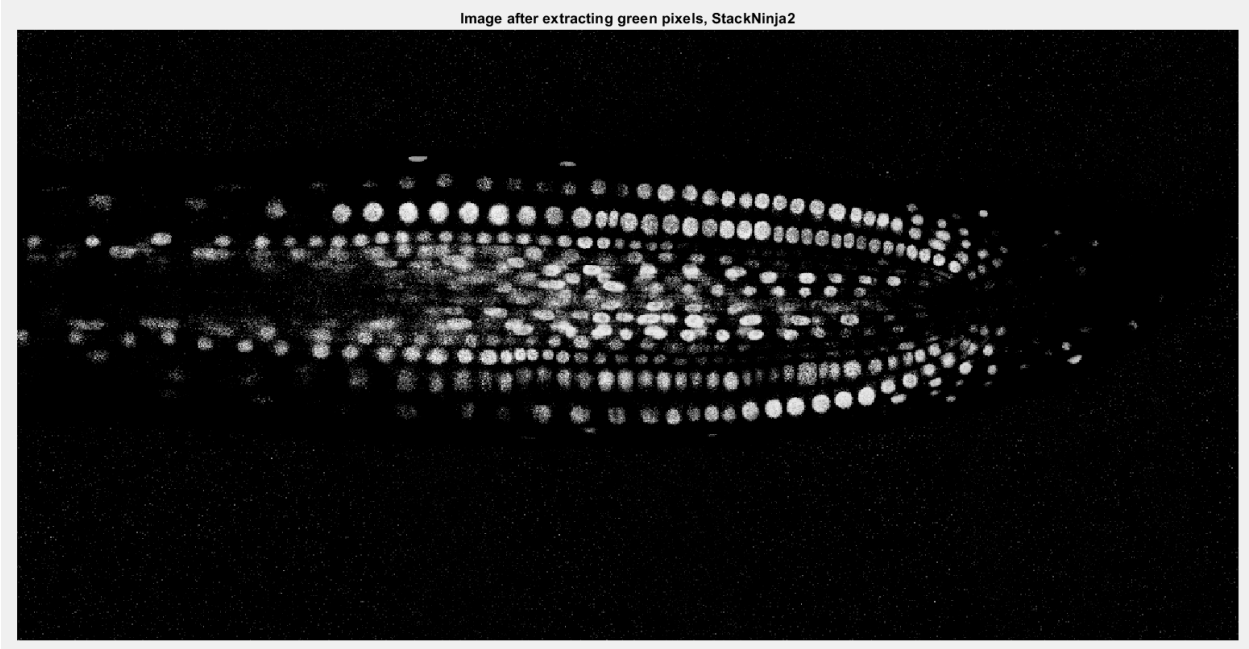

Figure 1.4.2 : StackNinja2 image after extracting green pixels

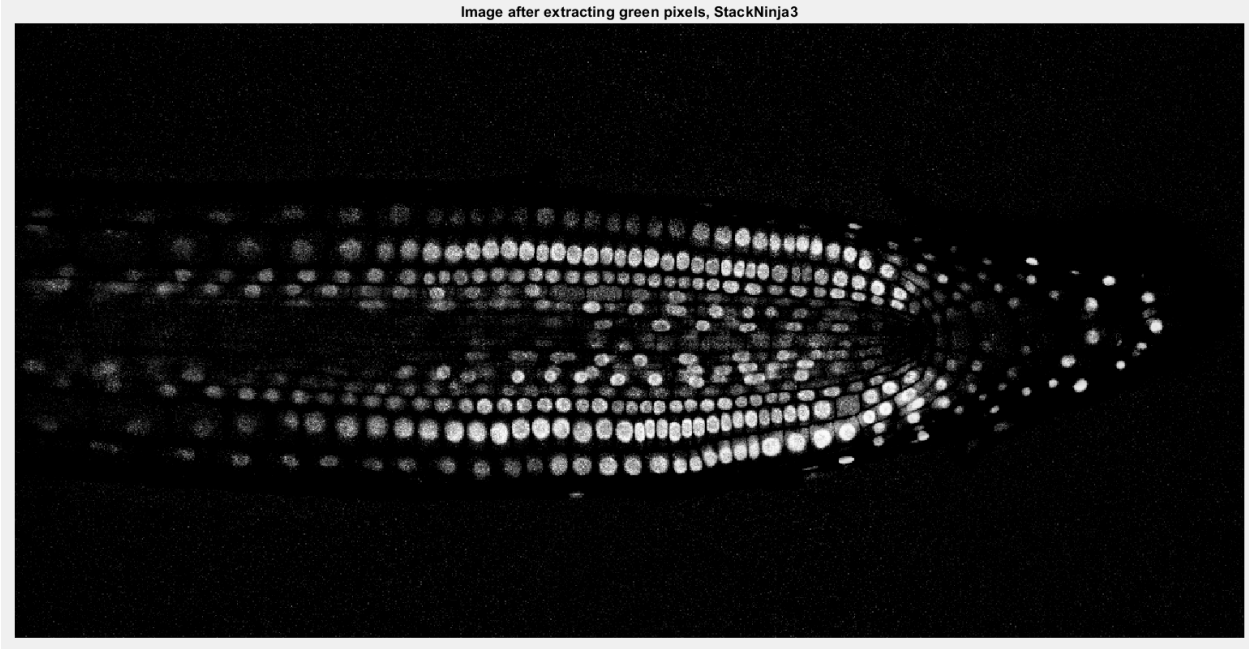

Figure 1.4.3 : StackNinja3 image after extracting green pixels

Figure 1.4.1, 1.4.2, and 1.4.3 shows the images of StackNinja1, StackNinja2, and StackNinja3 respectively. As shown in these figures, the problem with noise is present in all of the images. The noise will disturb the process of colouring the regions of interest,specifically the green cell nuclei.

To tackle the issue of noise, a 2-D median filter using the matlab's function *medfilt2* is then applied to the RGB image to smoothen out the image and remove excess noise so that each nuclei can be easily identifiable.

Gupta (2011) explains that Median Filter is an edge preserving and noise removing filter that replaces the value of each pixel with the median value of the neighbouring pixels. The size of the neighbourhood can be defined through a kernel or window size. Median Filter is good for removing impulse noise, also known as “salt and pepper” noise as it is less sensitive to extreme values compared to a mean filter.

The default value of the neighbourhood size is usually a [3 3] matrix. After experimenting with a [3 3], [5 5], and [7 7], I have concluded that a neighbourhood that is too big causes over-smoothing, which merges many of the cells. Between a [3 3] and [5 5], a [3 3] still had some remaining noise, causing the overall image to be over segmented due to additional pixels. Therefore, I chose to use a [5 5] kernel size as it brings a good balance between noise removal and avoiding over-smoothing.

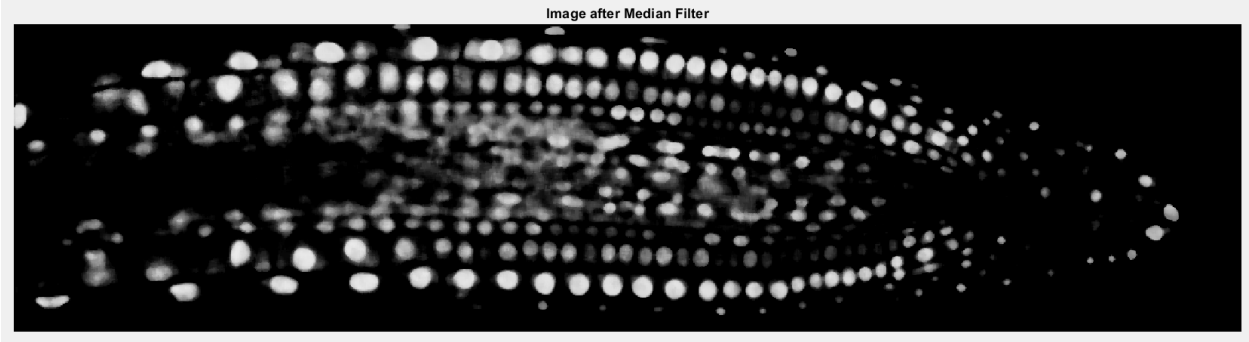

Figure 1.4.4 : Image after undergoing Median Filtering

Figure 1.4.4 shows the image after applying Median Filtering. When compared to Figure 1.4.1, the noise around and within the area of the cell nuclei has been reduced and the remaining cell nuclei appear to have more distinct edges. However, not all the noise has been removed, and the filter has caused some of the cells to merge together, which proves to be a disadvantage of this filter. In this scenario, the pros outweigh the cons as the drawbacks can be addressed in the subsequent processes.

2 Implementation

2.1 Binary Image

Once the green pixels have been extracted from the RGB colour space and the relevant filter has been applied, we will then convert the image into a binary image using the *imbinarize* function. The function will apply a threshold to the image where the pixels that meet the threshold criteria, with a set sensitivity value, will be converted into white pixels while the pixels that do not meet the criteria will be converted to black. The threshold technique used to calculate the threshold criteria is the adaptive method with a sensitivity value of 0.3. This method calculates the threshold value for each pixel based on the local intensity of neighbouring pixels (Roy, 2014). Figure 2.1.1 shows the binarised image.

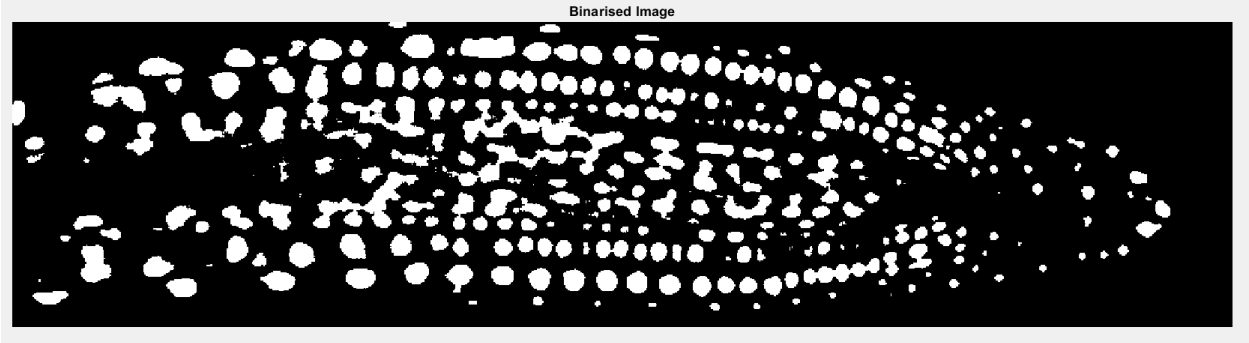

Figure 2.1.1 : Binarised Image using adaptive method

Matlabs provides two options for binarisation methods: global or adaptive. Global uses the Otsu method to calculate a global image threshold while the adaptive method takes a local approach. Kim (2017) states that the adaptive method often outperforms the global Otsu method because it takes into account the local characteristics of the image using the first-order image statistics of each pixel. Figure 2.1.2 and Figure 2.1.1 shows the binary image using global method and adaptive method respectively. There is a noticeable difference between the adaptive and global method as the global method contains fewer cells.

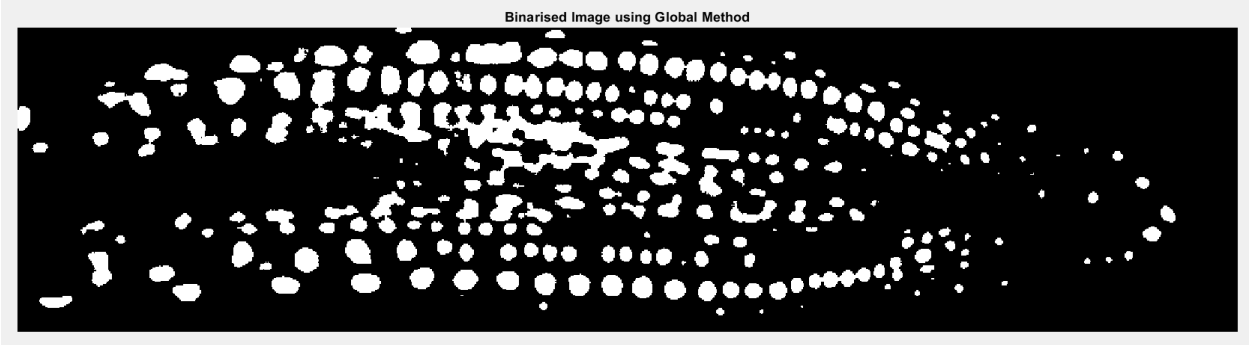

Figure 2.1.2 : Binarised Image using global method

Adaptive thresholding is more suitable for images with varying levels of lighting throughout the image. In the original image, many of the cell nuclei are of varying intensities that indicate they are overlapping each other, as shown in Figure 1.1.1. Global method would create a threshold based on the entire image, resulting in many of the lightly shaded cells to disappear. Since the adaptive method creates local thresholds, more of the cell nuclei are retained. Therefore, I have selected the adaptive method to preserve as many cell nuclei as possible.

For the sensitivity, the value ranges from 0 to 1. A low sensitivity will pick up fewer details while a high sensitivity will pick up more details, possibly noise. After testing with values 0.2, 0.3, 0.4, and 0.9, I have decided to use value 0.3 as it provides a good balance between extracting details and not picking up too much noise.

2.2 Watershed Segmentation

If you examine Figure 2.1.1 carefully, you will notice several issues regarding the segmentation process. The original image contains multiple overlapping cells, distinguished by the different shades of green that they are coloured. However, after the process of converting the grayscale image to the form of binary, all of the cell nuclei would subsequently be converted to white pixels, resulting in the “merging” of the cell nuclei. In addition, the median filtering will have smoothen the image, causing more merging. This problem is present in all 3 provided images.

To address this problem, I have decided to implement the Watershed technique. Hamarneh and Li (2009) have stated that watershed is a segmentation algorithm that splits the image into

regions using intensity values. The image is treated as a topographic map, where the height of the terrain is controlled by the intensity values of the pixels. The algorithm then floods the “basins” from the local minima. At one point, the flooding between different regions will meet, and the watershed lines, or ridges, will form at this line.

Hamarneh and Li (2009) have compared the watershed algorithm to other methods and the results showed that the watershed method outperformed the other methods in terms of the accuracy of the segmentation lines. Due to the numerous overlapping cells in the image, I have chosen to implement the Watershed technique.

The steps to implement the watershed algorithm are as follows :

Calculate the distance transform of the complement of the binary image. This creates a distance map where every pixel value in the output image is the distance between that pixel and the nearest non zero pixel. The method used for calculating the distance transform is the Euclidean.

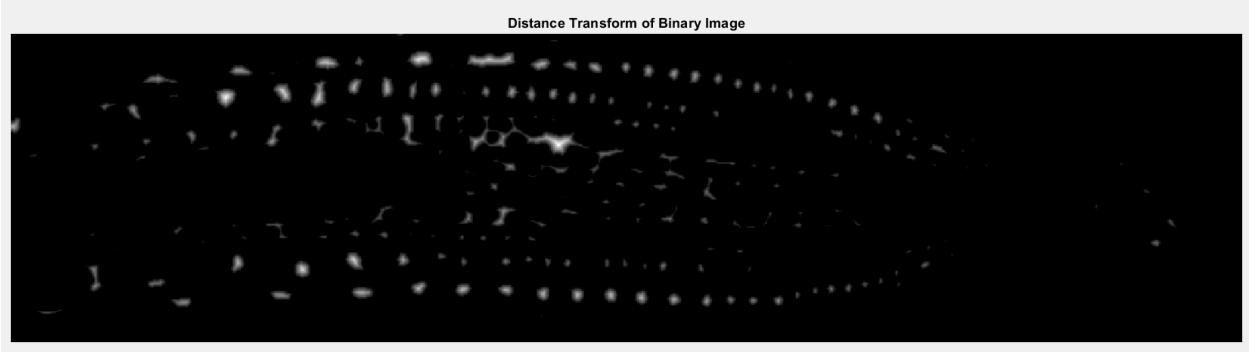

Figure 2.2.1 : Distance Transform of Binary Image

The function *imhmin* is used to suppress the minima *h*, all values with height less than *h* will be set to h. I have selected the value 3 for *h*.

Invert the distance transformed image so that the light pixels represent high elevations and dark pixels represent low elevations for the watershed transform.

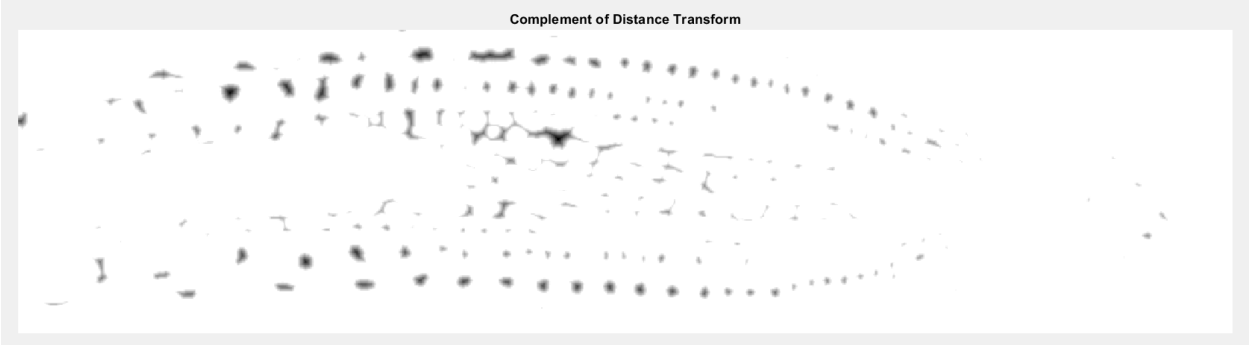

Figure 2.2.2 : Complement of Distance Transform

Finally, calculate the watershed transform and set pixels out of the regions of interest to 0. Convert the resulting image to grayscale and convert all non white pixels to white and the white background to black.

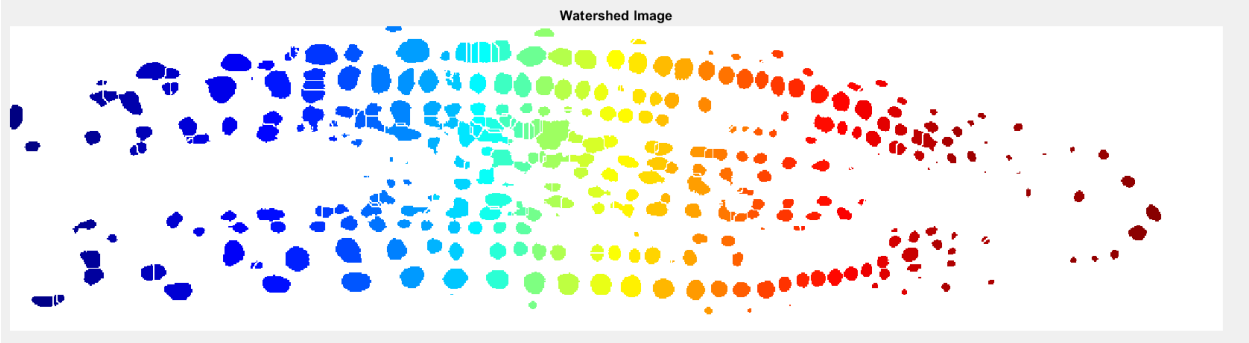

Figure 2.2.3 : Segmented and Coloured RGB Image

I have chosen to use the Euclidean method to calculate the distance transform. Matlabs provides 3 other types of methods for calculating the distance transform, chessboard, cityblock, and quasi-euclidean. Grevera (2007) states that the method used for calculating the distance transform is crucial to properly capture and preserve the details of the image.

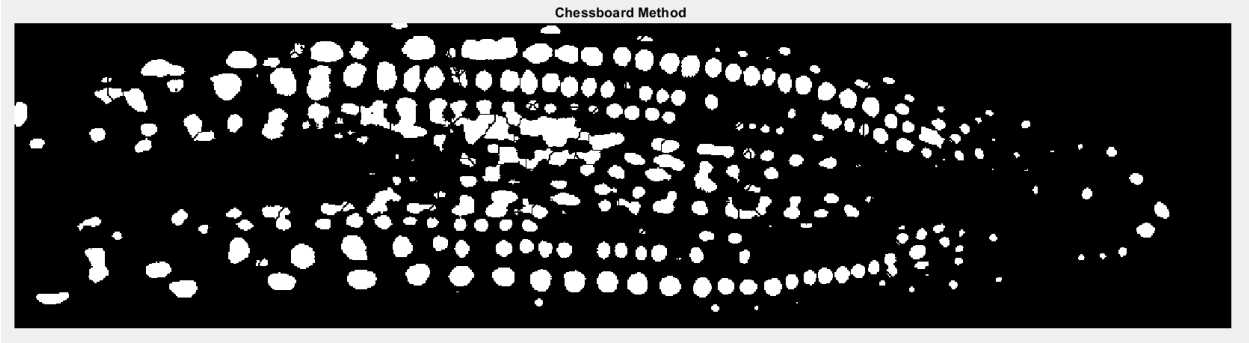

Figure 2.2.4 : Watershed generated using Chessboard method

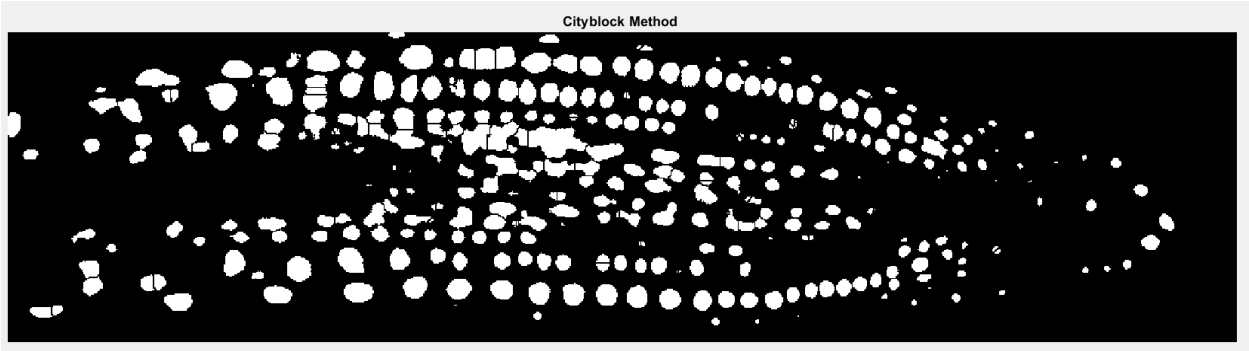

Figure 2.2.5 : Watershed generated using Cityblock method

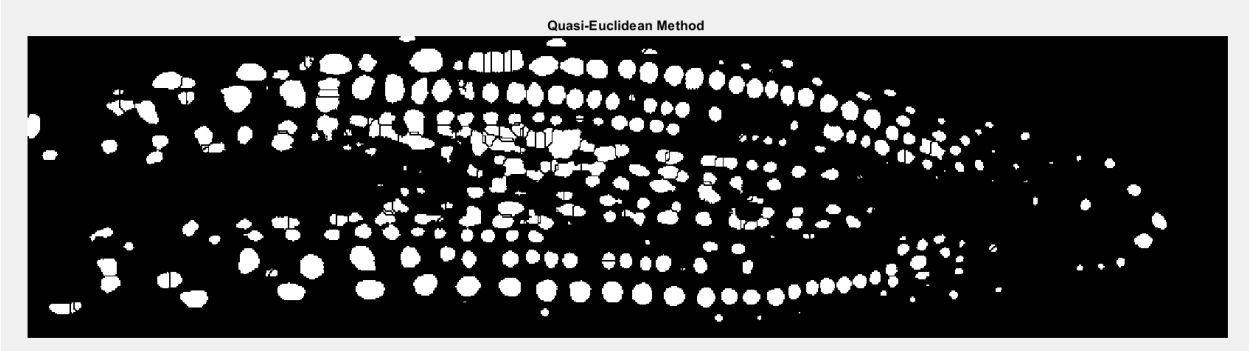

Figure 2.2.6 : Watershed generated using Quasi-Euclidean method

Figure 2.2.4, 2.2.5, 2.2.6 shows the images after applying the chessboard, cityblock, and quasi-euclidean method respectively to calculate the distance transform. When comparing the images to the one produced using the Euclidean method in Figure 2.2.3, you can see that Figure 2.2.4 and Figure 2.2.5 still contains a few merged cells that have not been properly segmented, while Figure 2.2.6 shows a few cases of over segmentation of the cell nuclei. Although the Euclidean method still produces a few cases of over segmentation, it gives a good balance of

properly segmented cells while avoiding excessive over segmentation, hence my reason for selecting that method.

Belaid and Mourou (2011) mentions that the watershed algorithm's main disadvantage is over-segmentation, which is when the algorithm divides the cells into too many smaller regions. To address this problem, I've used the function *imhmin* which suppresses the minimas that have a value less than *h*, for in this case, *h* is 3. In my approach, the combination of the Euclidean distance transform along with the *imhmin* function has helped me achieve better segmentation results.

Figure 2.2.7 : Closeup with imhmin

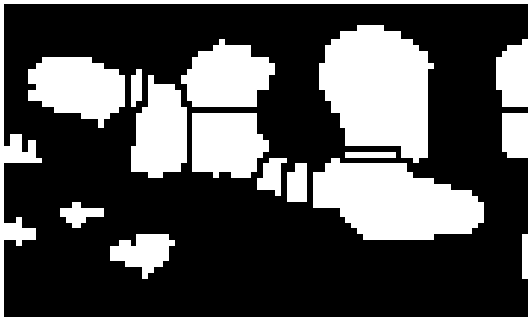

Figure 2.2.8 : Closeup without imhmin

The reasons for using *imhmin* is to prevent over segmentation and to also ensure the outline of the original object is properly depicted. Figure 2.2.7 and 2.2.8 show the binarised image after watershed with *imhmin* and without respectively. Figure 2.2.8 shows that a section of the cell is over segmented into multiple parts.

Figure 2.2.9 : h = 2

Figure 2.2.10 : h = 3

Figure 2.2.11 : h = 4

When choosing a value of h, it's important to note that a small value of h will result in the preservation of small, segmented regions while a larger value of h will result in the merging of

these smaller regions. I decided to use value 3 for h as an optimum value to prevent over segmentation or over merging after trying it out with other values such as 2 and 4.

2.3 Opening

I will then perform Opening in order to remove the small objects while preserving the larger objects. This would help remove too many small pixelated objects. Matlab's process to apply opening first consists of defining a structuring element *strel()* followed by *imopen*. The structuring element I have chosen is a “disk” shape of radius 2.

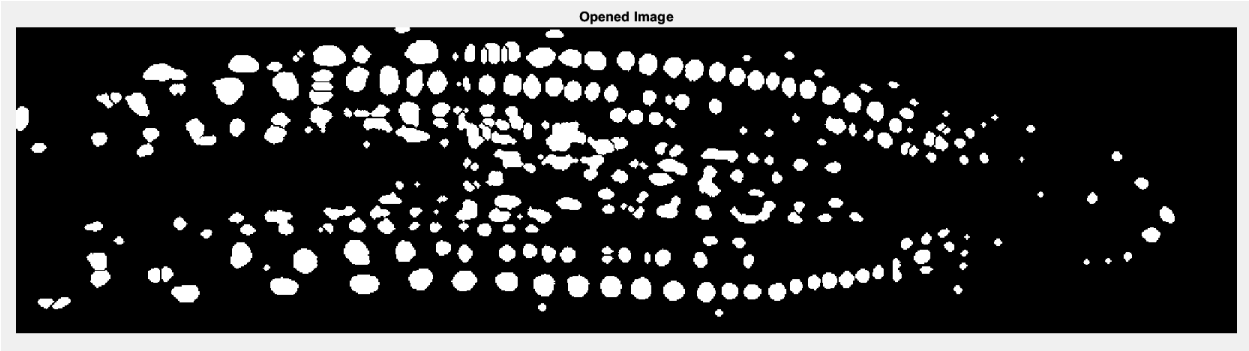

Figure 2.3.1 : Binarised Image of Segmented Image

Opening involves 2 morphological operations, erosion and dilation. It will first erode the image, removing small objects or thin lines. The resulting image then undergoes dilation, which will expand the remaining objects, smoothing out the edges while filling any gaps within the object.

I have selected a disk shaped structuring element of radius 2 as many of the cell nuclei are of spherical shape. A structuring element of the shape disk would expand in a circular symmetrical shape, preserving the roundness of the cell nuclei. The radius determines the size of the features removed from the image. A larger radius may remove some of the cell nuclei while a smaller radius may have negligible effects. After trying out structuring elements of radius 1, 2, and 3, I went with a radius of 2 as 1 had many left over small objects while 3 removed many of the smaller cells.

2.4 Random Colouring

The last step would be to randomly colour the cell nuclei. I would first use the function *bwlabel* to label the connected components in the image, producing a 2-D matrix. Once I have the matrix, I would use the *max* function to calculate the maximum label value, which is the total number of cell nuclei found in the image.

Using the *rand* and *label2rgb* function, I will generate a random colour for each cell from the three colour channels, resulting in the final image shown in Figure 2.4.1, 2.4.2, and 2.4.3

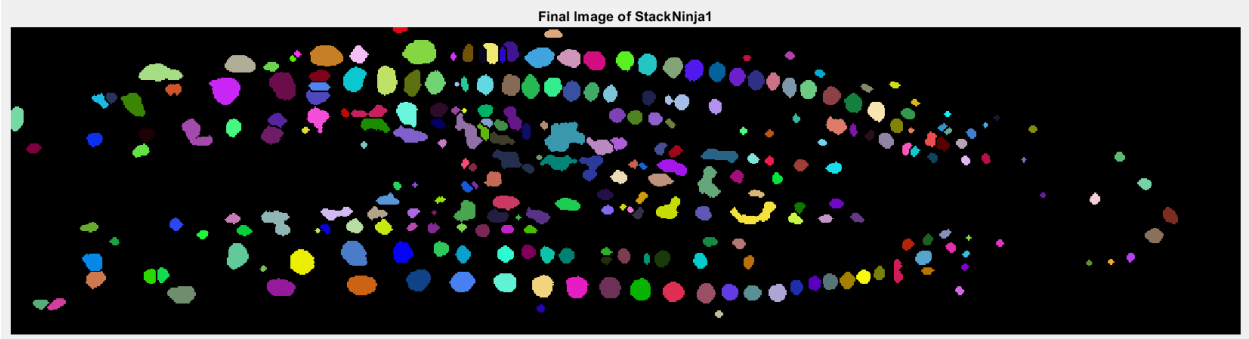

Figure 2.4.1 : Final Image with Randomly Coloured Cells of StackNinja1

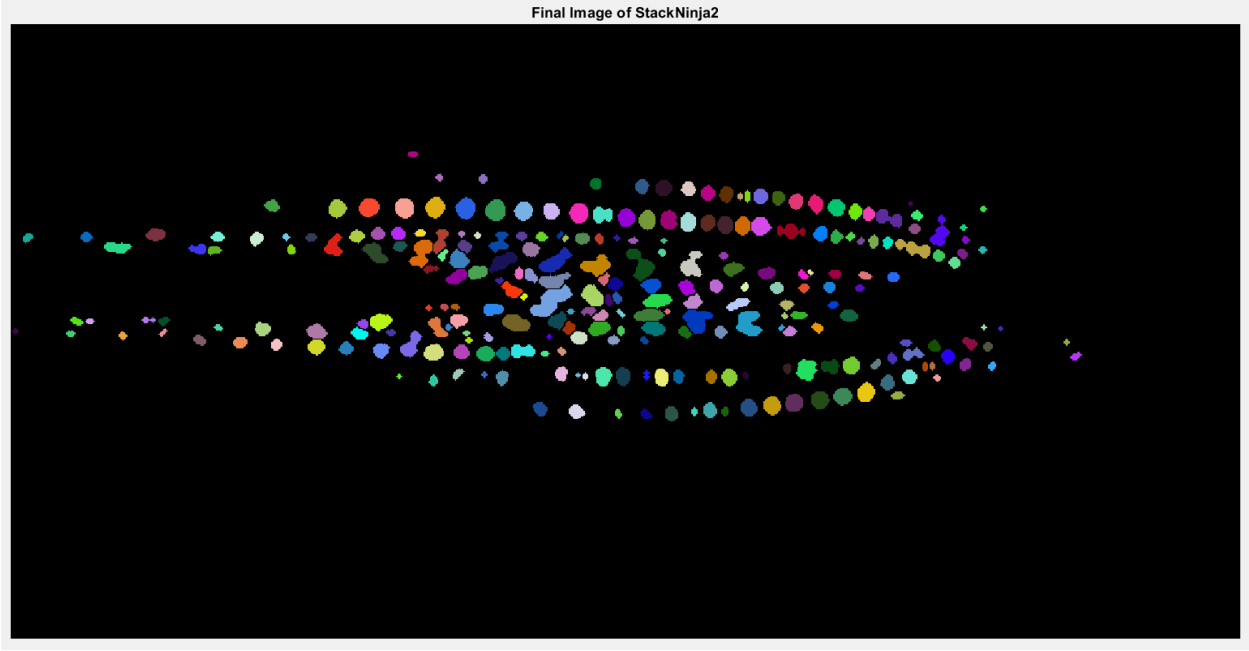

Figure 2.4.2 : Final Image with Randomly Coloured Cells of StackNinja2

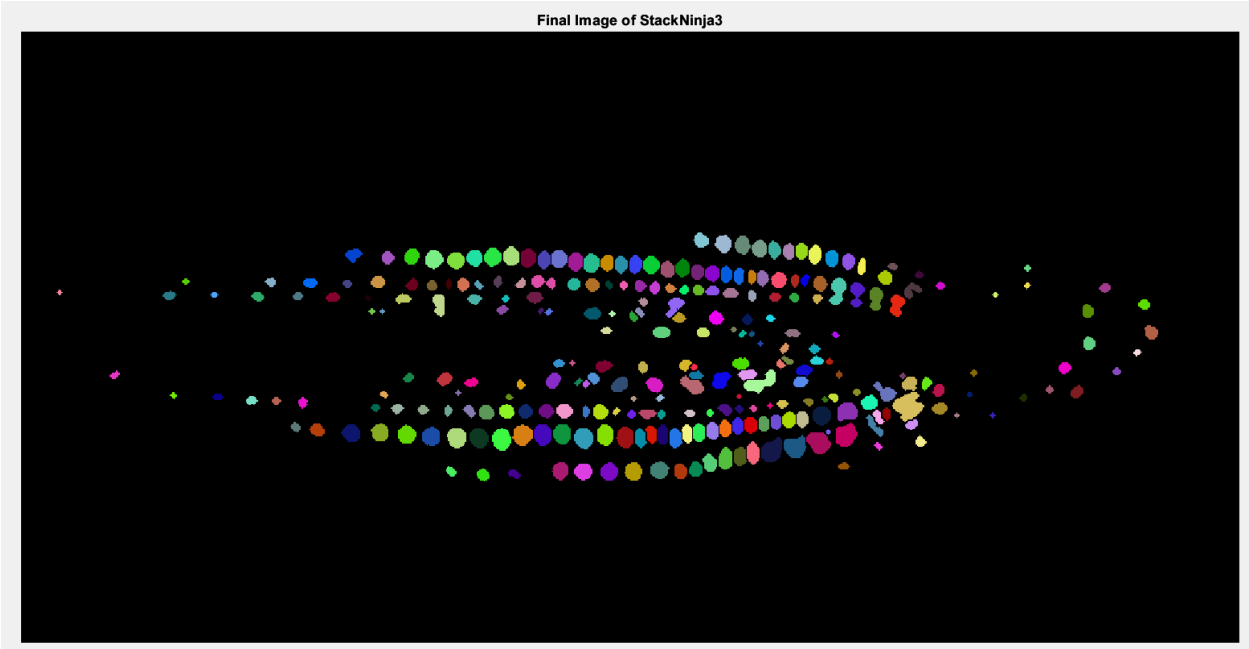

Figure 2.4.3 : Final Image with Randomly Coloured Cells of StackNinja3

3 References

1. Belaid, L.J, and Mourou, W (2011). “Image Segmentation: A Watershed Transformation Algorithm.” *Image Analysis & Stereology*, vol. 28, no. 2, p. 93., https://doi.org/10.5566/ias.v28.p93-102
2. Bressan, M, (2007), et al."Local contrast enhancement," Proc. SPIE 6493, Color Imaging XII: Processing, Hardcopy, and Applications, 64930Y; https://doi.org/10.1117/12.724721
3. Grevera, G. (2007) “Distance Transform Algorithms and Their Implementation and Evaluation.” *Deformable Models*, pp. 33–60., https://doi.org/10.1007/978-0-387-68413-0_2
4. Gupta, G. (2011). Algorithm for image processing using improved median filter and comparison of mean, median and improved median filter. International Journal of Soft Computing and Engineering (IJSCE), 1(5), 304-311
5. Hamarneh, G, and Xiaoxing, L. (2009) “Watershed Segmentation Using Prior Shape and Appearance Knowledge.” *Image and Vision Computing*, vol. 27, no. 1-2,, pp. 59–68., https://doi.org/10.1016/j.imavis.2006.10.009
6. Kim, H (2017), et al. “Comparative Analysis of Image Binarization Methods for Crack Identification in Concrete Structures.” *Cement and Concrete Research*, vol. 99, pp. 53–61., https://doi.org/10.1016/j.cemconres.2017.04.018
7. Palus, H. (1998). Representations of colour images in different colour spaces. In: Sangwine, S.J., Horne, R.E.N. (eds) The Colour Image Processing Handbook. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-5779-1_4
8. Roy, P (2014), et al. “Adaptive Thresholding: A Comparative Study.” *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, https://doi.org/10.1109/iccicct.2014.6993140