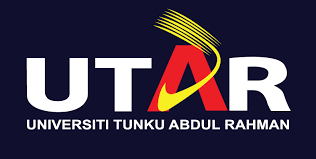
****

**Faculty of Green Energy and Technology**

**BACHELOR OF TECHNOLOGY (HONOURS) IN ELECTRONIC SYSTEMS**

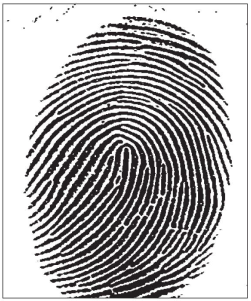
**Work Based Learning - Machine Vision *I***

**Assignment 1 Image Segmentation**

1. **Anand Low Hong Ren 1801371**

**Introduction**

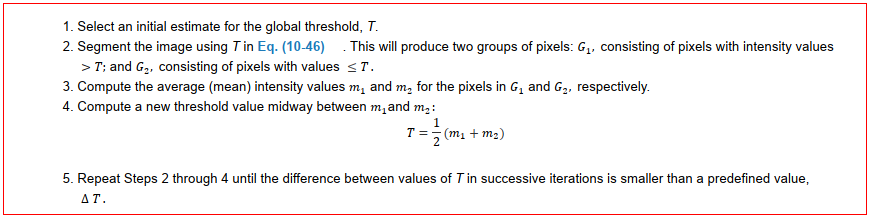
Image Segmentation in digital image processing and computer vision is defined as the process of partitioning a digital image into multiple segments or sets of pixels. The goal is to simplify or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is usually used to identify objects and boundaries such as edges, lines, curves in images. This is done by assigning a label to every pixel in an image such that pixels with the same label share certain characteristics ( Gonzalez & Woods, 2018) . In Vitrox, image segmentation is used to define more clearly the object and the background, so that image processing can be performed easily on the image. Thresholding is the simplest method of image segmentation. The basic idea is that it will turn a gray-scale image into a binary image. This is done by selecting a threshold value, any pixel whose value is above the threshold value is set to a binary 1, while any whose value is equal or below the threshold value is set to a binary zero.



**Figure 1 Image of fingerprint before and after basic global thresholding**

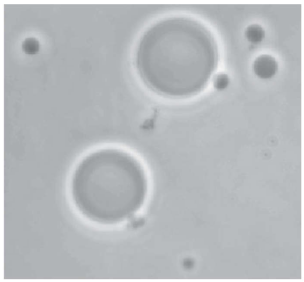
**Methodology**

The most basic thresholding is global thresholding. This simple method is best applied when the intensity distributions of the objects and background pixels are sufficiently distinct, it is possible to use one single threshold for the whole image. This is done by selecting a threshold value, any pixel whose value is above the threshold value is set to a binary 1, while any whose value is equal or below the threshold value is set to a binary zero. An iterative algorithm is used to estimate the threshold value for the image. The cons is that it is easily affected by noise, or unevenly distributed lighting which will produce an output which is undesirable ( Gonzalez & Woods, 2018).

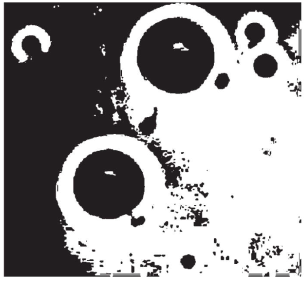


**Figure 2: iterative algorithm for basic global thresholding**

Another alternative to basic global thresholding is Otsu’s Method. This method is optimum as it maximizes the between-class variance. The basic idea is that properly thresholded classes should be distinct with respect to the intensity values of their pixels and a threshold that gives the best separation between classes in terms of their intensity values would be the best threshold. Otsu’s method computations are performed on the histogram of an image, which is an easily obtainable 1-D array ( Gonzalez & Woods, 2018). The algorithm for Otsu’s method can be referred here <http://www.ijircce.com/upload/2017/june/164_21_Otsu.pdf>. The results of Otsu’s method can be seen below.



(a) (b)

****

(c)

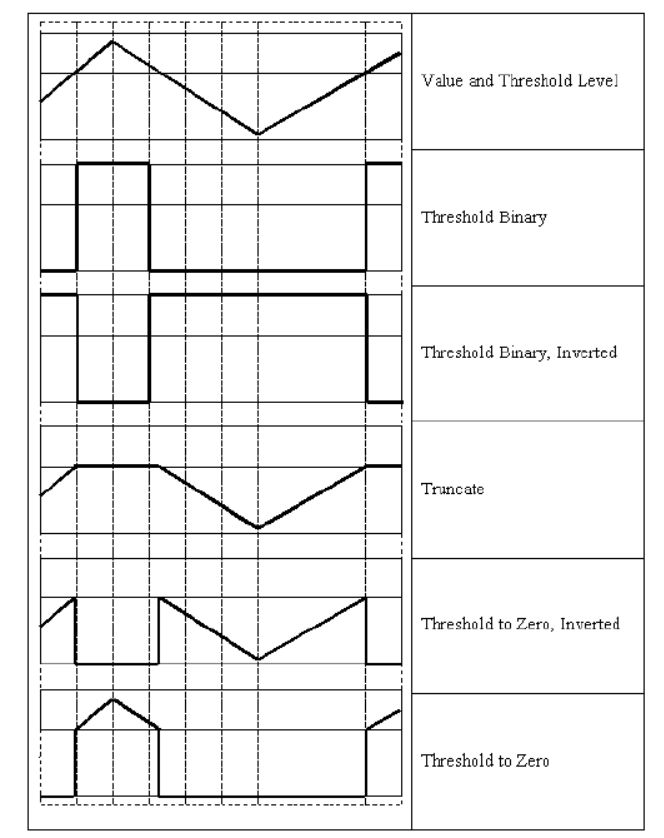
**Figure 3: (a) Optical microscope image of polymersome cells**

**(b) Results using Otsu’s method**

**(c) Results using basic global algorithm**

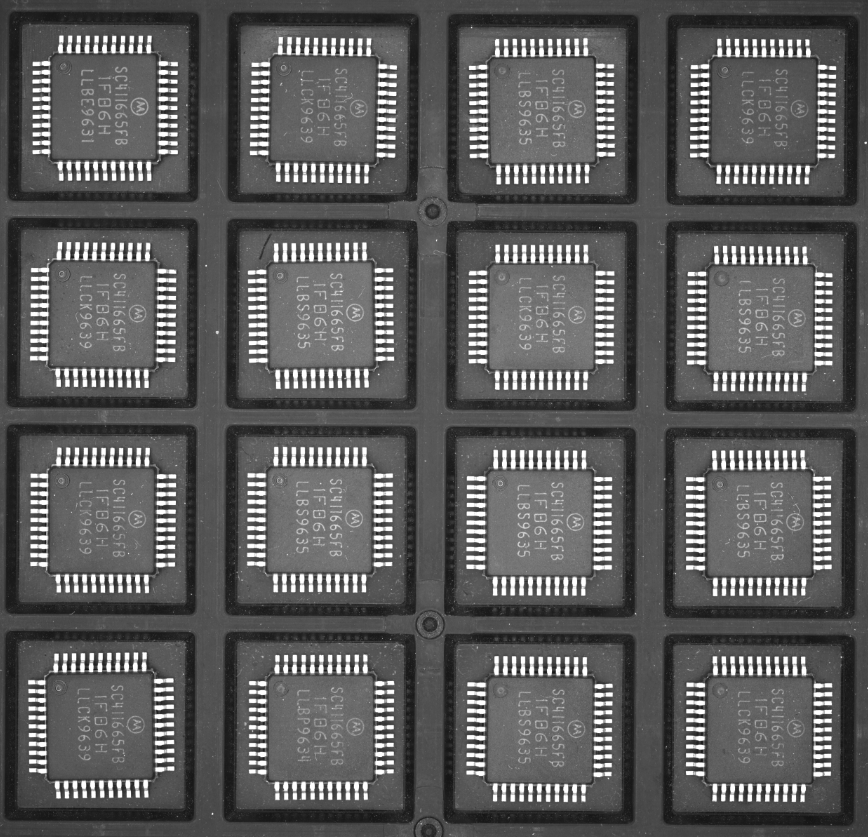
As we can see in figure 3, the basic global threshold algorithm failed to achieve the desired segmentation, while the results obtained using Otsu’s method is superior to the basic global thresholding.

In OpenCV, there are other global threshold algorithms included in the threshold function such as truncate, threshold to zero and Otsu’s thresholding algorithm. As we can see in Figure 5 below, the first diagram shows the grey value of the pixel of the original image and the threshold level. For threshold binary (basic global thresholding) it is as explained in the introduction. For truncate, any value above the threshold is modified till it equals the threshold value. For threshold to zero, any value below the threshold value is reduced to a binary 0. The three types are inverted to show a different kind of output. Adaptive thresholding algorithms are also available. Adaptive thresholding works by compute a threshold value at every point, (x, y), in the image based on one or more specified properties in a neighborhood of (x, y). Although this may seem like a laborious process, modern algorithms and hardware allow for fast neighborhood processing, especially for common functions such as logical and arithmetic operations. Thus, an input threshold value is not required for the adaptive threshold function.



**Figure 4: Different types of threshold operations**

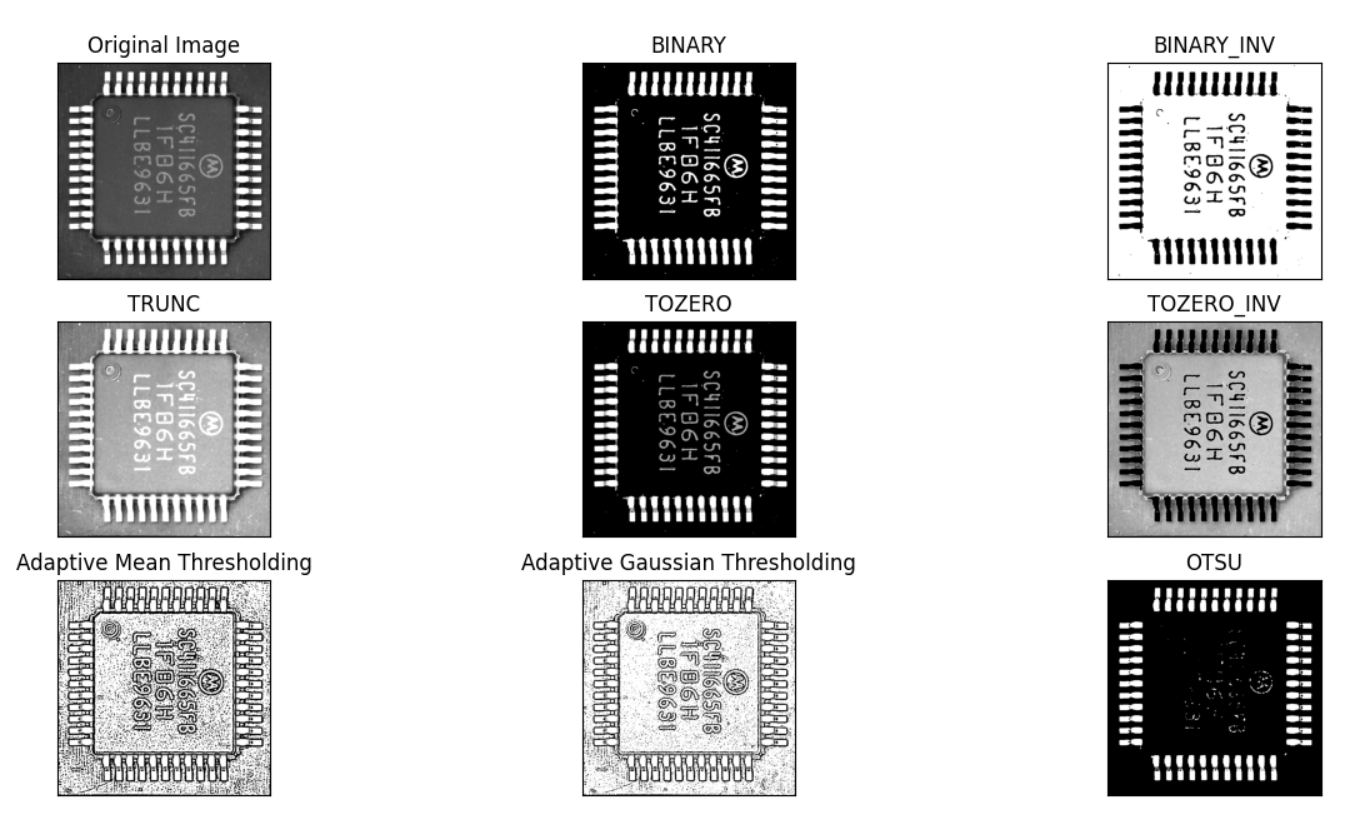
Since the image (Figure 4) has 16 chips, a nested for loop is used to crop into the region of interest for each of the 16 chips. After cropping, the function image\_result() will insert the cropped image into the openCV threshold function to apply thresholding to the image. The image is then displayed using the matplotlib library. Each of the processed images are observed. Median blurring of kernel size 5 is applied to reduce some of the dust in the image. Threshold value, v of 127 is set for every global threshold algorithm.



**Figure 5: The image to be processed**

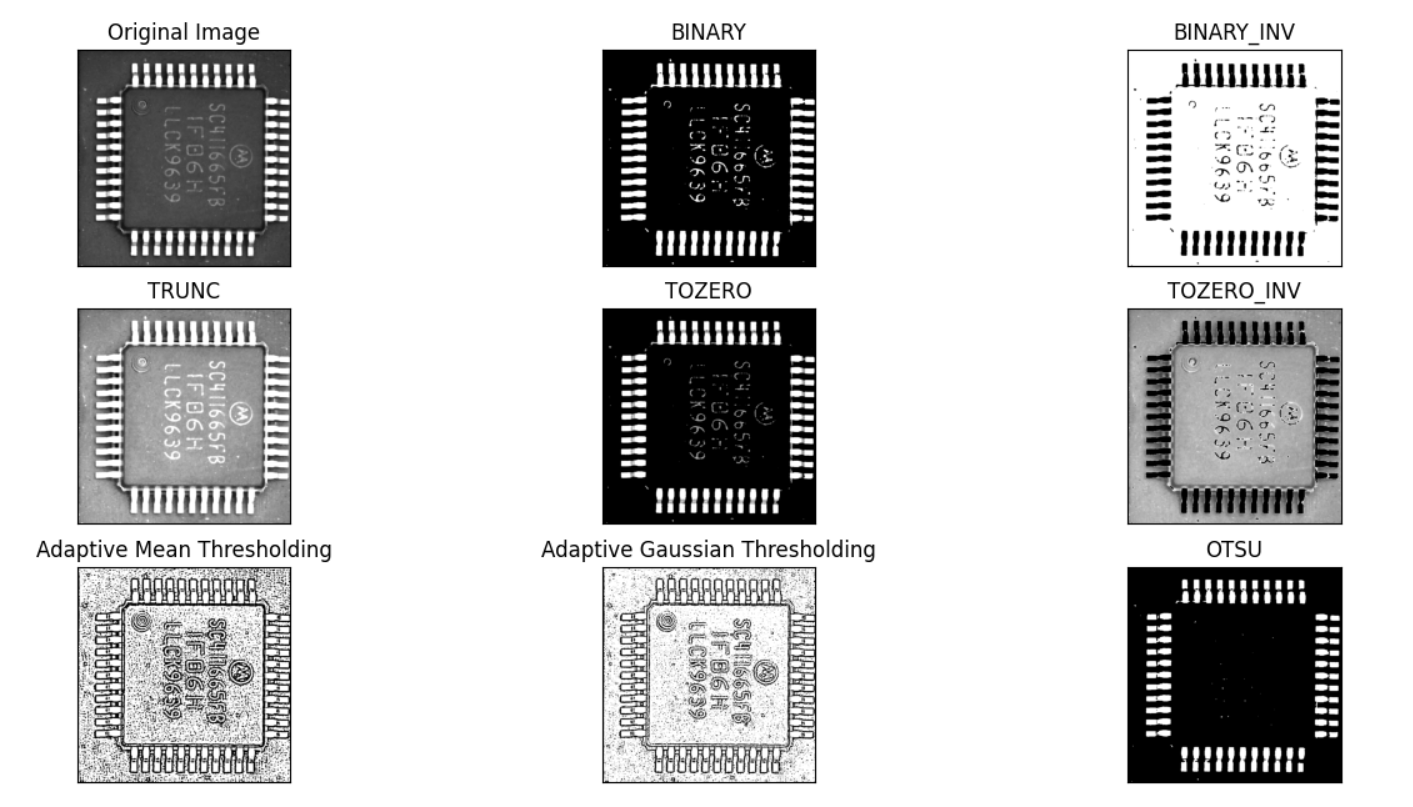
**Results and Discussion**

Threshold value, v = 127, median blur with kernel size 5



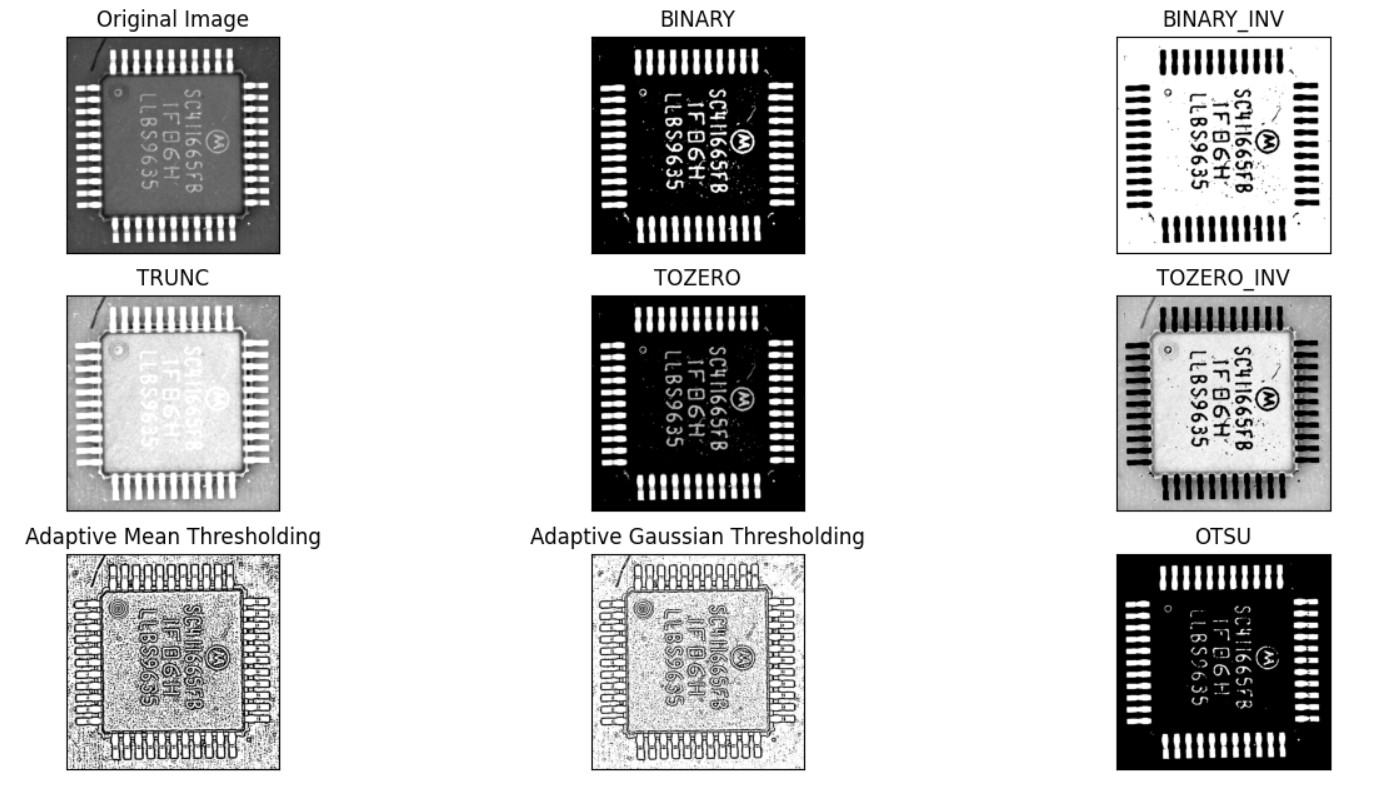
**Figure 6: 1st row 1st column chip applied with different threshold algorithm (v =127)**

The results of the first chip are quite good. The results of binary, binary inverse, truncate, threshold to zero and threshold to zero produce very clear leads and characters. These algorithms would be best for this chip. The adaptive mean thresholding and adaptive gaussian thresholding have clear edges around the lead, characters and the chip itself, but have some noise which makes the chip not so distinct with the background. The Otsu method results in clear leads, but is unable to show the characters on the chip, making it unsuitable.



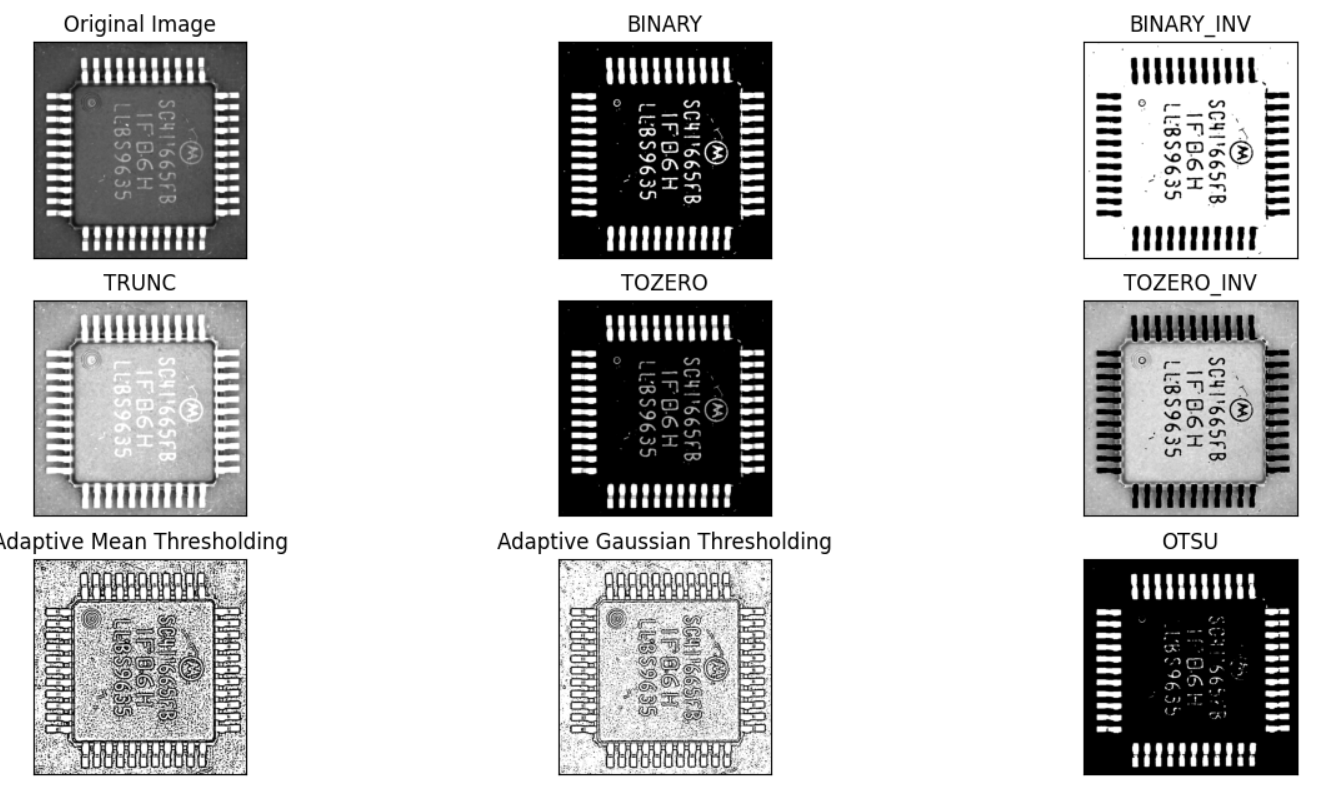
**Figure 7: 1st row 4th column chip results ( v= 127, characters segmenting error)**

The global thresholding algorithms with the exception of the truncate threshold algorithm, produce leads that are very clear to identify, however there is a problem segmenting the characters on the chip. This may be due to the dimmer light shone on the chip compared to other chips. The character reflects less light which makes some part of the character lower than the threshold value which causes some part of the character to be black instead of white. The Otsu threshold algorithm totally cannot differentiate the character from the chip, showing total blackness on the chip. Otsu’s threshold method has been unsuccessful so far. Adaptive threshold algorithms do a great job of finding the edges of the characters and the leads, so they can be identified.

****

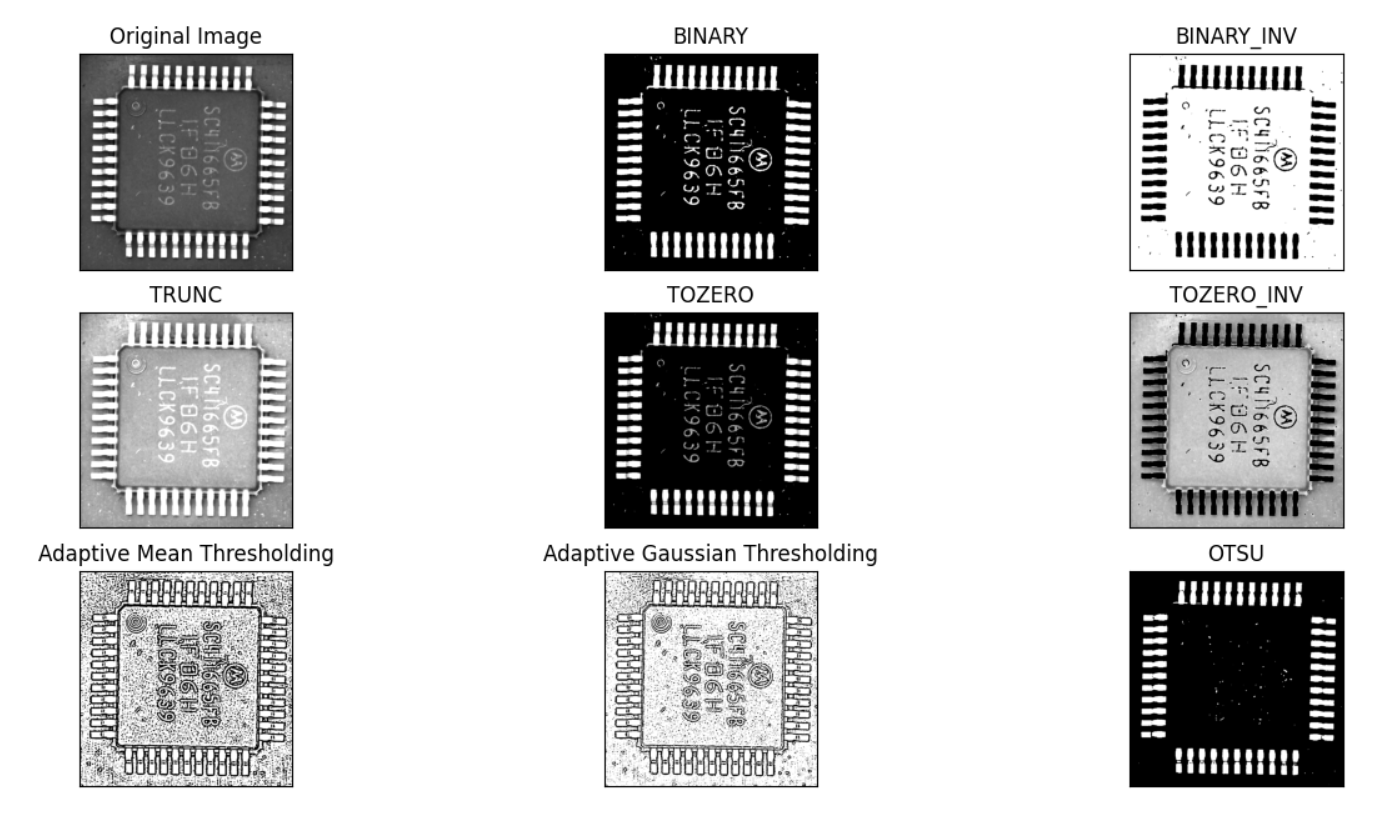
**Figure 8: 2nd row 2nd column chip results ( v = 127, lighting)**

The result to note here is the truncate threshold algorithm. The chip itself is very bright, which makes it hard to identify the characters. This is due to the chip shone by brighter light. However the other algorithms work well except for the Otsu’s method where the character is hardly visible. On the flip side, binary global thresholding works very well in brighter light.



**Figure 9: 3rd row 4th column chip results (v = 127, defect in character)**

The binary global thresholding algorithm works best here. We are able to see the defect in the characters.The 1 in the characters was printed only halfway. The defect can be easily detected and the chip will be rejected.

****

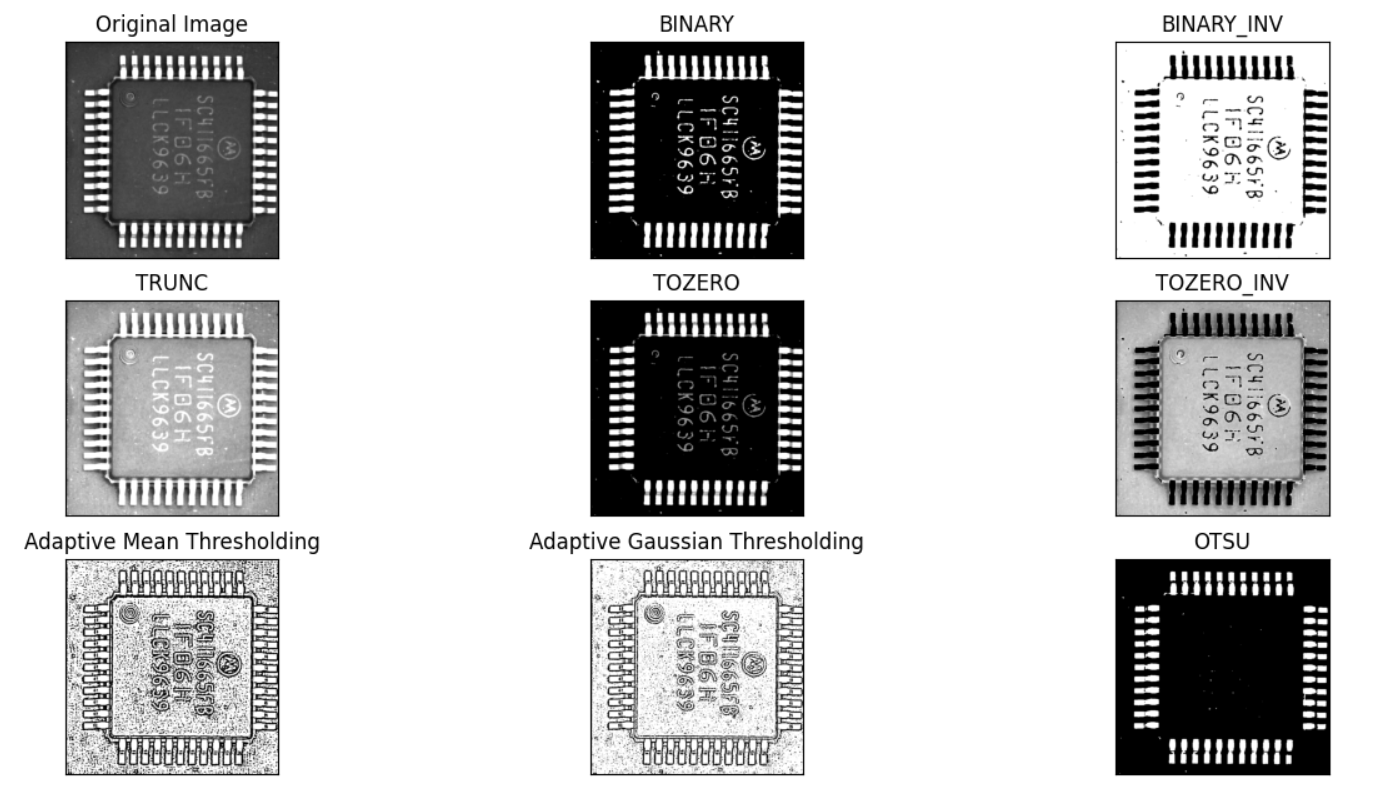
**Figure 10: 1st row 2nd column chip results (*v* = 127, dust)**

As we can see in the original image, the characters reflect some of the light, resulting in a dark greyish colour compared to Figure 5, 7 and 8 where the characters reflect more light and appear to be brighter. The dust on the chip is almost the same colour as the characters, so the thresholding algorithm also changes the value of the dust to binary 0. Thus, the dust will appear as if it is part of the characters, which will affect the readability of the characters. In Figure 9, we can see that the dust is rendered as part of the character between the 1s in the characters. This will make it hard for the later image processing to recognise it as two 1s. The characters itself have no problems but due to this dust the chip will be categorised as a defect and will be rejected.

Overall for threshold value,*v* = 127 and median blur of kernel size 5, the best algorithm will be binary as it shows the clearest characters and leads, and it has a high percentage of success. The only con for this algorithm is if the characters are shone by dimmer light, it has a hard time segmenting the characters from the chip. Threshold to zero is the same as the binary, just that the bright parts such as characters and leads retain their original value, while the rest below threshold would be set to black. The leads are very reflective, so they appear white, however for characters who are less reflective, they appear grey instead of white like binary. Threshold to zero inverse algorithm is almost the same with a binary inverse algorithm but it provides more depth with it’s different grey values instead of just black and white. Truncate threshold algorithm would be my second best algorithm because it works well in dim or average lighting conditions as well. It’s only drawback is that it can be hard to segment a brightly lit image. Thus, I would use truncate for dim or average lighting conditions, while I would use binary for bright conditions.

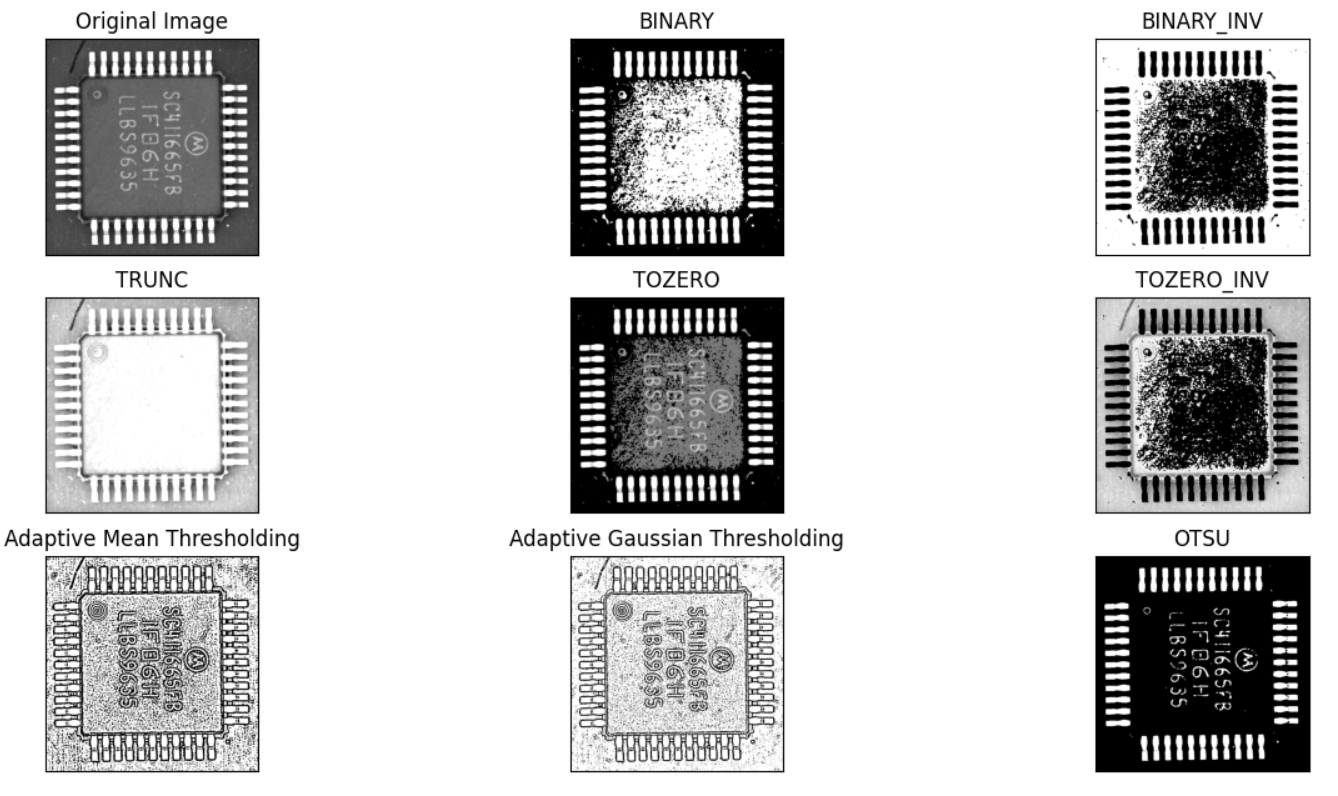
Since the chip on the 1st row 4th column has problems segmenting the characters, I tried different threshold values, *v* to see whether there is an ideal threshold value to allow all the characters to be rendered.

Threshold value, v = 115, median blur with kernel size of 5



**Figure 11: 1st row 4th column chip results (v= 115)**

The character after binary thresholding can be seen more clearly, but the part with not so clear print is not shown after thresholding. It can be seen that it is not the algorithm problem, but is the threshold value set which affects the output image.

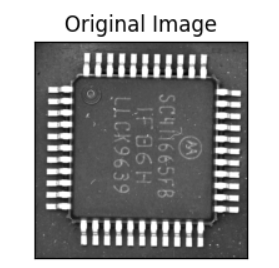


**Figure 12: 2nd row 2nd column chip results (v= 115)**

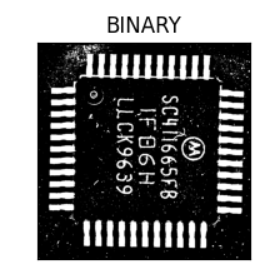
When the threshold value, v is set to 115 it is too low for some of the chips. This problem happens when the light is brighter and the chip is brighter. Thus, this threshold is too low for brighter lit chips.

After trying several threshold values it is impossible to find the right balance where all the images can be segmented properly. If the threshold is set too high the characters cannot be segmented properly. If the threshold is set too low, the brighter lit images will have problems segmenting the characters. Overall, the best threshold value is 127 as it segments the characters and leads properly.

Another factor that can affect the quality of the output image is the noise of the image. There is not much noise in the image but there is a lot of dust which affects the segmentation of the image. To counter this error, image smoothing or image blurring is used. There are 4 types of blur function in OpenCV. These are averaging, gaussian blur, median blur and bilateral filtering.

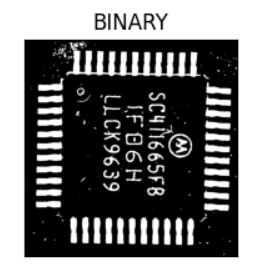


**Figure 13: Original image 1st row 2nd column**

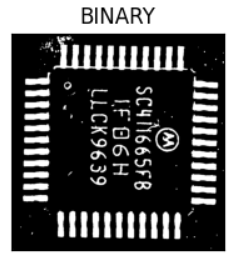
****

**Figure 14: Binary image 1st row 2nd column (no blur)**

As we can see here without any blur, the dust on the tray and on the chip reflects the light, it affects the segmentation, the output does not look so clean.

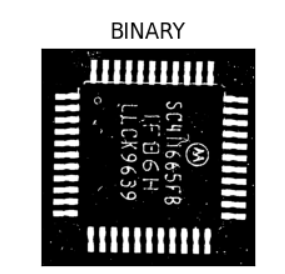


**Figure 15: Binary Image 1st row 2nd column (after median blur with kernel size 5)**

****

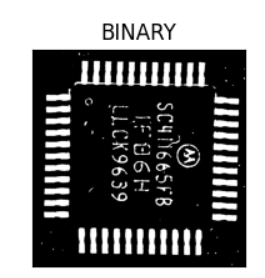
**Figure 16: Binary Image 1st row 2nd column (after median blur with kernel size 7)**

As we can see here when applied median blur the dust size decreases, making the overall output image cleaner. If the kernel size is increased, the dust size decreases as a result, and some dust is even removed. However the kernel size cannot be set too large as the edge will be blurred and becomes not accurate.

’

**Figure 17:**  **Binary Image 1st row 2nd column (after Gaussian blur)**

Comparing Gaussian blur to median blur, it removes dust that is clumped up together or larger in size, however it does not remove the smaller salt and pepper like dust effectively.



**Figure 18:**  **Binary Image 1st row 2nd column (after Bilateral filtering)**

Bilateral filtering is the best solution among all the small and large dust that are all removed.

**Conclusion**

The image given is lit very evenly with little discrepancy. This makes global thresholding work very well. However a slightly dim or bright image affects the quality of the output of the thresholding. A suggestion to improve the result is to develop an algorithm which is able to determine the most ideal threshold value for the image based on the intensity level of the image. Overall, I will use truncate for all conditions and binary for bright conditions. Adaptive thresholding makes the edges very distinct which is useful for some applications. All in all the best algorithm for this image is a binary global threshold algorithm with bilateral filtering.

Reference

1. Gonzalez, R. C. & Woods, R. E. (2018). *Digital Image Processing (4th edition)*. Hudson Street, New York: Pearson
2. OpenCV. (n.d.). Miscellaneous Image Transformations. Retrieved from <https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html>
3. OpenCV. (n.d.). Smoothing Images. Retrieved from <https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html>
4. OpenCV. (n.d.). Image Thresholding. Retrieved from <https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html>