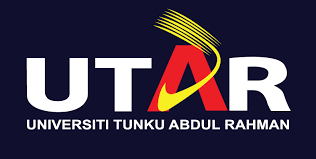
****

**Faculty of Green Energy and Technology**

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

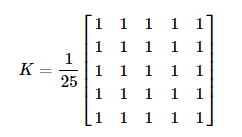
**Work Based Learning - Machine Vision I**

**Assignment 2 Edge Blob Detection**

1. **Anand Low Hong Ren 1801371**

**Introduction**

The objective of this assignment is to find the edges of the leads of the chip, and to calculate the length, width and the distance between the leads. For this assignment, thresholding, image smoothing or blurring, edge detection and other methods are used. Image Smoothing is mainly used to reduce noise within an image. Image smoothing is a key technology of image enhancement, which can remove noise in images. So, it is a necessary functional module in various image-processing software. Smoothing can be performed by spatial or frequency filters. Spatial filtering is the filtering operations that are performed directly on the pixels of an image. The process consists of moving the filter mask from point to point in an image. Smoothing filters are used for noise reduction and blurring operations. It takes into account the pixels surrounding it in order to make a determination of a more accurate version of this pixel. By taking neighboring pixels into consideration, noisy pixels can be filtered out. However some fine details such as edges may be lost due to the smoothing process. In OpenCV, a kernel, which is a 2D array of various sizes, is used to calculate the new value for every pixel in a picture.



**Figure 1: a kernel with size 5x5**

The kernel will be kept above a pixel, and it will add all the 25 pixels around the central pixel, take the average and replace the central pixel with the new average value. This operation is continued for all the pixels in the image. In OpenCV there are 4 types of image smoothing, which are the basic averaging, Gaussian blurring, median blurring and bilateral filtering (OpenCV, n.d.).

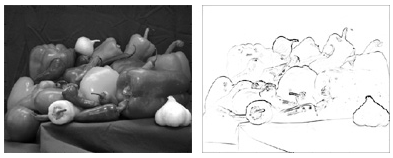
Other than Image Smoothing, edge detection is also a key part of this assignment. Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness or gray values. Common edge detection algorithms include Sobel, Canny, Prewit, Roberts, and fuzzy logic methods.



**Figure 2: Edge detection using Sobel’s method**



**Figure 3: Edge detection using Canny’s method**



**Figure 4: Edge detection using Fuzzy Logic method**

In OpenCV, there is only the Canny edge detection method. It is a 5 stage algorithm that is based on 5 objectives. Low error rate, edge points should be well localized, and the detector should return only one point for each true edge point. The first stage is noise reduction by using a 5x5 Gaussian filter. After that, the algorithm will find the intensity gradient of the image. The image is filtered with a Sobel kernel in both horizontal and vertical direction to get the first derivative in horizontal and vertical direction. After getting gradient magnitude and direction, a full scan of the image is done to remove any unwanted pixels which may not constitute the edge. Every pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Basically, you will get a binary image with thin edges. After that comes the last stage which is hysteresis thresholding. This stage decides where all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so removed. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also removed. This stage also removes small pixels noises on the assumption that edges are long lines.

**Methodology**

First, the image is read as a grayscale image. After that the image is cropped into the region of interest which is the chip. After that, the image is blurred and is applied with a binary threshold to segment the leads and characters clearly from the background. After that, findContours() method is used to find all the leads, characters, noise which are white in colour. After that, it calculates the area of each blob and erases blobs which are smaller than the area of 150, which are most likely to be noise and dust. After that, the program calls the function thresh\_callback() which finds all the remaining blobs, which are the leads, and bound them within rectangles. The rectangle’s height and width are calculated to find the approximate height and width of the leads. However, the top part of the lead( connected to the chip) is higher than the bottom part of the lead ( the part where it connects to the board). Thus the bottom part is either slightly to the left or to the right of the top part. If we measure the whole lead, the width measured does not equals to the width of the top and bottom part. To solve this problem, the middle part of the lead is sliced so that the top and bottom part of the lead can be measured separately. This gives a more accurate reading of the width of the lead. The most important aim of this assignment is to calculate the width and length of the bottom part of the lead because that is the part connecting to the board. Any defects there would affect the connection between the chip and the board so that is very important to check it. Another important thing to check is the pitch between leads, or the distance between the two leads. This is also important because if the two leads touch each other, this will cause a short circuit which will spoil the chip. After separating the top and bottom part, a Canny Edge filter is applied to highlight all the edges. The **cv2.findContours()** method finds all the contours and draws it on an empty image. However, there are some contours that are not the leads themselves, or multiple contours for a single lead. Another problem faced is that the method finds contours randomly, with no specific order. To solve that problem, I created a function **Sort\_Contour()** which sorts the contour from left to right After that, **cv2.boundingRect()** will return 4 parameters in a list, which are x-coordinate, y-coordinate, width and height to the **oldboundRect** variable. The small contours are filtered out by calculating the area of the rectangle formed, and a threshold is set to filter out rectangles with areas too small to be defined as lead. Since there are some leads with multiple contours, this will create two or more rectangles for one lead. To solve this, if the 4 parameters of the rectangle returned by the **cv2.boundingRect()** is the same as the previous rectangle, it means that it is repetitive, so this will be filtered out as well. The remaining parameters for the 22 leads will be inserted into **boundRect**. The contour will be drawn by the **cv2.drawContours()** method, while the rectangles will be drawn by the **cv2.rectangle()** method. This will happen for all of the 16 chips. The height and width of the leads will be shown, and the pitch will be calculated by subtracting the x-coordinate of the current rectangle with the x-coordinate plus width of the previous rectangle. The units of the length are all counted in pixels. This image has a scale of 16 microns per pixel. The program is able to check all 4 sides of the chip’s leads.



**Figure 5: Brief rundown of the program**

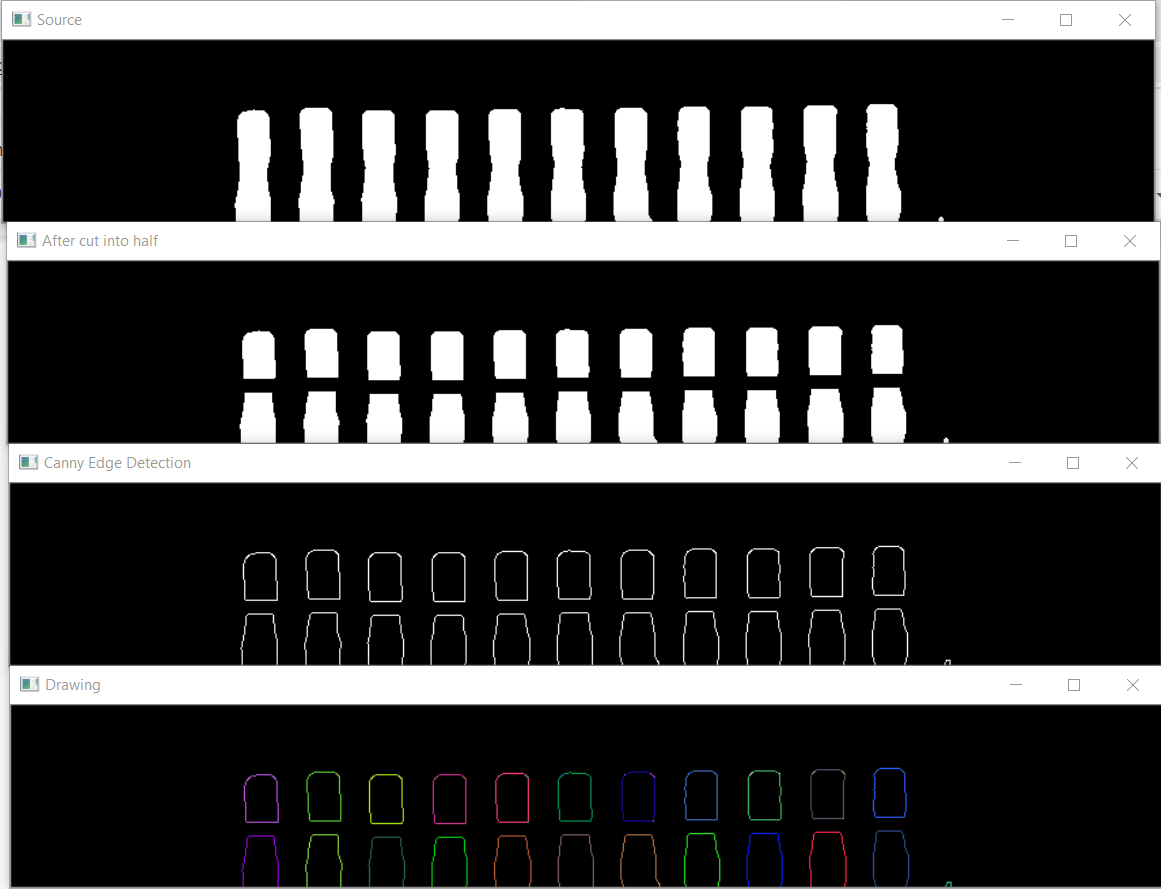


**Figure 6: The lead**

****

**Figure 7: pitch calculation**

**Results and Discussion**

****

****

**Figure 7:**

1. **Original image after binary thresholding and removing noise**
2. **After cutting the lead into half**
3. **After applying Canny Edge Detector**
4. **After drawing the contour**
5. **Final Image after bounding the Rectangle to the contour**

****

**Figure 8: The lead’s are number like this**

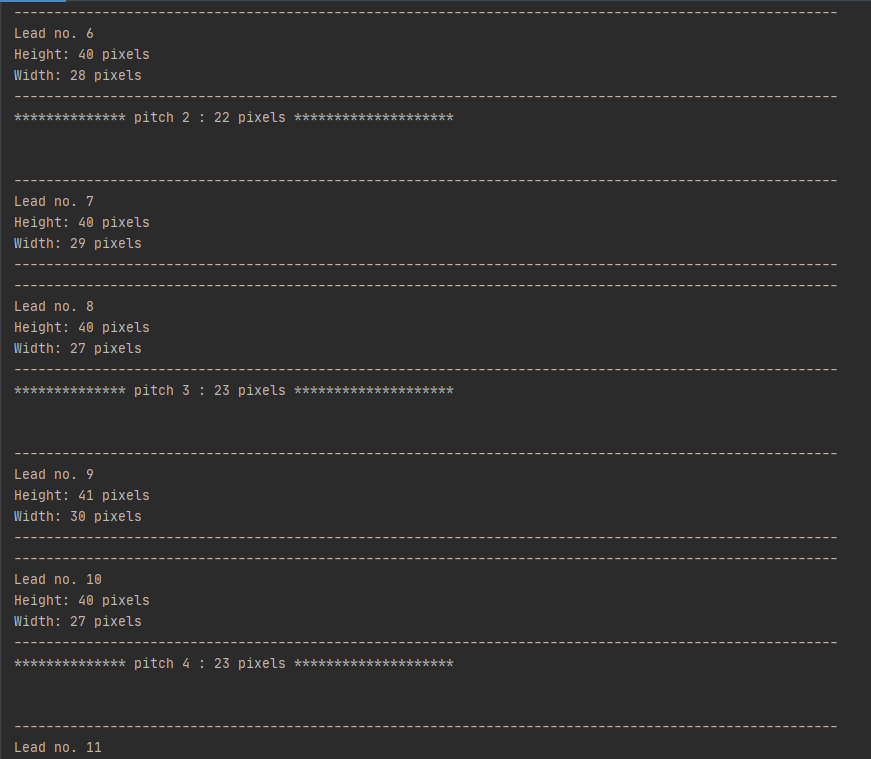
The contours have various ways of sorting it for example, top-to-bottom, bottom-to-top, left-to-right and right-to-left. However after testing all of them, left-to-right provides the most consistent results, thus it is being used. The number goes like a zig zag pattern, with odd numbers in the bottom row and even numbers in the top row. This way is done for top and bottom. For left and right side of the chip, it uses a top-to-bottom way to count the leads.

****

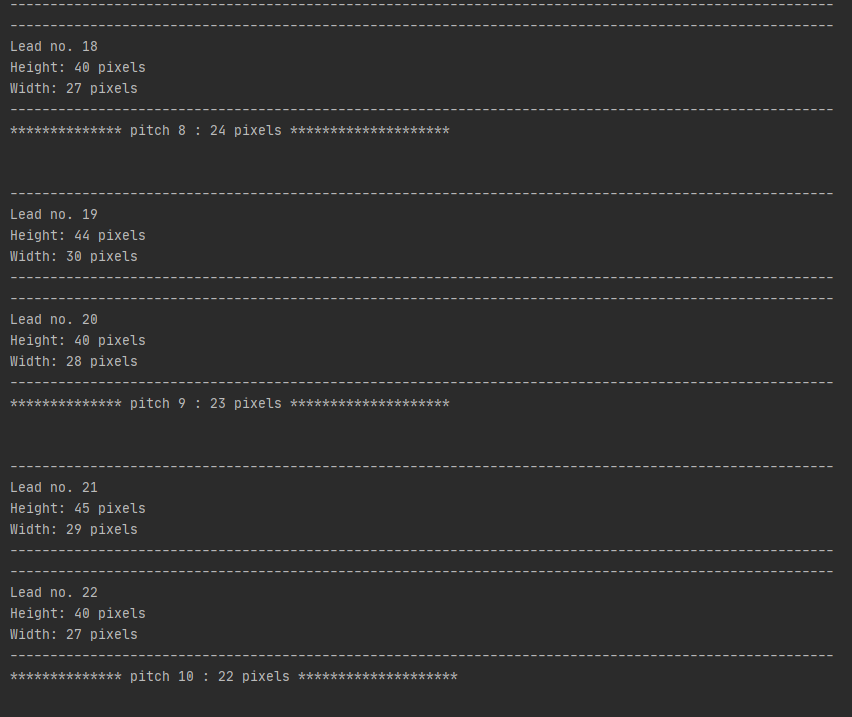
**Figure 8: The chip’s row and column is shown, and the lead no. is shown with the**

**Height and width**

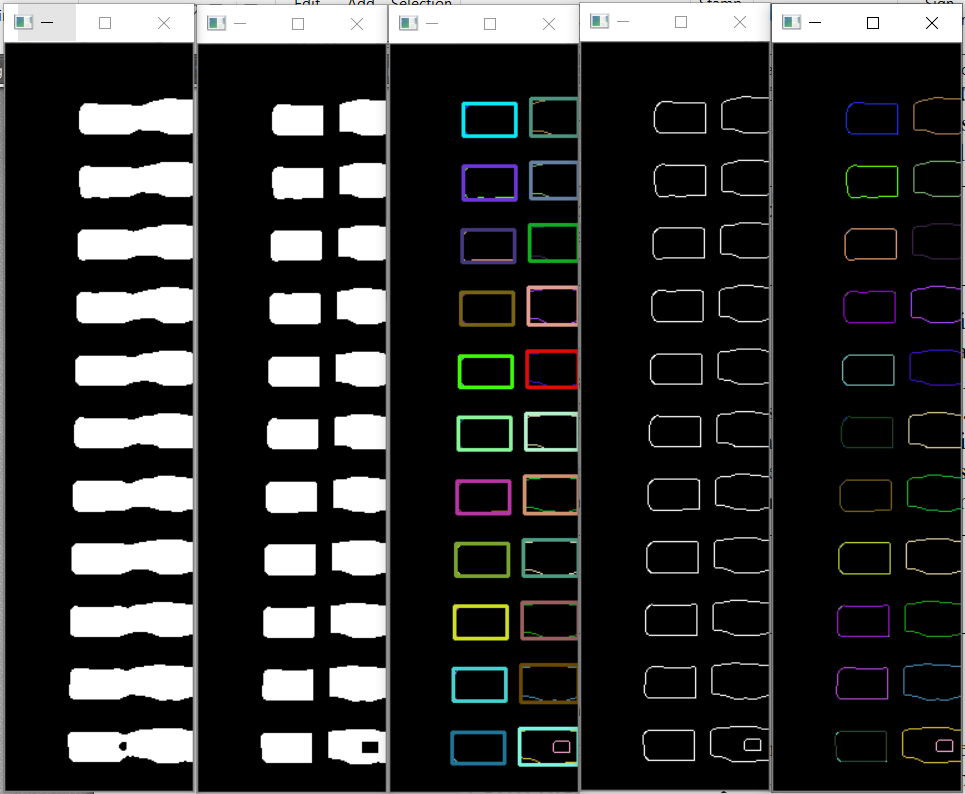
**The pitch is counted starting from the 4th lead, which is the top row 2nd lead from the left**

****

**Figure 9: Continued till 22nd lead**

****

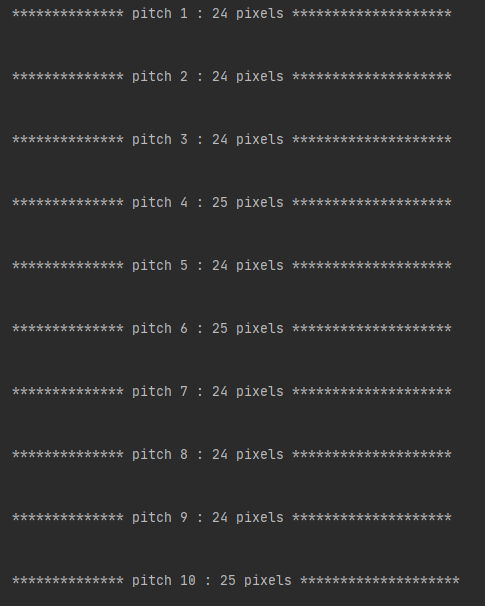
**Figure 10: Top row 11 leads and bottom row 11 leads add up to 22 leads. There are a total of 10 pitches in between the 11 leads.**

****

**Figure 9: Left side of B4 Chip**

****

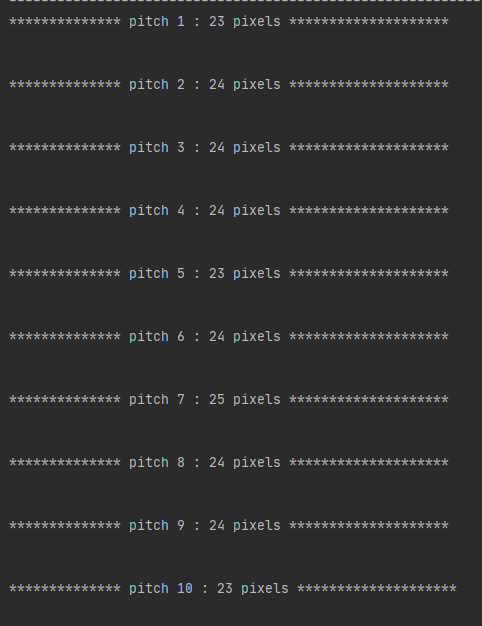
**Figure 10: Lead is labelled like this for left side of B4 Chip**



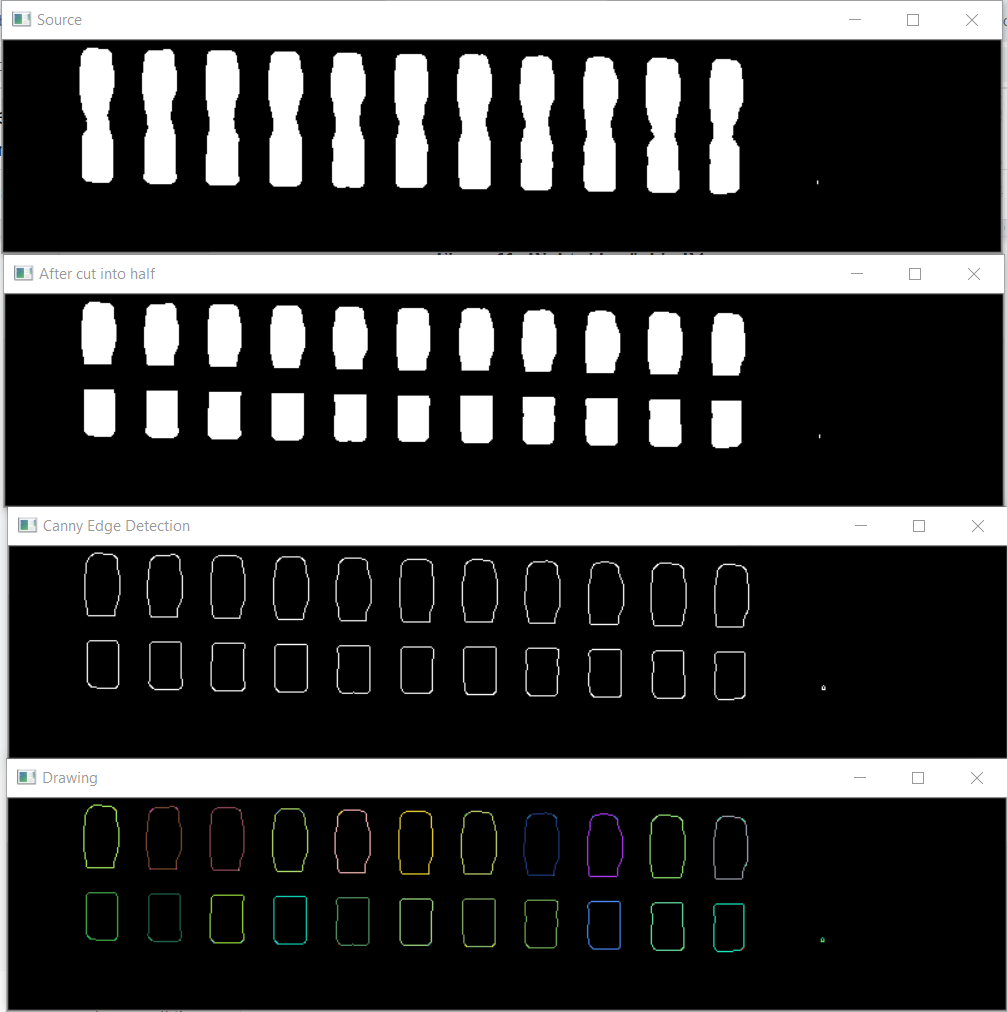
**Figure 10: The pitch of the left side leads (B4 chip)**

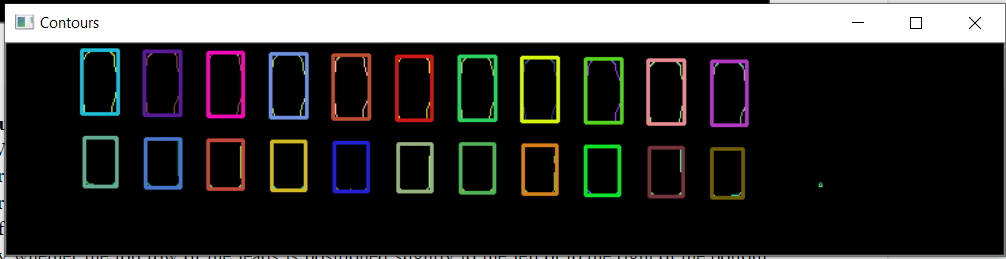
****

**Figure 11: Right side of chip B4**

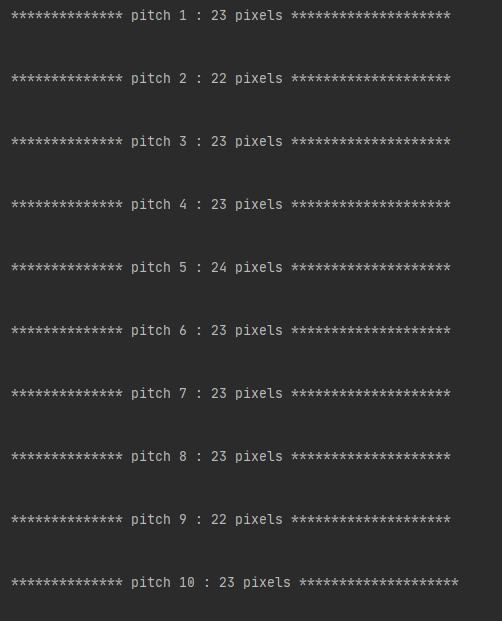
****

**Figure 12: Pitch of right side B4 chip**

****

****

**Figure 13: Bottom side of C2 chip**

****

**Figure 14: Pitch of bottom side C2 chip**

****

**Figure 15: Error in measuring the actual width of chip**

**Suggestion and Improvements**

With these algorithms, there are many bugs that have to be debugged before it can work properly. Even if the algorithm can work fine, it is not robust. If the chip is placed slightly above or below, this will affect the cropped image, and the lead may be cropped away. If large portions of the lead is cropped away, the measurement will be affected. Another problem is that when the background near the lead is not threshold away, it will cause error in measurement of width (Figure 15). The lead labelling is also affected by whether the top row of the leads is positioned slightly to the left or to the right of the bottom row. My algorithm assumes that the top row is positioned slightly to the right of the bottom row lead, which will lead to wrong labelling for other conditions. This will also affect the pitch calculations as it is done also based on this assumption.

**Conclusion**

Overall, the algorithm gives accurate pitch, height and width when the image is ideal. Pitch ranges from 23 pixels to 26 pixels in length. However, the algorithm will still face some errors calculating the width if there is some blob extending out of the lead.

**Reference**

1. OpenCV. (n.d.). Contour Features. Retrieved from <https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html>
2. OpenCV. (n.d.). Canny Edge Detection. Retrieved from <https://docs.opencv.org/master/da/d22/tutorial_py_canny.html>
3. OpenCV. (n.d.). Creating Bounding boxes and circles for contours. Retrieved from https://docs.opencv.org/3.4/da/d0c/tutorial\_bounding\_rects\_circles.html