Radwan duadu
14342606

School of Electronic Engineering - Dublin City University

Module ee453 PROJECT (2.5 ECTS)

TASKS

a) Using grey scale morphological reconstruction techniques develop a robust, fully automated, data driven Matlab program that will automatically isolate the alpha numerical characters from the calculator image 'calculator.tif'. Your solution should remove all shading and reflection artefacts from the image. [7/25]

Plan:

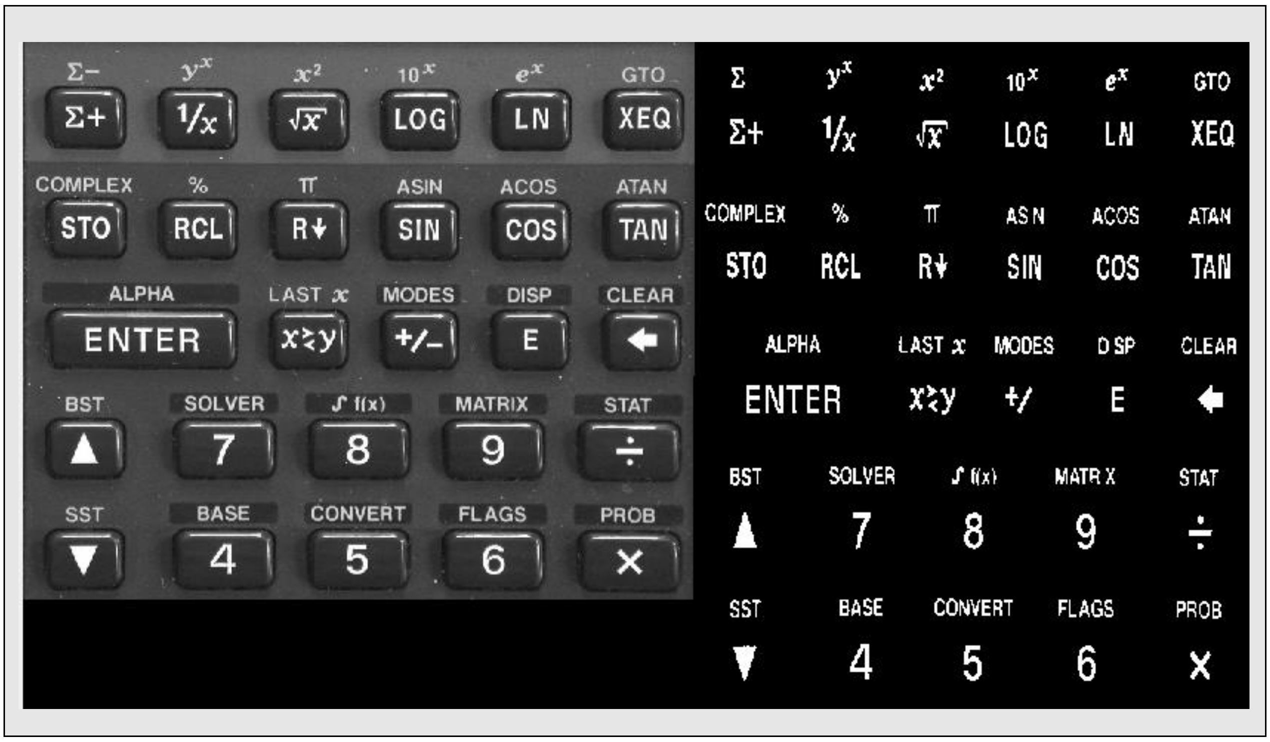
1. The first procedure is to do some initial filtering to convert the data into a simpler for example binary to make manipulating the data faster and easier.
2. The second step is to find a way to remove all the noise (the random data that is not an alpha numerical character example horizontal lines and random points).
3. Representation of output and analysis of result.

pseudo-code:

1. Load in the image and convert to greyscale.
2. Use a function to predefine the scale of every image, this is done so the structuring elements used later in the code will not need to alter with each image.
3. Use histogram to automatically select a data driven image threshold. As we have good contrast between the alpha numerical characters and the background this can be done by simple examination of the histogram range and finding a midpoint across this range.
4. The vsg method Open is used this method will first erode then dilate the image this acts as filter removing a lot of the random bits of data and shrinks the vertical and horizontal lines that represent the border of each button making it easier to remove them in the following steps.
5. Errored the vertical using image processing method **imerode**, this method takes in a structuring element for the vertical lines a square structuring element is used to errored anything that is less than 4 pixels vertical. A higher value is not selected because letters like **I** would start being eroded.
6. The Horizontal lines removal was completed using a line structing element using a big value in this case the value 10 is used because it is assumed that anything that is horizontally long won't be considered a letter.
7. In both cases after the images were eroded they were then added back into the image i.e. The image was reconstructed this will remove them from the image and only leave the information that we require as the output.
8. Result is then seen, and issues are analysed.

Radwan duadu
14342606

Result:



Σ− y^x x² 10^x e^x GTO
Σ+ 1/x √x LOG LN XEQ
COMPLEX % π ASIN ACOS ATAN
STO RCL R↓ SIN COS TAN
ALPHA LAST x MODES DISP CLEAR
ENTER x≷y +/− E ←
BST SOLVER ∫f(x) MATRIX STAT
▲ 7 8 9 ÷
SST BASE CONVERT FLAGS PROB
▼ 4 5 6 ×

Table 1 - Result input vs output

As can be seen above the result is missing some characters, These characters are for example the I as the I is so small it had to be removed during the erosion operation a workaround to try and keep the I was not possible because if the I remained then small bits of the button borders remain, this can be seen in the input image as well some of the border reflection is very clear where as the I in the word matrix is very hard to make out making it very hard trying to get around the problem.

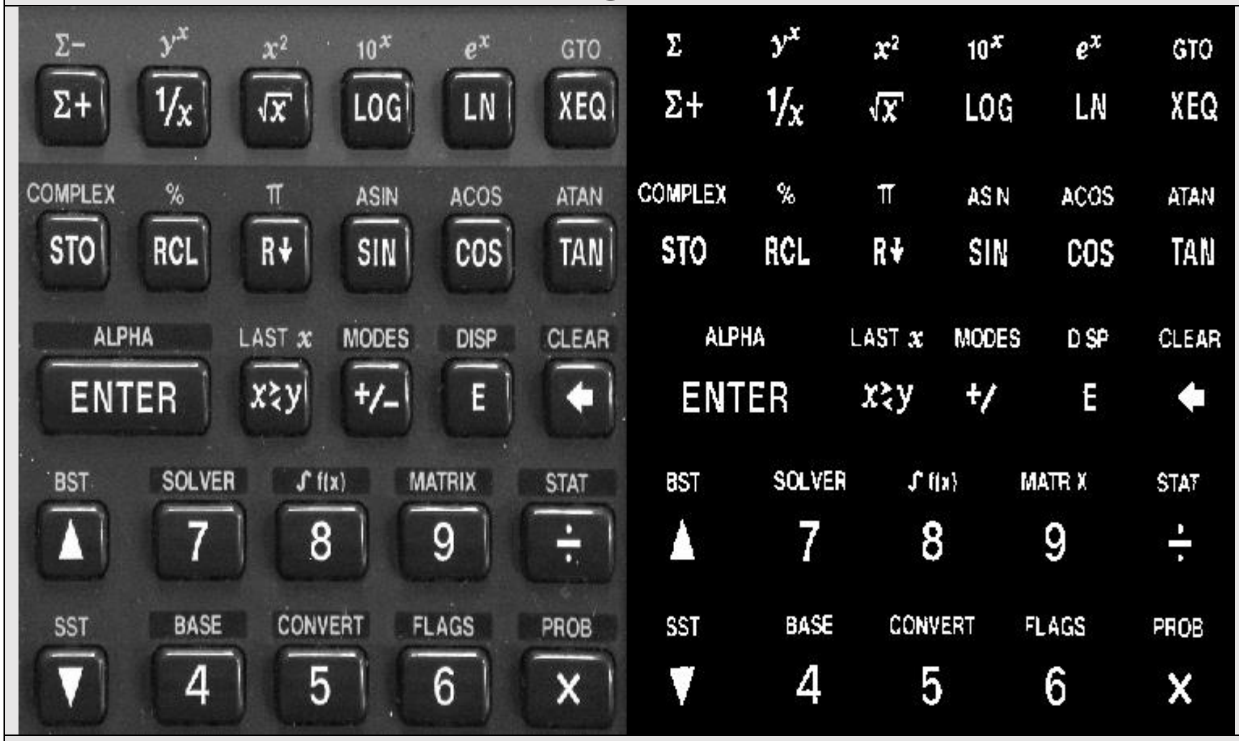
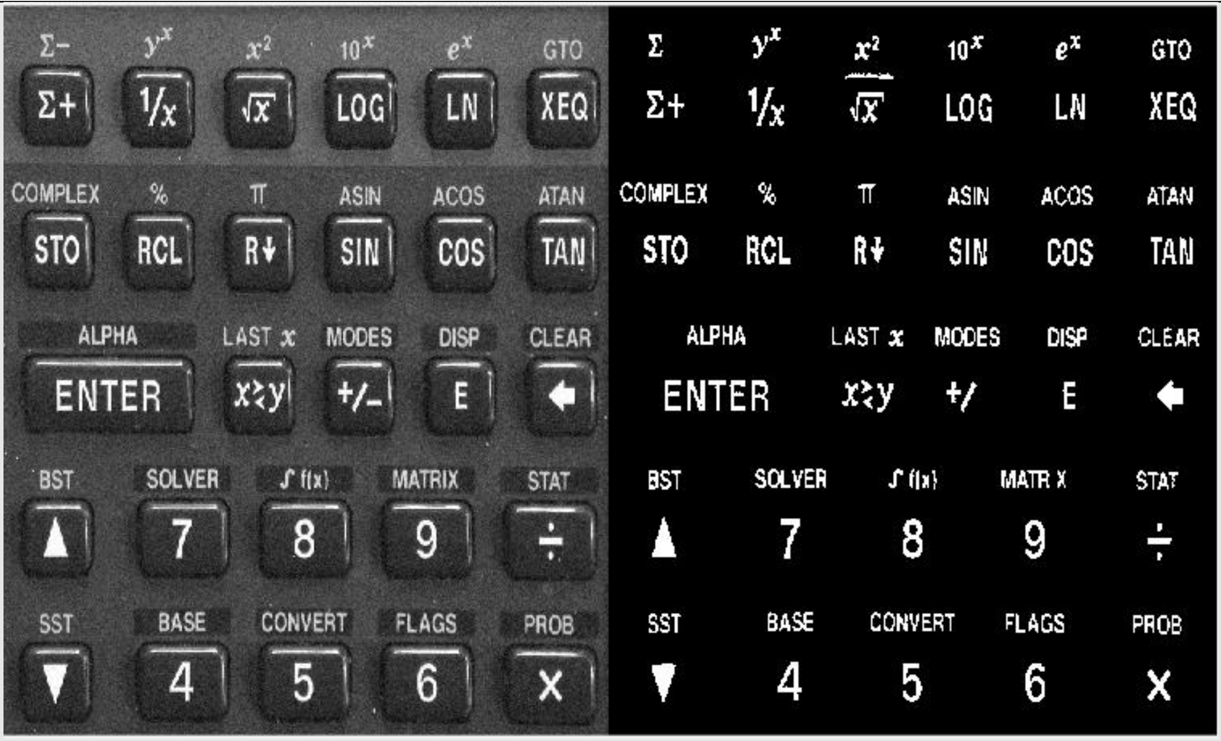
It also can be seen that the minus (**-**) sign in the images are removed this is also again due to erosion as they are too small and some of the button border horizontal lines are very thick and clear making it very hard to remove them without influencing the minus sign image.

b) Test your solutions robustness to both noise and scale variations. [4/25]

Testing noise:

The noise robustness for the solution is not very strong as the solution will result in a few horizontal or vertical lines appearing after the noise is increased past 3% when using the **vsg GuassianNoise** function, although this may not be desirable it is still functioning correctly. It remains working until the noise is increased past 33%. After this value is passed the letters start to fade making it hard to make out the letters and believe this is an accurate representation of the breaking point of the code.

Below are a few images at the different noise ranges.

Gaussian Noise @ 0-3% below at 3%

Gaussian Noise @ 4-33% in this case 10% and 30%

30%

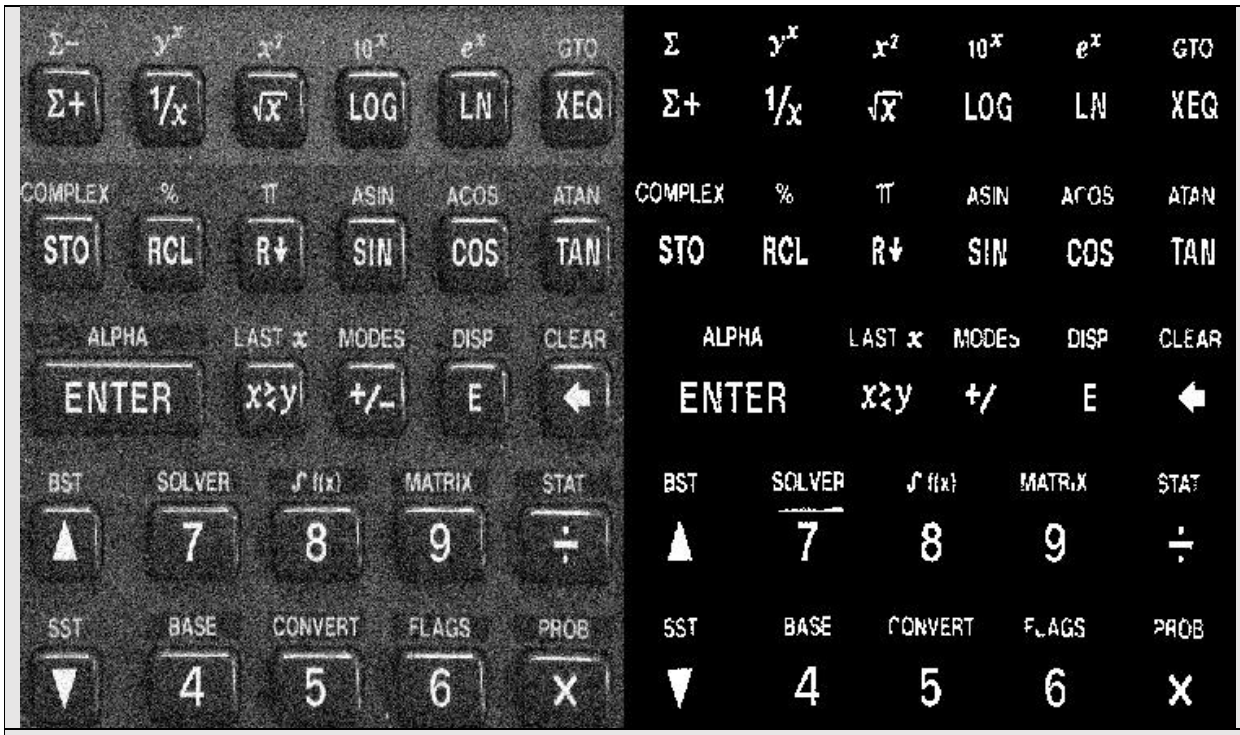
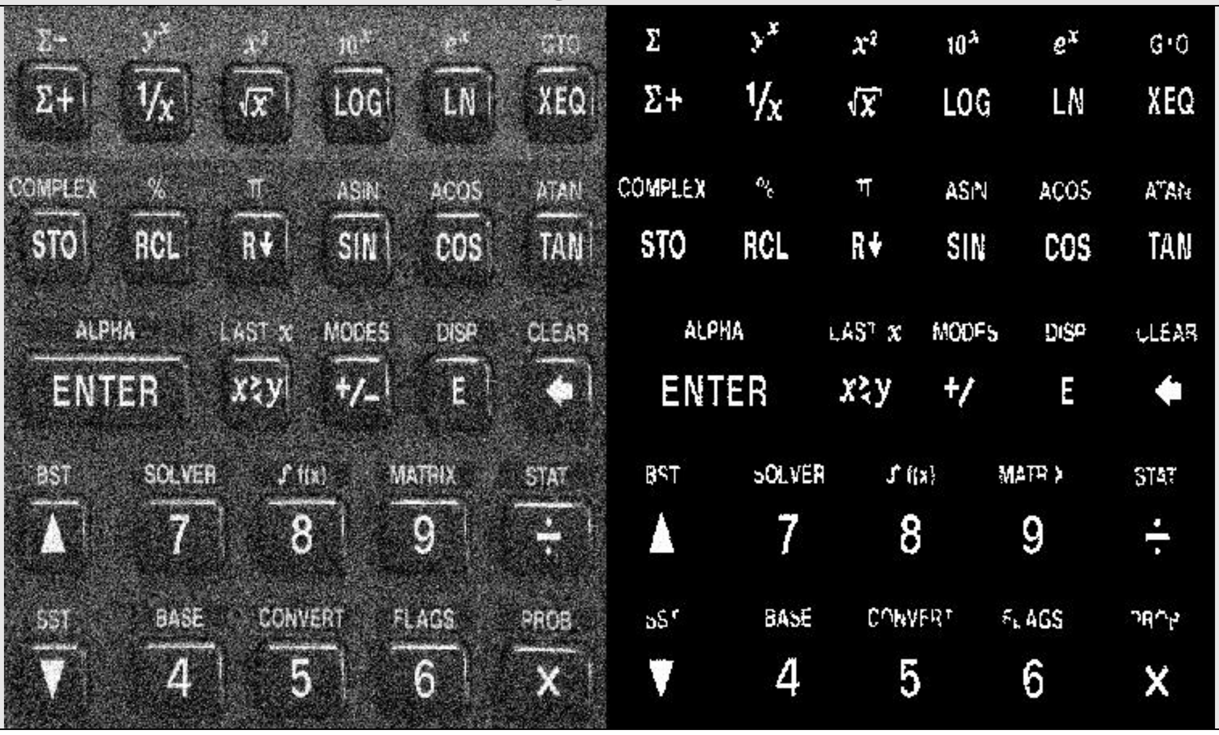

Gaussian Noise @ 33-100% below at 40%


Table 2- Noise Tests

As can be seen from the result even though after 3% the horizontal borders reappear but what also occurs is also that the vertical lines used to represent the I is returned therefore the result is improved and decreased at the same time it depends if the priority is that every letter shows up clearly or if the output must get rid of all the background with the exception of the letters and shapes.

Scale variation:

Since the code was designed to always readjust the scale of an image to a predefined scale it would not matter if the image is scaled up or down as the solution itself automatically sets the dimensions for the image. Using the vsg **ScaleImg** function.

Therefore, the result of scaling will not influence the result, it can be said that the solution is immune to scale variations.

c) Apply Otsu thresholding to both your final reconstructed image and the original image. Undertake a quantitative comparison of the resultant images? [2/25]

This was done in the second section of the code and can be found in the matlab code below under part C.

The Otsu thresholding was applied using the **multithresh** method provided by Matlab which will apply an Otsu thresholding effect to the images.

In the code some of the code was repeated as the original code using reconstruction to reconstruct only the data that is required therefore the original greyscale image is not included and that is why in the second part the greyscale image was included to make it possible to compare the two images accurately.

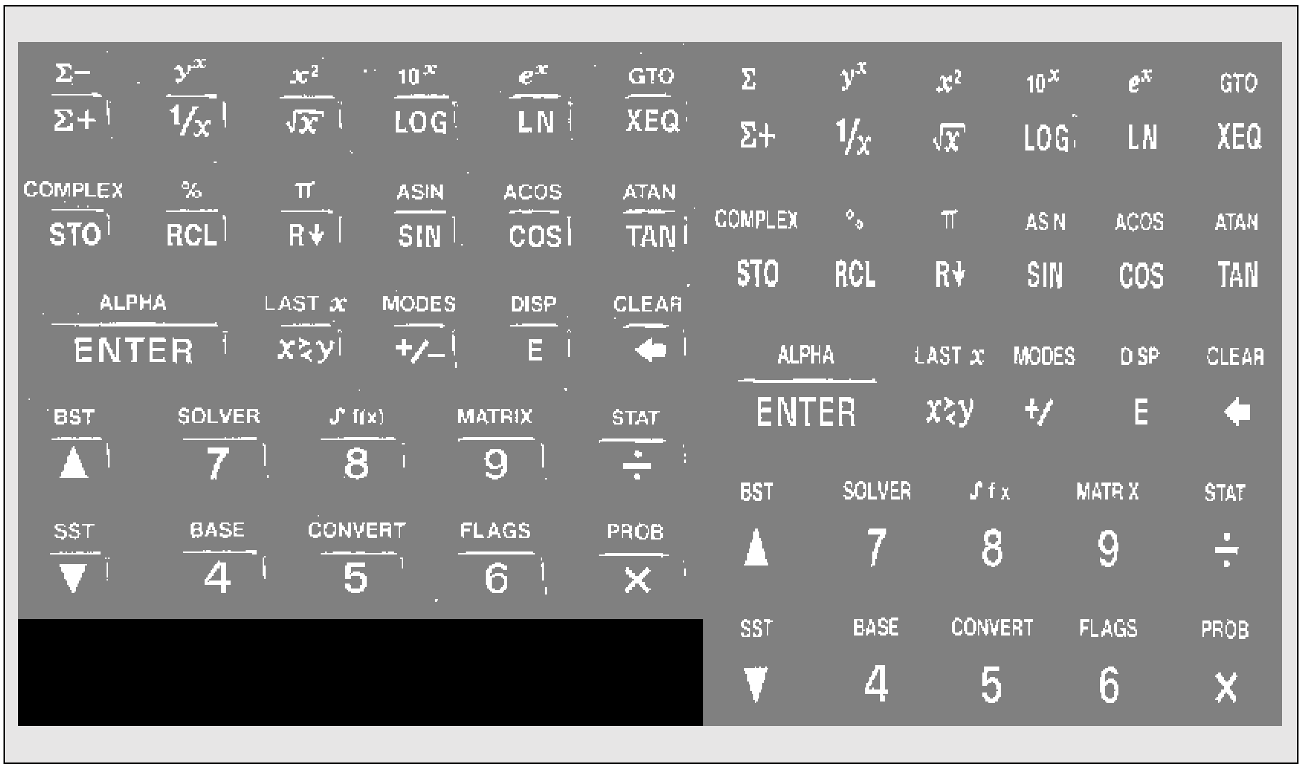

Σ− y^x x² 10^x e^x GTO
Σ+ 1/x √x LOG LN XEQ
COMPLEX % π ASIN ACOS ATAN
STO RCL R↓ SIN COS TAN
ALPHA LAST x MODES DISP CLEAR
ENTER x≷y +/− E ←
BST SOLVER ∫f(x) MATRIX STAT
▲ 7 8 9 ÷
SST BASE CONVERT FLAGS PROB
▼ 4 5 6 ×
Σ y^x x² 10^x e^x GTO
Σ+ 1/x √x LOG LN XEQ
COMPLEX % π ASIN ACOS ATAN
STO RCL R↓ SIN COS TAN
ALPHA LAST x MODES DISP CLEAR
ENTER x≷y +/ E ←
BST SOLVER ∫f x MATRIX STAT
▲ 7 8 9 ÷
SST BASE CONVERT FLAGS PROB
▼ 4 5 6 ×

Table 3-Otsu comparison

From the images the results are very different as the one on the left represents the original image after using Otsu thresholding method and the one on the right is the final image after it goes using morphology. Original images retain all the light reflections that are on the boarder including some noise random pixels scattered around, this results in a very different result then what was provided as an optimal solution in the Project sheet, whereas the second image contains only one line which is wrong so therefore the quantity of lines out of place in the original image can be seen is many times more than the final output.

d) Comment on the selection (size and shape) of the structuring elements used in your solution. [2/25]

In the solution used the structuring elements used were composed of two line structing elements that were set to the angles of 0 degrees and 90 degrees.

The first structuring element used was a vertical element it was used to erode all the random vertical lines. The vertical lines represent the reflection that is seen at the borders of the buttons from the light around and because these lines are not useful pieces of information the vertical line structuring element is used. The element was set to erode only the lines that are under for pixels that is because if they are bigger many of the vertical lines that used to compose the alpha numeric characters will be removed.

The use of square structure element was tested, but since the square most be symmetrical e.g. (5x5 or 6x6) it was not possible to use it as it would end up removing even more characters then the line structuring element does.

The second structuring element used was also a line structuring element used to remove the horizontal lines this structuring element was set at 10 pixels and 90 0 degrees denoting horizontal as the horizontal lines in the image were quite large.

e) Develop a new enhanced programme that is capable of working on at least two other sample calculator keypad images. Comment on the robustness and accuracy of your enhanced solution. [10/25]

The main two things that enable the program to be able to work with two other images are the use of the filtration and the use of the VSG Open method which filter out the background and the fact that the images are scaled to specified dimensions making it possible for the erosion of the lines to be accurate.

Below are the results:

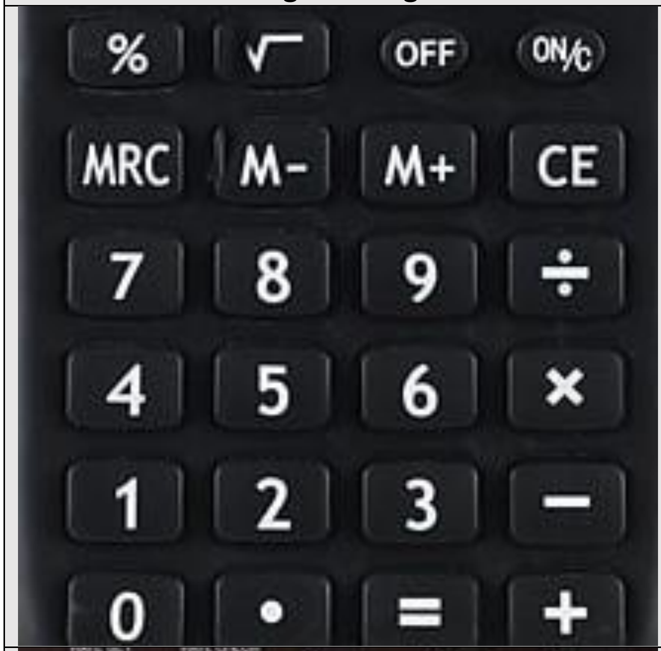
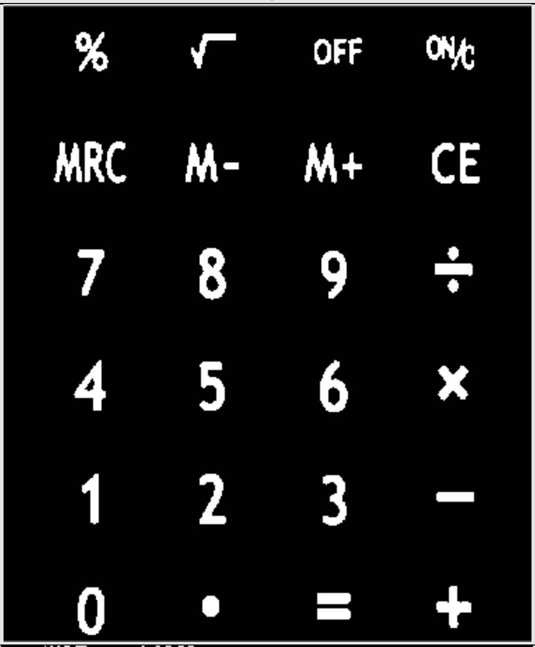
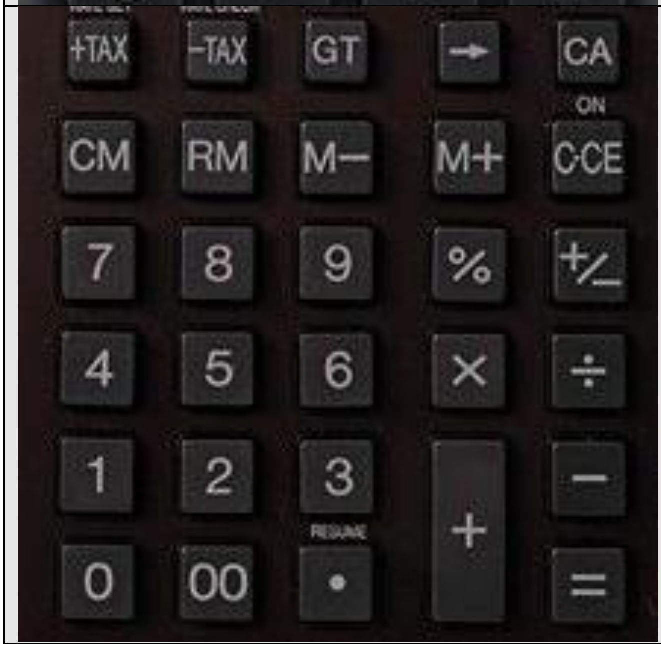
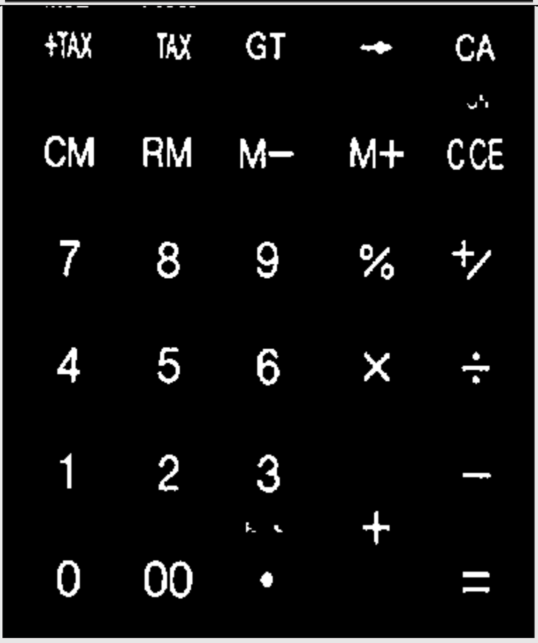
Original image	output
	
	

Table 4- Two other calculators

From the first image everything was picked up and it was clear, and the solution works perfectly in that case. This s mainly due to the filter that is put at the start of the code removing the background and there was not much reflection and the reflection that is there is removed using the Open VSG method in the image making it easier to process.

The second image is a good example even though the corners can be seen to be also removed, the second image contains issues though and that is because this solution has an issue and that is it is very hard for it to keep the small words and very small symbols and that is because the erosion used will usually pick them up making them partially appear or be removed completely. Therefore, This is a good solution to use if images that are close to the clarity and size of the given images.

Radwan duadu
14342606

The faults of the solution are many as it is not a perfect solution for example if the buttons are red then the solution would not work as it will pick them up as light background assuming they are a part of the text. Some problems can be avoided be using other methods other than morphology to extract the data which will directly just extract any characters without needing the processing of looking at morphology.

Also, if the background is white it makes the program fail, the images also need to be close because if the images are not close to the keypad of the calculator the characters will be very small resulting in a bad result.

Below is an example of a case where the solution works very well finding the bigger characters and lines, but the smaller characters are only partial solutions and not clear. Another main reason why the big words like RANDOM in this solution do not show up is due to them being a different colour resulting in them being partial removed when a threshold is applied to the image.

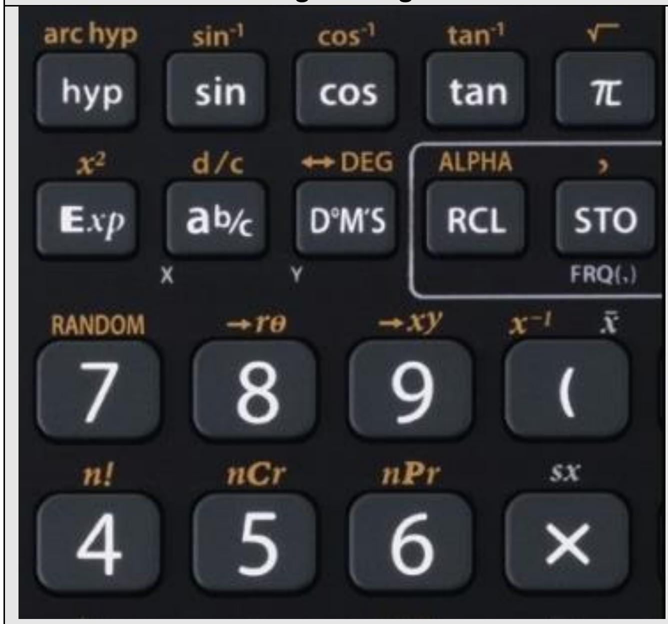
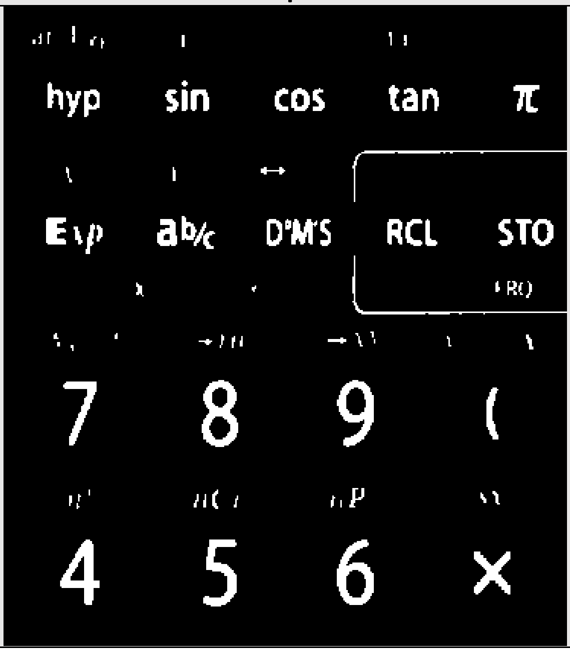
Original mage	output
	

Table 5- Example of issue

MATLAB-CODE:

```
clear all
clc
close all

% images calculator calc2 calc1
image = imread('calculator.tif');
% scaling images after input
[out_img1]=vsg('ScaleImg',image,1134,1360);

grey = vsg('GreyScaler',out_img1);
% adding noise
grey2 =vsg('GaussianNoise',grey,0);

%chnage image to binary and filter noise
```

```
[hi_g]=vsg('HighestGrey',grey2);
[lo_g]=vsg('LowestGrey',grey2);
thresh=uint8((hi_g+lo_g)/2);
BW1=vsg('Threshold',grey2,thresh);

image2 =vsg('Open',BW1,8);

%erossion using structuring elements
marker = imerode(image2, strel('line',4,0));
Iclean = imreconstruct(marker, image2);

marker2 = imerode(Iclean, strel('line',10,90));
Iclean2 = imreconstruct(marker2, Iclean);
BW3 = Iclean2;

% display input and output image
figure;
imshowpair(grey2, BW3, 'montage');
%% part c

% orginal image otsu threshold
org = multithresh(image);



% getting output reconstructed image
test= imreconstruct(image2, grey2);
markerc = imerode(test, strel('line',7,0));
Icleanc = imreconstruct(markerc, test);
markerc2 = imerode(Icleanc, strel('line',10,90));
Icleanc2 = imreconstruct(markerc2, Icleanc);
final= multithresh(Icleanc2);

% change the images to binary
BW = imquantize(image,org);
BW2 = imquantize(Icleanc2,final);

%display result
figure;
imshowpair(BW,BW2,'montage');
```