

**A Project Report**

**on**

*Automatic detection and recognition of the text using the Maximum stable extremal regions and Stroke width transform in unstructured scenes*

By,

Monishkanna Barathan

(1589389)

CENG 5131-Engineering Applications

**Abstract**

A real time text recognition system for natural scenes is a topic that quite captures every ones attention because it sounds imaginative and exciting to even think of the possibility of a computer to process videos almost cognitively as any human. Reading text in any natural scenes has been brought back into focus of many researchers due to the increasing availability of image capturing devices such as mobile phones, video cameras etc. There has been considerable amount of research done on machine reading systems for more than fifty years and as a result the current OCR packages now operate with high speed and accuracy on standard text documents.

The commercial OCR systems do not work for natural scene images because in those images the text is usually present on complex backgrounds and there are many other factors like geometric distortions, partial occlusions, changes in illumination, different font styles, font thickness, font color and texture etc. Therefore, the problem of text recognition of such images remains an active and exciting research topic.

The main aim of the project is to implement the idea of detecting and recognize the text automatically from the captured image in the random environment. This is different than structured scenes, which contain the known scenarios where the position of the text is known beforehand.

In this project, a four step system which automatically detects and extracts the text in the images is proposed. First, detecting the texts using the MSER technology in which several non-text regions are detected alongside the text. Second, those non-text regions are removed on geometric properties. Third, those non-text regions which are detected in the previous steps are removed using stroke width variation and then better text bounding boxes are generated by using the binarized text as strokes. Text is then cleaned and binarized from these new boxes. Finally, if the text is of an OCR-recognizable font, it is passed through a commercial OCR engine for recognition.

The system is stable, robust, and works well on images (with or without structured layouts) from a wide variety of sources, including digitized video frames, photographs, newspapers, advertisements, stock certificates, and personal checks.

**1. INTRODUCTION**

The aim of the project is to detect, identify and recognize text from the natural images. Unlike determined scenarios where the location of the text is known, this project aims at the unstructured scenes. Detecting the text from the natural scenes which is different from the text detecting in the images which is in the printable form. Retrieving the text in several environment provides the contextual step for the number of computer vision applications. If we use the OCR in the case of natural images the results drop drastically and also OR are designed for the scanned texts. As the natural images exhibit several range of color noise, blur and occlusions OCR find its way harder to recognize it.

OCR results is accurate only if the text is located on a uniform background and is formatted like a document. When the text appears on the non-uniform background OCR experience several challenges in obtaining results. So there is an additional preprocessing steps are required to get the better OCR results. In the image if the text presents is sparse and also the background of the image is irregular, then the heuristics used for document layout analysis with OCR might be failing to find the block of text within the image.so the text recognition fails. In those situations automatic layout analysis disable is a convenient method. If that is also the case where the exact text location not able to find, then several approaches came into existence. In that case we can use MSER regions finder helps us to identify the text regions. It uses maximum stable extremal regions algorithm to find the regions. The MSER algorithm has been used in text detection by Chen by combining MSER with Canny edges. Canny edges are used to help cope with the weakness of MSER to blur. MSER is first applied to the image in question to determine the character regions. To enhance the MSER regions any pixels outside the boundaries formed by canny edges are removed.

The separation of the later provided by the edges greatly increase the usability of MSER in the extraction of blurred text. An alternative use of MSER in text detection is the work by Shi using a graph model. This method again applies MSER to the image to generate preliminary regions. These are then used to construct a graph model based on the position distance and color distance between each MSER, which is treated as a node. Next the nodes are separated into foreground and background using cost functions. One cost function is to relate the distance from the node to the foreground and background. The other penalizes nodes for being significantly different from its neighbor. When these are minimized the graph is then cut to separate the text nodes from the non-text nodes. To enable text detection in a general scene, Neumann uses the MSER algorithm in a variety of projections. In addition to the greyscale intensity projection, he uses the red, blue, and green color channels to detect text regions that are color distinct but not necessarily distinct in greyscale intensity. MSER has greater advantages when compared to other text detector as follows.

**Region density** - in comparison to the others MSER offers the most variety detecting about 2600 regions for a textured blur scene and 230 for a light changed. Scene, and variety is generally considered to be good. Also MSER had a repeatability of 92% for this test.

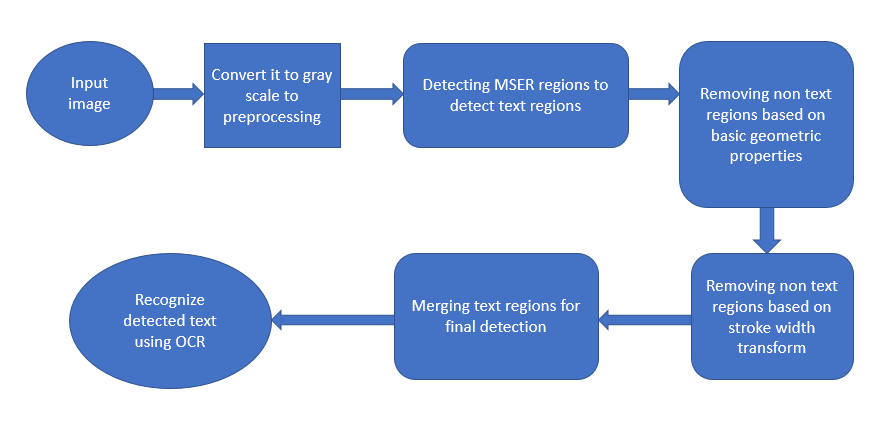
**Region size** - MSER tended to detect many small regions, versus large regions which are more likely to be occluded or to not cover a planar part of the scene. Though large regions may be slightly easier to match.

**Viewpoint change** - MSER outperforms the five other region detectors in both the original images and those with repeated texture motifs.

**Scale change** - Following Hessian-affine detector, MSER comes in second under a scale change and in-plane rotation.

**2. METHODOLOGY**

**2.1 BLOCK DIAGRAM**



2.2 DETECTING TEXT USING MSER REGIONS.

The MSER feature detector works well for finding text regions. It works well for text because the consistent color and high contrast of text leads to stable intensity profiles. Use the detectMSERFeatures function to find all the regions within the image and plot these results. Notice that there are many non-text regions detected alongside the text.

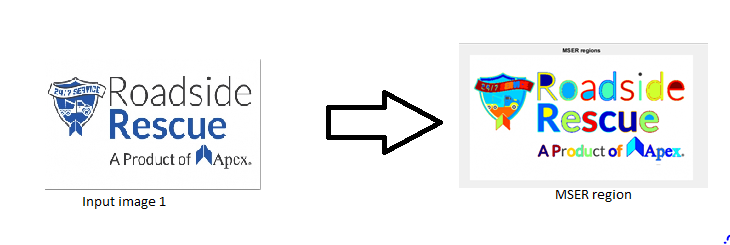


Figure 1. Results of image 1 for MSER regions

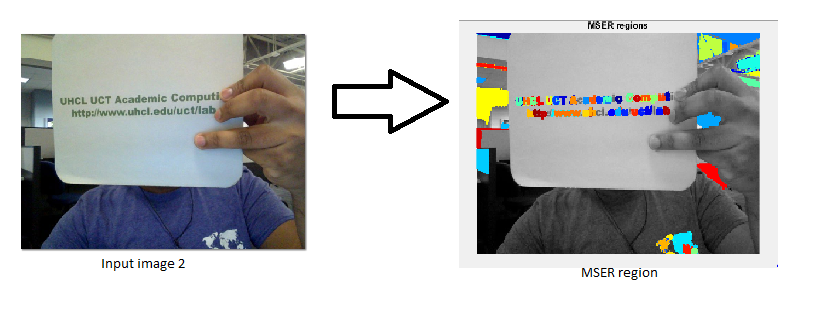


Figure 2. Results of image 2 for MSER regions

2.3. REMOVING NON TEXT REGIONS BASED ON GEOMETRIC PROPERTIES

Although the MSER algorithm picks out most of the text, it also detects many other stable regions in the image that are not text. You can use a rule-based approach to remove non-text regions. For example, geometric properties of text can be used to filter out non-text regions using simple thresholds. Alternatively, you can use a machine learning approach to train a text vs. non-text classifier. Typically, a combination of the two approaches produces better results [4]. This example uses a simple rule-based approach to filter non-text regions based on geometric properties.

a. Eccentricity:

It returns the scalar that specifies the eccentricity of the eclipse that has the same second moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major length. The value is between 0 and 1.

b. Extent:

Returns the scalar that specifies the ratio of the pixels in the region to the pixels in the total bounding box. Computed by the area divided by the area of the bounding box.

c. Euler number:

The Euler number is the total number of images in the images minus total number of holes in those objects. Euler number greater than 1 is having discontinuities.

d. Solidity:

It returns the scalar specifying the proportion of the pixels in the convex hull that are also in the region. The solidity or proportion of the pixels in the convex hull that also in the region of letters is typically low, so that the connected components with the high solidity can be rejected.

2.4. REMOVING NON TEXT REGIONS BASED ON STROKE WIDTH VARIATION

For the sake of completeness, also mentioning that SWT or Stroke Width Transform was devised by Epstein and others in 2010 and has turned out to be one of the most successful text detection methods till date. It does not use machine learning or elaborate tests. Basically after Canny edge detection on the input image, it calculates the thickness of each stroke that makes up objects in the image. As text has uniformly thick strokes, this can be a robust identifying feature.

SWT works by first identifying high contrast edges in a given image. By traversing the image at each edge pixel, in the direction normal to the edge, until another normal edge is found, we can effectively identify strokes in the image. A stroke is an element of finite width with two roughly parallel sides, as you might find in a pen stroke.

By measuring the width of this stroke, and connecting adjacent pixels with similar associated stroke widths, we can extract each "pen stroke" from the image. A contiguous stroke typically represents a man-made character, since most natural or otherwise noisy shapes don't exhibit stroke-like characteristics, and those that do usually don't possess consistent stroke width

SWT operates in a few steps:

1. Use matlab to extract image edges using Canny edge detection
2. Calculate the x- and y-derivatives of the image, which can be superimposed to calculate the image gradient. The gradient describes, for each pixel, the direction of greatest contrast. In the case of an edge pixel, this is synonymous with the vector normal to the edge.
3. For each edge pixel, traverse in the direction θ of the gradient until the next edge pixel is encountered (or you fall off the image). If the corresponding edge pixel's gradient is pointed in the opposite direction (θ - π), we know the newly-encountered edge is roughly parallel to the first, and we have just cut a slice (line) through a stroke. Record the stroke width, in pixels, and assign this value to all pixels on the slice we just traversed.
4. For pixels that may belong to multiple lines, reconcile differences in those line widths by assigning all pixels the median stroke width value. This allows the two strokes you might encounter in an 'L' shape to be considered with the same, most common, stroke width.
5. Connect lines that overlap using a union-find (disjoint-set) data structure, resulting in a disjoint set of all overlapping stroke slices. Each set of lines is likely a single letter/character.
6. Apply some intelligent filtering to the line sets; we should eliminate anything to small (width, height) to be a legible character, as well as anything too long or fat (width: height ratio), or too sparse (diameter: stroke width ratio) to realistically be a character.
7. Use a k-d tree to find pairings of similarly-stroked shapes (based on stroke width), and intersect this with pairings of similarly-sized shapes (based on height/width). Calculate the angle of the text between these two characters (sloping upwards, downwards?).
8. Use a k-d tree to find pairings of letter pairings with similar orientations. These groupings of letters likely form a word. Chain similar pairs together.
9. Produce a final image containing the resulting words.



2.5 MERGING THE TEXT REGIONS FOR FINAL DETECTION RESULT

At this point, all the detection results are composed of individual text characters. To use these results for recognition tasks, such as OCR, the individual text characters must be merged into words or text lines. This enables recognition of the actual words in an image, which carry more meaningful information than just the individual characters. For example, recognizing the string 'EXIT' vs. the set of individual characters {'X','E','T','I'}, where the meaning of the word is lost without the correct ordering.

One approach for merging individual text regions into words or text lines is to first find neighboring text regions and then form a bounding box around these regions. To find neighboring regions, expand the bounding boxes computed earlier with regionprops. This makes the bounding boxes of neighboring text regions overlap such that text regions that are part of the same word or text line form a chain of overlapping bounding boxes.

The overlapping bounding boxes can be merged together to form a single bounding box around individual words or text lines. To do this, compute the overlap ratio between all bounding box pairs. This quantifies the distance between all pairs of text regions so that it is possible to find groups of neighboring text regions by looking for non-zero overlap ratios.

Once the pair-wise overlap ratios are computed, use a graph to find all the text regions "connected" by a non-zero overlap ratio. Use the bboxOverlapRatio function to compute the pair-wise overlap ratios for all the expanded bounding boxes, then use graph to find all the connected regions. After detecting the text regions, use the OCR function to recognize the text within each bounding box. Note that without first finding the text regions, the output of the OCR function would be considerably noisier.

3. MATLAB CODE

monish\_input = imread('a3.png');

figure,imshow(monish\_input),title('Original Image')

m\_grayImage = rgb2gray(monish\_input);

figure,imshow(m\_grayImage),title('Original Image in gray')

% Detect MSER regions.

[mserRegions, mserConnComp] = detectMSERFeatures(m\_grayImage, ...

'RegionAreaRange',[200 7800],'ThresholdDelta',4);

figure

imshow(m\_grayImage)

hold on

plot(mserRegions, 'showPixelList', true,'showEllipses',false)

title('MSER regions')

hold off

mserStats = regionprops(mserConnComp, 'BoundingBox', 'Eccentricity', ...

'Solidity', 'Extent', 'Euler', 'Image');

% Compute the aspect ratio using bounding box data.

bbox = vertcat(mserStats.BoundingBox);

w = bbox(:,3);

h = bbox(:,4);

aspectRatio = w./h;

% Threshold the data to determine which regions to remove. These thresholds

% may need to be tuned for other images.

filterIdx = aspectRatio' > 3;

filterIdx = filterIdx | [mserStats.Eccentricity] > .995 ;

filterIdx = filterIdx | [mserStats.Solidity] < .3;

filterIdx = filterIdx | [mserStats.Extent] < 0.2 | [mserStats.Extent] > 0.9;

filterIdx = filterIdx | [mserStats.EulerNumber] < -4;

% Remove regions

mserStats(filterIdx) = [];

mserRegions(filterIdx) = [];

% Show remaining regions

figure

imshow(m\_grayImage)

hold on

plot(mserRegions, 'showPixelList', true,'showEllipses',false)

title('After Removing Non-Text Regions Based On Geometric Properties')

hold off

regionImage = mserStats(6).Image;

regionImage = padarray(regionImage, [1 1]);

% Compute the stroke width image.

distanceImage = bwdist(~regionImage);

skeletonImage = bwmorph(regionImage, 'thin', inf);

strokeWidthImage = distanceImage;

strokeWidthImage(~skeletonImage) = 0;

% Show the region image alongside the stroke width image.

figure

subplot(1,2,1)

imagesc(regionImage)

title('Region Image')

subplot(1,2,2)

imagesc(strokeWidthImage)

title('Stroke Width Image')

% Compute the stroke width variation metric

strokeWidthValues = distanceImage(skeletonImage);

strokeWidthMetric = std(strokeWidthValues)/mean(strokeWidthValues);

% Threshold the stroke width variation metric

strokeWidthThreshold = 0.4;

strokeWidthFilterIdx = strokeWidthMetric > strokeWidthThreshold;

% Process the remaining regions

for j = 1:numel(mserStats)

regionImage = mserStats(j).Image;

regionImage = padarray(regionImage, [1 1], 0);

distanceImage = bwdist(~regionImage);

skeletonImage = bwmorph(regionImage, 'thin', inf);

strokeWidthValues = distanceImage(skeletonImage);

strokeWidthMetric = std(strokeWidthValues)/mean(strokeWidthValues);

strokeWidthFilterIdx(j) = strokeWidthMetric > strokeWidthThreshold;

end

% Remove regions based on the stroke width variation

mserRegions(strokeWidthFilterIdx) = [];

mserStats(strokeWidthFilterIdx) = [];

% Show remaining regions

figure

imshow(m\_grayImage)

hold on

plot(mserRegions, 'showPixelList', true,'showEllipses',false)

title('After Removing Non-Text Regions Based On Stroke Width Variation')

hold off

% Get bounding boxes for all the regions

bboxes = vertcat(mserStats.BoundingBox);

% Convert from the [x y width height] bounding box format to the [xmin ymin

% xmax ymax] format for convenience.

xmin = bboxes(:,1);

ymin = bboxes(:,2);

xmax = xmin + bboxes(:,3) - 1;

ymax = ymin + bboxes(:,4) - 1;

% Expand the bounding boxes by a small amount.

expansionAmount = 0.08;

xmin = (1-expansionAmount) \* xmin;

ymin = (1-expansionAmount) \* ymin;

xmax = (1+expansionAmount) \* xmax;

ymax = (1+expansionAmount) \* ymax;

% Clip the bounding boxes to be within the image bounds

xmin = max(xmin, 1);

ymin = max(ymin, 1);

xmax = min(xmax, size(m\_grayImage,2));

ymax = min(ymax, size(m\_grayImage,1));

% Show the expanded bounding boxes

expandedBBoxes = [xmin ymin xmax-xmin+1 ymax-ymin+1];

IExpandedBBoxes = insertShape(monish\_input,'Rectangle',expandedBBoxes,'LineWidth',3);

figure

imshow(IExpandedBBoxes)

title('Expanded Bounding Boxes Text')

% Compute the overlap ratio

overlapRatio = bboxOverlapRatio(expandedBBoxes, expandedBBoxes);

% Set the overlap ratio between a bounding box and itself to zero to

% simplify the graph representation.

n = size(overlapRatio,1);

overlapRatio(1:n+1:n^2) = 0;

% Create the graph

g = graph(overlapRatio);

% Find the connected text regions within the graph

componentIndices = conncomp(g);

% Merge the boxes based on the minimum and maximum dimensions.

xmin = accumarray(componentIndices', xmin, [], @min);

ymin = accumarray(componentIndices', ymin, [], @min);

xmax = accumarray(componentIndices', xmax, [], @max);

ymax = accumarray(componentIndices', ymax, [], @max);

% Compose the merged bounding boxes using the [x y width height] format.

textBBoxes = [xmin ymin xmax-xmin+1 ymax-ymin+1];

% Remove bounding boxes that only contain one text region

numRegionsInGroup = histcounts(componentIndices);

textBBoxes(numRegionsInGroup == 1, :) = [];

% Show the final text detection result.

ITextRegion = insertShape(monish\_input, 'Rectangle', textBBoxes,'LineWidth',3);

figure

imshow(ITextRegion)

title('Detected Text')

ocrtxt = ocr(m\_grayImage, textBBoxes);

[ocrtxt.Text]

4.MATLAB FUNCTIONS

[[regions](https://www.mathworks.com/help/vision/ref/detectmserfeatures.html" \l "outputarg_regions),[cc](https://www.mathworks.com/help/vision/ref/detectmserfeatures.html#outputarg_cc)] = detectMSERFeatures([I](https://www.mathworks.com/help/vision/ref/detectmserfeatures.html" \l "inputarg_I))optionally returns MSER regions in a connected component structure.

[stats](https://www.mathworks.com/help/images/ref/regionprops.html#outputarg_stats) = regionprops([BW](https://www.mathworks.com/help/images/ref/regionprops.html" \l "inputarg_BW),[properties](https://www.mathworks.com/help/images/ref/regionprops.html#inputarg_properties)) returns measurements for the set of properties specified by properties for each 8-connected component (object) in the binary image, BW. stats is struct array containing a struct for each object in the image. You can use regionprops on contiguous regions and discontinuous regions

[CC](https://www.mathworks.com/help/images/ref/bwconncomp.html#outputarg_CC) = bwconncomp([BW](https://www.mathworks.com/help/images/ref/bwconncomp.html" \l "inputarg_BW)) returns the connected components CC found in the binary image BW. bwconncomp uses a default connectivity of 8 for two dimensions, 26 for three dimensions, and conndef(ndims(BW),'maximal') for higher dimensions.

C = vertcat(A1,...,AN) vertically concatenates arrays A1,...,AN. All arrays in the argument list must have the same number of columns. vertcat fills in default row names when some of the inputs have names and some do not.

 vertcat assigns values for each table property (except for RowNames) using the first nonempty value for the corresponding property in the tables A1,...,AN. If the inputs are timetables, then column names must be the same.

[A](https://www.mathworks.com/help/matlab/ref/or.html#inputarg_A) | [B](https://www.mathworks.com/help/matlab/ref/or.html#inputarg_B) performs a logical OR of arrays A and B and returns an array containing elements set to either logical 1 (true) or logical 0 (false). An element of the output array is set to logical 1 (true) if either A or B contain a nonzero element at that same array location. Otherwise, the array element is set to 0.

[A](https://www.mathworks.com/help/matlab/ref/accumarray.html#outputarg_A) = accumarray([subs](https://www.mathworks.com/help/matlab/ref/accumarray.html" \l "inputarg_subs),[val](https://www.mathworks.com/help/matlab/ref/accumarray.html#inputarg_val)) returns array A by [accumulating elements](https://www.mathworks.com/help/matlab/ref/accumarray.html#bt3696b-9) of vector val using the subscripts subs. If subs is a column vector, then each element defines a corresponding subscript in the output, which is also a column vector.

The accumarrayfunction  collects all elements of val that have identical subscripts in subs and stores their sum in the location of A corresponding to that subscript (for index i, A(i)=sum(val(subs(:)==i))). Elements of A whose subscripts do not appear in subs are equal to 0.

For an *m*-by-*n* matrix subs, each row represents an *n*-dimensional subscript into output A. The ith row of subs corresponds to the ith element in the vector val.

hold on retains plots in the current axes so that new plots added to the axes do not delete existing plots. New plots use the next colors and line styles based on the ColorOrder and LineStyleOrder properties of the axes. MATLAB® adjusts axes limits, tick marks, and tick labels to display the full range of data. If axes do not exist, then the hold command creates them.

hold off sets the hold state to off so that new plots added to the axes clear existing plots and reset all axes properties. The next plot added to the axes uses the first color and line style based on the ColorOrder and LineStyleOrder properties of the axes. This option is the default behavior.

imshow([I](https://www.mathworks.com/help/images/ref/imshow.html" \l "inputarg_I)) displays image I in a figure, where I is a grayscale, RGB (truecolor), or binary image. For binary images, imshow displays pixels with the value 0 (zero) as black and 1 as white. imshow optimizes figure, axes, and image object properties for image display.

D = bwdist(BW) computes the Euclidean distance transform of the binary image BW. For each pixel in BW, the distance transform assigns a number that is the distance between that pixel and the nearest nonzero pixel of BW. bwdist uses the Euclidean distance metric by default. BW can have any dimension. D is the same size as BW.

[BW2](https://www.mathworks.com/help/images/ref/bwmorph.html#outputarg_BW2) = bwmorph([BW](https://www.mathworks.com/help/images/ref/bwmorph.html" \l "inputarg_BW),[operation](https://www.mathworks.com/help/images/ref/bwmorph.html#inputarg_operation)) applies a specific morphological operation to the binary image BW.

[txt](https://www.mathworks.com/help/vision/ref/ocr.html#outputarg_txt) = ocr([I](https://www.mathworks.com/help/vision/ref/ocr.html#inputarg_I)) returns an [ocrText](https://www.mathworks.com/help/vision/ref/ocrtext-class.html) object containing optical character recognition information from the input image, I. The object contains recognized text, text location, and a metric indicating the confidence of the recognition result.

4. RESULTS

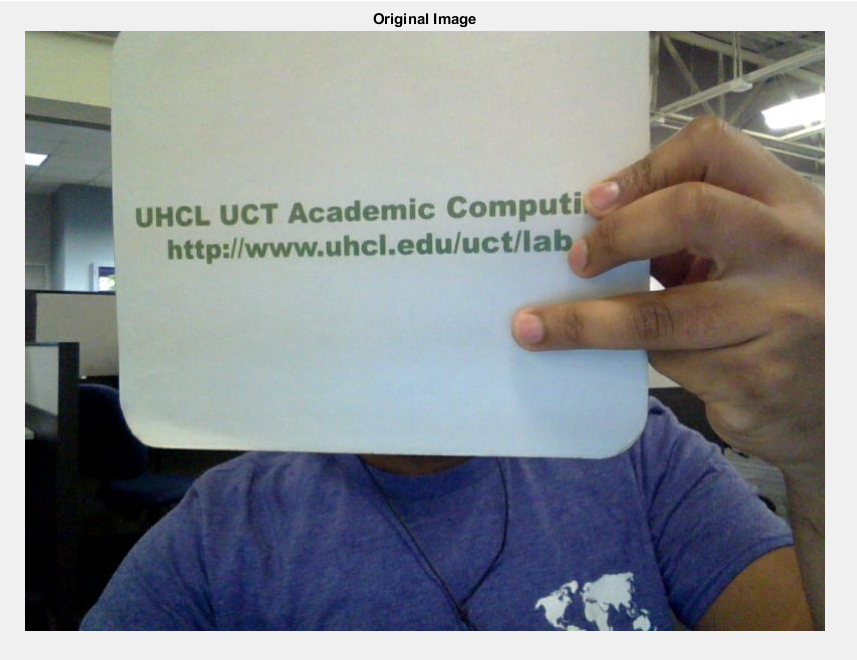
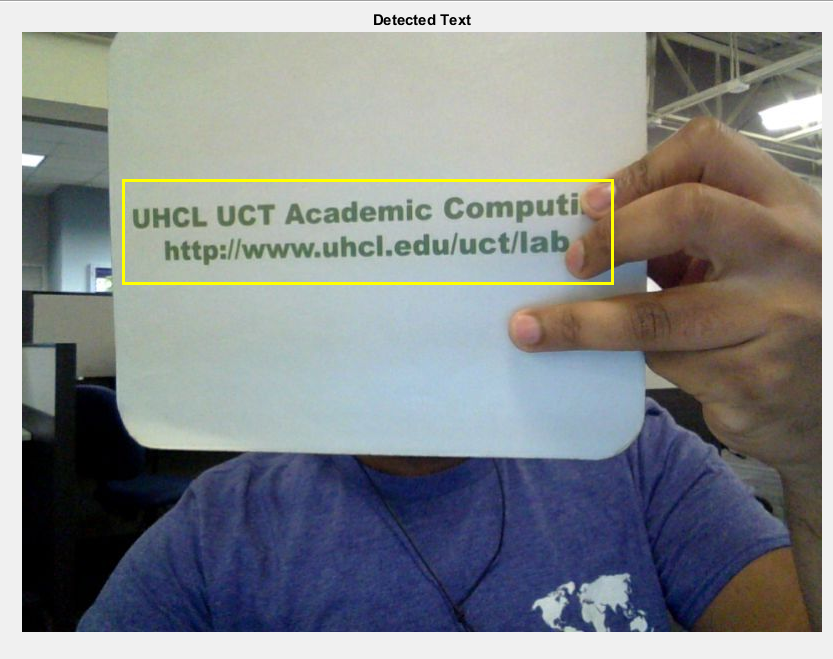
 

Figure 3. Bounding boxes to obtain final result

5. REFERENCES

[1] Chen, Huizhong, et al. "Robust Text Detection in Natural Images with Edge-Enhanced Maximally Stable Extremal Regions." Image Processing (ICIP), 2011 18th IEEE International Conference on. IEEE, 2011.

[2] Gonzalez, Alvaro, et al. "Text location in complex images." Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE, 2012.

[3] Li, Yao, and Huchuan Lu. "Scene text detection via stroke width." Pattern Recognition (ICPR), 2012 21st International Conference on. IEEE, 2012.

[4] Neumann, Lukas, and Jiri Matas. "Real-time scene text localization and recognition." Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012.

[5] website: <https://www.mathworks.com/help/>