# EE610 Project : Tilt Correction For Text Images

Devanshu Singh Gaharwar
*16D070042*

Pankaj Singh
*183079036*

Amit Lohan
*183079033*

## I. Problem Statement

Storing and processing of text documents in digital mode is far more convenient than physical mode, however scanning these text documents is the key problem. Due to some mechanical or human errors certain inclination angle exist in text images and will cause obvious deformation in characters, that is unfavorable for character recognition. It can also affect the layout analysis of the text images. Thus, it becomes very important to correct the tilt in text images. At present, there is a lot of work already done for designing efficient tilt correction algorithm's. A big example for this would be CamScanner app. So we aim to solve this problem of tilt correction, using the concepts taught in the course and in the process realize various ways for the same.

## II. Methods

A short description of our implementation is as follows:
1) Firstly, as a pre-processing step we need to convert the input RGB text images into gray images.
2) Then we do low pass filtering of this image using a Gaussian blurring function for noise removal and also to remove the unnecessary details of image which are irrelevant to the future blocks.
3) After this, we do the edge detection using the Canny operator (which extracts the edges using the gradient of the image).
4) Then we extract the straight lines from the output after edge detection using the Hough Transform and then using these lines,find the 4 corner points of the image.
5) Finally, we do the perspective transform which is just a affine transformation (rotation and translation) of the image to get the corrected image.
Note that this perspective transform is applied on the original image itself and thus output will be a coloured image like the input image i.e. no text information of the original image will be lost.

Now, we explain each step of our implementation in detail. This would be supported by an example image for better visual understanding. The standard example image that we use is

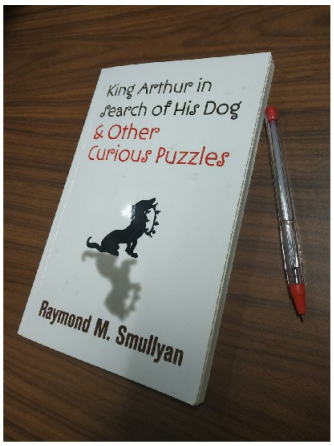


Figure 1: Input Image for Tilt Correction

### A. Pre-Processing

As pre-processing we only need to convert our input coloured image into gray image. Another general step in such procedures is to also do the binarization of the image, but that is not needed in our setup.
After this pre-processing our input image changes to :

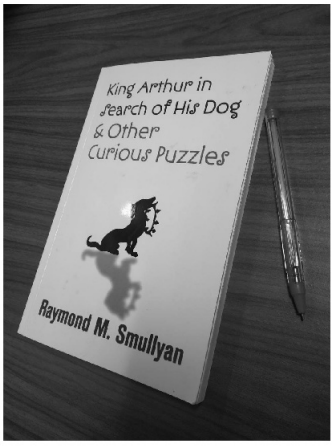


Figure 2: Input Image after Pre-Processing

### B. Low-Pass Filtering

Our aim is to detect the 4 corner points of the image correctly so that we can quantify the tilt. For this aim, we don't care about the specific text information inside the image. Also, since in the later stage we use the Canny Operator, we need

to do some sort of low pass filtering in our original image, since canny operator uses gradient and we know that noise gets amplified by the gradient. Thus doing low pass filtering serves both our purposes i.e. removing irrelevant information and suppressing noise. In particular, to do the low pass filtering we use a Gaussian Filter.
The output image after this is as shown:

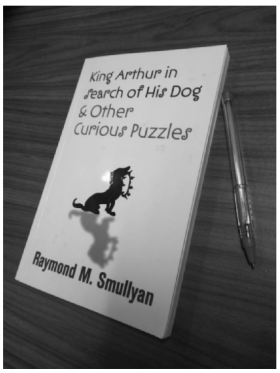


Figure 3: Image after Low Pass Filtering

### C. Canny Operator

Now, after getting a low-passed version of the image we need to identify the edges in our images. For this we use the Canny Operator. The steps involved in the Canny Operator is as follows:
(1) Convolution of the input image with the partial derivatives of the Gaussian to get $I_x$ and $I_y$.
(2) Then, calculate the magnitude and direction of the gradient using $I_x$ and $I_y$.
(3) Now apply non-maxima gradient suppression i.e. suppress all the points which are not the maxima by looking at the neighbouring points by interpolating.
(4)Finally, Hysteresis Thresholding is done in which we adopt a double threshold detection and edge connection scheme.
The output image after Canny Operator is as shown:

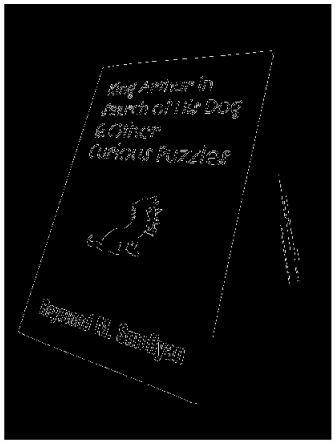

Figure 4: Image after Canny Operator

### D. Hough Transform

For this, we have as an input the edges of the image and we aim to extract lines from it. And then using those lines we can find the 4 corner points of the image.
The Hough Transform can be intuitively understood as the discrete version of Radon Transform. In general, Hough Transform is used for detecting simple shapes, such as straight lines, circles or ellipses. In our case we use the Canny edge detector's output as the input to this but due to imperfections in either the image data or the edge detector, there may be missing points or pixels on the desired curves as well as spatial deviations between the ideal line and the noisy edge points as they are obtained from the edge detector. For these reasons, it is often non-trivial to group the extracted edge features to an appropriate set of lines, circles or ellipses. The purpose of the Hough transform is to address this problem by making it possible to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects.
Thus, using Hough Transform we can extract lines properly. The detected lines in our image is as shown:

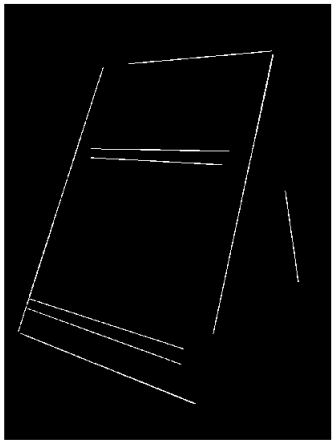

Figure 5: Lines detected using Hough Transform

### E. Corner Point Detection

Now, after getting the Hough Transform, we have successfully extracted the lines in the image. Now, using these lines we have to find the location of the corner points. An immediately obvious approach for this would be to solve the equations of line and thus get the intersection points. In-fact, this is precisely the approach of our reference paper. But, we have followed a different approach because on implementing the suggested method on various test images, we saw that the point that we get by solving the line equations can be a bit offset from the actual corner point because even after Hough Transform there might be some error in the lines detected. And if there is error in even one of the lines in the sense that they

are extended or shortened, then the intersection point would be the interpolation of that line, which would result in some offset in the corner point.
To tackle this issue, in our implementation for detecting the corner points we do the following:
(1) For detecting, the top right corner point, among the various lines we have as input, we find the point having the maximum value of $|x + y|$, i.e. find the point having the sum of x and y co-ordinate as maximum. So, top right corner = argmax $|x+y|$
(2) Similarly, bottom left corner = argmax $|-x + (-y)|$
(3) Top left corner = argmax $|-x + y|$
(4) Bottom right corner = argmax $|x + (-y)|$
Now, some intuition for why would our approach solve the issue in the previous approach. In this approach we essentially pick the farthest corner points and thus no interpolation is done to find the corner points. Thus if we flag a point as the corner point, it would correspond actually to some edge in the image. And we saw that our approach detects the corner points correctly even when the previous approach taken by our reference fails.
The detected corner points in our image is shown:

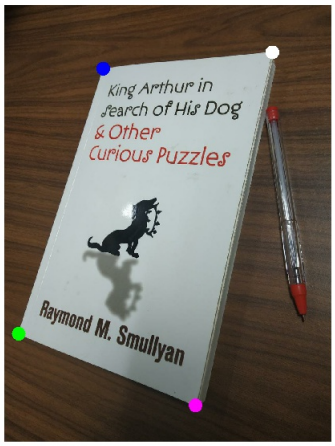


Figure 6: Corner points detected

### *F. Perspective Transform*

In our final stage we have as input the 4 corner points and our original image. Now we do the perspective transformation. The perspective transformation means that the image is transformed into the form of orthographic projection by rotation and translation of the object. For doing this transformation, we need to find the correction matrix and to find the correction points we only need the mapping of the input to the output points. In our case we already have the input corner points and as the output points, we want the extreme points of the image, to display the corrected image in full size. Thus we obtain the corresponding correction matrix and then perform the affine transformation.
And the final output image from our tilt correction method is as shown:

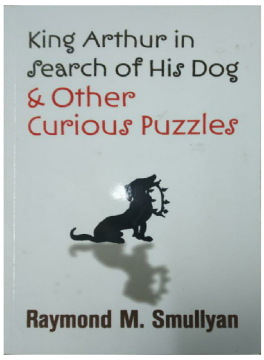


Figure 7: Final Output Image after Perspective Transformation

## III. Results

In this section, we show the performance of our algorithm on various images.

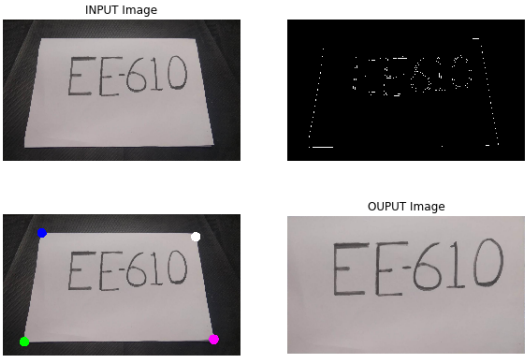


Figure 8: Test Image 1 with outputs at different stages


Figure 9: Test Image 2 with outputs at different stages

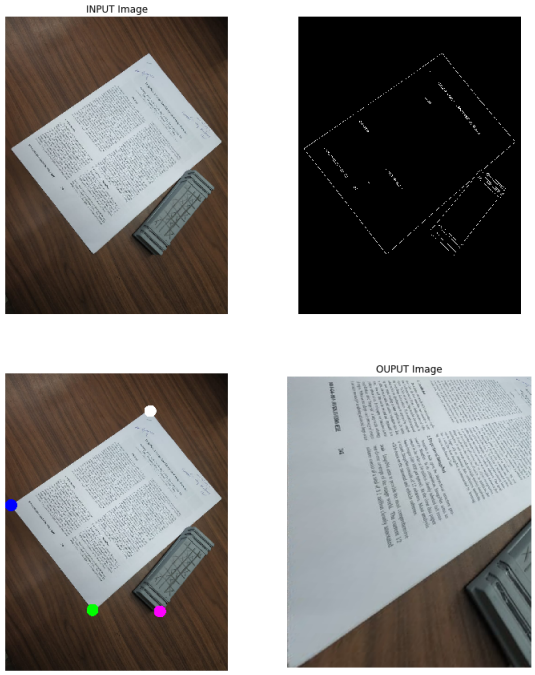


Figure 10: Test Image 3 with outputs at different stages

In the results we have shown, three other test cases. In the test case 1 and 2, it is clearly evident that our approach detects the corner points correctly. Also, it is seen in general that higher the contrast between the image(object) and background, better will be the performance of our approach seen accross various images.
We have also shown a test case($3^{rd}$ image) in which our approach doesn't detect all corner points correctly. But that is because there is another object at one of the extremes of the image. So, although 3 corner points are identified correctly, the fourth corner point is detected as the corner point of the object and this is because from our approach that corner point will maximize the difference between x and y co-ordinate. But note that in real situations we don't get this type of input images in this setup having another object. So, this type of wrong detection is not frequent.

## IV. Conclusion

In this project, we studied a tilt correction algorithm based on using Hough Transform and Canny Operator to correct tilt test images. We see that our method performs well for almost all cases. Our method will work well even if there are multiple objects in the image given that the object of interest is the largest. But we also had few cases in which the corner point's were not classified correctly as shown in the results. This mis-classification happens only in the case when we have multiple objects in our image placed at corners of the image and which is not the cases for the general text images in this setup. Thus, we have a robust algorithm which would perform proper tilt correction. Our code can be accessed via this link :- https://github.com/Pankaj1357/Tilt-Correction-for-Text-Images

## V. References

Our primary reference for this Project was this paper.
https://aip.scitation.org/doi/pdf/10.1063/1.4981624
Some other useful links were:
(1) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html
(2) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html