CONVERSION OF PRINTED EQUATIONS TO $\LaTeX$ CODE

Arka Sarkar (2018222)
arka18222@iiitd.ac.in

Kinshuk Chopra (2018239)
kinshuk18239@iiitd.ac.in

Indraprastha Institute of Information Technology Delhi

ABSTRACT

Latex is a very powerful tool and is massively used in the Industry. This also raises a lot of concerns regarding the ease of use of this program. In this paper, we propose a method that converts printed mathematical equations to latex code. We have used multiple Digital Image Processing methods such as binarization, Otsu thresholding, skew connection, Hough transformation , edge detection, centroid mapping etc. to segregate and spatially map the mathematical symbols. The mapped symbols were identified using a translation-scale-Rotation Invariant robust method.

***Index Terms*—** binarization, Otsu thresholding, skew connection, Hough transformation , edge detection, centroid mapping

1. MOTIVATION

Latex being a very powerful tool is massively used in the Industry for drafting various documents in a professional manner. This wide use of Latex demands a highly intuitive and easy system for drafting documents especially mathematical equations. Once a document is typed and outputted it is very challenging for one to change the contents of the document without having the full access to the source code. Re coding the source code again is a very time consuming and a daunting task which is very susceptible to errors. In this paper, we propose a method that converts printed mathematical equations to latex code. This was achieved using multiple Digital Image Processing methods such as binarization, Otsu thresholding, skew connection, Hough transformation , edge detection, centroid mapping etc. to segregate and spatially map the mathematical symbols. The mapped symbols were identified using a translation-scale-Rotation Invariant robust method with central moment of Inertia, Hu Moments and Circular Topology. Using these identified characters were generated the Latex Code.

2. LIERATURE REVIEW

A lot of this work was already implemented but using huge and more complicated modules. The characters pallet with the moment of inertia,circular topology and hu's moment is something which was very useful and we also used and referenced a pre existing character pallet which helped in making the character identification really smooth. Majority of the pre-existing works was modelled really well to handle complicated equations such as the one consisting of limits and square roots which required the additional knowledge of convex hull to extract the precise location of the square root.

3. METHODOLOGY

We have chosen to go with a modular structure of our system as it provides a more robust system which is least prone to errors. The input provided would be scanned mathematical equations to the system. The entire system pipeline was divided into 6 modules :

- Input RGB image was binarized to segregate the image into a background and foreground.
- The skew was corrected of the binarized image.
- Next the image was segmented into characters, the bounding boxers, centroids of the characters.
- After each character is segmented, we generate a feature vector using translation-scale-Rotation Invariant robust method .
- The characters were matched with the ground truth character vectors and mapped.
- Generating the Latex code (Equation Forming).

We would be discussing all the above mentioned modules in detail in the upcoming subsections.

3.1. Binarization

The scanned images were first binarized, and were segregated into foreground and a backgroud. This binarization was achieved by using the Otsu's thresholding method, where an individual pixel was converted to 1 if it was higher than the threshold else to 0. Otsu's thresholding is a global thresholding method that uses the histogram of the image and maximises the in-class variance of the segment classes.

Since we are only working with scanned images, we are assuming that the lighting is even around the image. We are currently not handling the case when the images passed to the system has uneven lighting. Hence, no lighting compensation would be required in our case.

3.2. Skew Correction

Skew correction was one of the major modules of our system as it is essential for the mathematical equation to be horizontal, for accurate results in the upcoming modules. To correct the skew, we had to rotate the entire equation from the dominant gradient orientation. We have used hough transformation to compute the dominant orientation, since most of the mathematical expressions consist of straight lines, fractions and horizontal elements, thus the dominant orientation is mostly horizontal.We had calculated the majority Hough peaks and on the basis of the peaks we decided whether to go with the peak highest peak 4 unique peaks or to go with the peak which has the maximum number of occurences. We made this choice depending on the result we got by calculating the Hough peaks.

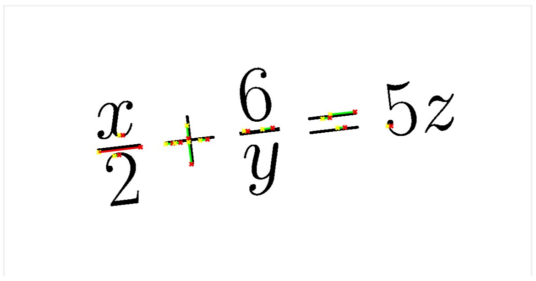

Fig. 1. Gradient Vectors after Hough transform

The lines in the above figure 1 denotes the gradient vectors of the horizontal lines in the image that were outputted from Hough transformation. Using the theta value of the Hough peak(as decided above), we convert that to an angle between -45 to 45 so that we can make the final image rotation to deskew the initially skewed image.

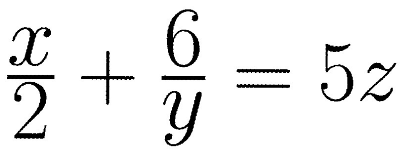

Fig. 2. Deskewed Image

3.3. Segmentation

To detect the equation properly, we had match each character separately . To tackle this, we extracted each character from the image by using centroids and bounding boxes of each character. Firstly, we extracted the edge maps by eroding the inverse of the image and then taking the xor with the original image, this gives us white edge maps on a black background. Secondly, we extracted the centroid, bounding box for each edge which are visually shown in the figures 3 and 4 below and each character properties were saved for further use in the modules. This methods successfully extracts most characters but in some cases like in $\sqrt{}$ (sqaure root), the bounding boxes over lap with each other so it cannot be extracted using this method.

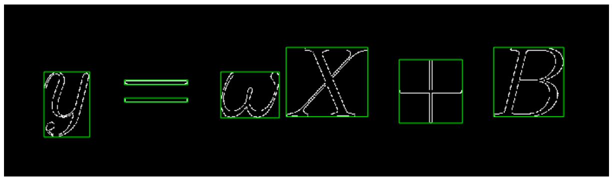

Fig. 3. Bounding Boxes generated

Since, bounding boxes Figure 3 give an accurate representation of the localization of the individual characters, it is much more convenient to run our feature extraction and character identification algorithm on this implementation rather than using any machine learning techniques.

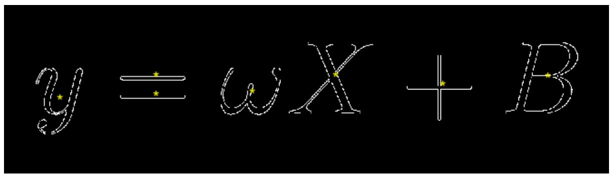

Fig. 4. Centroids generated generated

The centroids Figure 4 generated give the precise location of each object with respect to each other, which helps immensely while formulating the Latex code.
The above mentioned techniques were implemented using inbuilt matlab regionprop function for better and efficient computation.

3.4. Character Feature Extraction

After extracting each character from the image, the main task at hand is to generate a feature vector that will be used to match the characters with the ground truth values to get an accurate match.
For this to work we needed to extract characteristics that were

scale-invariant, translation-invariant and rotation-invariant. We were inspired from the from the literature on circular topology [1], normalized central moment of Inertia and Hu Invariant Moments to generate a 22 element vector as a feature from each character segregated. The first element in the vector would be normalized central moment of Inertia which is translation and rotation-invariant, it is given by the following equation.

$$I = \frac{\Sigma_{i=0}^{N}((x_i - C_x)^2 + (y_i - C_y)^2)}{N^2}$$

, where N = number of pixels in the image; C_x and C_y denote the respective centroids.
The next 6 elements of the vector would be from the Hu Invariant Moments [2], which is invariant to scaling, translation and rotation. We skip the first Hu moment as it is normalised central moment of Inertia which was already incorporated in the vector. The rest 6 Hu Invariant Moments are as follows :

$$\begin{aligned}
&\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \qquad \gamma = 1 + \frac{p+q}{2} \\
&H_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
&H_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
&H_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} - \eta_{03})^2 \\
&H_5 = (\eta_{30} - 3\eta_{12})^2(\eta_{30} + \eta_{12})^2[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&H_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{12} + \eta_{03}) \\
&H_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{3}$$

where μ_{pq} is the central moment of order p and q.
The Final 15 elements were extracted using circular topology [1]. We are aiming to extract rotation invariant characteristics using circular topology, Now, since circles are geometric in nature, a character along a circular path would be rationally invariant. We are calculating the values by logically plotting eight concentric equidistant circles from the centroid of the character. The spacing between the circles is determined by taking the maximum distance of the centroid of the character to the corner pixels and dividing it by $k + 1$, in this case we are taking $k = 8$. The first 8 elements of the topology elements would the number of times each circle intersects character edges. We are only considering the first k circles as the outermost circle mostly moves of out the frame of the character and gives no intersections.
The figure 5 shows the circles visually plotted on a character.

The last 7 topology elements were the distance between the character crossing for each circle for better segregation from the characters having equal number of intersections.

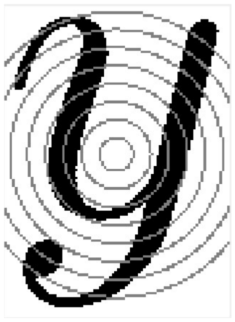

Fig. 5. Concentric circle intersections

The measure was very intuitive, we took the the difference of the two longest arcs from each circle and divide it with the circumference.

$$m = \frac{arc_1 - arc_2}{circumference}$$

We have done this for all the $k - 1$ circles, we have ignored the innermost circle as it mostly either complete background or complete foreground.

3.5. Character Matching

After generating the feature vectors from the extracted characters, we have to find out which ground truth character it closely matches. To tackle this, we have matched the ”Manhattan” and ”Eucledian” distances of the generated feature vector to all the ground truth labels in our character pallet. This ground truth character that gives the minimum distance is picked and outputted as the matched character
This approach works really well as the feature vector is generated using scale-translation-rotation invariant methods so the feature vectors remain similar for the same characters.
Currently our implementation uses the ”Manhattan” distance as default but can be changed by the user.This is due to the fact that manhattan distance is a better way to match distances when dealing with small numbers to avoid any possible underflow in finite precision.

3.6. Equation Formation

Using the bounding boxes and centroids calculated previously, we parse the characters one by one and at the same time, using the previous characters centroid we check whether the current character is either in superscript ,subscript,numerator or denominator.
We used a few observations while parsing the string. The special case of the equal sign, if the $-$ matches only with the bounding box of another $-$, this would imply that it’s a $=$ sign.On the other hand if it does not match with any other $-$ we would consider it to be the dash in a fraction or a minus sign.

Such observations helped us to parse and form the final equation but at the same time led to some limitations.

4. RESULTS

We are pleased to inform that the results were very promising on a lot of the equations that we tested. We tested our implantation with around 25 different printed scanned equations with ranging characters and mathematical symbols and the results were very nearly completely accurate . One example is shown in the figure 6.
The equations which were rotated to some angle was current

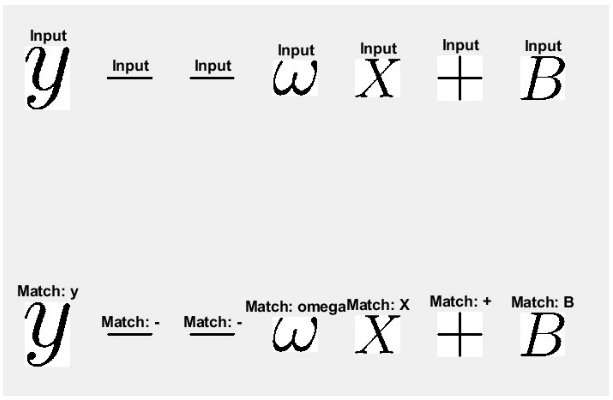

Fig. 6. Sample Test Equation

identified and corrected by our skew correction 3.2 module. Our choice of choosing the feature extractor as a aggregation of Hu moments and circular topology is also a success which is evident from our results.
The current implementation has some limitations which are discussed in the following section.

5. LIMITATIONS

The current implementation is very sensitive to uneven lighting as the binarization modules doesn't work very well on those kind of images. One major bottle neck of the current implementation is that the skew correction 3.2 depends heavily upon that assumption the dominant orientation of the gradient vectors is horizontal and hence can use it as a reference to correct the skewness of the image. Equations that are short and have diagonal lines such that $\,, |, /, !$ will not be deskewed properly as these gradient vectors which are pointing upwards could dominate the entire gradient of the equation.
Another limitation of our implementation is that the bounding boxes generated are accurate for most characters, but certain characters such as $Band\sqrt{}$ will create issues as the bounding boxes get overlapped within each other and the character inside the $\sqrt{}$ will be present inside its bounding box and won't be segregated separately. figure 7 is such a example where $\sqrt{x}$ could not be accuracy segregated due to the aforementioned reason.

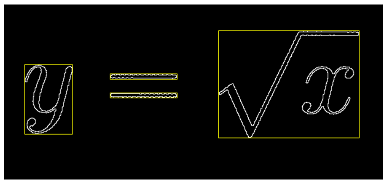

Fig. 7. Sample failed Equation

6. FUTURE DEVELOPMENTS

Future work can be done on the robustness on the algorithm to make it more acceptable to a wider variety of characters. Making the system more robust uneven lighting changes would be of great importance as in that case pictures taken from a smartphone camera could also be accuracy predicted from our implementation.
More future work can be made on making the system detect mathematical equations from text and other jargon in the background. This idea can be combined with modern machine learning techniques to make more accurate and wide technique.

7. REFERENCES

[1] L. A. Torres-Mendez, J. C. Ruiz-Suarez, L. E. Sucar and G. Gomez, "Translation, rotation, and scale-invariant object recognition," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 30, no. 1, pp. 125-130, Feb. 2000, doi: 10.1109/5326.827484.

[2] Vishnu Muralidharan (2020). Hu's Invariant Moments (https://www.mathworks.com/matlabcentral/fileexchange/52259-hu-s-invariant-moments), MATLAB Central File Exchange. Retrieved December 19, 2020.

[3] A. Pradhan, M. Pradhan, and A. Prasad, "An approach for reducing morphological operator dataset and recognize optical character based on significant features", in Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on, vol. 2, Aug. 2015, pp. 1631–1638.

[4] B. Potocnik, "Visual pattern recognition by moment invariants," in Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006 focused on, Jun. 2006, pp. 27–32.

[5] Ming-Kuei Hu, ”Visual pattern recognition by moment invariants,” in IRE Transactions on Information Theory, vol. 8, no. 2, pp. 179-187, February 1962, doi: 10.1109/TIT.1962.1057692.

[6] Jim brewwer and James Sun, ”LaTeX generation from Printed Equations”

[7] Joseph Chang, Shrey Gupta and Andrew Zhang ”Painfree LaTeX with Optical Character Recognition and Machine Learning”