LaTeXGeneration from Printed Equations

Jim Brewer, James Sun
Department of Electrical Engineering
Stanford University
Stanford CA, 94305
Email: {jebrewer, jsun2015}@stanford.edu

Abstract—**This article describes a system that takes a photograph of a printed equation and produces a LaTeXcode representation. The process uses adaptive thresholding with mean filtering, morphological edge smoothing, and the Hough transform for image binarization and skew correction. The project uses Hu invariant moments and circular topology to match characters against a database of characters. The algorithm then assembles the appropriate LaTeXcode from the detected characters. This system is able to detect 93.6% of characters in ideal images and 86.2% of characters in real-world photographs when combining two different skew methods (74%-79%).**

I. Introduction

LaTeXis a powerful typesetting system that is extremely useful for technical documents, particularly mathematical equations. However, once rendered, the output cannot be modified without access to the underlying code. Re-coding lengthy equations is time consuming and prone to errors. The ability to take a photograph of an existing equation printed in a textbook, homework assignment or technical article and to produce the editable LaTeXcode solves this problem. The challenges to accomplishing this involve cleanly converting the photograph to a binary image, correcting skew from the horizontal, correctly segmenting each individual character, matching the characters to a database of characters, and finally generating the correct LaTeXrepresentation of the equation.

II. Related Work

The advent of smartphones has prompted the rise of applications that automatically recognize and solve mathematical equations using a built in camera; one example is the popular PhotoMath application [1]. However, less work has gone into translating equations into LaTeXcode.
The problem of character recognition has also seen numerous techniques that have had varying levels of success.This includes a basic template matching algorithm using morphological operators with additional character properties [2]. Some have used morphological operators with only portions of the character to reduce the operator dataset and detect characters based on certain combinations of matched operators [3]. Others have used various region-based invariant moments such as the Hu, Zernike, and Krawtchouk moments [4] and circular topology of individual characters in an effort to have scale, translation, and rotation robustness [5]. This project combines the use of the Hu moments with circular topology.

III. Process Flow

The following processing pipeline converts an image of an equation to LaTeXcode. First, the image is captured; the target use-case does this through a smartphone camera. Next, the image is binarized with a white background and black characters. Then, skew is corrected so the equation in the image is horizontal. Afterwards, segmentation algorithms find each contiguous character in the equation, extract a feature vector, and identify the character using nearest neighbor classification. Finally, an algorithm assembles the recognized characters into LaTeXcode. The current implementation of the process outputs the resultant code as a .tex file to be used as needed. The following sections describe each part of the pipeline in more detail.

A. Binarization

The input RGB image is first converted into a binary image for processing. We found that treating scanned images differently from smartphone photographs gave the best results. In order to differentiate between the two image types, an image is first converted to grayscale. Then, we look at the proportion of pixels that are mid-gray, defined as having an 8-bit intensity value between 15 and 240, inclusive. If this proportion is less than 0.1, we classify the image as a scanned or screenshot image; otherwise, we classify the image as a photograph. The following two sections describe the binarization method carried out for each class of image.

1) Smartphone Photographs: Photographs taken by a smartphone invariably contain uneven lighting and page imperfections. We perform adaptive thresholding with noise removal to compensate. First, the image is blurred with a Gaussian filter to smooth edges and remove high frequency noise, if the image is high-resolution. This pre-filtering is very beneficial for character recognition in high-resolution images but actually has a detrimental effect for low-resolution images due to the small number of pixels in the image. We empirically determined that an image with more than 2000×1000 pixels is well classified as high-resolution for use with a 10×10 Gaussian filter with standard deviation of 3.

We use adaptive thresholding for the binarization in order to compensate for uneven lighting. The window size is set at $1/60^{\text{th}}$ of the smaller dimension of the image. This adaptive window size was chosen in order to adequately handle uneven lighting while still being small enough to preserve the integrity of most of the characters. To eliminate the overhead of

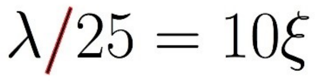

Fig. 1: Image with Problematic Non-Horizontal Line

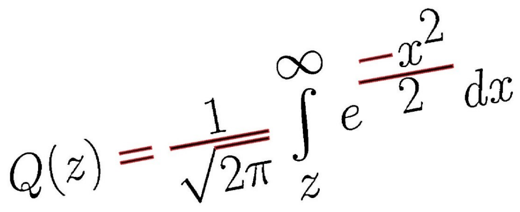

Fig. 2: Skewed Binarized Image

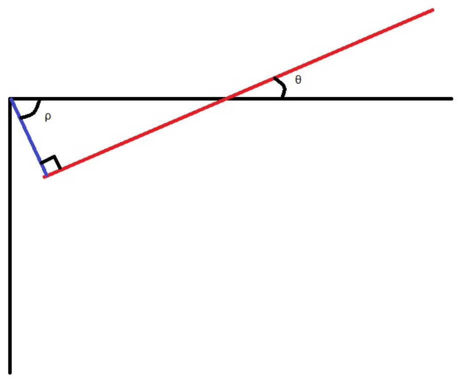

Fig. 3: Skew angles

performing Otsu's method on each window, we leverage the fact that each image consists only of black text or white background. Thus, we use the window mean minus 10 as the threshold, with the offset of 10 to eliminate the effect of noise in the background. The window means are calculated quickly by convolving with an averaging filter; subtracting the offset from the output creates our threshold matrix. Then, we take the difference of the image and the thresholds, allowing us to produce the binary image by thresholding this difference matrix with 0.

After producing the binary image, we invert it so that the foreground is now the text and noise. We remove the noise in the background by performing a morphological opening operation. Afterwards, we perform a closing operation to close gaps in character edges. Finally, since the chosen window size may have created gaps in the middle of some characters with thick strokes, we perform small hole filling. Afterwards, we return the output image to the original polarity in order to obtain the final binarized output.

2) Scans and PDF Screenshots: Scans and PDF screenshots should not need to have lighting correction performed. To prevent excessive runtime and unnecessary distortion, we perform Otsu's method for these images.

B. Skew Correction

We then correct the binarized image for rotations. To compute the dominant orientation, we take the Hough transform. Fortunately, most equations have multiple horizontal lines, such as fraction bars, equal signs, and negative signs, as seen below. This means that the dominant orientation is usually given by the correct, horizontal orientation. In order to guard against large magnitude peaks given by long diagonal lines, such as the diagonal division bar in Figure 1, we consider the top four magnitude Hough peaks and choose the mode of the orientations.

Also, while the image in Figure 2 is rotated by a small positive angle, the orientation in Hough space is given as the angle between the normal and the horizontal axis, which is the complement of our desired angle.

For visualization, in Figure 3, we want θ, but the Hough transform gives us ρ, but we easily correct for this by subtracting $90°$. After derotating the image we perform edge softening if the detected rotation angle is large. Significant rotations introduce quantization errors along straight lines, manifesting in jagged edges that are detrimental to character recognition. Currently, we soften the edges using an opening operation. The output of the Skew Correction algorithm is fed into the Segmentation algorithm of the Character Matching portion of the pipeline, described in the next section.

C. Segmentation

Because characters are matched individually, the characters are first extracted from the derotation algorithm output. We considered several approaches, such as recursive projection segmentation [2], but decided to use centroids and bounding boxes of edge maps for simplicity and for the ability to extract characters surrounded by others (such as under a square root). First, the edge map is obtained by eroding the inverted image and then XORing that with the original inverted image, resulting in a white edge map on a black background. Then, for each edge, we extract its centroid, bounding box and convex hull. This successfully extracts many characters, but we still must handle numerous edge cases. For example, this method can produce extraneous segmentations with characters such as B that have inner contours, as shown in Figure 4.

Thus, if a character's convex hull is fully contained within another convex hull, we examine the edge map to determine if the outer character fully surrounds the inner character. If so, we discard the inner segmentation. This additional edge check is necessary before discarding the inner segmentation for proper behavior on equations with a square root symbol,

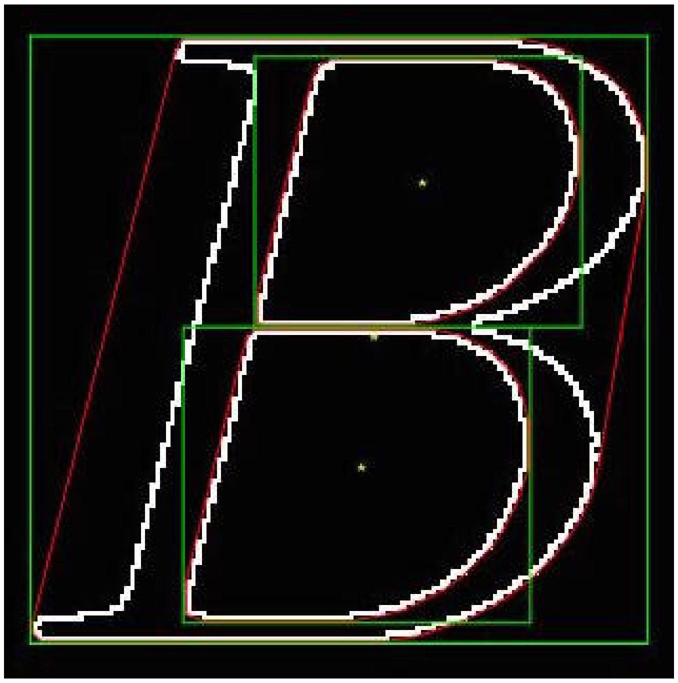

Fig. 4: Segmented B

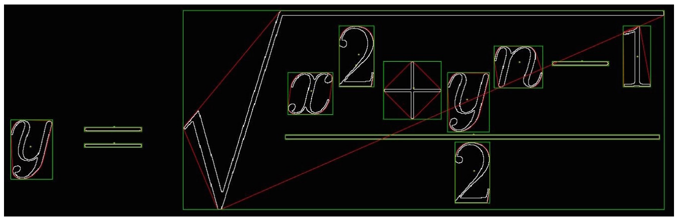

Fig. 5: Segmented Square Root

such as in Figure 5 where several characters underneath the square root are completely within its convex hull.

Finally, we extract each bounding box region and keep the largest single character within each region to only extract the character of interest in cases like the square root where other characters are contained in its bounding box.

D. Character Identifier

Afterwards, we build an identification profile for each segmented character. The identification profile is a twenty-two element vector used to match the segmented character with its equivalent in the template database. We chose these characteristics to be invariant to translation, scaling, and rotation. Specifically, we integrated the usage of the following features that have been used separately in literature: the normalized central moment of inertia, circular topology [5], and Hu Invariant Moments [4]. The first element in the vector is the normalized central moment of inertia [5]. The central moment of inertia is translation- and rotation-invariant; normalization renders it scale-invariant. The following equation calculates this:

$$I_N = \frac{\sum_{i=0}^{N}((x_i - c_x)^2 + (y_i - c_y)^2)}{N^2} \tag{1}$$

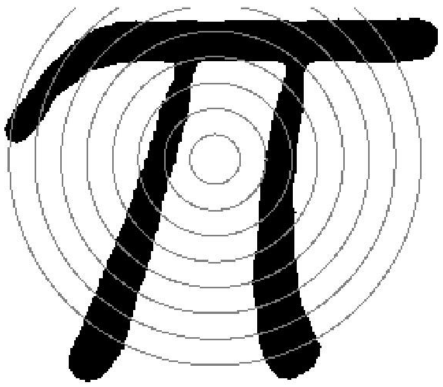

Fig. 6: Count: $[0\ 3\ 3\ 4\ 3\ 4\ 4\ 4]$

where N is the total number of character pixels in the image, and c_x and c_y are the coordinates of the character's centroid.

The next fifteen elements are extracted from circular topology. Since the circle is geometrically rotationally invariant, the topology of the character along a circular path is likewise invariant. Eight equally spaced concentric circles are centered at the characters centroid. We determine the spacing of the circles by finding the maximum distance from the centroid to an edge of the character and then dividing by $k+1$ where k is the number of circles. We used $k = 8$ based on empirical results and previous literature [5]. The first eight elements of these topology elements are the number of times each circle crosses the character.

Note that we used $k+1$ as the spacing but only considered the first k circles. We found that having the outer circle on the edge of the character created topological aliasing errors due to imperfections on the character edges.

Figure 6 shows the eight circles along with the topology count from the inner circle to the outer circle for the template character of π. The counts can be obtained by unwrapping a circle into a line segment and then counting the number of black segments. Note the extra count for the second circle from double counting on the right where the circle is "cut" to unwrap it. Removing this extra count did not improve matching performance, so it was left.

The last seven topology elements measure the spacing between the character crossings for each circle to differentiate between characters with equal numbers of circle intersections. We take the two longest background arcs from each circle, find the difference, and normalize by the circumference of that circle:

$$D_i = \frac{arc_2 - arc_1}{\text{circumference}} \tag{2}$$

This is done for the $k-1$ largest circles; we ignore the

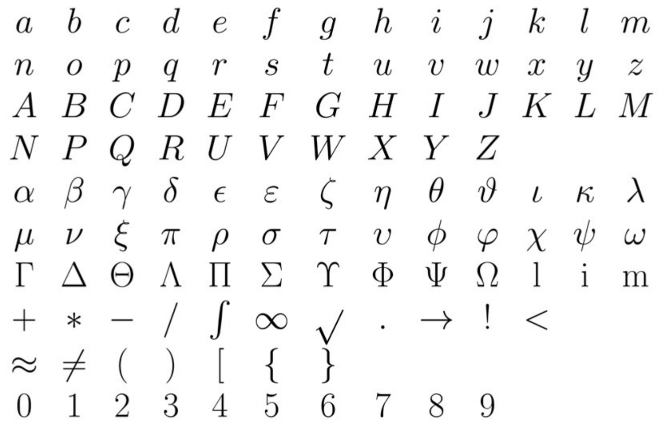
$a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m$
$n\ o\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$
$A\ B\ C\ D\ E\ F\ G\ H\ I\ J\ K\ L\ M$
$N\ P\ Q\ R\ U\ V\ W\ X\ Y\ Z$
$\alpha\ \beta\ \gamma\ \delta\ \epsilon\ \varepsilon\ \zeta\ \eta\ \theta\ \vartheta\ \iota\ \kappa\ \lambda$
$\mu\ \nu\ \xi\ \pi\ \rho\ \sigma\ \tau\ \upsilon\ \phi\ \varphi\ \chi\ \psi\ \omega$
$\Gamma\ \Delta\ \Theta\ \Lambda\ \Pi\ \Sigma\ \Upsilon\ \Phi\ \Psi\ \Omega\ \mathrm{l}\ \mathrm{i}\ \mathrm{m}$
$+\ *\ -\ /\ \int\ \infty\ \surd\ .\ \rightarrow\ !\ <$
$\approx\ \neq\ (\ )\ [\ \{\ \}$
$0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$

Fig. 7: Character Palette

innermost circle which is nearly always either completely background or completely character.

Finally, we calculate the Hu Invariant Moments [6]. The seven Hu moments are invariant to translation, scaling, and rotation [7]. The first Hu Moment is essentially the normalized central moment of inertia previously calculated, so only the last six moments are used. With μ_{pq} as the central moment of order p and q:

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

E. Matching

For feasibility, we defined a character palette that contains the possible characters that can be identified, shown in Figure 7. We created a template database by pre-calculating the character identifiers for every character in the palette and associating with the appropriate LaTeXcode.

Then, we classified each character found by the segmentation algorithm through a nearest neighbor classifier using Manhattan distance, which produced the best results compared to other types of distances tested, and we output the best match as the detected character. This along with the centroid and bounding box with respect to the original equation are passed to the equation assembly function.

F. Equation Assembly

We assemble each equation sequentially from left to right using each recognized character's bounding box. Notably, as we process each character, we keep track of a "previous" centroid to detect the presence of superscripts and subscripts. This is always the centroid of the previous matched character. For example, if the previous character were a part of a fraction, we need to consider the centroid of the fraction bar instead. The basic flow of the algorithm is as follows:

If the current character's bounding box does not overlap with any subsequent bounding boxes, we add the character to the equation directly using the following rules:

1) If the lower-left corner of the bounding box is above the height of the previous centroid, either a superscript or the end of a subscript has occurred. We keep track of a subscript flag to differentiate the two.
2) If the upper-left corner of the bounding box is below the height of the previous centroid, we carry out the opposite operation from above.
3) Otherwise, if the character is a LaTeXcontrol sequence, $\backslash$ is pre-appended to the character and a space is post-appended. Then the aggregate is added to the equation.
4) Otherwise, the character is added directly with no modifications.

If the current character overlaps with subsequent bounding boxes, we check for specific cases:

1) If the character is '-' and only overlaps with a singe other '-', we append = to the equation and skip the overlapping '-'.
2) If the character is a square-root, we select all of the subsequent overlapping characters and recursively assemble a sub-equation. Then, we append the appropriate LaTeXcommand to the total equation.
3) If the character is '-' and does not only overlap with a single other '-':
   a) If the width of the - is a large portion of the width of the union of the overlapping bounding boxes, we recognize the overlapping region as a fraction. We divide the characters into denominator and numerator sets. Then we recursively assemble these two sets into sub-equations. Finally, the fraction is assembled into the proper LaTeXformat and appended to the total equation.
   b) Otherwise, it is a negative sign, and we continue to the next case.
4) We look for a limit-enabled control sequence (summation, integration, and product), and then recursively assemble the equations of the upper and bottom limits as appropriate.

This logic is limited in the number of LaTeXconstructions supported. For example, we do not support all of the numerous LaTeXcontrol sequences. Also, variable modifiers such as the dot or bar notation will not be correctly assembled.

IV. Results

The binarization method yielded no problems on our test cases. Compared to performing Otsu's Method Adaptive Thresholding, our heuristic implementation runs in about half the time. The results are shown in Table I.

$$Q(z) = \frac{1}{\sqrt{2\pi}} \int\limits_{z}^{\infty} e^{\frac{-x^2}{2}} dx$$

Fig. 8: Sample ”Clean” Equation

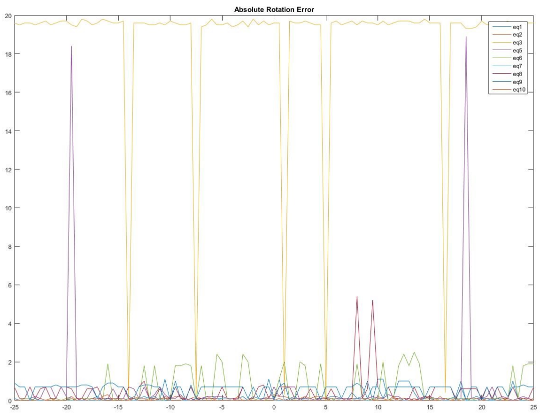

Fig. 9: Derotation Error

TABLE I: Time for Binarization on Demo_equation.jpg

	10 Runs	20 Runs
Mean Threshold	5.743 seconds	11.529 seconds
Otsus Method	11.232 seconds	23.221 seconds

We also performed derotation tests on our fifteen ”clean” equations. This ”clean” set was obtained by exporting direct LaTeXcompilations as .jpg files. See Figure 8 for an example. We manually rotated each ”clean” equation by an angle and ran the derotation algorithm on the resulting image. We then defined the error to be the absolute value of the difference between the detected angle and the rotated angle. The results are graphed in Figure 9. Our algorithm resulted in errors below 2 degrees in magnitude for most of our test equations. Equation 3 is the outlier. This equation was shown above in Figure 1. Despite the guards described in the Derotation section, our algorithm still registers the diagonal division bar as the dominant orientation for most rotation angles.

We also tested the matching algorithm in several ways. First, the database characters were rescaled before performing character matching. Over a scale range of 0.5 to 1.25 the overall correct percentage was 95.3% as shown in Table II.

TABLE II: Robustness to Scaling

Scale	**# Correct**	**% Correct**
0.5	103/116	88.8%
0.75	111/116	95.7%
1.0	116/116	100.0%
1.25	112/116	96.6%

Next, we ran the full pipeline on the fifteen equations in the clean set. We observed a matching performance of 147/157 characters for a rate of 93.6%. The missed characters are partly from false positives and partly from an equation that was incorrectly derotated due to the presence of a Λ which acted like the division bar in the test equation in Figure 1. We also tested the overall system for robustness to rotation. The fifteen clean equations above were rotated from $-25°$ degrees to $+15°$ in increments of $5°$, and the entire process was run on each rotated equation.

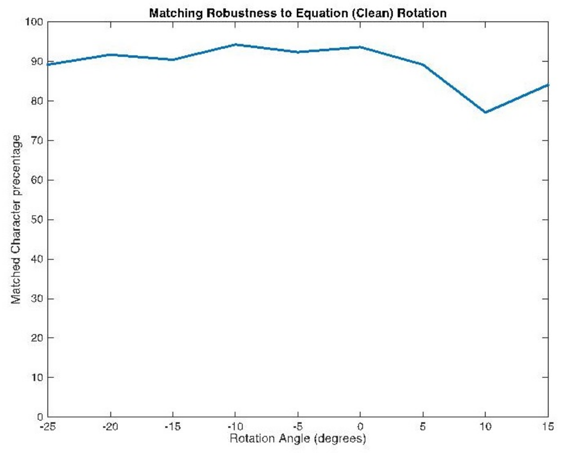

Fig. 10: Robustness to Rotation

Figure 10 shows the detection percentage throughout the rotation test. The overall matching percentage is not significantly affected, except at $10°$ where the deskewing algorithm again was confused by a specific image.

Finally, the algorithm was tested on ten photographs of equations under uneven lighting with a skew, such as in Figure 11. Again, the deskewing method would sometimes be confused by odd characters or by the ”\” division symbol and ”=”. We investigated a modification to our deskewing algorithm in which we kept only the dominant Hough peak rather than considering the top four peaks. This modification actually handled the ”\” worse but handled the odd characters better. The original and modified algorithms produced recognition rates of 98/123 for 79.7% and 91/123 for 74%, respectively. Combining the best equation results from these two methods gave a combined result of 86.2% correctly matched characters.

V. Limitations

The deskewing algorithm presented the most limitations. Most of these limitations manifested only in short equations where diagonal lines in characters were dominant enough to outweigh the horizontal lines in the Hough space. However, in practice, this could be overcome by underlining each equation by hand. In fact, this would be a very reasonable augmentation for extending this project to handwritten equations.

Also, our current character palette for character matching is limited in comparison to the huge variety of LaTeXcharacters available. Our process is modular enough that adding additional characters would be simple; we would just add the character to the palette with the appropriate label and the

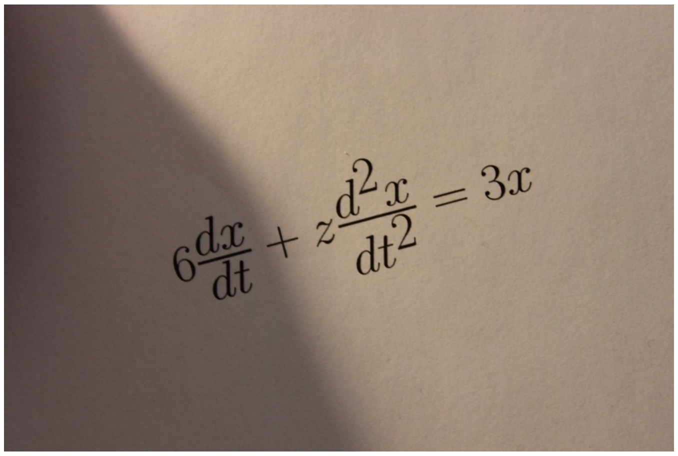

Fig. 11: Sample Test Image

algorithm would proceed as usual. However, we would need to perform additional validations to ensure that no aliasing effects would be introduced. In the same vein, our equation assembly algorithm is limited to recognizing certain patterns. However, in this case, additional LaTeXequation patterns should be easy to implement. However, one major limitation of the assembler is fractional exponents. As seen in Figure 8, the fraction in the exponent is very large. Our current algorithm correctly detects the exponent, and even the nested exponent, but it fails to detect that the following dx is outside of the exponent. Therefore, we have imposed the constraint that all fractions within exponents must occur at the end of the exponent, and the algorithm will automatically end any exponent after a fraction is appended. Further work must be done to remove this limitation.

VI. Further Work

Relevant future additions to our algorithm include the ability to detect and extract an equation from a page with background text and clutter. The current algorithm only supports equations with completely empty backgrounds.

This program has also been architected to be readily extended to recognize handwritten equations. We chose the features for character matching to be invariant to scale and rotation to support the huge variances in handwriting. Possible next steps towards supporting handwritten equations include machine learning to train a linear predictor. This would require the gathering of a training data set rather than a single character palette, but the same feature vector could be utilized.

Further work can also be done to increase the reliability of the current matching algorithm. The matching algorithm exhibits susceptibility to binarized and smoothed characters that were not perfect PDF images. This could be improved by reducing distortion from the binarization algorithm and adding non-perfect characters to the template database.

The robustness could also be improved by implementing a heuristic based error correction algorithm for the matching system. We could experimentally determine characters that tended to produce consistent false positives and take appropriate compensation actions for these characters. For example, we have noticed that the ”z” is often misclassified as an uppercase ”Z”. Knowing this, we could add an additional check when detecting the uppercase ”Z” before final classification and check whether the top is straight or curved.

Finally, our algorithm is rather slow. We made some attempts to decrease our algorithm's runtime such as implementing the heuristic based adaptive thresholding and investigating frequency based deskewing algorithms [8]. However, speed was not an objective in our original project formulation. As the project matures, though, algorithm optimization will become a central goal for practicality. Future tasks related to this include migrating to a compiled language such as C++ and rigorous algorithm analysis.

Acknowledgments

The authors would like to thank Professor Gordon Wetzstein for making this project possible and Kushagr Gupta for invaluable consulting.

Appendix

The project workload was divided between the group members:
Jim Brewer: Data Acquisition, Character Segmentation, Character Identification, Character Matching, Testing.
James Sun: Data Acquisition, Lighting Compensation, Image Derotation, Equation Assembly, Testing.

References

[1] Microblink. (2015, Nov.) Photomath. [Online]. Available: https://photomath.net/en/

[2] S. Naqvi and U. Sikora. (2015, Nov.) Ee368/cs232 digital image processing. course project. [Online]. Available: http://web.stanford.edu/class/ee368/Project_13/index.html

[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] L. Torres-Mendez, J. Ruiz-Suarez, L. Sucar, and G. Gomez, “Translation, rotation, and scale-invariant object recognition,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 30, pp. 125–130, Feb. 2000.

[6] V. Muralidharan. (2015, Nov.) Hu's invariant momentss. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/52259-hu-s-invariant-moments

[7] M. K. Hu, “Visual pattern recognition by moment invariants,” *IRE Trans. Info. Theory*, vol. IT-8, pp. 179–187, 1962.

[8] M. Kaur and S. Jindal, “An integrated skew detection and correction using fast fourier transform and dct,” *International Journal of Scientific & Technology Res*, vol. 2, Dec. 2013.