Mobile Camera Based Calculator

Liwei Wang
Department of Electrical Engineering
Stanford University
Email: liweiw@stanford.edu

Jingyi Dai
Department of Electrical Engineering
Stanford University
Email: jingyi89@stanford.edu

Li Du
Department of Electrical Engineering
Stanford University
Email: lidu@stanford.edu

***Abstract*—This article introduces the design and implementation of an Android-platform based calculator, which can recognize the formula captured by a mobile phone camera, compute the result, and display it on the screen. This application enables a faster calculation on a mobile device by avoiding inputting the formula to some device.**

Keywords-formula recognition; OCR; Android

I. INTORDUCTION

The idea of this application is inspired by an App “Youdao”, which can detect and translate a word captured by the camera of mobile phone into another language. Our application can detect a printed formula with the following operators: plus, minus, multiplication and division. Also, the captured picture can be rotated, perspective, and with some clutter.

The application we developed applies region labeling, the automatic techniques of text detection, OCR (optical character recognition) and formula computation [2]. Most of the work is done on the server, instead of on the phone, because of the high complexity cost.

A. *Prior and Related work*

1) Connected-component labeling: It is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled. This technique can detect connected regions in binary digital images, which can label independent objects in an image [3]. Based on the labeling, some properties can be applied to improve the quality of the image.

2) Optical Character Recognition: OCR, is a tool for extracting text from images, which can help us detect the numbers and operands of the formula. There are many versions of OCR, and the one we use is Tesseract [1], supported by the server blackhole1.stanford.edu and called by a python script.

3) Client-Server Communication: Since the memory of mobile phone is much less than a computer, it would be very hard for a phone to complete the complicated computation, so the picture captured by the phone can be transmitted to a server, which contains a corresponding php script to invoke the server-side application to recognize the formula.

II. System Flow

The equation recognition and calculation algorithms are based on the following steps:

0) Convert to grayscale image
1) Denoise and Deblur
2) Binarization
3) Formula Feature Filtering
4) Rotation and Perspective Adjustment
5) Optical Character Recognition
6) Formula Correction
7) Formula Calculation
8) Display the Calculated Result

A. *Step 1 - Denoise and Deblur*

The grayscale image then undergoes nonlinear noise reduction/sharpening. This is done by first feed the image into HPF to sharpen the edges. The HPF will boost noise so we then use soft coring to suppress the noise (Figure 1).

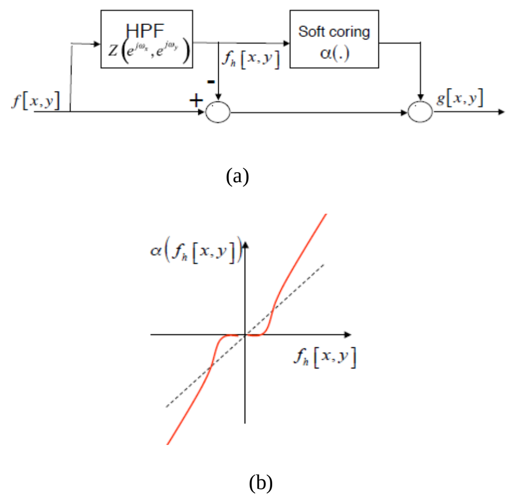


Figure 1. Denoise soft coring

To deal with images blurred due to out-of-focus effect, we adopt the following algorithm:

```
Im := Input Image
For Iteration = 1:NumIterations
    Im_d = dilate(Im, W)
    Im_e = erode(Im, W)
    Im_h = 0.5(Im_d + Im_e)
% Perform the following test for each pixel
    If Im > Im_h
        Im := Im_d
    Else
        Im := Im_e
    End
End
```

The rationale behind this algorithm (dilation and erosion are in grayscale) is we want to make the bright part brighter and dark part darker to sharper edges and get rid of the out of focus effect.

B. Step 2- Binarization

To binarize the image taken under non-uniform lighting condition, we need to compensate this non-uniformity before we do Otsu's global thresholding. To do this, the image is first dilated using local max by a 31X31 SE and then processed by a rank filter. The resulted image is the background of the original image. By subtracting it from original image, we get a uniformly lighted image which is safe to undergo global thresholding.

C. Step 3- Localization

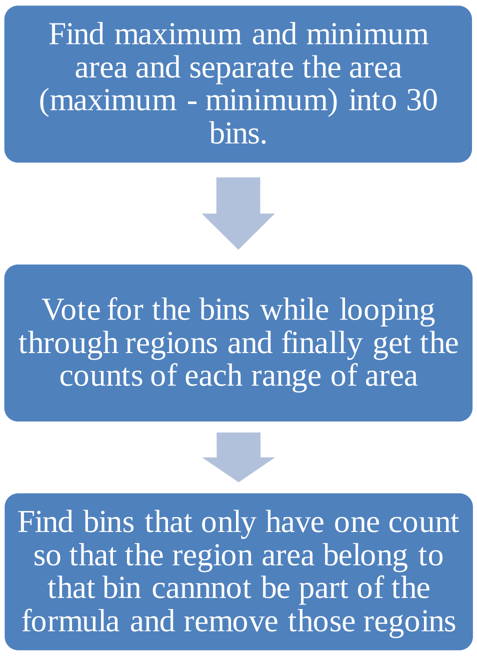
The localization of the equation in the image is done in three phases:

Phase1: Remove extremely small regions

We define these extremely small regions as having area of only order of 10 pixels, because these regions cannot be part of the formula.

Phase 2: Get rid of regions having area of small amount of counts in the image

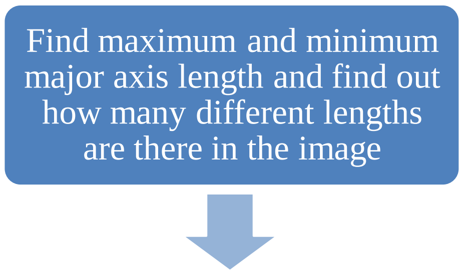
We are assuming regions belonging to the formula should have similar areas which occur at least three times. Because we do not know the area of formula (area changes depending on zoom in and out of targets when the picture is taken), we have to find statistics of the areas of regions and process the image according to these dat

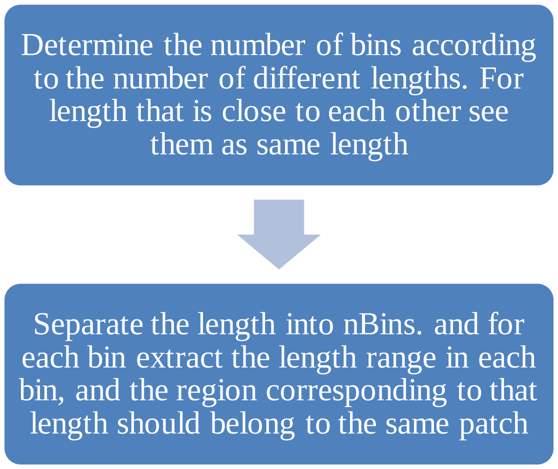

Find maximum and minimum area and separate the area (maximum - minimum) into 30 bins.
Vote for the bins while looping through regions and finally get the counts of each range of area
Find bins that only have one count so that the region area belong to that bin cannnot be part of the formula and remove those regoins

This process is only done if the relative difference between maximum and minimum area is larger than 10 (if the image only contain target then there is no need to filter according to area), otherwise we skip this step.

Phase 3: Separate image into patches

After filtering the image according to area property, the regions left should not contain clutters having unusual area. Normally what has been removed are chunk region in the background. We then need to find patches in the image that can potentially be out target. Usually these include formula and other text clutter in the image. To separate them, we're assuming that the texts in the same patch should have the same size and in order to separate one patch from another, texts in different patch should have different size. This assumption is reasonable in the sense that most of the time clutters are of different size from target, otherwise there is no way to distinguish one from another. The property we use in this section is major axis length:


Find maximum and minimum major axis length and find out how many different lengths are there in the image


Determine the number of bins according to the number of different lengths. For length that is close to each other see them as same length

Separate the length into nBins. and for each bin extract the length range in each bin, and the region corresponding to that length should belong to the same patch

Each resultant patch has the same dimension as the original image. What we actually do is removing regions not belonging to that patch. A bounding box is then applied to the original image using the patches to determine which part in the image we want to clip out. After this section, we have a list of candidates which will be transformed and fed to OCR test.

D. Step 4 – Rotation and Perspective Adjustment

The images taken are highly possible to be rotated or have some perspective distortions. This will lower the recognition ability of OCR. We did an adjustment on the rotated or oblique images before feeding each detected candidates into OCR.

We first built a database that contains a wide range of different rotated or oblique images. These images were fed into OCR. A statistic of the OCR results was summarized. We found out that the version of OCR we used is relatively sensitive to rotation and oblique angles, for instance, an equation rotated by about 10 degree cannot be recognized correctly. Hence, an algorithm should be built to adjust the images to an appropriate angle.

Rotation Adjustment:

Considering the formula might not be in a horizontal line, rotation is needed. First label all the regions of the image, and find their centroids. If the y values (the value in the vertical direction) of the first label and the last label are very similar (the difference is less than 2), rotation is not needed, because the formula is almost horizontal. If the difference is bigger than that, calculate the horizontal difference and vertical difference of the first and last label, and then calculate the corresponding angle, and finally rotate the image to horizontal level.

Perspective Adjustment:

We use the method of perspective transformation to adjust the oblique image. First, we know it is easier and more obvious to do perspective transformation if there are reference lines on the image. However, this is highly unlikely to happen in realistic cases. So we decided to implement it assuming there is no reference line on the image. Since this is a pretty hard task, we only implement one case of perspective distorted image, which is shown in Figure 2 below. Considering the user experience of our mobile application, users are highly likely to take pictures by this oblique angle, other cases merely happen. Hence, our simplification is reasonable.

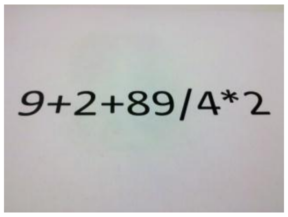


Figure 2. The most possible case of perspective distortion

The input of the adjustment algorithm was the detected patches bounded by a bounding box. Assuming the bounding box of the formula is an isosceles trapezoid, we need to find three most important points of the trapezoid, i.e. the topleft, topright, bottomleft points (point A, B, C shown in Figure 3).

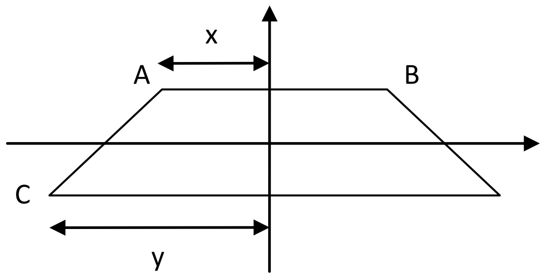


Figure 3. Geometry structure of the oblique image patch

Since it is highly unlikely to find the exact topleft, topright, and bottomleft points, we set up some standards to determine whether the points we found satisfies the condition to be a correct point we need. These standards of judgment are listed below:

- Start from the 1st row, search for the left and right point on each row. If the distance between two points exceeds $0.8 * \text{width of the image}$, we can treat the two points as the topleft and topright points (shown as point A, B in Figure 3 above).
- Start from the last row of the image, find the left point. If it is on the left of the topleft point we found before, we can recognize it as the bottomleft point (shown as point C in Figure 3 above).

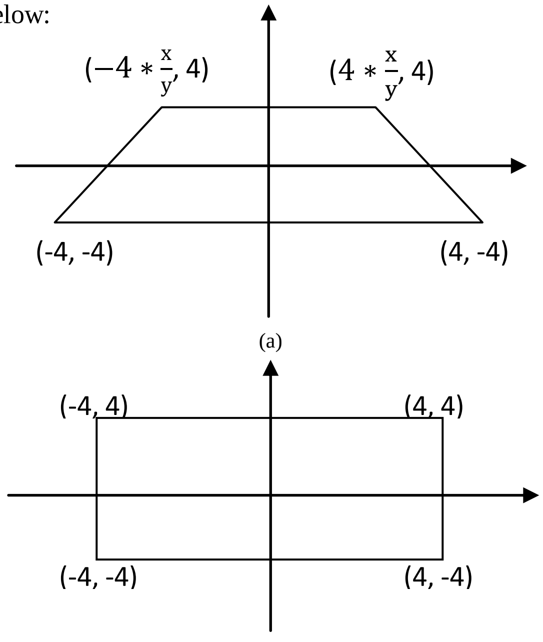
Based on the coordinates of the points we found, we can calculate the oblique rate of the patch.

We choose four basic points on u-v coordinate after transformation, which are (4,4)(-4,4)(-4,-4)(4,-4). Hence, the corresponding coordinate on the x-y axis before transformation can be calculated as:

$$|\text{x axis value}| = 4 * \frac{x}{y}$$

where x, y are shown in Figure 3 above.

Using imtransform function in MATLAB[4], the equation should fit into the bounding box shown in Figure 4 below:



(b)

Figure 4 (a) bounding box of formula before transformation (b) bounding box of formula after transformation

However, because of the variety of the images and no reference line are assumed to be on the images, this method may not be perfect. Further work should be done to improve the algorithm.

E. Step 5-8 – Formula Recognition, Correction, Calculation, and Result Display

1) OCR

We used python to perform OCR on blackhole1 server, and wrote a wrapper between matlab and python.

2) Correction

We implemented a post processing algorithm in the following steps:

- Remove spaces in the string
- Replace some special characters which is highly possible to be numbers, such as replacing 'l' by '1', 'O' by '0'.
- Calculate the probability of occurrence of alphabets. A patch has the lowest probability is most likely to be the correct formula we need.

3) Calculation

Since the result returned by OCR is a string, we need to change it to numbers and operators. We extracted the operators and operands of the result, and established a function with the operators using MATLAB inline function. Then we replaced the variables with the operands to get the numerical expression of the formula.

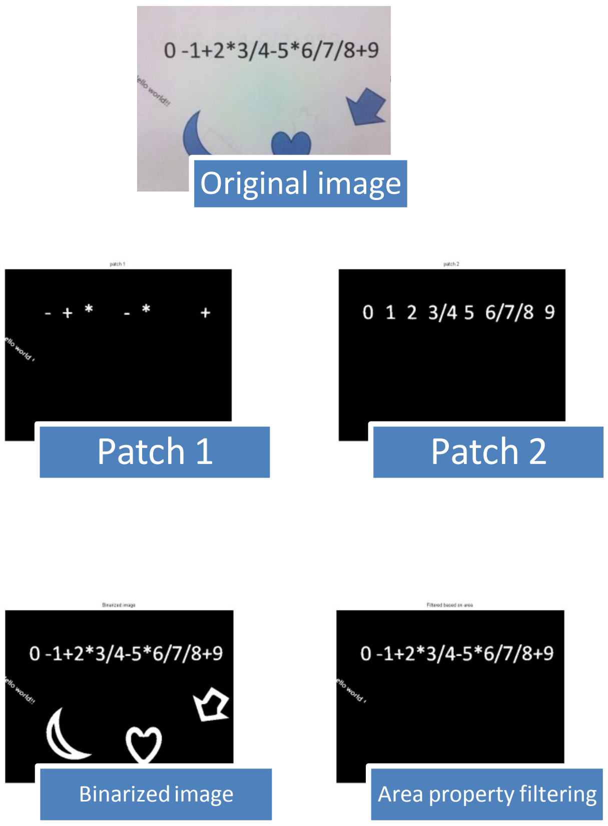
4) Result Display

A white image was created. Results were written on it as text. Then the image contains the result was sent back to Andriod phone and displayed on the screen.

III. Tests and Results

The followings are two examples of flow of the testing result.

General Flow:

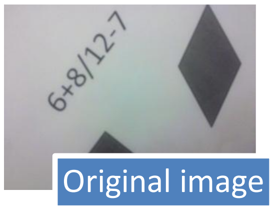


candidate to be sent to OCR

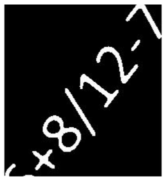
0-1+2*3/4-5*6/7/8+9
=8.9643

Result

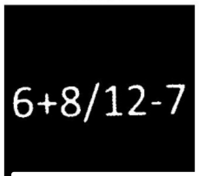
Process rotation:



Original image

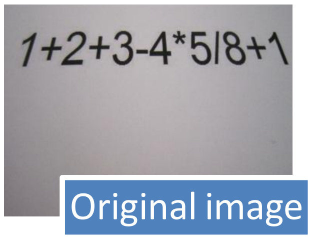


Before transform

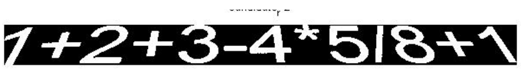


After transform

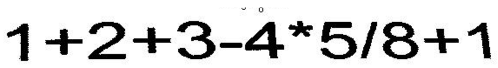
Process perspective adjustment:



Original image



Before transform



After transform

Statistics

The followings are the statistics of test results. Our training set and test set include about 100 images which can fall into the following categories:

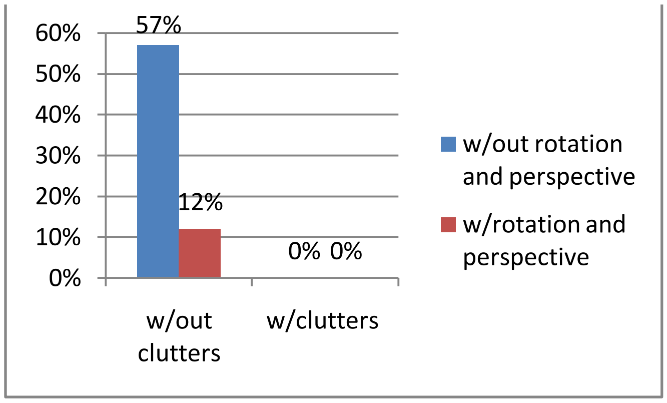
With clutters
- Without image distortion
- With image distortion

Without clutters
- Without image distortion
- With image distortion

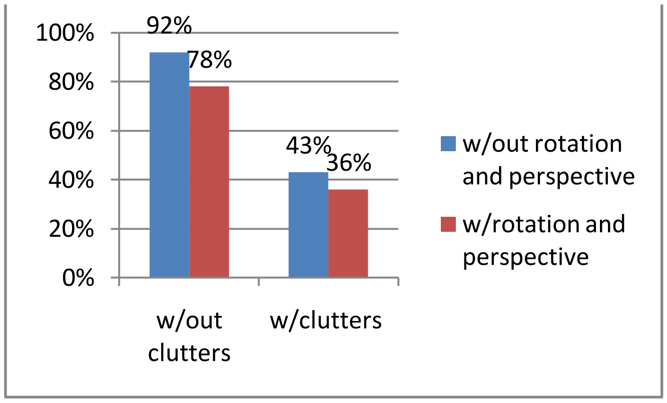
If we feed the un-preprocessed images directly into OCR, the result is fairly unreliable when the condition under which the image is taken is bad.

The contrast between with and without preprocessing illustrates that our preprocessing of image and post processing on the results we get from OCR can improve the performance to some extent.

	w/out clutters	w/clutters
w/out rotation and perspective	%	0%
w/rotation and perspective	12%	0%



	w/out clutters	w/clutters
w/out rotation and perspective	75%	0%
w/rotation and perspective	12%	0%



IV. CONCLUSION

The current edition of this application can detect formula in different conditions, such as rotation, perspective and so on, and the average accuracy is about 60%. Though this application can complete the fundamental functions of a camera-based calculator, some improvements could make it better with higher accuracy:

- Remove clutter of the same size with the formula
- Remove "number" clutter besides "letter" clutter
- Calculate more than one formula
- Detect the formula from a more complicated background
- Calculate more complicated formula which might include sin, log, root square and so forth

ACKNOWLEDGMENT

We would like to thank Professor Bernd Girod for giving us so wonderful lectures on advanced digital image processing techniques. We also would like to thank David Chen, Derek Pang, and our mentors Huizhong Chen and Sam Tsai for their help.

REFERENCES

[1] Derek Ma, Qiuhau Lin, Tong Zhang, Mobile Camera Based Text Detection and Translation

[2] http://code.google.com/p/tesseract-ocr

[3] http://en.wikipedia.org/wiki/Region_labeling

[4] www.mathworks.com/help/toolbox/images/ref/imtransform.html

Appendix – Individual Work Breakdown

Localize each patch using region labeling.	Liwei Wang
Perspective distortion adjustment. Create bounding box of each patch. Formula correction.	Jingyi Dai
Rotation distortion adjustment. Formula calculation.	Li Du
All other code and related tasks.	Done by us together