Ben Gurion University

OCR of Math Expressions

Optical Character Recognition System of Math Expressions & Testing Framework

Yakir Dahan*

Vitali Sepetnitsky†

Work Report

February 17, 2011

Abstract: In this project we present an application for performing OCR of mathematical expressions, based on segmentation and correspondence finding, based on correlation coefficient. In addition we present a framework which translates the results into a structural data using clues regarding the syntax of mathematical expressions. The framework allows to convert the result to a format compatible with Wolfram Alpha computational engine and transfer it to the engine.

* Student at Department of Software Engineering, Ben-Gurion University, Beer-Sheba, Israel
† Student at Department of Software Engineering, Ben-Gurion University, Beer-Sheba, Israel

Contents

Part I
Project Workflow Description

1 Introduction

1.1 Motivation

Have you ever been looking on a printed image of a mathematical expression, or even on an expression you typed using a formula editor (like the one of Microsoft Word), and were interested to know its definition. However, you don't want or don't have the time to perform the exhausting process of copying the expression to a computational engine command line while preserving the correct syntax of the expression. Moreover, this process requires you to switch between the engine's window and the image which is a very uncomfortable process.

A lot of effort, time and maybe nerves :) would be saved if you could take your mobile phone, capture the image and got automatically its analysis performed by some computational engine.

If you agree with us (and you must agree if you ever learned a calculus course :-)) than you got into the right place! The motivation of this project is to make the first steps towards the wonderful application described above. **In this project we take a mathematical expression given as an image, perform an OCR and further analysis process, retrieve the expression from the image and send it to Wolfram Alpha computational engine.** The project combines a computer vision approach, of performing OCR of single characters and symbols, and some kind of artificial intelligence approach, which is applied after the basic OCR process. At the second stage we use syntactic rules of mathematical expressions to repair recognizing errors of the OCR stage, thus trying to reach a syntactically correct expressions and decrease the inaccuracies of the OCR.

Currently, we perform OCR by using the same approach used to solve the correspondence problem when performing stereopsis - by maximizing a correlation similarity measure computed using a set of pre-defined templates.

The application we propose here can be usable for:

- People working with mathematical expressions, especially students.
- Finding errors and inconsistencies in a typed mathematical expression.
- Rewriting and redesigning the typed expression.
- Transferring the expression between different applications and text files.

1.2 OCR - Definition and Applications

OCR is an abbreviation for *O*ptical *C*haracter *R*ecognition. OCR allows a machine to recognize characters through an optical mechanism. The naming OCR refers to all technologies which perform translation of scanned/captured images of text (which can be handwritten or typewritten) to a standard, machine-encoded text with the same interpretation. The purpose of an OCR system is to classify optical patterns, contained in the image, to corresponding alphanumeric or other characters. After performing OCR, a further processing can be applied to the text, such as text-to-speech and text mining. OCR is a field of research in computer vision and artificial intelligence.

1.3 OCR evolution – Important dates

1929 A first patent on OCR was first obtained in Germany by Gustav Tauschek, for a mechanical device which performed OCR using a photodetector and templates.

1950 The first commercial OCR application was invented in USA, by David H. Shepard, a cryptanalyst at the Armed Forces Security Agency. A corporation was founded and named *I*ntelligent *M*achines *R*esearch (IMR). It delivered machines which converted printed texts which were especially credit card numbers and teletype typewritten messages, to a computer text.

1973 The first development of an omni-font OCR system – A computer program which was capable of recognizing text printed in any normal font. This program was intended to the blind people in order to have the computer read text to them loudly.

1992-1996 The Information Science Research Institute (ISRI) conducted the Annual Test of OCR Accuracy in order to encourage the development of OCR systems for understanding machine printed documents.

1.4 Current state of OCR technology

Today, there exist a lot of commercial applications for recognition of Latin script. According to a study based on recognition of 19th and early 20th century newspaper pages concluded that character-by-character OCR accuracy for commercial OCR software vary from 71% to 98%. Other areas in OCR - including recognition of hand printing, cursive handwriting, and printed text in scripts other than Latin —are still a subject of active research.

2 Problem Domain and Assumptions

In order to define the problem formally we decided about an initial set of characters and mathematical symbols that our application should recognize:[1]

1. Capital English letters ($'A'$, ..., $'Z'$) and small English letters ($'a'$, ..., $'z'$).
2. Small Latin letters which are usually used in mathematical expressions: $'\alpha'$, $'\beta'$, $'\gamma'$, $'\delta'$, $'\varepsilon'$, $'\theta'$, $'\lambda'$, $'\pi'$ and $'\sigma'$.
3. Digits ($'0'$, ..., $'9'$).
4. Binary operators: $'+'$, $'-'$, , $'*'$, $'\div'$ and $'/'$.
5. Boolean operators: $' <'$ and $' >'$.
6. Parentheses signs: left parenthesis ($'('$) and right parenthesis ($')'$).
7. The root character ($'\sqrt{}'$).
8. The equals ($' ='$) character.
9. Sum ($'\sum'$) and product ($'\prod'$) symbols.
10. Integral ($'\int'$) and derivative ($'''$) symbols.
11. The infinity symbol ($'\infty'$).

As it was said above, the goal of this project was to combine **computer vision approach** which allow us to perform OCR of single symbols. Furthermore we wanted to use syntactic rules of mathematical expressions to repair errors of the OCR stage and construct correct expressions from the read characters, thus leading us to application based on some kind of **artificial intelligence**.

We assume that the images given to the application, contains only characters from the set given above, which are syntactically correct. For example, the following expressions aren't acceptable: $+^6$, e^- and $\sum^k$. The second assumption is useful especially for the operation of retrieving the mathematical expression from the read characters.

In addition, we assume that the intensity of the characters seen on the image is clearly different from the intensity of the background. As it can be seen from the next section, our algorithm can deal with noisy images, but it is limited to images in which the characters can easily be extracted from the background.

3 Principles of the Algorithm

Our approach for solving the problem was divided to 4 stages:

1. First, we perform a simple image processing of the image using the following operations:

   (Ła) Converting the image to gray scale by eliminating the hue and saturation information while retaining the intensity. We remind here that by assumption the intensity of the expression's characters on the image should be strictly lower than the intensity of the background and therefore, converting to the image to gray scale, shouldn't change drastically the number of expression's pixels.

   (Łb) Converting the gray-scaled image to black&white, by using Otsu's method. Since we assume that the initial image consists of a background and the mathematical expression only, we should get a white background with characters and mathematical symbols lying on it.

   (Łc) Removing all connected groups of pixels with less than 9 pixels - made in order to reduce noises (but may cause problems in handling small fonts).

   (Łd) Cropping the result image to be of size of the minimal bounding rectangle - for retaining the expression only and omitting the background.

2. Next, we has to divide the given expression into single characters with a true order. This problem is solved easily by thinking about the way we read a mathematical expression, constructed from the symbols given above. Usually the expression is read with **left-to-right** and **top-to-bottom** order. Fortunately, the number of exceptional expressions is limited (according to the set of symbols given above) and refers only to the second rule:

   (Ła) The expression $\sum_{k=1}^{10} x$ is read as "*sum* $(x,\ k = 1\, to\, 10)$". Similarly, the expression $\prod_{k=1}^{10} x$ is read as "*product* $(x,\ k = 1\, to\, 10)$". In other words, these expressions are read **left-to-right** but **bottom-to-top** (regarding the $k = 1$ and 10).

   (Łb) The expression $\int_{-\infty}^{\infty} x^2$ is read as "integral from $-\infty$ to ∞ of x^2. Here, again, the expression is read **left-to-right** but **bottom-to-top**.

   For the reasons stated here, we assumed that OCR can be performed on the expression's characters in the left-to-right and top-to-bottom order. The process of inverting the order of characters in the special cases, is done during the construction of the full expression from the single characters read by the OCR process.

   The division of the image into single characters and symbols was done as described at section 3.1.

3. For each character/symbol we got from the division described above, we has to perform and OCR and return an interpretation of this character. The interpretation was done by maximizing a similarity measure as described at section 3.2. In order to perform the process of finding the most corresponding interpretation of the image, we first re-size it to a standard size which is the size of all the templates supplied (for details see section 3.2) which is 42 rows X 24 columns.

4. We called the result of the above stages a **"pseudo-result"**, since, it isn't the final result of the algorithm - the final result is retrieved after performing a conversion of the pseudo-result to a mathematical expression, using syntactic clues of expressions of this type.

3.1 Dividing the image into columns, rows and clumps

Given a binary (black&white) image, let's define some definitions:

A valid column is a column whose height equals to the height of the image and width is at least 1. In addition, all its vertical, unit width, columns, have at least one black pixel.

A maximal valid column is a valid column of maximal size. Meaning, that if another unit-pixels column of the image is added to this valid column (on left or right), it becomes invalid (the new column doesn't contain any black pixels).

A valid row is a row of pixels whose height is at least 1. In addition, all its horizontal, unit width, rows, have at least one black pixel.

A maximal valid row is a valid row of maximal size. Meaning, that if another unit-pixels row of the image is added to this valid row (on top or bottom), it becomes invalid (the new row doesn't contain any black pixels).

A connected component is a set of pixels, in which any pixel has a neighbor from the set in one of the 8 directions (N, NW, W, SW, S, SE, E, NE).

A clump is a connected component of maximal size. Any pixel, external to the clump, will break the connectivity requirement if added to the clump. Dividing the a black&white image into clumps is the simplest kind of segmentation.

When carefully thinking about a given (black&white) image representing a mathematical expressions, if we want to divide it into single symbols, while preserving the order of symbols in the expression, we can use the following way:

Algorithm 1 Division of the image to single symbols

---

1. Divide the image into a set, C, of maximal valid columns.
2. For each column $c \in C$:
   - (Ła) Divide c into a set, R, of maximal valid rows (c is treated as a standalone image) .
   - (Łb) For each valid row $r \in R$:
     i. Divide r into a set of clumps, S - each clump $s \in S$ is assumed to be a single symbol.

In order to demonstrate the above algorithm, we present its operation on the mathematical expression given below:

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Figure 1: The mathematical expression on which Algorithm 1 is applied

After computing the maximal valid columns, we obtain the following set C of columns (the columns are highlighted):

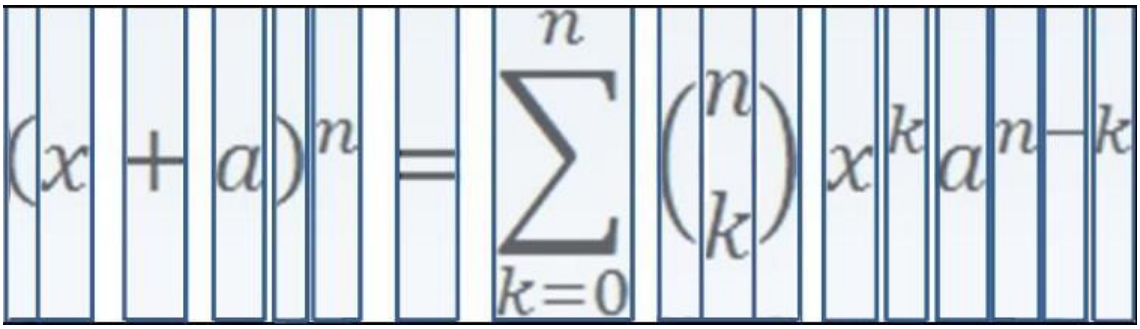

Figure 2: Retrieving columns from the image

Here, we present the maximal valid lines of the two central columns from C:

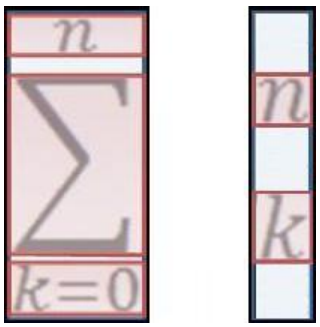

Figure 3: Retrieving lines from two central columns

As it can clearly bee seen, the left column consists of 3 rows and the right column consists of a 2 rows. Now, we divide each row into clumps, which are numbered according to $\langle m, n \rangle$ pattern where m is the number of the row (from top to bottom) and n is the number of the clump (left to right, top to bottom). Here is the result we should get:

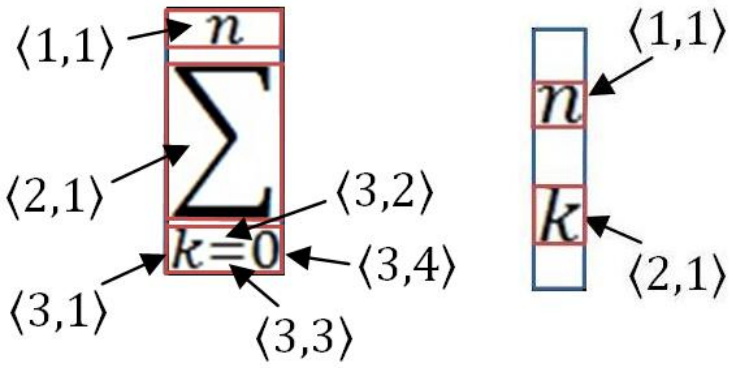

Figure 4: Dividing the retrieved lines into clumps

3.2 Performing OCR of a single character

One of the main parts of this project is taking an image, representing a single character or mathematical symbol, and giving it the interpretation of this character/symbol. We decided to implement our OCR system by using an intensity based method by establishing correspondence of the image to a pre-saved set of characters with known interpretations. The measure of similarity we chose to maximize is a **2-D correlation** which is described below:
Given two matrices A and B of the same size, the 2-D correlation coefficient r is calculated by the following way ($\overline{A}$ and $\overline{B}$ represent the mean values of A and BŁ accordingly):

$$r = \frac{\sum_m \sum_n \left(A_{mn} - \overline{A}\right)\left(B_{mn} - \overline{B}\right)}{\sqrt{\left(\sum_m \sum_n \left(A_{mn} - \overline{A}\right)^2\right)\left(\sum_m \sum_n \left(B_{mn} - \overline{B}\right)^2\right)}}$$

The correlation coefficient ranges from -1 to 1. A value of 1 implies that a linear equation describes perfectly the relationship between A and B, with all data points lying on a line for which B_{mn} increases as A_{mn} increases. A value of -1 implies that all data points lie on a line for which B_{mn} decreases as A_{mn} increases. A value of 0 implies that there is no linear correlation between the variables. We can see that the expression $\left(A_{mn} - \overline{A}\right)\left(B_{mn} - \overline{B}\right)$ is positive if and only if A_{mn} and B_{mn} lie on the same side of their respective means. Thus the correlation coefficient is positive if A_{mn} and B_{mn} tend to be simultaneously greater than, or simultaneously less than, their respective means.
In our case, the two matrices are:

1. A is the image on which OCR should be performed.
2. B is the template image, which represents a character or a mathematical symbol whose interpretation is known.

Moreover, in our case A and B are binary matrices, since we convert the input image to black&white and the templates are stored as black&white. Therefore, high value of a correlation coefficient means that there are a lot of pixels, lying at the same places in A and B, with same values and therefore A can get the interpretation of B (which is known).

In order to use this method, a set of template images should be pre-saved. These templates serve as B matrices when computing the correlation coefficient. For this purpose, we created a set of templates which is supported with our application and can be found in the *templates* folder.

4 The Final Algorithm we Used

Given an image I, assumed to be an image of a mathematical expression:

Algorithm 2 The whole algorithm

1. Compute: $GrayScaleI \leftarrow image\,to\,gray\,scale\,(I)$
2. Compute: $Black\&WhiteI \leftarrow image\,to\,black\,and\,white\,(GrayScaleI)$
3. Initialize: $pseudo\_result \leftarrow \emptyset$
4. Divide $Black\&WhiteI$ into a set of columns, C, according to the method specified in section 3.1.
5. For each column $c \in C$ do:
   (Ła) Divide c into a set of lines, L, according to the method specified in section 3.1.
   (Łb) For each line $l \in L$ do:
      i. Divide l into a set of clumps, S, according to the method specified in section 3.1.
      ii. For each clump, $s \in S$, assumed to be a single character (or a part of a single character):
         ŁA. Perform character recognition using the statistical method described above and obtain a character c.
         ŁB. Analyze c for being start or end of a power:
         ŁC. In case c is a start of a power:
         $pseudo\_result \leftarrow pseudo\_result \cup \wedge(c$.
         ŁD. Else, in case c is an end of a power or a square root:
         $pseudo\_result \leftarrow pseudo\_result \cup c)$.
         ŁE. Else - c is a regular symbol:
         $pseudo\_result \leftarrow pseudo\_result \cup c$.
6. Compute the final mathematical expression from the pseudo-result, using statistical clues of mathematical expressions:
   return $ComputeFinalResult(pseudo\_result)$.

5 The Application (For a complete user-manual see part 2)

For the purpose of demonstrating our OCR system capabilities we have created an application which serves in addition as a testing environment to our OCR implementation and its further improvements. Our application consists of two parts: **The OCR system** and **The GUI framework**:

5.1 The OCR System

The first part of the system, which performs the OCR of the expression by using the first 3-stages of the algorithm described above, was implemented in **MATLAB**. The implementation is divided into the following scripts:

1. The main file *OCR.m*: This file contains one main function named *OCR*(), which performs the OCR of an image stored in the current folder of MATLAB, by using the auxiliary functions described below. The pseudo-result of the OCR is written to a file and is used by the second part of the implementation (see below) in order to finalize the expression which was recognized.
   In order to change the name of the image on which the OCR is performed, the name written in line 11 of the script, can be replaced with the desired name. In addition, the name of the file, into which the pseudo-result is written, can be changed at line 12 of the script.
   Note: Because of the way we implemented the testing framework - neither the name of the image-file, nor the name of the output text-file, should be changed in order to perform OCR on different images. For further information see the explanation for the second part of the implementation.

2. The functions *columns*() and *lines*() which are stored accordingly in the files *columns.m* and *lines.m*. are intended to divide the image, on which the OCR is performed, to columns and lines accordingly, as described in section 3.1.

3. The function *create_templates*(), which is stored in the file *create_templates.m*, creates a dataset of templates from the templates of characters stored in the *symbols* folder. This dataset is used later for performing OCR by the *read_letter*() function (see below).

4. The file *read_letter.m* contains the function *read_letter*() which is the function that practically performs OCR of a given character and returns the resulting character. This function finds the correspondence by maximizing the correlation coefficient, calculated on each of the templates of the *templates* dataset, as described in section 3.2.

5.2 The Visualization & GUI framework

This part of the implementation includes the implementation of the 4th stage of the algorithm. It is responsible of taking the pseudo-result returned by **MATLAB**, and performing its finalization and conversion to a valid mathematical expression which is then sent to Wolfram Alpha computational engine by opening it in a Mozilla Firefox browser.

In order to make our OCR system standalone and more convenient to the user (while getting closer to the application goals described above) we have implemented a GUI based system in **Java**, using the **Swing GUI toolkit**. In order to allow our application capture images from a web-camera, we have used the **JMF** (*J*ava *M*edia *F*ramework) library that enables audio, video

and other time-based media to be added to Java applications.

Furthermore, in order to allow our application to connect with MATLAB computation engine, we used an open source Java RMI-based tool called **JAMAL** (*J*ava *M*atlab *L*inking), which made it possible to call the MATLAB functions performing OCR - from the Java application, without requiring the user to actually open MATLAB.

Here we present a block diagram describing our application:

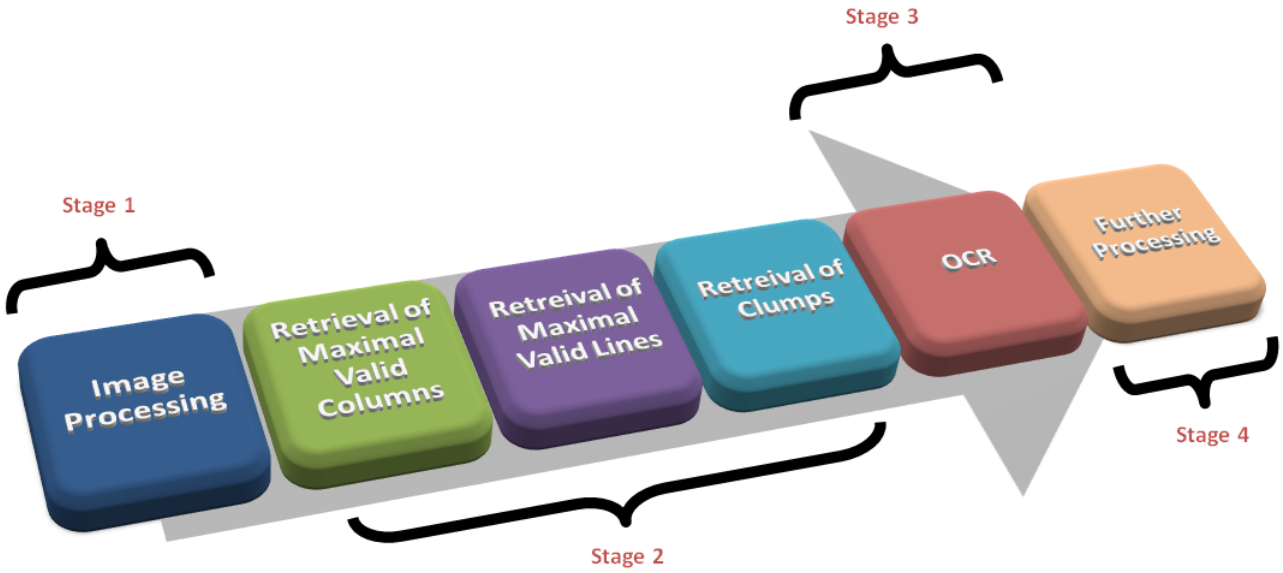

Figure 5: Block Diagram of the Algorithm

6 Experiments and Results

In order to test the proposed algorithm we had to define a common measure of accuracy rate. The accuracy rates of these kinds of algorithms can be measured in several ways, and how they are measured can greatly affect the reported accuracy rate. For example, if the structure of mathematical expressions (the context) is not used to correct the resulting expression, a character error rate of 5% (95% accuracy) may result in an error rate of 9% (91% accuracy) or worse if the measurement is based on whether each composed expression was recognized with no incorrect letters.

We decided to measure the accuracy of the recognized expression returned after the both parts of the algorithm - the OCR recognizer and the final converter. There can be two types of mistakes in the recognized expressions:

1. Replacing characters with wrong characters - the number of these mistakes was denoted as α and they got a weight of 1.
2. Placing true characters in wrong places - the number of these characters was denoted as β and they got a weight of 0.5 (less than the first type).

The measuring of the accuracy rate was done using the formula: $1 - 100 \cdot \frac{(\alpha+0.5\beta)}{\gamma}$. In this formula γ represents the total number of characters and symbols in the input formula.

The experiments were conducted on a set of **20** mathematical expressions with different degrees of complexity, written by the Microsoft Word 2007 formula editor. Here are the results returned by the algorithm and the success rate analysis (the mistakes are shown in **bold red**):

100% accuracy

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Figure 6: Test 1, The result is: "(x+a)^(n)= sum(C(n, k)x^(k)a^(n-k) , k=0 to n)"

100% accuracy

$$(1+x)^n = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \cdots$$

Figure 7: Test 2, The result is: "(1+x)^(n)=1+((nx) / (1!))+((n(n-1)x^(2)) / (2!))+"

97% accuracy

$$\cos\alpha + \cos\beta = 2\cos\frac{1}{2}(\alpha+\beta)\cos\frac{1}{2}(\alpha-\beta)$$

Figure 8: Test 3, The result is: "cos **a**+ cos beta =2 cos ((1) / (2))(alpha + beta) cos ((1) / (2))(alpha - beta)"

94% accuracy

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Figure 9: Test 4, The result is: "x=((-b+ sqrt(b^(2)-4a**C**)) / (2a))"

100% accuracy

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Figure 10: Test 5, The result is: "A=P(1+((r) / (n)))^(nt)"

93% accuracy

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \left(\int_{-\infty}^{\infty} e^{-x^2}\,dx \int_{-\infty}^{\infty} e^{-y^2}\,dy\right)^{1/2} = \sqrt{\pi}$$

Figure 11: Test 6, The result is:
"integrate(e^(-x**2**) , - infinity, infinity)=(integrate(e^(-x**2**) , - infinity, infinity)* integrate(e^(-y**2**) , - infinity, infinity))^(1/2)= sqrt(pi)"

96% accuracy

$$\frac{1}{2\pi}\int_0^{2\pi} \frac{d\theta}{a + b\sin\theta} = \frac{1}{\sqrt{a^2 - b^2}}$$

Figure 12: Test 7, The result is: "((1) / (2 pi))2 pi **L**0((d theta) / (a+b sin theta))=((1) / (!!a^(2)-b^(2)))"

87% accuracy

$$a(b + c) = ab + ac$$

Figure 13: Test 8, The result is: "a(b+**C**)=ab+a**C**"

94% accuracy

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}$$

Figure 14: Test 9, The result is: "((a) / (b))+((**C**) / (d))=((ad+bc) / (bd))"

86% accuracy

$$y\sqrt{a^2+b^2}+tan(b)=\frac{y(a+b)}{tan(b)}$$

Figure 15: Test 10, The result is: "y**!**/**7**a^(2)+b^(2)+ta**l7**(b)=((y(a+b)) / (**L**a**a**(b)))"

90% accuracy

$$(a\,x+b\,)(cx+d)=acx^2+(ad+cb)x+bd$$

Figure 16: Test 11, The result is: "(ax+b)(**C**x+d)=a**C**x^(2)+(ad+**C**b)x+bd"

100% accuracy

$$\left(\frac{a}{b}\right)^n=\frac{a^n}{b^n}$$

Figure 17: Test 12, The result is: "(((a) / (b)))^(n)=((a^(n)) / (b^(n)))"

100% accuracy

$$f(a)=\frac{1}{2\pi i}\int\frac{f(z)}{z-a}dz$$

Figure 18: Test 13, The result is: "f(a)=((1) / (2 pi i))* integrate(((f(z)) / (z-a)))"

100% accuracy

$$\tan\theta=\frac{\sin\theta}{\cos\theta}$$

Figure 19: Test 14, The result is: "tan theta =((sin theta) / (cos theta))"

80% accuracy

$$tan(a) + tan(b) = \frac{sin(a+b)}{cos(a)cos(b)}$$

Figure 20: Test 15, The result is: "**L**a**a**(a)+**t**a**a**(b)=((si**ll**(a+b)) / (c**O**s(a)c**O**s(b)))"

96% accuracy

$$f(z) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(z-a)^n$$

Figure 21: Test 16, The result is: "f(z)= sum(((f^((n))(a)) / (n^(1))(z-a)n) , n=0 to infinity)"

89% accuracy

$$f(x) = \frac{1}{\sqrt{x+1}} + \sqrt{x^2+3}$$

Figure 22: Test 17, The result is: "f(x)=((1) / (!!x+1))+!/**7**x^(2)+3"

100% accuracy

$$\cos n\theta + i \sin n\theta = (\cos\theta + i \sin\theta)^n = e^{in\theta}$$

Figure 23: Test 18, The result is: "cos n theta +i sin n theta =(cos theta +i sin theta)^(n)=e^(in theta)"

100% accuracy

$$\sqrt[x]{4s^{-2} - 7b} + \ln(2b)' - \cos' 5d * \frac{\partial f(x)}{\partial x} = \frac{\partial}{\partial x} g(x)$$

Figure 24: Test 19, The result is: "2(4s^(-2)-7b)^(1 / x)+ ln (2b)'- cos '5d*((df(x)) / (dx))=((d) / (dx))g(x)"

100% accuracy

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a \cos\frac{n\pi x}{L} + b \sin\frac{n\pi x}{L} \right)$$

Figure 25: Test 20, The result is: "f(x)=a0+ sum((a cos ((n pi x) / (L)))+b sin ((n pi x) / (L)), n=1 to infinity)"

7 Discussion ans Conclusions

In this project we tried to create a reliable and applicable implementation of an algorithm for performing OCR on mathematical expressions. The total accuracy of the algorithm, according to the experiments we conducted, is close to **95%** for typed images. These recognition rate is quite surprising since we have used a very simple method for performing OCR of a single character/symbol.

One of the disadvantages of our system is that it requires calibration to read a specific font. The current version of the application works well on characters written in fonts similar to the one of the templates. However, we created the templates dataset using the font of the formula editor supplied by Microsoft Office programs, which is a very common formula editor and therefore the usability of our application is still very high.

One of the limitations of our algorithm is a very high sensitivity to rotation of the expression appearing in the image (its orientation) - the mathematical expression should be placed horizontally or very close to horizontally. In the table below we can see the result of the algorithm on rotated images and it is clearly seen that the total accuracy decreases drastically as the expression is rotated from 0°.

Image	Obtained result	Accuracy rate
$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$	(x+a)^(n)= sum(C(n, k)x^(k)a^(n-k) , k=0 to n)	100%
$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$	(x+a)^(n)= sum(C(n, k)x^(k)a^(n-k) , k=0 to n)	100%
$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$	(x+a)^(n).+ sum(C(k, n)x^(k)a^(n1k) , k!!0 to h)	82%
$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$	cx+a)^(h)!! sum(ckn, k=0 to n)x^(k)a^(n!k	60%

Figure 26: The results of the algorithm on an image with different rotations

Another disadvantage of our method is characters and mathematical symbols which looks very similar and therefore the correspondence coefficients of images on one of them are very close. As examples we can give the following pairs of characters:

1. The little-o character ($'o'$) is very similar to the zero character ($'0'$) and to the big-o character ($'O'$).

2. The little-h character ($'h'$) is very similar to the little ($'n'$) character - can be seen at the last row of the table.

These similarities can cause mistakes in which an image representing a symbol, can get an interpretation of other (similar) symbol. Sometimes, that kind of mistakes can be repaired by our second part of the implementation. For example, we know to repair the mistake in which the string $'cos'$ is interpreted as $'c0s'$ (the character $'o'$ is replaced by the character $'0'$).

While testing the application on images captured by a web-camera, we noticed another disadvantage of the algorithm; Our algorithm is very sensitive to little errors, generally caused by insufficient quality of captured images. For example, the image below is treated as $(1+x^r)^n$ since, if we look close on the x character, we can see it has a little defect on its top left side which is recognized as another clump (marked with a red circle):

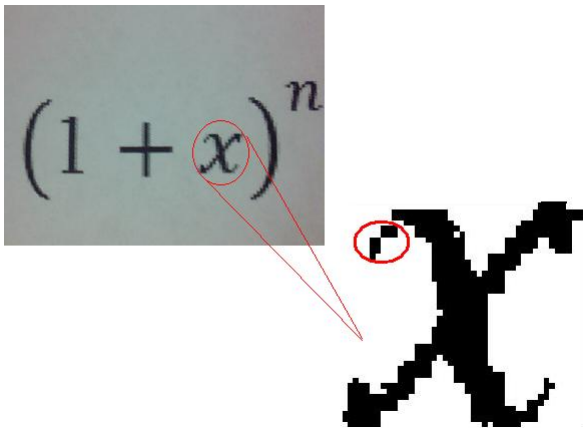

Figure 27: Captured image with a defect

8 Future Work

The application presented here can be considered as a prototype of more extensive and complicated application which has more capabilities. Here we propose improvements and additional features which can be applied to the current version of the application in order to improve it:

- First of all, the process of performing OCR to a single character can be improved. As it was stated before, the current application is limited to a font used by the formula editor of Microsoft Word or similar fonts. An important improve is to add more templates of another fonts or even change the OCR method of a single character and use a more sophisticated method, capable to deal with any font.
- A recursive process can be implemented in order to deal with clumps; currently the application can't deal with a composite expressions located in a single line, like the expression below. In next version a recursive handling of clumps can be implemented - allowing to start the process of finding columns, lines and clumps again, after treating the first clump of the line. This will allow the algorithm to treat expressions of the above type.

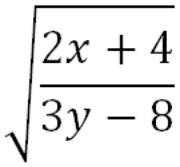

Figure 28: An expression which is badly treated by the current version of the algorithm and can be treated better in next implementations

- A method for treating the rotation of the given expression can be proposed - in order to minimize the damage of the final result, caused by rotation issues.
- A more complicated noise reduction can be implemented, allowing the application to treat noisy images captured from camera. The implementation of a better noise reduction method, may allow the application to be installed on a mobile device with a camera, thus extending its capabilities.

9 References

Here we list the sources of information which were used while working on the project and writing this document:

1. The Association for Automatic Identification and Data Capture Technologies (AIM) (2000), "Optical Character Recognition (OCR)".
   Online version is available here.

2. Øivind Due Trier, Anil K. Jain and Torfinn Taxt, "Feature extraction methods for character recognition-A survey", Elsevier Science B.V., 1996.
   Online version is available here.

3. Vishvjit S. Nalwa. (1993). Stereo, correspondence establishment. "A Guided Tour of Computer Vision". Addison-Wesley. p226-227.
   Online version is available here.

4. Ben Shahar, Ohad, "3D Shape and Depth Inference IV - Stereopsis", Lecture notes from "Introduction to Computational and Biological Vision" course, 2011.
   Online version is available here.

5. Wikipedia, A free web encyclopedia, `http://en.wikipedia.org/wiki/Main_Page`.

10 Useful Links

Here you can find references to websites containing tutorials to the technologies and tools used to create the application:

1. Mathworks official website:
   `http://www.mathworks.com/products/matlab/`.

2. JAMAL official website:
   `http://jamal.sourceforge.net`.

3. SUN tutorials to Java Swing toolkit: Creating a GUI With JFC/Swing:
   `http://download.oracle.com/javase/tutorial/uiswing/`.

4. Wolfram Alpha computational engine:
   `http://www.wolframalpha.com/`.

5. "Introduction to Computational and Biological Vision": course homepage:
   `http://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision`.

Part II
User Manual

10.1 System Requirements

- Our application was tested on a 32 bit machine with a Windows XP operating system. Since it is written in MATLAB and Java, there should not be a problem to install it on another type of machine with other kind of operating system, but we can't guarantee an installation success.
- The machine should have MATLAB and Mozilla Firefox browser installed and in addition it should have Java 1.6 or higher.
- Since the application reads and writes files from and to the installation directory, you should have the permissions to perform this operations. In addition, you should have a permission to edit the directory where MATLAB is installed and the MATLAB classpath, this is order to install the JAMAL server and client.

10.2 Installation

The whole application consists of a single .jar file which is ran by simply double clicking on it. The directory which contains the application .jar file should contain also the *OCR.m*, *columns.m*, *lines.m*, *create_templates.m* and *read_letter.m* scripts.

Before running the application you should install the JAMAL .jar file, please follow the following instructions:

1. Copy the supplied jar file named "$jamal-2.1.jar$" to the "$/java/jar$" sub-folder located in the MATLAB folder.
2. Get into MATLAB.
3. Type in the MATLAB prompt: "*edit classpath.txt*".
4. In the opened file, add the line. "$matlabroot/java/jar/jamal-2.1.jar$".
5. Save the file and exit.

10.3 Application Loading

When opening our application, it first connects to the MATLAB computational engine by running a server on MATLAB and a client on our application. This process can take several seconds during which the "Loading Please Wait!" message is presented on the screen. After that process and in case of a successful operation, the main screen of the application is shown.

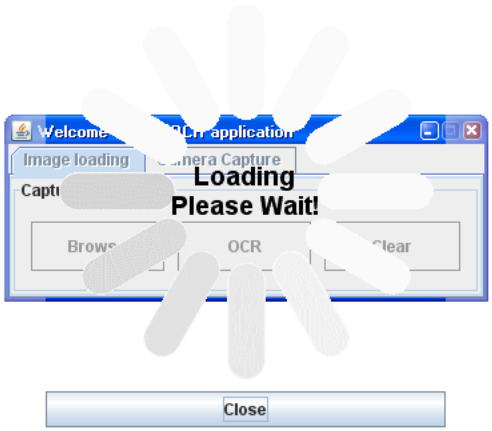

Figure 29: Loading screen of our application

Notes:

1. During the connection process, you can cancel and exit by pressing the "Close" button shown below the loading animation. In this case, the application unloads securely in order to allow the MATLAB stay in a stable state. This process can take several seconds - please be patient!

2. Sometimes the connection to MATLAB fails at first run and in this case our application ens. The fault may be caused by connection timeout or licensing errors. In case you have a right licensed version of MATLAB, which operates correctly while it is ran as a standalone application, please try to run our application again.

10.4 Performing OCR

Our application provides two ways for uploading images and performing OCR. These ways are shown on the tabs of the application main screen and are loading image from a pre-saved file (the first tab - "Image Loading") and capturing an image from a web-camera connected to the machine on which the application is installed (the second tab - "Camera Capture").

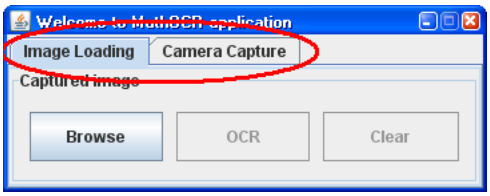

Figure 30: The two ways of image loading provided by the application

10.4.1 "Image Loading" Tab

As it was said above, this tab allows the user to load a pre-saved image file which represents a mathematical expression on which OCR should be performed. The loading is done by pressing the "Browse" button, choosing an image and pressing "Open". Currently, the supported formats of images are ".GIF", ".PNG", ".JPG" and ".BMP". After pressing "Open" the image is loaded and shown on the application screen.

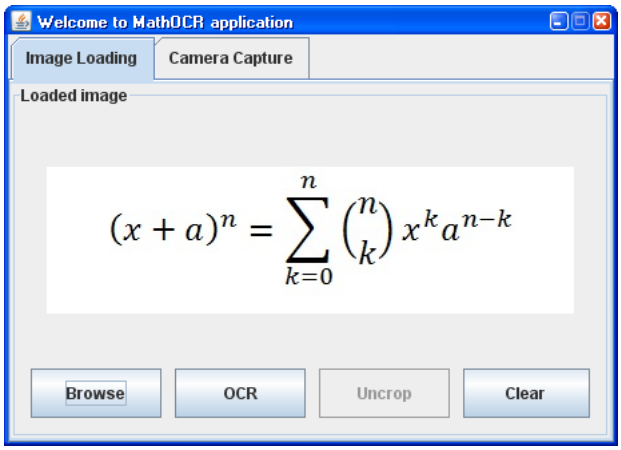

Figure 31: The image loading screen with a loaded image

After an image was loaded you can change it by pressing the "Browse" button again and choosing other image. In addition you can delete the image from the view by pressing the "Clear" button. In this case the image will be removed from the application screen and you will return to the main screen shown in figure 32.
If you want to crop the image horizontally you can use the mouse in order to change its top and the bottom boundaries. Pressing the "Uncrop" button cancels the last cropping. The crop feature is helpful when you have an image which contains several lines of mathematical expression. The cropping is necessary in this case, in order to focus the OCR system on a single expression. Here is a screenshot of a cropping made to an image:

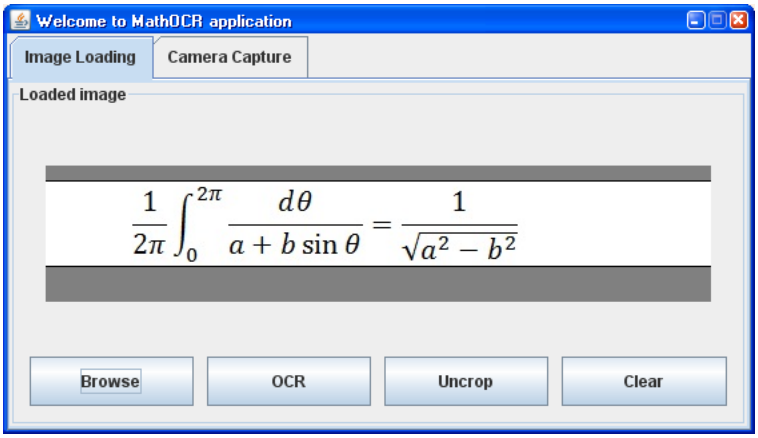

Figure 32: Cropping an image

Now, after an image representing an mathematical expression has been loaded, OCR can be performed by pressing the "OCR" button. The OCR process can take several seconds, depends on the complexity of the given expression. During that process, all the buttons are made not-accessible and you can't exit the application (in order to keep MATLAB in a stable state).

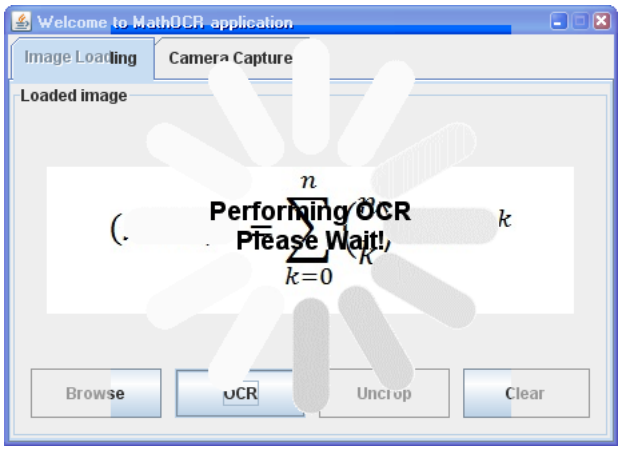

Figure 33: The application view while performing OCR

After a successful OCR has been performed, the application window spreads out and its right part shows the result of the OCR. You can see the result in the text-box at the top of the right side. For convenience we allow the option to wrap the result (useful when it is wider than the window) or to see it not wrapped - as a single line. The option is "wrap" by default. You can change the wrap state by checking/un-checking the check-box "wrap ocr answer" below the OCR result.
In addition, the result window provides the option to clear the OCR result and make the application window look like in Figure 33. This can be done by pressing the "Clear OCR" button. In case you press the "Clear" button now, the OCR result will be cleared too and hidden.

Note: The OCR result shown in the text-box is after performing further processing of the pseudo-result returned by MATLAB. While transferring the result to Wolfram Alpha computational engine, additional processing is made in order to make the result conform to Wolfram Alpha query structure.

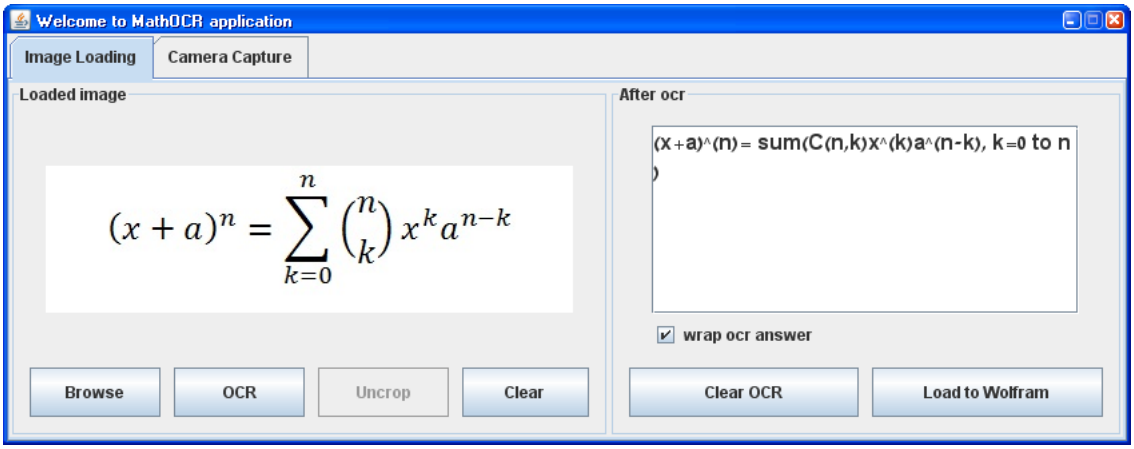

Figure 34: The result window after performing OCR

The result of the OCR is automatically delivered to to Wolfram Alpha. Thus, immediately after returning from the call to OCR, a new firefox browser window is opened and the result is shown, after processing by Wolfram Alpha. In case you closed the browser window with the result of the OCR and you want to see it again, you can press the "Load to Wolfram" button and the browser window will be loaded again.

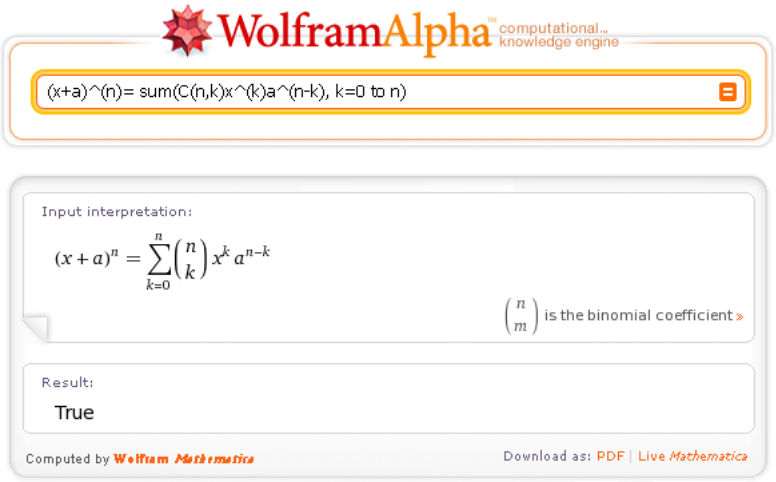

Figure 35: The result after performing OCR, delivered to Wolfram Alpha

10.4.2 "Camera Capture" Tab

In addition to performing OCR on pre-saved images, we provide the option to capture images on-the-fly, by using a web-camera. This feature is implemented in the application and can be reached by pressing the "Camera Capture" tab. In case the camera is connected and initialized properly, the application detects it and presents a window similar to image loading window shown at figure 33. Below the window which shows the image currently viewed by the camera, you can see the "Capture" button, which allows to capture the currently shown image and perform an OCR on it. The image is shown with a horizontal and vertical grid-lines in order to help you capturing the mathematical expression with a horizontal orientation. As it was shown in the conclusion part, the rotation of the image is critical to the algorithm success.

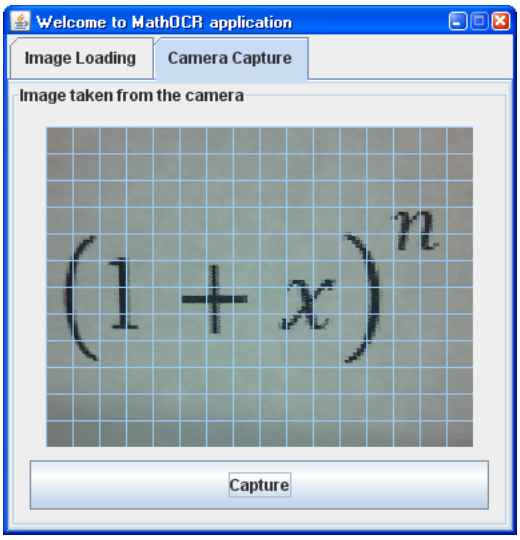

Figure 36: The camera capturing screen with math expression in front of camera

After pressing on "Capture" the application window spreads out and its right part shows the captured image. Here, again you can perform cropping of the image using the mouse and clean the crop result by pressing "Uncrop". In addition, you can save the captured image by pressing

the "Save as" button. Currently only saving to ".JPG" format is allowed. Note that the saving operation is optional and it isn't required in order to perform an OCR on the captured image. Also, please note that the image that will be saved is the cropped version of the captured image (in case a cropping was done). Like in the "Image Loading" mode, you can clear the captured image by pressing the "Clear" button.

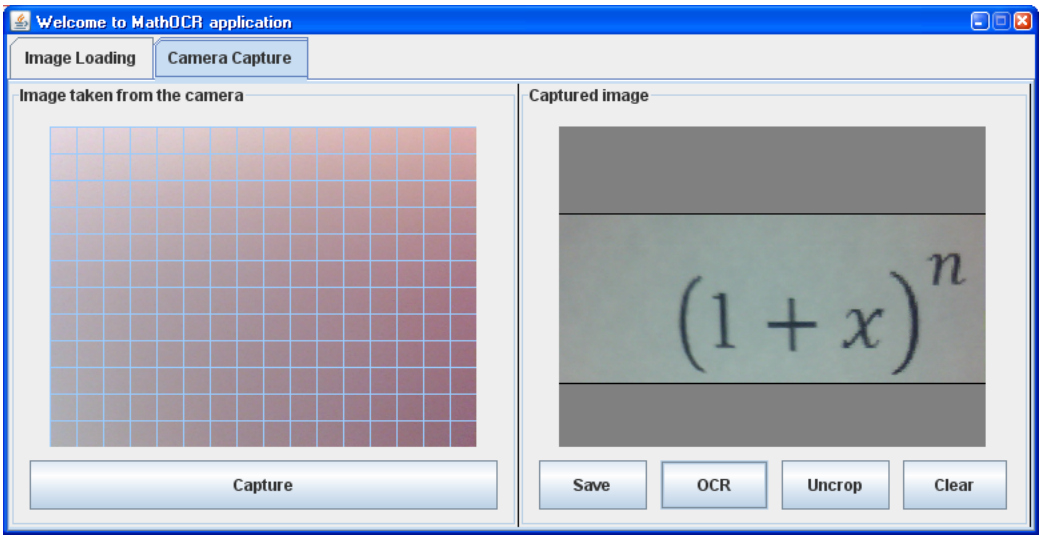

Figure 37: The camera capturing screen after capturing an image

In order to perform OCR press the "OCR" button on the "captured image" side. The process of performing OCR is similar to the one described above (in the "Image Loading" mode). After a successful OCR has been performed, the application window spreads out again and its most right part shows the result of the OCR. You can see the result in the text-box at the top of the right side, clear the OCR result by pressing on the "Clear" button and load the result to Wolfram Alpha by pressing "Load to Wolfram".

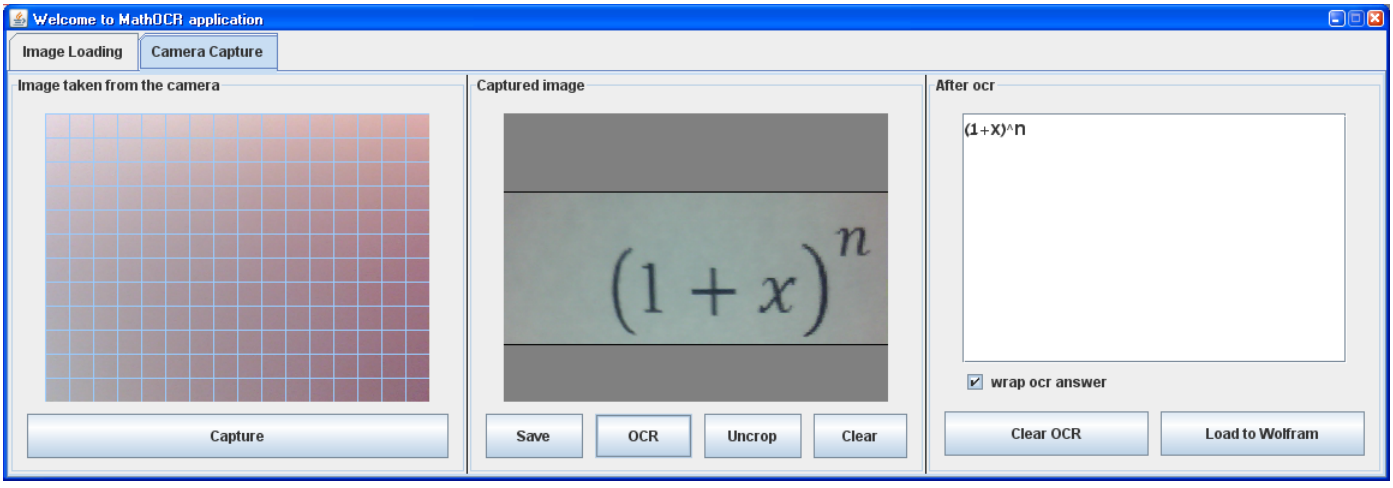

Figure 38: The camera capturing screen after performing OCR on a captured image

Notes:

1. If you press on the "Clear" button in the middle window, the captured image and the OCR result will be removed from the screen.

2. Pressing the "Capture" button again will capture the image currently shown in the leftmost screen and the currently captured image will not be saved (except if you saved it before). The previous OCR result will disappear too.

3. In case of a failure with the connection to the camera, the screens of the "Camera Capture" mode disappear and a "Connect to camera" button is shown. After resolving the connection problem, press the button and the screen with the camera output will be shown again.

Figure 39: The screen of "Camera Capture" mode in case of a camera connection failure