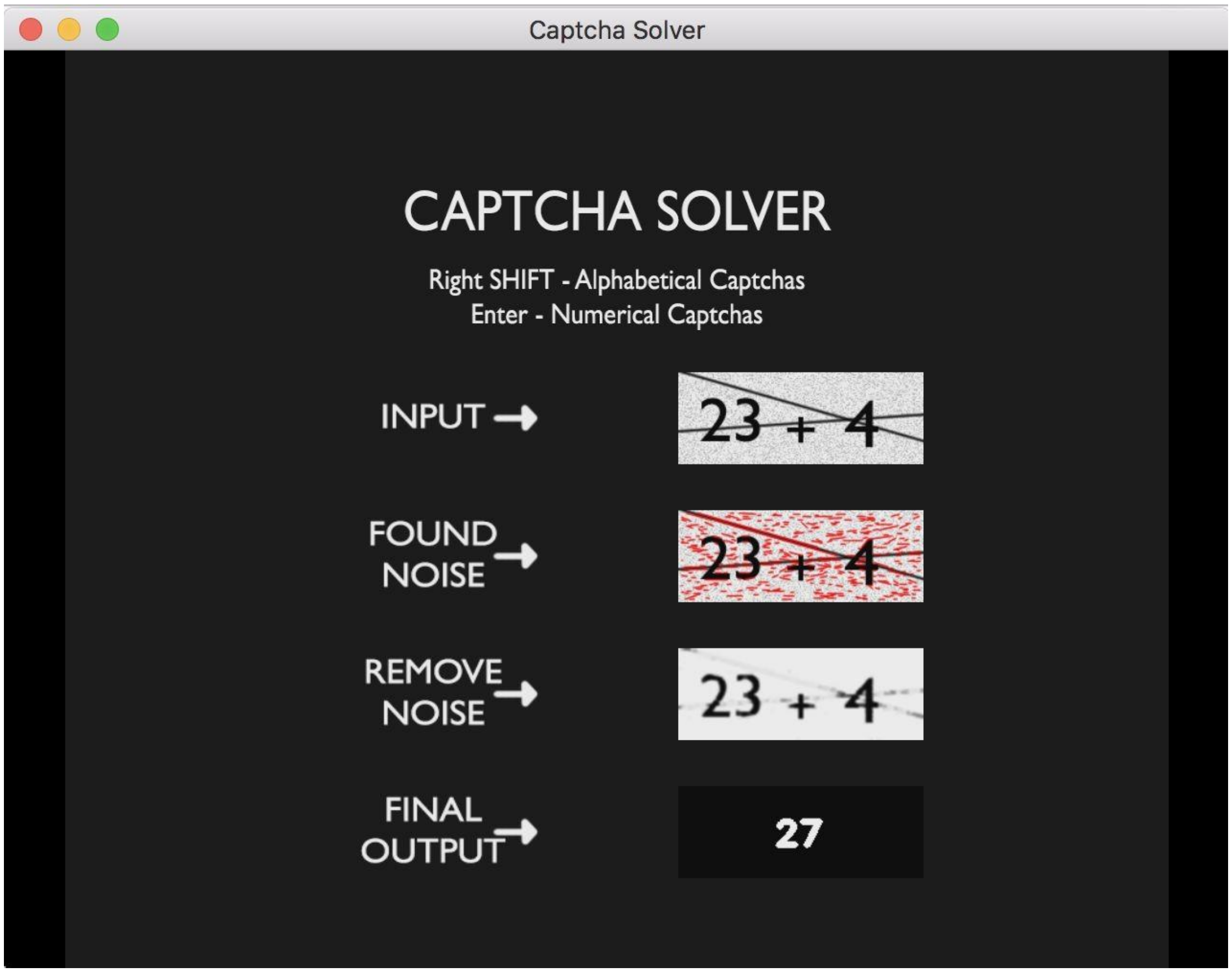
ECE 6122 Advanced Programming Techniques

Captcha Solver

By Aditya Chavan, Sapan Desai, Vishwateja Reddy



Introduction

CAPTCHA or "Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge–response test used in computing to determine whether or not the user is human. The most common type of captcha uses a computer generated distorted image containing a sequence of random letters and numbers, which the user would have to type in. Since this test is administered by a computer, and meant to detect a human, this procedure is often called as a 'Reverse Turing Test'. Captchas are widely used by web developers to prevent automated bots from accessing a particular part of the website deemed particularly sensitive to DDoS attacks.

Setup Instructions for Mac:

A few libraries will need to be installed first. To make the installation process easier, homebrew will need to be installed.

1. Visit brew.sh, copy the link and run it in the terminal.

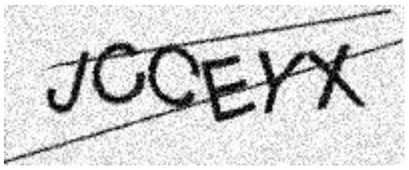
Run the following commands in terminal to download the libraries, if you already have the libraries, please make sure they're up to date.

2. brew install glfw3
3. brew install glew
4. brew install opencv3 -force
5. brew install tesseract In

the project directory, run:

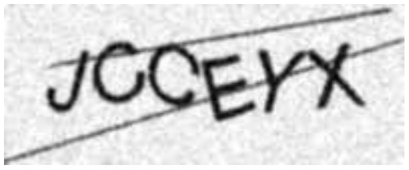
6. cmake .
7. ./APT

Steps for Image Processing and Noise Removal :

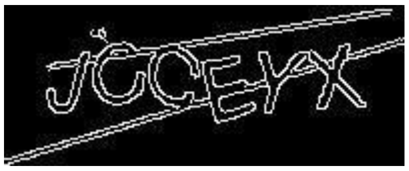


We start with an input image as shown:

We notice the captcha (may) have a certain amount of granular noise in the background. To remove this noise we use a technique called Gaussian Blur. Gaussian blur is a smoothing technique widely used in Computer Vision to reduce granular noise in the images. It uses a 'smoothing kernel' to produce an effect resembling that of viewing the image through a translucent screen. Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function, also known as a two-dimensional Weierstrass transform.

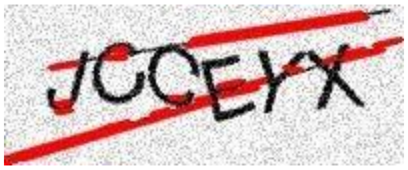


The resulting image is as shown. The noise while still visible in the background, is now blurred and wont pose a problem to our edge detection algorithm.

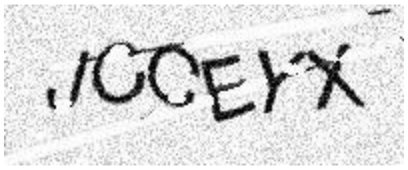


Next to remove the horizontal lines, we need edge detection, so we use a Canny edge detector. It uses a multi stage algorithm to detect a wide range of edges in images. Details below in specifics about the techniques used section.

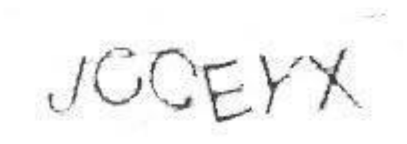
Hough Linear Transform is used to detect and remove lines that are above a certain length, and between a certain angle. We get the coordinates of the lines and superimpose them on the previously cleaned image for illustration.



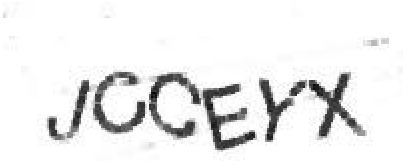
We can observe that the horizontal lines in the background are detected, while the lines that overlap with the alphabets are left relatively untouched. We then subtract these lines from the image as shown:



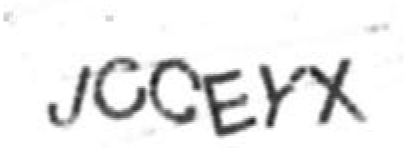
Next we apply dilation to the image to get rid of the remaining lines and granular noise. Dilation thins out every line in the image. This not only removes much of the background noise, but also takes care of parts of the horizontal line that are left.



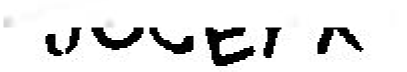
To counter for dilation, the image is eroded, which thickens out every line from the image.



This image is almost cleaned up, to get rid of any remnant granular noise and to make the words more of a continuous lines , we apply the Gaussian blur again. The resulting image looks like the following:



Finally, we take a threshold of the final image. Thresholding basically replaces all the pixels above a certain threshold value with black pixels and others to zero. The final output is:



This image is given to the Tesseract OCR, which then detects the letters in it.

Since this project was aimed at solving numerical captchas as well, we had to add a special case for this. We take a look at the result buffer searching for any addition (+) or subtraction (-) operators. If found, the numbers on the two sides of the mathematical operator are 'cleaned up', essentially meaning, any alphabet wrongly detected by the tesseract-ocr would be simply ignored and the final output would be calculated. Special care had to be taken for '~' character as the subtraction operator was sometimes detected as '~'.

More Specifics about the image processing techniques used:

Canny edge detection: This was used to detect edges in the image. Canny edge detection basically extracts the most useful information for further image processing and also dramatically reduces the amount of data that needs to be processed. It uses a multi stage algorithm to do so. After Gaussian blurring, we find the intensity gradients of the image to detect all edges and corners. These detected edges might have ridges which are removed using non-maximum suppression. Then double threshold is applied to determine potential edges. The last step is to finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges. This is done by using edge hysteresis.

Hough Probabilistic Line: This was used detect stray lines in the captcha image. Hough Transform is a popular technique used to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit.

Thresholding: This is a parameter used to remove/hide unwanted data from an image. If the pixel value is lower than the threshold, the pixel is set 0, otherwise its set to 255. This results in a monochrome image with noise like dust and scratches mostly removed.

Gaussian Blur: It is the result of blurring an image using a gaussian function. This is done to greatly reduce speculative noise. A gaussian blur effect is typically got by convolving a gaussian kernel or filter with an image resulting in a blurred image.

Erosion and Dilation: The basic effect of the erosion operator on a binary image is to erode away the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels *shrink* in size, and holes within those areas become larger. The basic effect of the dilation operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels *grow* in size while holes within those regions become smaller.

The UI:

The user interface was made using basic opengl techniques learnt during this course. The user interface is built using multiple rectangles on to which textures have been mapped. Textures were loaded into the program using SOIL2 and stb_image libraries. While SOIL2 was more versatile compared to the latter, we found that moving the project folder from one system to another could break functionality and hence we decided to stick with stb_image. Using png images gave us better results in terms of quality presumably due to JPEG compression.

The program registers key presses like Enter, Right Shift and Escape. Right Shift is used to cycle through the text based captchas provided and Enter is used to cycle through the numerical/math based captchas. Pressing the Escape key exits the program. The aforementioned image processing steps are carried out in the glfw render loop so that textures for the rectangles are update in real time, while this may unnecessarily consume system resources, we've encountered problem where textures don't render properly if the image processing was carried out on a key press.

The output of the image processing section needs to be displayed somewhere, so we used the puttext function in OpenCV to overlay the calculated captcha output on an image so that it could be mapped on to a rectangle for display on the UI.

Results:

We have had promising results with this project. We have only submitted 25 of the captchas out of the 100s we have tested so that we don't make the submission folder too large. We've achieved an accuracy of 70% with text based captchas and 82% with our math based captchas. We consider this accuracy pretty good considering that captchas were not meant to be broken in the first place.

Possible Future Work:

This system uses the open source tesseract ocr library. While this is really good at doing what it does, it was trained to be able to recognize normal text, not captchas where fonts and font sizes can drastically vary. A way of getting around this would be to build a Neural Network and train it on a large database of labelled text and number images. The trained NN will be able to read the captcha with very little image processing. This sort of implementation will also require an image segmenter that can intelligently split up the image into multiple chunks so that each chunk has just one character in it. We predict that such a system will have a significantly better accuracy than tesseract for the sole reason that it was specifically trained to read captchas.