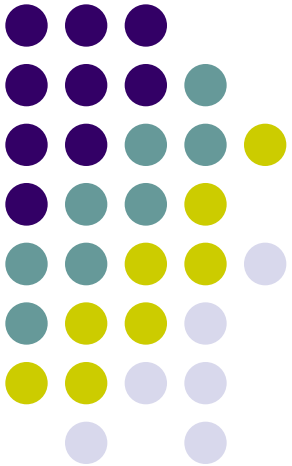
Captcha

CS 293 Final Project Demo

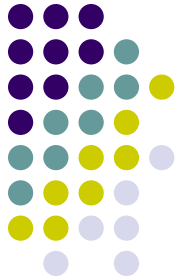
November 27, 2012

Astha Agarwal, 110050018

Deepali Adlakha, 11D170020

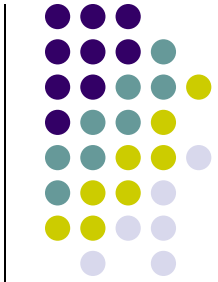


Outline <for a total of 30 mins>



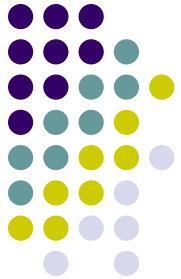
- Aim of project (1 mins)
- Demo (5 mins)
- Teamwork Details (0.5 min)
- Design Details –Algorithm (5 mins)
- Design Details – Implementation (8 mins)
- Viva (9 mins)
- Transition time to next team (2 mins)

Aim of the project



- A program that can decode simple captchas, by using techniques of Image Processing and Optical Character Recognition
- Background: A captcha is a challenge response test which is used to prevent spam bot attacks by checking whether the user is a human or a bot. They are usually images that contain distorted texts, which is readable by humans but not computer programs.

Demo



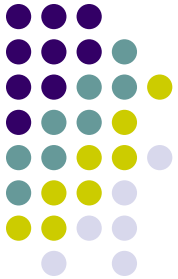
- *Demo of different types of captchas*
- *Demo of custom captcha decoder*

Teamwork Details

1. Astha: Image processing and Generating Training Data
2. Deepali: Training for Character Recognition and User Interface Development
3. Both: Testing for captchas by applying various filters, Creating Shell Scripts for training

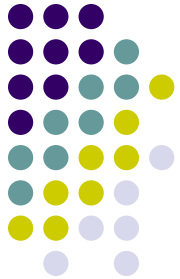
Overall Contribution by Team Member 1	Overall Contribution by Team Member 2
50%	50%

Design Details



- Algorithm
  - We pass a captcha image through various filters to get a standard image with white background and black text. Then we pass this image to tesseract library for identification of text based on our training.
- Implementation
  - Library: CImg, Tesseract-OCR
  - Platform: C++ , UI: QT Creator

Algorithm Design



- Problem: To decode a given captcha
- By applying various noise filters on the image and converting it to an image with white background and black text, and then identify the text using OCR techniques
- Algorithms & implementation of filters are described in following slides

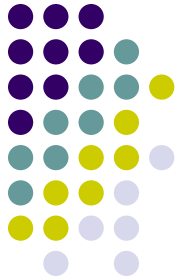
Algorithm Design- Functions

- Problem: To extract a single character from image file
- We traverse the image and find the first dark (black) pixel, then we apply a breadth first type of search on the image, traversing through all connected dark pixels. This is like a maze, where the dark pixels represent path and the white pixels represent hurdles.
- Now apply the same search again on remaining unvisited black pixels

Algorithm Design- Example

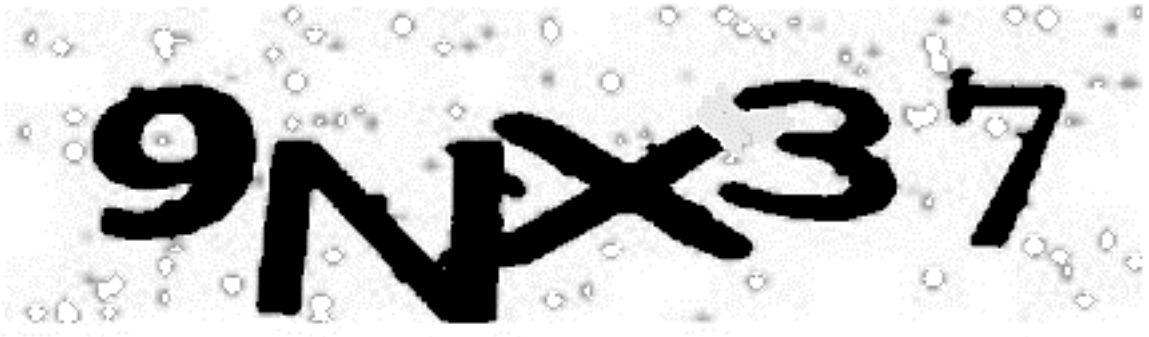
- *Original Image:*

Algorithm Design- Functions



- Problem: To remove dotted noise from image
- Traverse through the image and land on a black pixel. Now again apply the same breadth first search on black pixels connected to that pixel. If the number of connected pixels is very small (this threshold is determined according to total number of pixels in the given image) , we identify this as a stray dot in the image, and change the colour of those pixels to white.

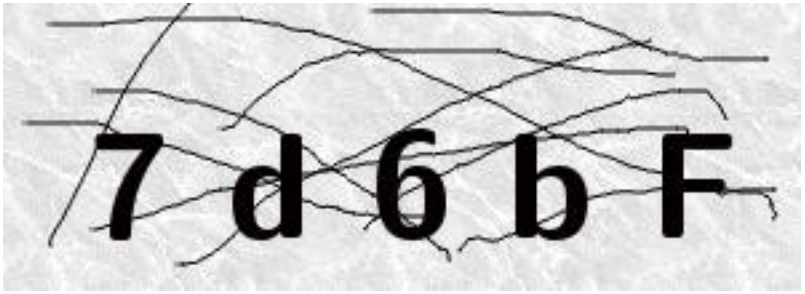
Algorithm Design- Example



Algorithm Design- Functions

- Problem: To remove thin stray lines in image
- For every dark pixel in the image, consider a k*k matrix around the pixel. For a pixel in a stray line, most of the pixels in this matrix will be lighter in shade; while for a pixel that belongs to the text, most of these pixels will be darker. If the number of dark surrounding pixels is small, we identify this as a pixel belonging to a thin stray line, and change its colour to white.

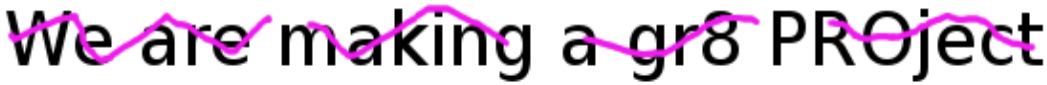
Algorithm Design- Example



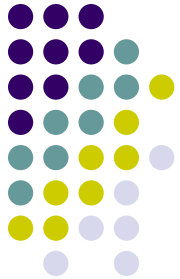
Algorithm Design- Functions

- Problem: To remove thick lines that cut through the text in image
- Depending on the frequency of occurrence of every colour, we differentiate the colour of background, text pixels and the cut through line. Now, we remove the cut through line and change those pixels to white. This will create breaks in text, because we removed all cut through pixels. So we go to all text pixels, check if we removed any nearby pixel and re-color it to black to remove the breaks created.

Algorithm Design- Example



Algorithm Design- Functions

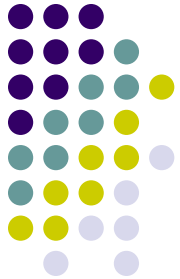


- Problem: To remove coloured noise when text is dark (blackish)
- We have utilized the fact that if the difference between R-G, G-B and G-R values of a pixel is small, then the pixel is either white or gray. We identify the pixels which have at least one of these differences large enough to make it a coloured pixel, and re-colour them to white.

Algorithm Design- Example

abcdefghicjklm

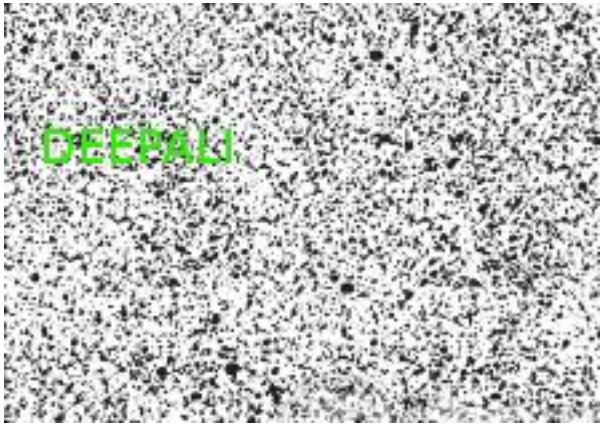
Algorithm Design- Functions



- Problem: To remove blackish noise when text is coloured
- We use the following formula to convert a coloured image to grayscale image:

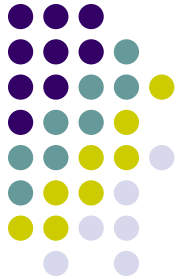
  New pixel value= (R*30 + G*60 + B*10)/100
- This is based on the relative weights and sensitivity of each colour to our eyes.
- Now, we re-color all the pixels that are in a lighter shade to white, because they were actually colored pixels.

Algorithm Design- Example



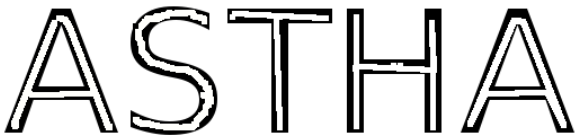
DEEPALI

Algorithm Design- Functions

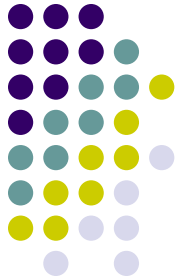


- Problem: To decode stencil captcha
- First, we colour all the white connected background (BFS) to black, and mark those areas as visited. Then, we invert the captcha. Now, we traverse the image and land on the first unvisited black pixel. We apply breadth first search to traverse all connected black pixels and convert them to gray. While traversing, we keep account of the min and max coordintes encoutered. Next, we go to the middle pixel of the rectangle thus visited and colour all connected black pixels to white. Now we traverse all gray pixels in the image, and turn them to black. The following images will give a clearer idea of the algorithm.

Algorithm Design- Example

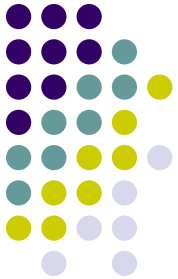


Algorithm Design- Functions



- Problem: To remove background in image, in case of coloured images
- The colour with the highest frequency is identified as the background colour and is re-colored to white.

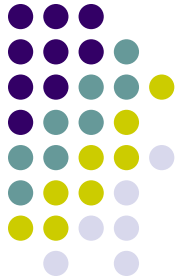
Algorithm Design- Example



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 o p q
r s t u v w x y z ! ? & $ - ; ,"
0 1 2 3 4 5 6 7 8 9

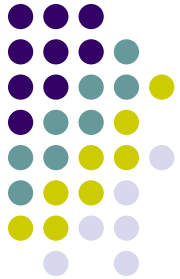
A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q
r s t u v w x y z ! ? & $ - ; ,"
0 1 2 3 4 5 6 7 8 9

Algorithm Design- Functions



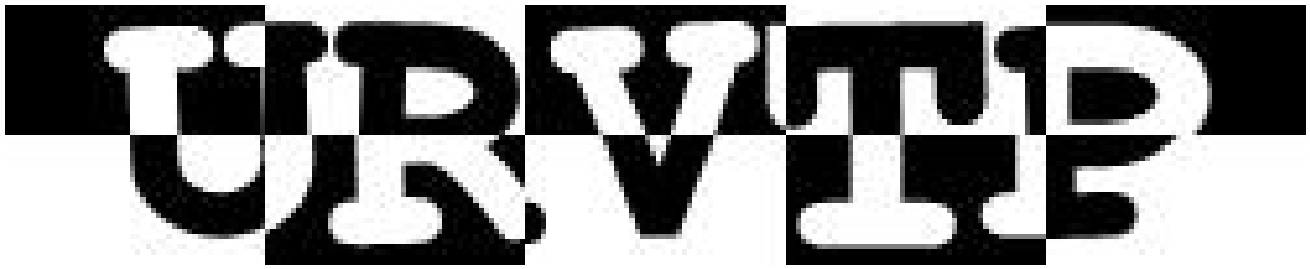
- Problem: To invert the colours in an image
- Sometimes, captchas have black background and white text, so we traverse through the captcha and simply invert the colours.

Algorithm Design- Functions

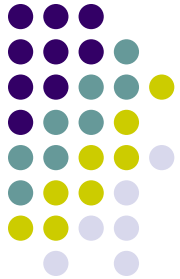


- Problem: To invert selective regions for a chess captcha
- We traverse through line and column of the image to figure out the regions where captcha colour is to be inverted, and accordingly apply the invert function on those regions. This gives us the image with white background and black text.

Algorithm Design- Example



Algorithm Design- Functions

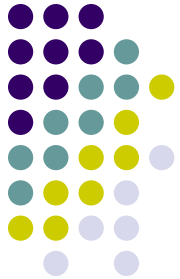


- Problem: To rotate back a rotated text to normal
- We extract all characters of the text by applying breadth first search (as explained before), and store the bottom left coordinates of all such characters. Now we apply regression on these coordinates to find the angle through which the text is rotated. Next, we use rotate function of CImg library to rotate the text back to normal orientation.

Algorithm Design- Example

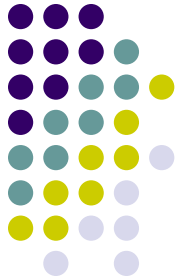
ROTATE ME

Class Design – High level



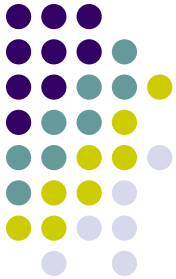
- A simple vector2d class, which can handle x and y coordinates of the image easily. This is useful when we do breadth first traversal in an image, or when we have to find neighbours of a certain pixel.
- We haven’t used any other classes as such. We take an image file, and apply the filters accordingly to get a standard text image that our OCR can identify.

Class Design – High level



- General design trade-offs: We are not able to remove distortions and decode skewed text images (like the ones used in Google captchas), as that involves a high level of machine learning and Character recognition techniques.
- Also, captchas with overlapping charaters cannot be decoded.

Class Design - Details



Class Name	Brief Description
vector2d	Stores information about the coordinates of a pixel, mainly used for easy access to neighboring pixels and comparing coordinates of two pixels by operator overloading

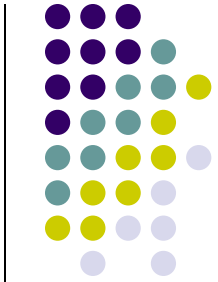
Data Structures Used

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Breadth First Search	Queue	STL
Regression on coordinates	List	STL
To find peaks in frequency distribution	Priority Queue	STL

Source Code Information- qt_gui

File Name	Brief Description	Author
allfunctions.h	Declares all filter functions	Deepali
allfunctions.cpp	Implements filter functions	Both
vector2d.h	Declares vector class and implements operator overloading for +, - and <	Astha
tester.cpp, tester.h, main.cpp	Implementing Graphic User Interface	Deepali

Source Code Information- training and input_files



File Name	Brief Description	Author
train.sh	Creates box files from training data	Deepali
trainb.sh	Feeds the training data into tesseract library (font name: ddb)	Astha
training	jpg text images & corresponding .txt files used for training	Astha
clusterwithlines.cpp	To take an image and text file, and generate box file, which will be used for training	Astha
solved_captchas	Folder containing sample captchas	Both

Brief Conclusion

- The program successfully runs with many captchas, however as more and more distortions are introduced, the accuracy falls.
- The program works with approximately 80% accuracy for trained fonts. For untrained fonts, the accuracy falls depending on the extent of distortion in the font.
- This program has not been made with the intension of spamming, but for a broader and more constructive motive. It can be used to digitize old texts, for which digital versions do not exist.

Thank You – Questions?