Assignment 1: Image Processing and Recognition Basics

Darshan Shinde(dshinde),
Virendra Ishwarappa Wali (vwali),
Subhojit Som (susom)

Approaches towards detection of marked segments from OMR sheet

Image pre-processing - extraction of the answer zone

A major task in recognition of the marked answer options in this task is to detect the segment of the image file which contains the answer options with their question numbers.
In order to achieve this, we followed a naive approach of plotting the entire image on an x-y coordinate system and extract the pixel indexes (horizontal/ vertical margins) which contain the answers. From plotting all the test-images given and manually deriving the top, bottom, left and the right margin coordinates for each image the mean values indicate, that the entire answer segment resides roughly within the following coordinates:
image_matrix[y_top: y_bottom, x_left:x_right]
where,
y_top = 650
y_bottom = 1550
x_left = 0
x_right = 1700

Horizontal and vertical line detection for demarcating boundaries of each question

On the cropped image segment extracted naively following the above method, the next task is to detect the set of horizontal and vertical lines, the intersection points of which would give the demarcation of the region of containing each question and more precisely its answers.
In order to get these set of lines a Hough transform was done on the cropped section, and the rest of the detection process depends heavily on the accuracy of the implementation of Hough Transform. In our implementation of Hough Transform, a major step involves edge detection for which the following was done.

Hough transform

Hough transform involves the following set of standard steps -

Parameter space transform and accumulation of votes

This step involves transforming parameters of each line that passes through every point in the image in the Cartesian plane to its polar coordinate form, which results into 2 parameters rho (perpendicular distance from origin to the line) and theta (angle produced by the perpendicular on the x-axis) for each slope and intercept in the Cartesian plane representation of the line.
A Hough accumulator space is constructed which has the dimensions of (rho x theta), where rho is the diagonal length of the image (as that can be the longest line) and theta are all values from 0 to 90 degrees evenly spaced by 1 degree. As we are only interested in horizontal and vertical lines which casts \theta values of 90 and 0 degree respectively.
For each point the entire set of rho and theta values are calculated using - rho = x*cos(theta) + y* sin(theta), and the corresponding rho and theta entries in the accumulator matrix is incremented.

Doing the above step results into an accumulator which has multiple local maximas which correspond to a straight line. Since we are only interested in the horizontal and vertical lines, we only take local maximas which have column index below 2 and above 88 (such that they correspond to theta values around 0 and 90 degrees respectively, keeping a margin of 2 degrees error to get appropriate number of needed lines)

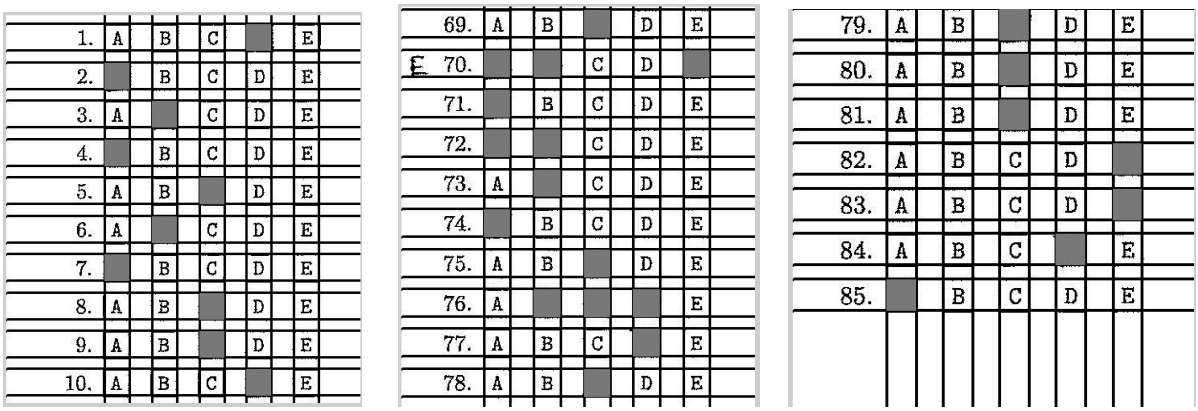
Extraction of straight lines by converting from Hough parameter space to Cartesian space

To extract lines from accumulator we sorted the rho and theta values in decreasing order of number of votes gathered by lines. For given image we need only 30 vertical lines and 58 horizontal lines. So accordingly, we chose lines from each category (lines with theta value in range 0 to 2 and lines with theta value in range 88 to 90). But this approach was very susceptible to the noise and hence was failing mostly in bottom part of the image. To address this problem, we adapted following approaches to make Hough transform more efficient.

The answer region was partitioned into 3 sections containing 10 contiguous lines each. This was done keeping in mind even if Hough Transform fails in 1 segment, the final output would not suffer, as line detection can carry on independently in the other sections of the answer zone. However, the line detection was not perfect till this point.

Hence at last, we have partitioned the answer part of the image into total 9 sub-parts (3 columns and 3 rows of 10 questions in each columns). This time we got perfect result from the Hough transform (the required set of horizontal and vertical lines that would later be used to intercept the grid structure).

After extraction of straight lines by running Hough transform on selected sections of the image, independently of one another we get the x, y coordinates of the grid. And each grid point is then parsed separately to extract the marked answer option.



Detection of marked answer from the grids intercepted by Hough transform

Average pixel value for each square area within the grid is calculated. The boxes which are marked will have relatively lower average pixel value than unmarked areas. This is how we identified marked boxes from the grid coordinates as obtained above.

The marked option index and question numbers are calculated from the grid index, of the block containing lowest pixel summation value. Using these pixel values, we have drawn green box to highlight the marked boxes.

For drawing the lines, we used basic line drawing functionality of ImageDraw of Pillow library.

Future work

We have considered all the edges with theta values between 0 to 90 degrees, however orientation angle beyond that does not work so well with the hyper parameters and their tunings that we have done to

achieve our desired results. To accommodate orientation changes of the scanned OMR sheet the parameters needs to be returned.

Performance

Although with our implementation of Hough Transform the accuracy is well above 90% with the supplied test images, however as mentioned previously, orientation angle of the scanned sheet will hamper the system's performance.

The detection of hand-written answers (for marking x in the detected answers), although is working with an accuracy over 80% right now, however it is purely dependent on the quality of image that is to say the noise induced within it.

We have considered all angles between 0 to 90 degrees, however we are only interested in horizontal and vertical lines so we can restrict the range of theta between 80 to 100 and -10 to 10 degrees, to improve performance.

Alternatives tried for detecting marked blobs

Canny edge detection

As a pre-processing step to Hough transform a binary edge map is extracted from the section of the image that contains the answers. To derive that a Canny Edge Detection algorithm has been implemented that contains a set of standard steps as mentioned below.

Gaussian blurring

The image section is convolved with a Gaussian kernel of shape (3 x 3) with a sigma=1, in order to smoothen the noise present in the image.

Sobel operator

Post that a Sobel operator is applied which gives the gradients in the X and Y directions. This process involves convolution of the image with 2 kernels which are approximations of gradients of a Gaussian kernel in X and Y directions. The kernel which produces the X gradient is [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], while that producing the Y gradient is: [[1, 2, 1], [0, 0, 0], [-1, -2, -1]].
After applying the Sobel operator, we get a set of 2 matrices which gives the X, Y gradients, G_x and G_y respectively. From G_x and G_y, the gradient magnitudes and angles for each pixel can be derived which produces.

Non-maximal suppression

This step involves suppression of multiple edges in the edge map o/p. In order to achieve this for each entry in the gradient magnitude, the corresponding gradient angle is used to check the values of the neighboring pixels in the direction of the gradient.
The angles are quantized into 4 sections which helps to carry out this task by examining only 4 neighboring pixels for each entry as below.

angle 0 degrees
0 <= gradient angle <= 22.5 and 157.5 <= gradient angle <= 180 are considered as 0 degrees, due to which we only need to check 2 neighboring pixels (1 before and another after the entry).

angle 45 degrees
22.5 <= gradient angle <= 67.5 are considered as 45 degrees, due to which we only need to check 2 neighboring pixels (top right, bottom left in the 3 \times 3 neighborhood).

angle 90 degrees
67.5 <= gradient angle <= 112.5 are considered as 90 degrees, due to which we only need to check 2 neighboring pixels: top, bottom in the 3 \times 3 neighborhood.

angle 135 degrees
112.5 <= gradient angle <= 157.5 are considered as 90 degrees, due to which we only need to check 2 neighboring pixels: top right, bottom left in the (3 x 3) neighborhood.

The maximum magnitude in the direction of the gradient is taken and the value of each magnitude intensity is replaced with that if the original intensity value is greater than or equal to it else the magnitude intensity is suppressed, by setting to 0. This helps to suppress gradient magnitudes which are not in the direction of the gradient and helps to get rid of multiple edges for each.

Thresholding and hysteresis

After non-maximal suppression, a thresholding is applied.

The thresholding step involves determining 2 threshold points which serves as an upper and lower threshold. To determine this, the value of the maximum intensity is obtained from the gradient intensity matrix and 10% and 7% of that is taken as the upper and lower thresholds, for the hysteresis stage. All magnitude Intensities below the lower threshold is set to 0.

The upper and lower thresholds obtained from the above step are used to check the local neighborhood of 3\times 3 size of each magnitude intensity. And then only if there is a maximum in the neighborhood, the gradient intensity is kept otherwise it is restored to 0.

Finally, we get an edge map, where the non-edge pixels are set to 0 and edge pixels are set to 255.
However, after experimenting, it was found that our Hough transform is able to perform reasonably well on the image without using Canny Edge detector output and hence we removed it to make the whole process run a little faster.

Image segmentation to extracted marked answer blobs

Initially we used image segmentation to extract foreground from background using Felzenszwalb & Huttenlocher segmentation algorithm.

Graph creation

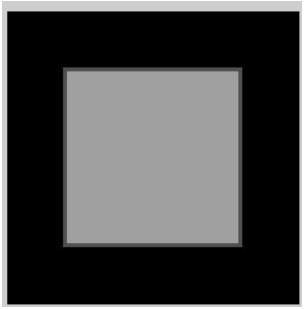
We created grid graph from an image using pixels as a vertex in graph edge weights as absolute difference between adjacent pixel values.

Minimum Spanning tree

Then from grid graph we extracted minimum spanning tree using Kruskal's algorithm with find and union.

Connected components

From minimum spanning tree we created different segments by removing edges with higher weights. (Number of segments = Remove number of high value edges)
However, the heuristic we used for edge removal in order to create the connected components, was an overtly simplified one - remove edges with highest weights, however this scheme fails for OMR sheets as the edge weights are very unevenly distributed,



Embedding encoded answers into OMR sheet for later retrieval

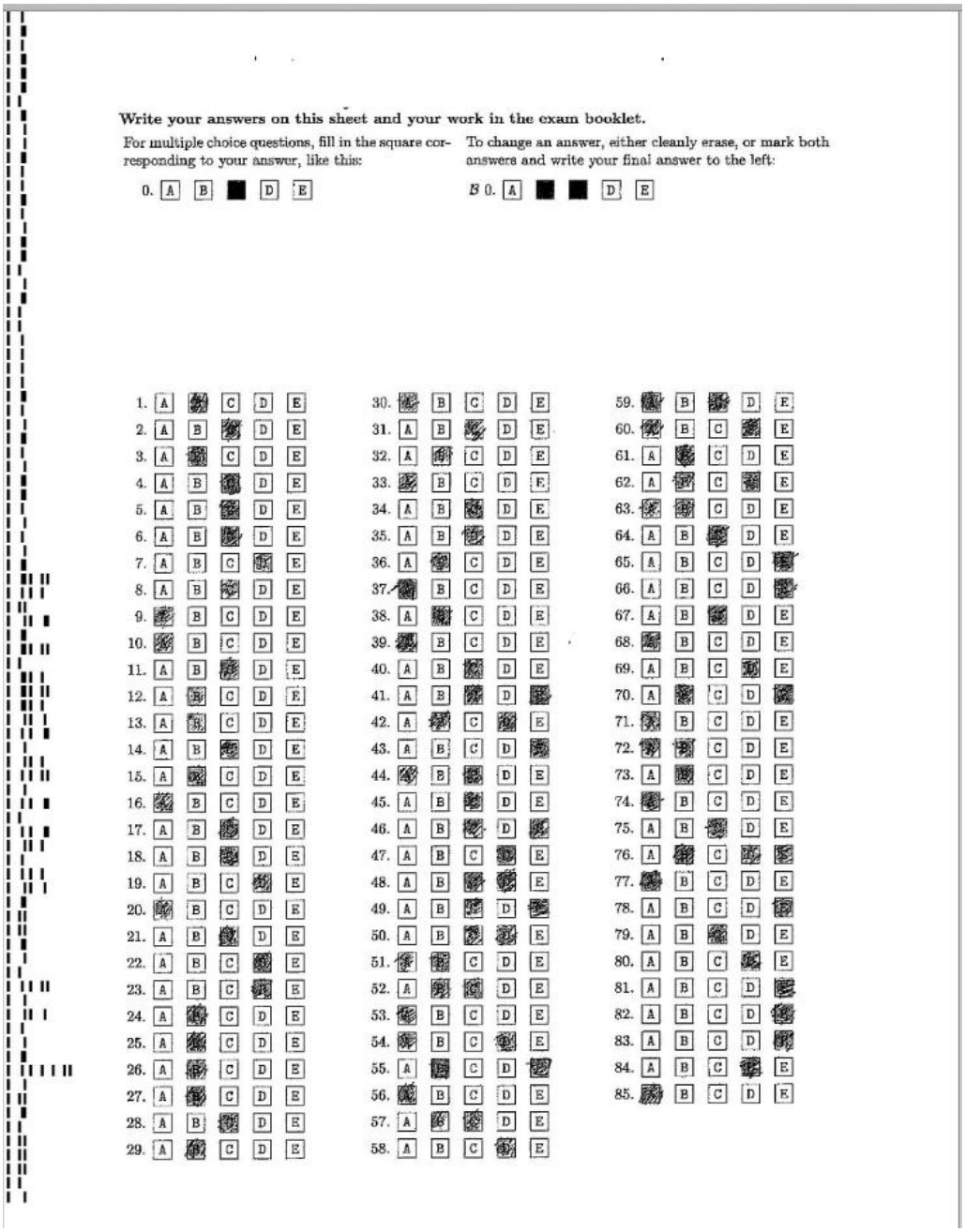
The concept of Steganography has been widely used for a host of purposes viz. Image Hiding, Processing, Compression etc. This concept was used for the practical implementation of the Assignment 1 of the Computer Vision Course wherein the purpose was to detect the answers for the Multiple-Choice Questions (MCQ) using the Optical Magnetic Reader technology.

Injecting encoded answers within the answer sheet

To achieve this a very simple method of bit encoding the answers are used, each ground truth answer character's ascii code is converted to a bit stream which represents the binary value of the ascii code and this code is replicated a fixed number of times both horizontally and each bit is replicated another fixed number of times vertically. This produces a binary matrix. Now since there are 5 options, to each question and all of them can be true at once, the matrix grows horizontally for each question and for the questions with less than 5 correct options appropriate number of 0 padding is appended to the end. This way we create a huge binary matrix but restricts its width and height such that it fits along the left vertical margin of the OMR sheet. This binary matrix is then embedded into the answer sheet.

Decoding answers from embedded image within OMR sheet

During extracting the embedded answer, the reverse of the encoding process is applied. The zone containing the answer is fixed, so we can extract it from the matrix representation of the whole image. Also, the number of times each 8 bit ascii code is repeated horizontally and vertically is known, so the matrix is parsed indexing appropriately and the extracted binary codes of the ascii values are then converted to the answer characters.



Write your answers on this sheet and your work in the exam booklet.

For multiple choice questions, fill in the square corresponding to your answer, like this:

0. [A] [B] [■] [D] [E]

To change an answer, either cleanly erase, or mark both answers and write your final answer to the left:

B 0. [A] [■] [■] [D] [E]

1. [A] [ ] [C] [D] [E]
2. [A] [B] [ ] [D] [E]
3. [A] [ ] [C] [D] [E]
4. [A] [B] [ ] [D] [E]
5. [A] [B] [ ] [D] [E]
6. [A] [B] [ ] [D] [E]
7. [A] [B] [C] [ ] [E]
8. [A] [B] [ ] [D] [E]
9. [ ] [B] [C] [D] [E]
10. [ ] [B] [C] [D] [E]
11. [A] [B] [ ] [D] [E]
12. [A] [ ] [C] [D] [E]
13. [A] [ ] [C] [D] [E]
14. [A] [B] [ ] [D] [E]
15. [A] [ ] [C] [D] [E]
16. [ ] [B] [C] [D] [E]
17. [A] [B] [ ] [D] [E]
18. [A] [B] [ ] [D] [E]
19. [A] [B] [C] [ ] [E]
20. [ ] [B] [C] [D] [E]
21. [A] [B] [ ] [D] [E]
22. [A] [B] [C] [ ] [E]
23. [A] [B] [C] [ ] [E]
24. [A] [ ] [C] [D] [E]
25. [A] [ ] [C] [D] [E]
26. [A] [ ] [C] [D] [E]
27. [A] [ ] [C] [D] [E]
28. [A] [B] [ ] [D] [E]
29. [A] [ ] [C] [D] [E]
30. [ ] [B] [C] [D] [E]
31. [A] [B] [ ] [D] [E]
32. [A] [ ] [C] [D] [E]
33. [ ] [B] [C] [D] [E]
34. [A] [B] [ ] [D] [E]
35. [A] [B] [ ] [D] [E]
36. [A] [ ] [C] [D] [E]
37. [ ] [B] [C] [D] [E]
38. [A] [ ] [C] [D] [E]
39. [ ] [B] [C] [D] [E]
40. [A] [B] [ ] [D] [E]
41. [A] [B] [ ] [D] [ ]
42. [A] [ ] [C] [ ] [E]
43. [A] [B] [C] [D] [ ]
44. [ ] [B] [ ] [D] [E]
45. [A] [B] [ ] [D] [E]
46. [A] [B] [ ] [D] [ ]
47. [A] [B] [C] [ ] [E]
48. [A] [B] [ ] [ ] [E]
49. [A] [B] [ ] [D] [ ]
50. [A] [B] [ ] [ ] [E]
51. [ ] [ ] [C] [D] [E]
52. [A] [ ] [ ] [D] [E]
53. [ ] [B] [C] [D] [E]
54. [ ] [B] [C] [ ] [E]
55. [A] [ ] [C] [D] [ ]
56. [ ] [B] [C] [D] [E]
57. [A] [ ] [ ] [D] [E]
58. [A] [B] [C] [ ] [E]
59. [ ] [B] [ ] [D] [E]
60. [ ] [B] [C] [ ] [E]
61. [A] [ ] [C] [D] [E]
62. [A] [ ] [C] [ ] [E]
63. [ ] [ ] [C] [D] [E]
64. [A] [B] [ ] [D] [E]
65. [A] [B] [C] [D] [ ]
66. [A] [B] [C] [D] [ ]
67. [A] [B] [ ] [D] [E]
68. [ ] [B] [C] [D] [E]
69. [A] [B] [C] [ ] [E]
70. [A] [ ] [C] [D] [ ]
71. [ ] [B] [C] [D] [E]
72. [ ] [ ] [C] [D] [E]
73. [A] [ ] [C] [D] [E]
74. [ ] [B] [C] [D] [E]
75. [A] [B] [ ] [D] [E]
76. [A] [ ] [C] [ ] [E]
77. [ ] [B] [C] [D] [E]
78. [A] [B] [C] [D] [ ]
79. [A] [B] [ ] [D] [E]
80. [A] [B] [C] [ ] [E]
81. [A] [B] [C] [D] [ ]
82. [A] [B] [C] [D] [ ]
83. [A] [B] [C] [D] [ ]
84. [A] [B] [C] [ ] [E]
85. [ ] [B] [C] [D] [E]

Performance

Although this scheme works with encoding and decoding images, but it fails to ensure robustness when the embedded answers are scanned through a scanner. As a scanning machine crops the actual physical copy of the printed OMR sheets and adds its own margins, the indexing scheme fails to intercept the precise starting index of each ascii code.
To ensure robustness a margin of error is to be kept on all 4 edges of the matrix representation of the ground truth answer, such that the actual answers start from a fixed number of randomized pixel values. Such methods can be experimented to extend robustness of the system.

Other implementations for the project

Encoding answer following a more robust approach

Following were 2 other methods which we tried to encode answers into the sheet. However, the results were not as per expected and due to limitation of time could not pursue our theory any further.

Approach 1 - Encoding ASCII values

1. The given image OMR sheets have a total of 85 questions with each question having five options viz a, b, c, d & e. The answer to each question could be one or one more than one of the five options mentioned above.
2. For each option per question, we extracted the ASCII value and converted the same into the binary format.
3. This was followed by the same value being encoded into the group of 8 bits i.e. 1 byte.
4. The bits were assigned a notion such that a bit having a value 1 was odd and the bit having a value 0 was an even value bit.
5. Proceeding through the entire image of answer sheet in an (N x M) matrix format, the pixel value was compared to the byte value of the character. If both the values were not in sync, then the pixel value was adjusted to reflect the byte value of the character.

Pros
Easy to implement

Cons
The downside with this approach was that it was not found to be suitable for the JPEG compression due to the encoding into the group of 8 bits and hence the following approach was devised.

Approach 2 - Encoding ASCII values by inverting the bit notions
In short, instead of encoding the extracted ASCII value into a group of 8 bits, the same was encoded using the two extreme values of ASCII chart.
This approach was like the previous one except for the fact the ASCII value extracted and converted into the binary format was assigned the meaning such that the bit holding a value of 0 was even (0) and the bit holding a value was considered odd (255).

Pros

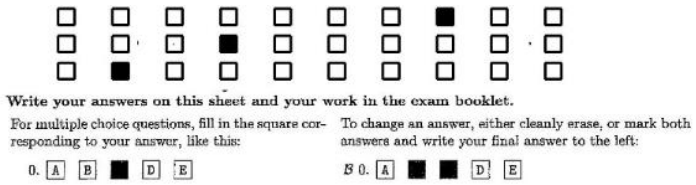
- This approach was found to be well suited for compressing the JPEG images.
- It was found that the pixel values of the resultant image deviated by 15 % to 20 %. However, this issue too was dealt with by applying a threshold value of 120. This value was considered considering the extreme values of ASCII chart used for encoding.
- Applying the threshold value equipped us with a clear segregation of the pixel values into odd and even group.

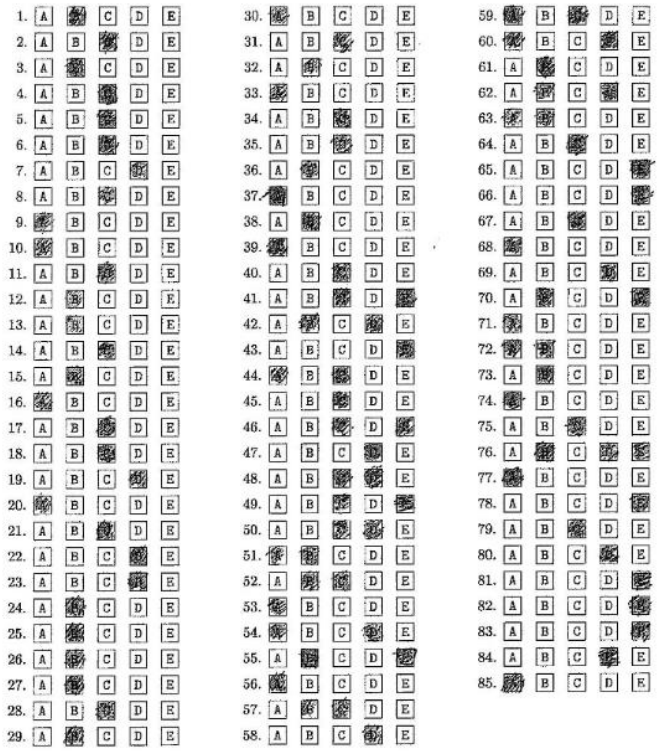
- Since encoding every answer option into the OMR sheet and then retrieving it is susceptible to noise induced by the scanning machine used to scan the PMR sheet. We tried a different approach for encoding the answers into the answer sheet.

Future work

Inject file name containing answers instead of answers

- Generate a random number for every question set (ground truth input)
- Create a text file with the same name as the generated number and place all the answers from the ground truth file into it.
- Post that, place the generated number onto the OMR sheet using each digit as a corresponding pixel value. The length of such a pixel encoding would depend on the length of the range of numbers we are generating the random number from.





Deriving answer file name

- The answer file name is encoded into a specific rectangular block on top of the OMR sheet, whose coordinates are known.
- To extract this zone, we can use Hough transform, which would place straight lines around this zone and then we can parse the enclosed area, and extract the file name by reading the pixel values.

Although we were able to get the encoding part of this scheme to work but due to limitation of time we could not pursue further on the implementation of decoding part of this scheme.

References

1. Medium article: [[https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123] [Canny edge detection]]
2. Github archive by alyssaq for [[https://github.com/alyssaq/hough_transform] [Hough Transform]]
3. Wikipedia [[https://en.wikipedia.org/wiki/Hough_transform] [Hough Transform]]
4. Wikipedia [[https://en.wikipedia.org/wiki/Edge_detection] [Edge detection]]
5. Digital image processing [[https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture06.pdf] [lectures]] by Professor Emmanuel Agu
6. Lecture [[http://me.umn.edu/courses/me5286/vision/Notes/2015/ME5286-Lecture9.pdf] [slides]] by Professor Saad Bedros