NEED TO BE STYLED

**Team Members**

1. Ahmed Alaa

2. Beshoy Morad

3. Zeyad Tarek

4. Waleed Mohamed

**Project Idea**

1. Grade Auto Filler
2. Bubble sheet correction

**Project Needs**

1. Python 3.10
2. OpenCV
3. Numpy
4. Skimage
5. Imutils
6. Pytesseract
7. Xlsxwriter

**Project Description**

The main idea of (Grade Auto Filler) is to give an image of a table with some data to the program and get an output of excel sheet containing the data that was in that image after mapping the symbols to the wanted grades.

The main idea of (Bubble sheet correction) is …

**Constraints**

1. The image should be clear, and the light distribution is appropriate and not too bad.
2. Handwritten numbers and symbols should be clear.

**Used algorithms**

To detect cells with normal image processing:

1. Enhance the image of the cell to remove the noises
2. Get the angle of lines in that cell and if it was in a certain range then this cell is **correct mark**
3. Perform opening operation on the cell with vertical and horizontal kernels to get the number of vertical and horizontal lines in that cell.
4. Get the contours and calculate the area then check if it was higher that a certain value then this cell is **Box**
5. Perform houghCircles with a certain value for min and max radius to check if that cell is **Question mark**
6. If it was nothing from the above, then check if it was **horizontal lines or vertical lines** cell
7. If it was not horizontal or vertical, then it is an empty cell

To detect cells using HOG:

1. We have to train the HOG model using a set of images with different scales and different shapes.
2. We can use that model to predict the upcoming cells.

**Experiment results**

We used 15 samples with different angles of capturing (Skewing, orientation, scale) and with different hand-writing fillings.

The results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sample Number | Number of wrong detected symbols with normal image processing | Number of wrong detected codes with OCR | Number of wrong detected numeric values with OCR | Number of wrong detected numeric values with features + classifier |
| 1 | 1 | 0 | 2 | 2 |
| 2 | 0 | 0 | 4 | 3 |
| 3 | 2 | 1 | 4 | 2 |
| 4 | 1 | 0 | 5 | 1 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 3 | 1 | 1 |
| 7 | 0 | 0 | 1 | 2 |
| 8 | 0 | 1 | 1 | 0 |
| 9 | 1 | 1 | 6 | 0 |
| 10 | 0 | 0 | 4 | 1 |
| 11 | 0 | 0 | 7 | 0 |
| 12 | 0 | 0 | 2 | 1 |
| 13 | 0 | 0 | 4 | 0 |
| 14 | 0 | 0 | 2 | 0 |
| 15 | 0 | 0 | 2 | 0 |

**Accuracy of Symbols**

Number of test cases = 17 cell \* 2 columns \* 15 samples = 510

Number of wrong detected symbols = 7

Accuracy of symbols detection with normal image processing techniques

= (510 - 7) / 510 = **98.6%**

Accuracy of symbols detection with HOG model = **100%**

**Accuracy of code and numeric values detection using OCR**

Number of test cases = 17 cell \* 15 sample = 255

Number of wrong detected codes = 6

Number of wrong detected numeric values = 46

Accuracy of code detection = (255 - 6) / 255 = **97.6%**

Accuracy of numeric values detection = (255 - 46) / 255 = **82%**

**Accuracy of code and numeric values detection using features and classifier**

Number of test cases = 17 cell \* 15 sample = 255

Number of wrong detected codes = SOON

Number of wrong detected numeric values = SOON

Accuracy of code detection = (255 - SOON) / 255 = **SOON%**

Accuracy of numeric values detection = (255 - SOON) / 255 = **SOON%**

**Analysis**

After testing the program with a lot of samples we noticed that OCR is good with codes but not very efficient with numeric values, also most of wrong numeric values detected was (**digit 1**) and sometimes (**digit 7**).

For symbols, most of wrong detected cells were right mark and question mark because the angle of lines of right marks sometimes gets out of the specified range, also question marks sometimes the radius of the circle gets out of the specified range.