IVP Mini Project Documentation:

By:- Shalaka Gedam BT16CSE028, Snigdha Patil BT16CSE066

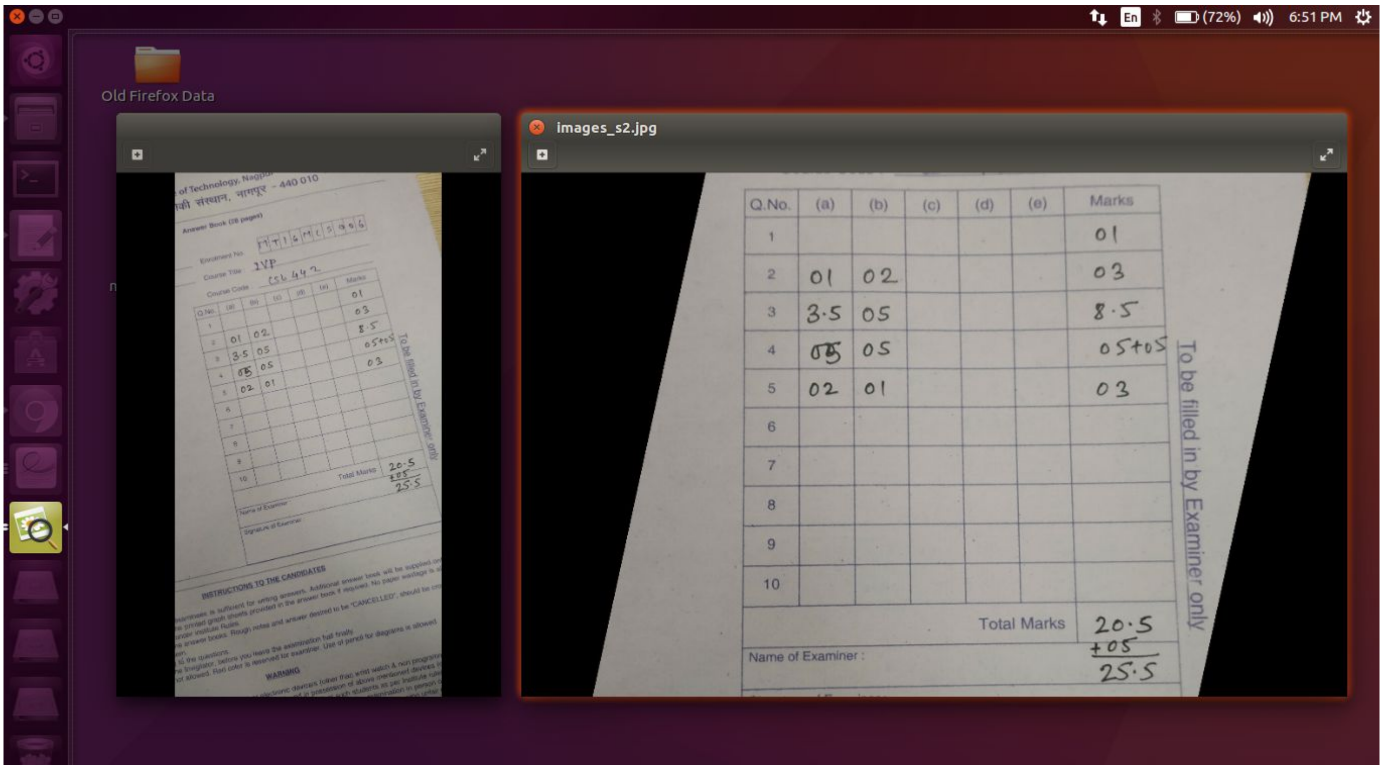
The python files were run in the following order:

1. stage2.py

   rotate_img() - Using Canny Edge detection all the edges are detected and then these edges are given to HoughLinesP to detect all the lines in the image. Then we rotate the image by the median of all the angles corresponding to each line to make it horizontal. **The resulting image could be horizontal or vertical.**
   In main function, we check if height < width. If yes, then we rotate by 90 degrees. **So now our image is vertical i.e. either straight or upside down.**

   show_grid() - We did morphological processing as follows:
   Median blurring, adaptive threshold, erode(thickens the lines).
   Then we store the dimensions of all the boxes and pass them to invert_img()

   invert_img() - Sort all boxes on the basis of 'y' coordinates. Calculate the difference between the 1st and the 7th boxes. If difference is greater than some threshold we rotate it by 180 degrees. **Now, the image is vertically straight.**
   After appropriate cropping we use these images to store the small cropped boxes of the grid to store the marks.

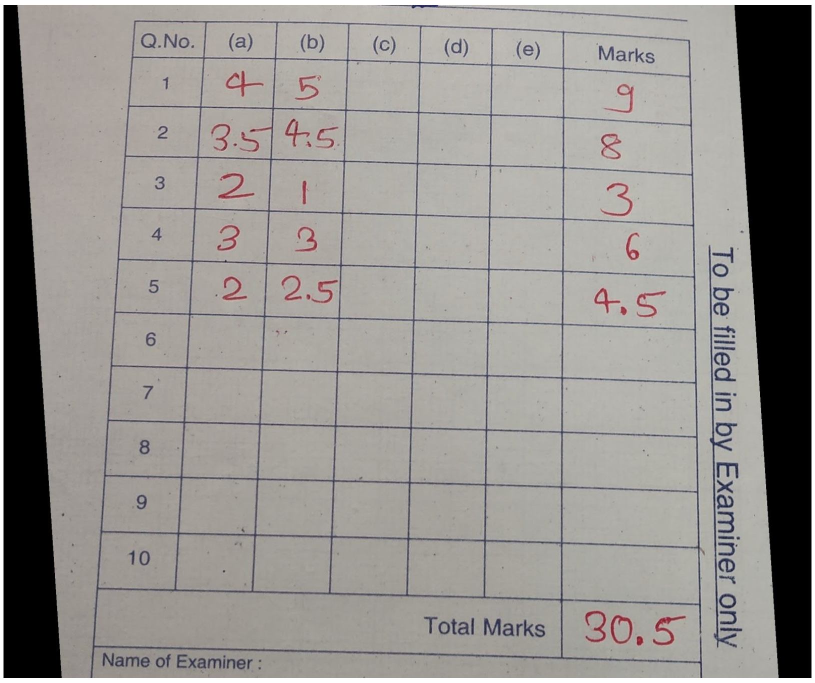


Q.No.	(a)	(b)	(c)	(d)	(e)	Marks
1						01
2	01	02				03
3	3.5	05				8.5
4	05	05				05+05
5	02	01				03
6						
7						
8						
9						
10						
					Total Marks	20.5 +05 25.5

Name of Examiner :

To be filled in by Examiner only

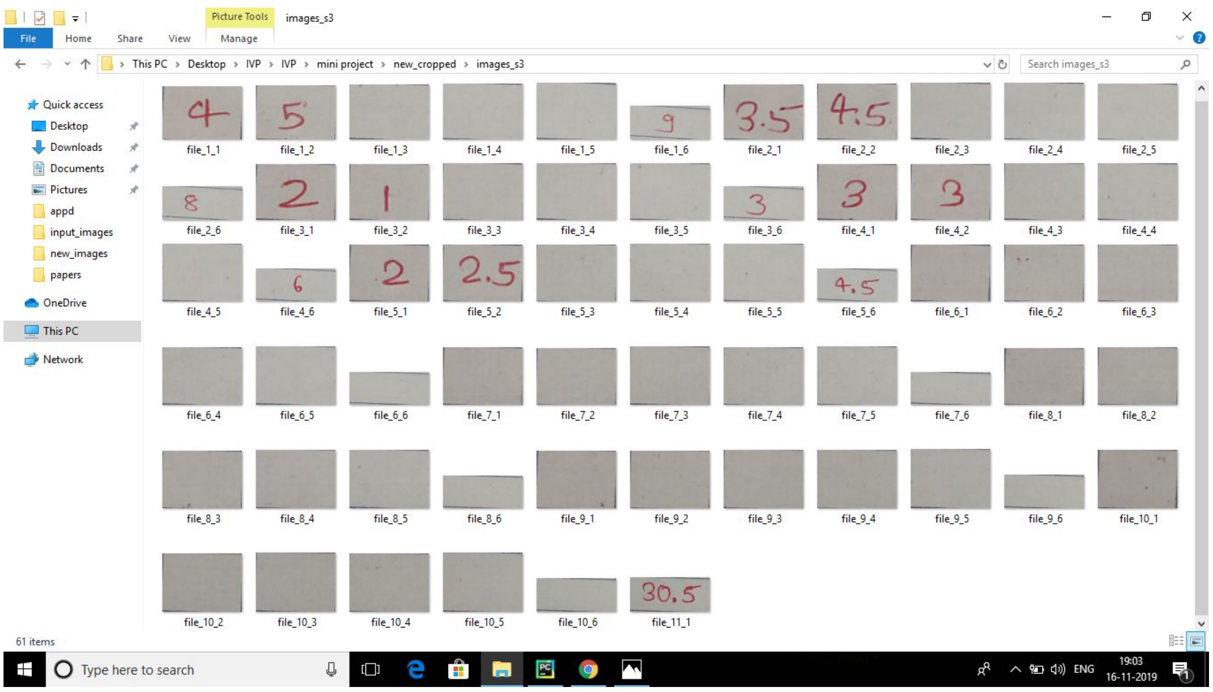
2. stage1.py
   The illumination of the paper was varying so we used adaptiveThreshold. Then after doing some morphological operations as given in the code we detect all the grid boxes which lie in a particular range of height and width. We find contours in each box. If we find contour then we store the cropped grid box along with question and sub-question number in a folder.
   For the following rotated image(obtained from stage2.py) we store the cropped boxes:



Q.No.	(a)	(b)	(c)	(d)	(e)	Marks
1	4	5				9
2	3.5	4.5				8
3	2	1				3
4	3	3				6
5	2	2.5				4.5
6						
7						
8						
9						
10						
					Total Marks	30.5

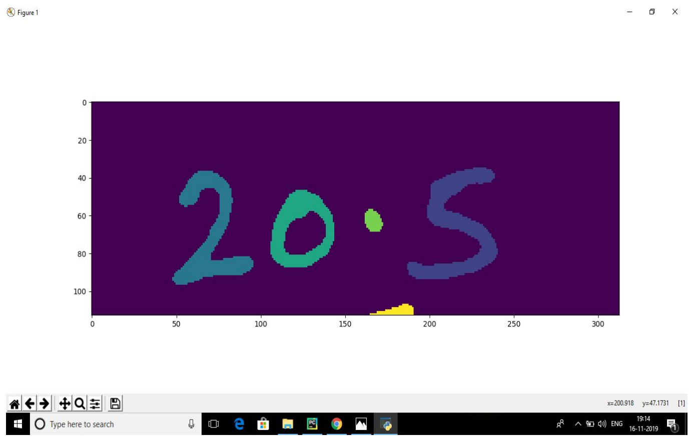
Name of Examiner :

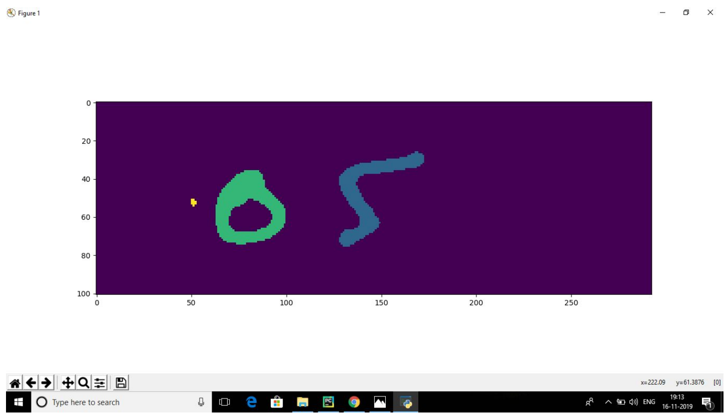
To be filled in by Examiner only



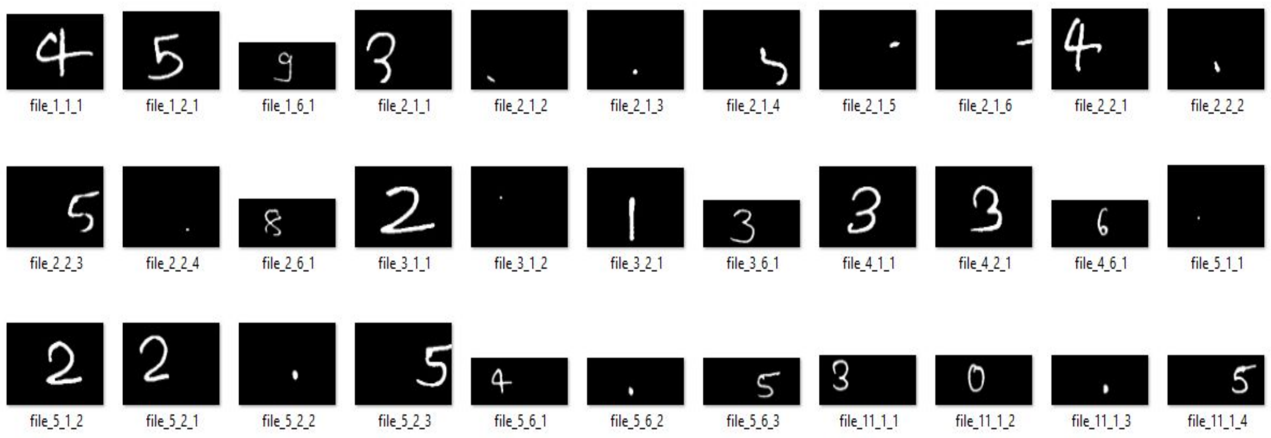
3. get_components.py

Taking the images one by one from above folder, we crop it by 10 on all four sides to get of border grid lines. We invert the binary image and dilate it. Then using connected components we detect the individual objects in the box, which includes digits, decimal point as well as some noise components. They look as follows:





We process these images only if the area of the largest component is greater than some threshold. Then we store all the individual components in separate blank images if the area is greater than some threshold and **it does not lie completely in the margin area.** Thus, we get rid of many noise components because of blurring and area condition. **We maintain the ques no. and sub-ques no. and the part no. for each component.**

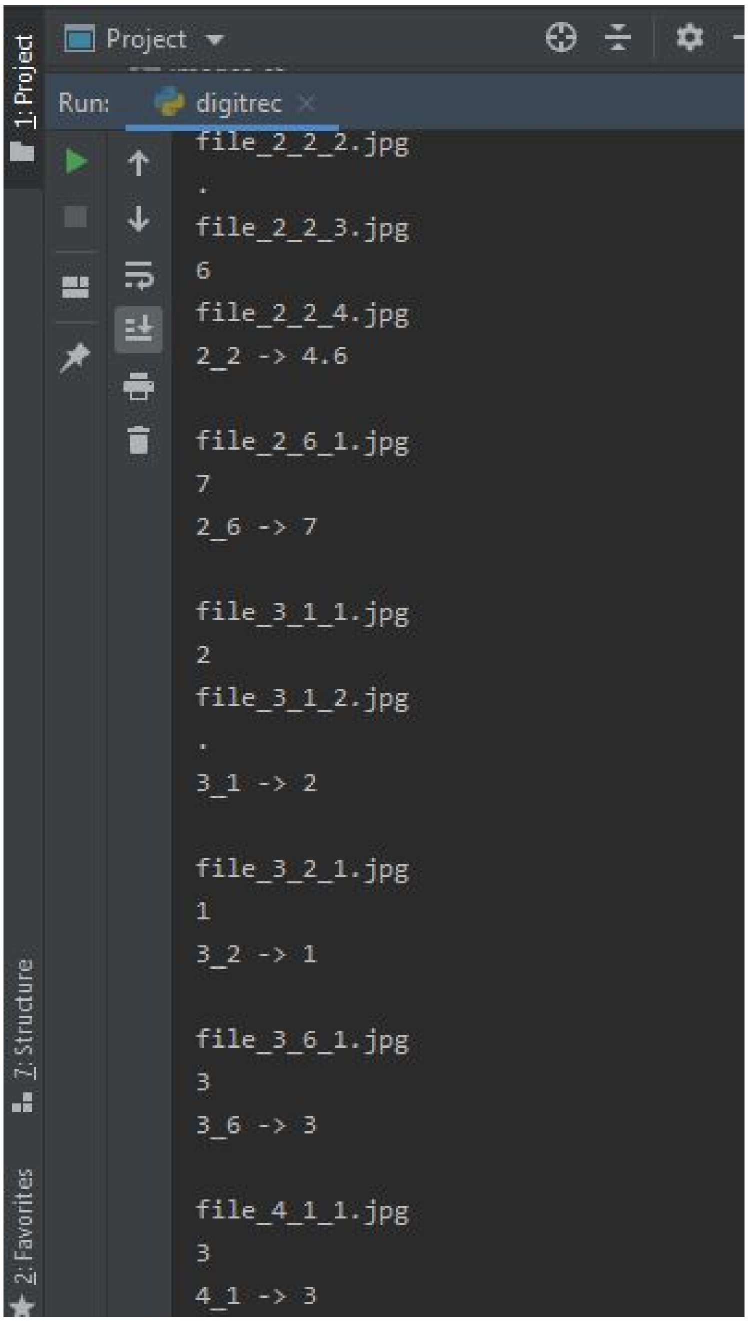


So as we can see there were 61 items in paper cropped folder and in the above folder there are only 33 items. So we got rid of most of the noise points. The remaining noise points were taken care of in digit_rec.py code.

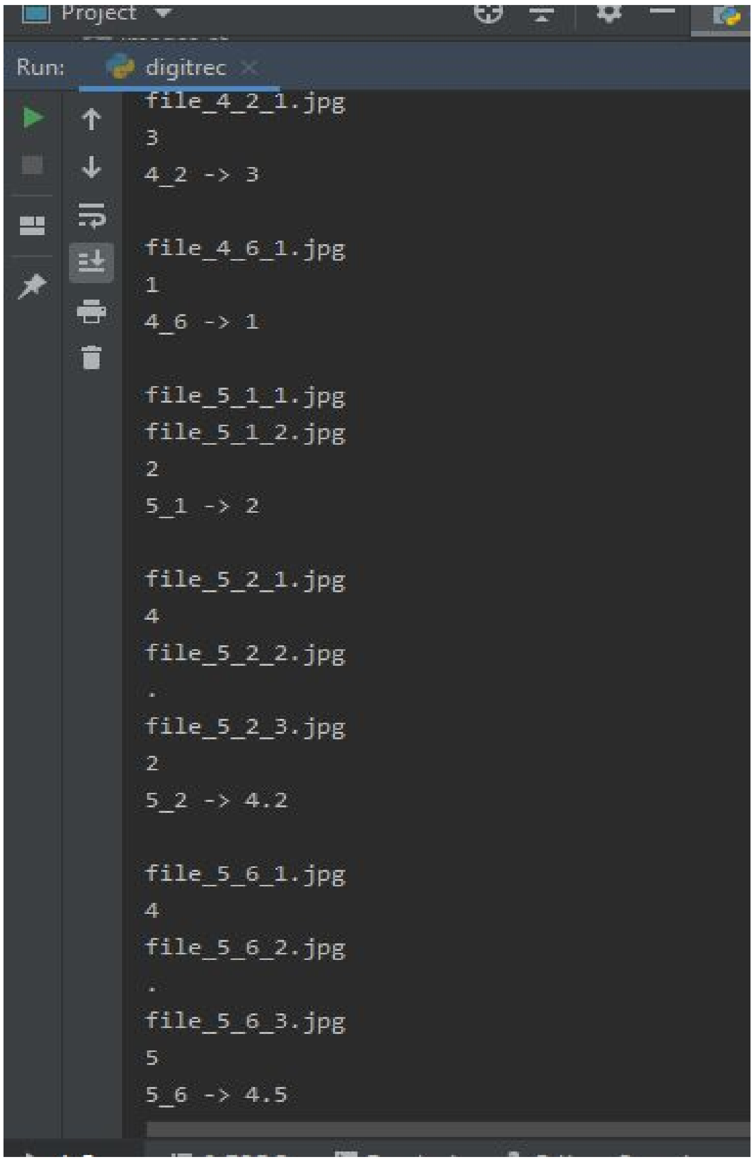
4. digitrec.py
   We maintained three flags as left_dig, right_dig, point to assure that decimal point occurs between two digits. So, we get rid of many noise points which used to be confused with the decimal point.
   We counted the number of white pixels and if it was greater than some threshold then we give it for prediction, else we say it is a decimal or a noise point. In 'marks' string we append the predicted values and output them.
   Thus, we are able to detect the marks inside the grid box along with its question and sub-question number.

   The following was our final output :

   In the output, for example,
   3_1 -> 2 means Ques no. 3 (a) has an entry 2
   Similarly,
   5_6 -> 4.5 means Ques no. 5 (total) has an entry 4.5'
   And 11_1 means grand total marks.
   (We have also outputted the individual component prediction)



```
file_2_2_2.jpg
.
file_2_2_3.jpg
6
file_2_2_4.jpg
2_2 -> 4.6

file_2_6_1.jpg
7
2_6 -> 7

file_3_1_1.jpg
2
file_3_1_2.jpg
.
3_1 -> 2

file_3_2_1.jpg
1
3_2 -> 1

file_3_6_1.jpg
3
3_6 -> 3

file_4_1_1.jpg
3
4_1 -> 3
```



```
file_4_2_1.jpg
3
4_2 -> 3

file_4_6_1.jpg
1
4_6 -> 1

file_5_1_1.jpg
file_5_1_2.jpg
2
5_1 -> 2

file_5_2_1.jpg
4
file_5_2_2.jpg
.
file_5_2_3.jpg
2
5_2 -> 4.2

file_5_6_1.jpg
4
file_5_6_2.jpg
.
file_5_6_3.jpg
5
5_6 -> 4.5
```