Format Of An MAT Object

→ MAT object-name (row, column, 3-channel (BGR))

Functions Used In Program

(a) void sampleFrameDraw (Mat resolved).

→ This function is just for programmer purpose.
→ It will create a chessboard like thing.
→ This is just to see the pixel size (resolved one).

(b) void invertImage (Mat Original);

→ The image which we are getting is a selfie image.
→ so to make it as per our actions, we have inverted it. (i.e, if we move our hand to right then, the image should do the same not opposite).

(c) void initializeMatObject (Mat);

→ Here we have used to initialise a MAT object obj to 0, which we will use further

Note:

MAT objects used.

Mat original :- As soon as we turn our web cam on, the real image is the Mat Original object.

Mat red :- This object is used for the detection of red light. Our original Mat object is now converted to Mat red object.

Mat canvas :- This is our screen where we will draw out different things and this is stored in Mat canvas object.

Mat resolved :- Our original screen is 720x1280; so we have resolved our pixels to 72x128 (by scaling factor 10) and then this is stored in Mat resolved object.

Mat scaled :- Whatever we have drawn and resolved; we have to show it in a fixed size image (e.g. 30x30); now this is stored in Mat scaled object.

Mat filtered :- Removing all the unwanted drawings / pixels from the resolved and ~~and~~ scaled one and store in Mat filtered object.

vector<Mat> spl; :- This is used to convert 3 channels into 3 independent frames of B, G and R.

(d) void thresholding ()

→ This helps in reading the red frame of spl.

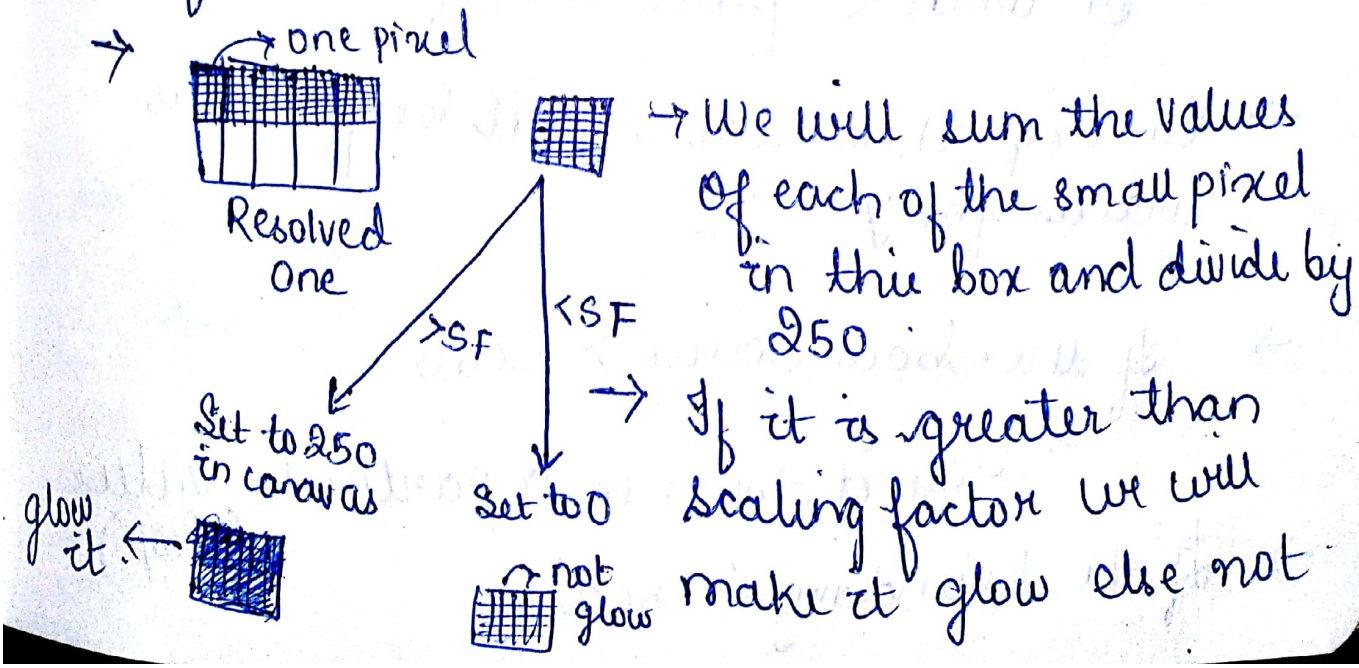
→ If the value of pixel >250
the set the value at canvas = 250
else set the value in canvas = 0.

→ This is used to just show our pixels with two value either 0 or 250.

(e) void setResolution (Mat Canvas, Mat resolved)

→ Decrease the size of image by scaling factor.

→



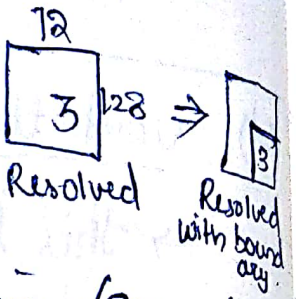
→ We will sum the values of each of the small pixel in this box and divide by 250.

→ If it is greater than Scaling factor we will make it glow else not.

(f) void findBoundary (Mat resolved)

→ This is used to find the boundary of image

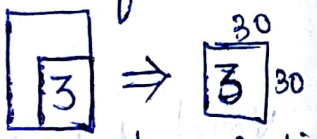
(i.e., min x of glowing pixel
min y where pixel is glowing
max x " " " "
, max y " " " ")



→ This we can used further in scaling (cropping)

(g) void scaling (Mat resolved, Mat scaled)

→ This is used to scale or crop our original image to a particular fixed scale size



→ There will be two cases :-

① When > particular fixed size

② When < particular fixed scale size

→ Each of these cases will be for X as well as for Y.

→ If the boundary > Scale

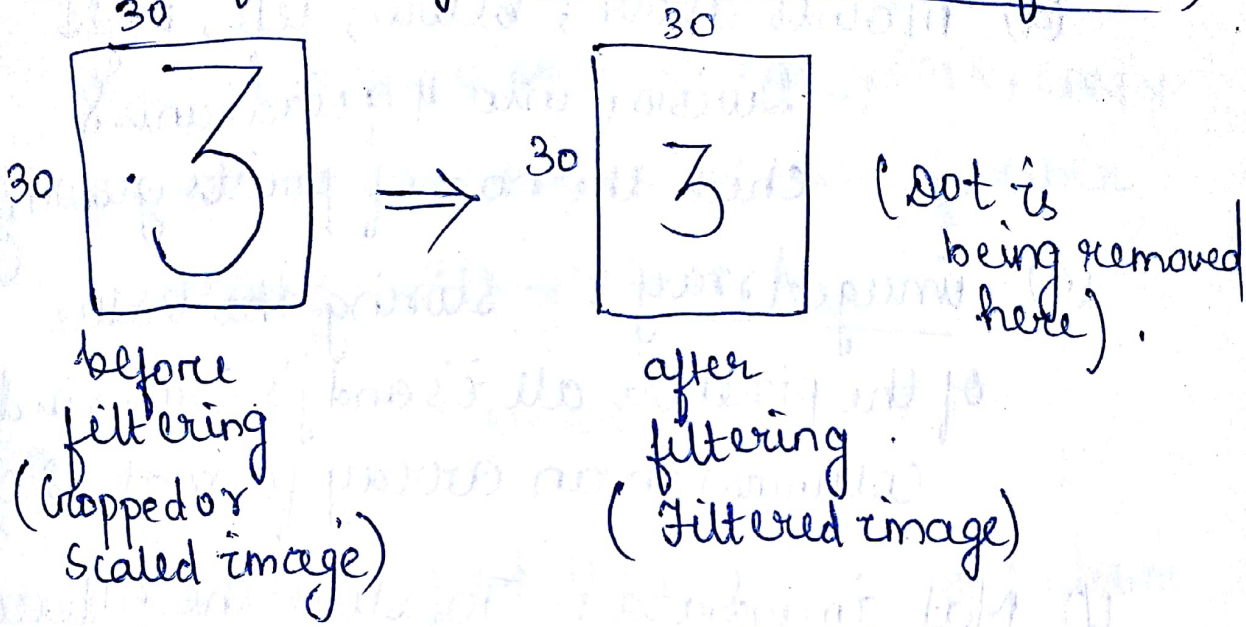
→ Scale it down to a particular value (Crop).

→ If the boundary < scale

→ Scale it up to a particular value (enlarge).

→ Then if the pixel is glowing ; then set that scaled value to 250 (glow)
else set it to 0 (Not glow)

(h) void filtering (Mat scaled, Mat filtered).



→ This function we are using to remove the unwanted dots, drawing, lines, etc.

→ Here we are watching that if near these glow pixels if nearly (most) pixels are not glowing then unglow that pixel also.

→ Else if all the pixels (nearly) are glowing then let it be glowing.

Recognition part.

→ Structure used (elements in the same)

(a) char id :- The id of the image sh

(b) ComX, ComY :- Centre of mass.

(c) nPoints :- No:- of points glowing.

(d) nPoints above ; below ; left, Right
:- Division into 4 quadrants & check the no:- of points glowing.

(e) imageArray :- storing the value of the pixel at all i's and j's (rows and columns) in an array format (2-D)

(f) Mat imageData :- To store the filtered image which we can use further.

@ initialiseImageToStruct

→ Store our filtered image into a Mat object (imageData).

(b) makeArrayFormat

→ To store all the values of the pixel in a 2-D array so that we can further use it.

(c) void setId (element * temp, int id)

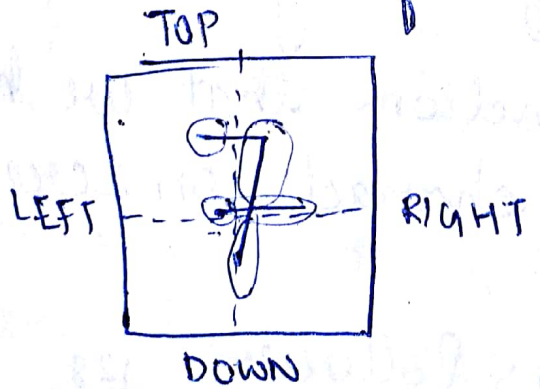
→ Initialising our data's id to the id (stored image ; i.e, trained data).

(d) setCommPoints (Element *temp)

→ This is used to find the COMX, COMY and npoints which we can further used for recognition.

(e) setnPoints (element * temp)

→ This is to find the npoints above, down, left and right that are glowing which can be further used for recognition



(f) int writeImage (int ch, int count)

→ This is used to further trained the data.

→ Whatever we are writing, are saved as an image in the given path; which we can use further to recognise data.

(g) void getImageForTraining (Element * temp, int id, int count)

→ Here we are storing the path of the trained image in the image Data which we can further used for recognition.

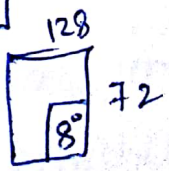
(h) void recognition():

→ This is basically used for setting all our windows.

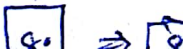
→ Some of the functions that we have used to write character on screen are called here.

The work goes as follows:-

→ setResolution 

→ Finding the Boundary ⇔ 

→ Scaling ⇔ 

→ filtering ⇔ 

(i) void training().

→ This is used to calculate the COM, npoints, setnpoints of the filtered image.

→ Here we call our functions :- setId, getImageForTraining, setCommPoints, SetnPoints.

→ And then push to our vector - trainedData.

(j) void displayTrainedData().

→ Whatever we have calculated of the data, we will display it using this function.

(k) int findNearestPixel (vector <element> :: iterator it, int i, int j, int *y, int *x)

→ This is used to find the glowing pixel. (By comparing with 4 pixels ↑ ← ● → ↓)

→ If the pixel is glowing, we got our desired pixel else we will push all its neighbours to stack and Pop by finding their nearest glowing pixel

(l) long long calculateSquaredDistance (-

→ After find the nearest glowing pixel by the previous function, we need do calculate the squared distance.

→ The formula used :-

distance += (abs(i-y) * abs(i-y)) +
(abs(j-x) * abs(j-x));

(m) void displaySquaredDistance (int *labelTo Accept)

→ This function is used to display the the squared distance of the nearest glowing pixel for every image in the data set.

(n) int SquaredDistanceLogic (-)

→ This will return the minimum of all the compared squared distance (which we have calculated earlier for every image).

o) void setIteratorValuesFromVectorElement(-)

→ Here we are going to a particular index of the IteratorArray and storing the iterator of that particular character in the dataset.

p) void getImageIntoMat(—)

→ Storing the image (basically for heading) in an Mat object called heading.

q) void displayImageAtPosition (—)

→ Here we are going to that particular x and y co-ordinate and then print the heading or any other image we want.

(We are basically using this to print SLATE, CALCULATOR, etc at top).

→ The image is overwriting again and again when we are calling this function again & again

(2) int secondActivity (—)

→ It just acts as a pathway from canvas to a particular activity.

→ Whatever character is recognised in activity, it is passed as a parameter to this function, and then it opens that particular activity.

#

- MusicActivity ()

→ If the character M is recognised in secondActivity then just run the particular command defined in this activity in terminal.

void setLabelFrequency (—)

→ It make the array of the frequen for labels that are passed as the parameters in this function.

[Labels → which we have defined at the start of the program

Calculator Activity:

→ Here we have made a calculator window

→ Reading of data is same as previous (Here we have used a canvasbackup window, anything we write in this window get saved in canvas window & viceversa).

→ we will print the heading at a particular position by using the previously described function.

→ If the key pressed is Escaped, then clear both Canvas & CanvasBackup.

→ If Enter key is pressed; then recog-nise the character and perform the desired statements related to it.

→ < → Back to Canvas.

= → Break & then perform the calculation.

→ void displayResultInCalculator (~) :

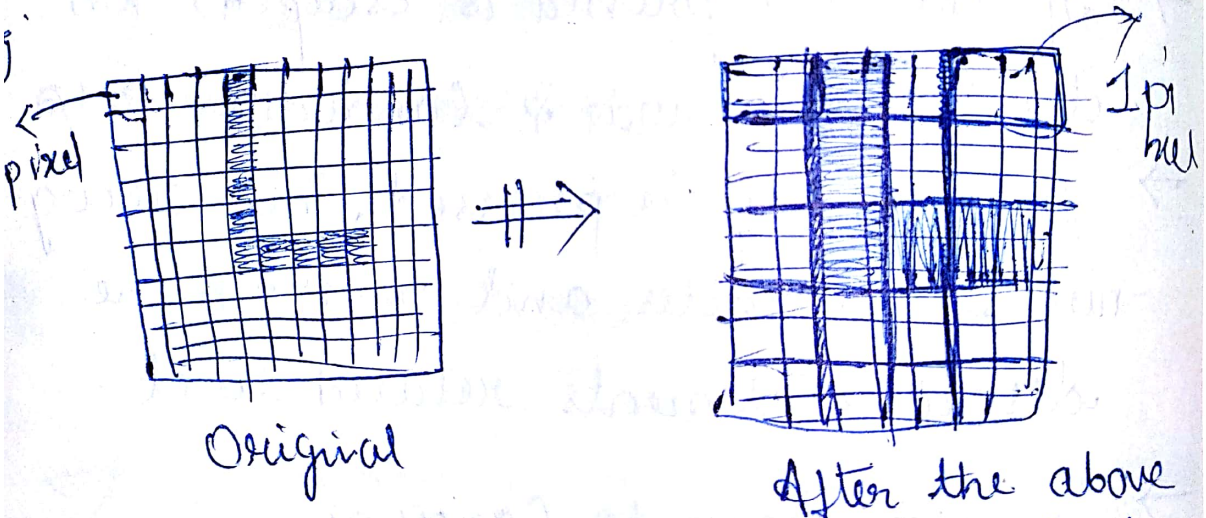
→ Push the result into the vector

→ Then print it at a particular position

→ After some time period, it will come back to the canvas window.

F AsciiArtActivity() :

→ Here we are doing thresholding, dividing our screen and glowing that particular by taking an average of all the pixels inside it.



Original

After the above done task.

→ Now we have a character set according to the brightness of the pixel.

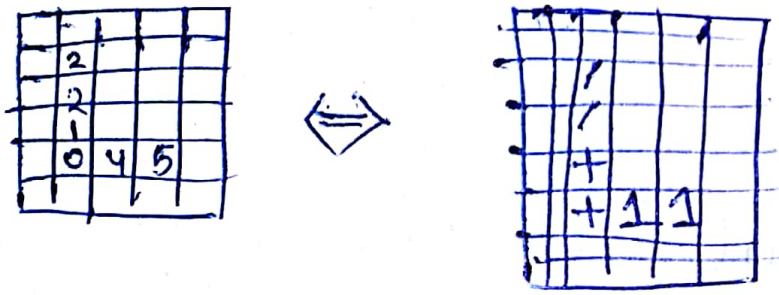
Each pixel will have value between 0 to 256. So according to the value

we have a character set (maybe less than 256). And we will print that character according to the value assigned to the ~~index~~ pixel (compare index to pixel value)

→ If the data set (character set) has less than 256 character then we can assign two index to one character.

Eg:- We need 29 characters.

We have 19 characters.



+ / 1 2 3 0 A B C ? D 4

0,1 2,3 4,5 6,7 8,9 10,11 12,13 14,15 16,17 18,19 20,21 22,23

void imageBoardActivity ().

→ This is used to draw any drawing you want.

→ Here there is no concept of recognition.

→ Here we will be using two modes - pen and Eraser and our default mode is pen mode.

→ If character p is pressed → pen mode
" " e " " → eraser mode
" Esc key is pressed → clear
If backspace key is pressed → back to canvas

→ In pen mode:
→ Thresholding
→ getImageIntoMat.

→ In Eraser mode.
→ function eraser (canvas is called) and then getImageIntoMat.

void eraser (Mat canvas).

→ At first we have nothing in eraserbBackup so we will initialize it with 0.

→ Then we need to glow at the pointer of eraser, which will be saved in the eraserbBackup and in the next iterator we will erase it.

This will be running continously in eraser mode.

void BrightnessActivity (—)

→ This is used to increase or decrease the brightness by swiping left or right.

→ It is basically done using starting X and end X.

If the difference > 0 (Brightness will increase)
else Brightness will decrease.

→ → → (Brightness increased)

← ← ← (Brightness decreased)