Project 1: Image processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

**Technology Used :**

1)Programming Language :- C++

2) Software :- Visual Studio Code (1.52.1) and OpenCV

3)Platforms :- Geeks for Geeks and openCv tutorial for C++(For learning purpose)

**Steps to run project :-**

1) Download ZIP and extract the file on your local system or clone repository using below command in command prompt :

2) Open cloned file in Visual Studio Code with OpenCV

3) Open Terminal >> Run Build Task.. (or Ctrl + F7)

4) In Terminal below , after successful build.

* Run following commands :  
  • gcc ip.cpp (program\_name.c) • .\imageprocessing.exe

Method / Approach Used :

Rotating images by a given angle is a common image processing task. Although it seems little bit complicated, OpenCV provides some built-in functions making it easy to do it. Here is a simple OpenCV C++ example code to rotate an image.   
  
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////  
#include **"opencv2/highgui/highgui.hpp"**  
**#include "opencv2/imgproc/imgproc.hpp"**  
  
**using namespace cv;**  
  
 **int main( int argc, char\*\* argv )**  
 **{**  
 **// Load the image**  
 **Mat imgOriginal = imread( "MyPic.JPG", 1 );**  
  
 **//show the original image**  
 **const char\* pzOriginalImage = "Original Image";**  
 **namedWindow( pzOriginalImage, CV\_WINDOW\_AUTOSIZE );**  
 **imshow( pzOriginalImage, imgOriginal );**  
  
 **const char\* pzRotatedImage = "Rotated Image";**  
 **namedWindow( pzRotatedImage, CV\_WINDOW\_AUTOSIZE );**  
  
 **int iAngle = 180;**  
 **createTrackbar("Angle", pzRotatedImage, &iAngle, 360);**  
  
 **int iImageHieght = imgOriginal.rows / 2;**  
 **int iImageWidth = imgOriginal.cols / 2;**  
  
 **while (true)**  
 **{**  
 **Mat matRotation = getRotationMatrix2D( Point(iImageWidth, iImageHieght), (iAngle - 180), 1 );**  
  
**// Rotate the image**  
 **Mat imgRotated;**  
 **warpAffine( imgOriginal, imgRotated, matRotation, imgOriginal.size() );**  
  
 **imshow( pzRotatedImage, imgRotated );**  
  
 **int iRet = waitKey(30);**  
 **if ( iRet == 27 )**  
 **{**  
 **break;**  
 **}**  
 **}**  
  
 **return 0;**  
**}**  
**///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

#### **Explanation**

* **Mat getRotationMatrix2D( Point2f center, double angle, double scale )**

This function returns 2x3 affine transformation matrix for the 2D rotation.

Arguments -

* + **center** - The center of the rotation of the the source image.
  + **angle** - Angle of rotation in degrees (Positive values for counter-clockwise direction and negative **values** for clockwise rotation)
  + **scale** - The scaling factor of the image. (Scaling factor of 1 means its original size)

* **void warpAffine( InputArray src, OutputArray dst, InputArray M, Size dsize, int flags = INTER\_LINEAR, int bordreMode=BORDER\_CONSTANT, const Scalar& borderValue=Scalar() )**

This OpenCV function applies [affine transformation](http://en.wikipedia.org/wiki/Affine_transformation) to an image.

Arguments -

* **src** - Source Image
* **dst** - Destination image which should have the same type as the source image(The transformed image is stored in this location)
* **M** - 2x3 affine transformation matrix
* **dsize** - Size of the destination image
* **flags** - Interpolation methods
* **borderMode** - pixel extrapolation method. (Try these values; BORDER\_REPLICATE, BORDER\_CONSTANT, BORDER\_REFLECT, BORDER\_WRAP, BORDER\_REFLECT\_101, BORDER\_TRANSPARENT and BORDER\_ISOLATED)
* **borderValue** - If you use BORDER\_CONSTANT for **borderMode**, this argument define the value used for the border