**Project on Image Rotation Using OpenCV with Visual Studio**

****

**Submitted By:**

Samar Maithani

B. Tech (C.S.E.)

III – Semester

Roll No.: 1918647

Session: 2020-2021

**Making of the Project of Image Rotation with Visual Studio**

1. Download Visual Studio latest version from the web at *https://visualstudio.microsoft.com/downloads/*

Choose the community version for free access to the software.

1. After downloading the software, it will start installing, during the installation process it will ask which Workload you want to install for your Visual Studio (Community version) software. Since we have to work with C++, so choose "Desktop development with C++" and the click install.

After that it will start to download all C++ related files for the Visual Studio, it will take some time since the size of the workload is big (~1.6GB) and its downloading and installing at the same time.

1. After all the downloading and installation is done it will open the software, but first it will ask us to sign in to your Microsoft account. Sign in to your account and then the Visual Studio will open.
2. In the Software window select the option of "Create a new project", then it will show us several templets to choose from that come as a part of Visual Studio, choose the template "Console App" and click Next.

Now We can name our Project (I personally named my project simply as 'Project'), then we have to choose which location we want our project to be stored, and then we'll give the solution name to our project (By Default it will be the same name as that of our project, and if you want you can just leave it like that) and then click on Create.

Visual Studio will now open and we will have our first file. We can see our file on the Solution Explorer on the left and the contents on the right. By Default, a Pre-Built C++ program printing "Hello World!" can be seen on the screen.

1. Now we have to add the OpenCV library, for that first we have to download the latest version of the OpenCV.

Navigate to the link *https://github.com/opencv/opencv/releases* there we can find the latest build. We will download the .exe file from there (it’s actually a self-extracting archive), after the download is complete, we will run it and extract its content to our desired location.

After extraction is finished, we will have a new folder called opencv, once inside we will navigate to the bin folder which will be inside **build\x64\vc15**, open the bin folder and copy its address. Then go to “This PC”, there right click on the screen and choose properties, a System window will open, in that window look for “Advance system settings” on the left-hand side of the screen and click it. A Dialog box will open named “System Properties” in it click on “Environment Variables” and then its dialog box will open, in it select “Path” under “System variables” and then click on Edit. Another Dialog box will open there click on New and then paste the copied path there.

1. Now coming back to the Visual Studio, we will first set the platform from x86 to x64 since the pre-built binaries are built for that platform. Now select Project form the menu bar and then select properties. A Dialog box will appear and in that dialog box under “Configuration Properties” on the Left-Hand Side select “VC++ directories” and then on the right under General select “Include Directories” and edit, here we will have to add the include directories which we will find inside the opencv folder **opencv\build\include**, copy its address and paste it in the “Include Directories”. We also need to add the “Library Directory” the same way, this directory is inside the **opencv\buildx64\vc15\lib**. One more addition is needed which is to add the .lib file in the additional dependencies. Simply go to Linker (on the LHS) in the same dialog box and under that select “Input”, Select “Additional Dependencies” there input the name of the **.lib** file which is present inside the opencv lib folder, since we currently are in debugging configuration therefore we need to add the .lib file having a ‘d’ at the end of the file name (in my case it was opencv\_world451d.lib).
2. Now finally we can get started with the programming. First, we need to include the **opencv2/opencv.**hpp header file in our program. This header file will include all the other necessary header files for our application and to compile our OpenCV code. **iostream** header file should also be included because we use cout and cin in our program to print an error message to the console.

All OpenCV functions, classes and data structures are declared inside **cv** namespace. So, we have to add the '**using namespace cv'** line in the top of our program. We have to use the **std** namespace also because cout, endl and cin functions are inside the std namespace.

1. Now in the main() function we first have to load the image from the file and return it as a **Mat** object. **imread(“imagename”, int flag)** is the function used to load an image from the file. In imagename we have to give the relative or absolute path of an image file which should be relative to our cpp file. jpeg, jpg, bmp, png, etc. image file types are always supported. In flag we have to mention how the image should be loaded whether it should be colored or gray-scaled. There are several arguments for the flag such as:

**IMREAD\_UNCHANGED** - The image will be loaded as it is.

**IMREAD\_GRAYSCALE** - The image will be load as a gray-scale image (i.e. - Single channel image, Black and white image)

**IMREAD\_COLOR** - The image will be loaded as a BGR image (i.e. - 3 channel image, color image)

If we didn’t give any value to the argument then by default **IMREAD\_UNCHANGED** argument will be used.

1. If **imread()** function fails to load the image, the returned Mat object will be empty. If the Mat object is empty, **image.empty()** function will return true. In such scenarios, our program will print an error message to the console and wait for any key press. When the user press any key in the keyboard, the program will exit returning -1. It's a good practice to check whether the image is empty and exit the program. Otherwise your program will crash while trying to execute the **imshow()** function.
2. Now we have to create a window for the image to open but first we have to give the window a name which will show on its Title Bar, then we create a window using the function **namedWindow(“windowname”, int flag).** Windowname is the name of the window. That name will display in the title bar of the newly created window. This name is also the identifier for this window, and it will be used in the later OpenCV function calls to identify the window. Here the argument flag is used to determine the size of the window. The arguments used here are:

**WINDOW\_AUTOSIZE** - User cannot resize the window. Image will be displayed in its original size.

**WINDOW\_NORMAL** - User can resize the window.

By Default, **WINDOW\_AUTOSIZE** is used if no argument is passed.

1. Next we will display the image in a window using the **imshow(windowname, Mat object).** This function call should be followed by **waitKey(0)** function call in order to provide sufficient time to paint and display the image in the window. The **waitKey(0)** function will wait until a key is pressed. When any key is pressed, this function returns the ASCII value of the key and your program will continue. If we do not call **waitKey(0)** function, the image will not be displayed in the window. Then we will use the **destroyWindow(namedWindow)** function to close the opened window identified with the name, windowName and will de-allocate any associated memory usage.
2. Now we will ask the user that at which direction does we want the image to be rotated (Clockwise or Anti-Clockwise) and by what angle. And if the user wants to rotate the image Clockwise, we will multiply the angle by (-1).
3. Now first we locate the center of rotation of the image, we use **Point2f** for that purpose with center as its object name - **“Point2f center((image.cols - 1) / 2.0, (image.rows - 1) / 2.0)”** we divide the length of the column and length of the row of the image object by 2 to locate its center. Next, we will create another image object for the rotated image – **“Mat rot = getRotationMatrix2D(center, a, 1.0)”** here we use the **getRotationMatrix2D** function to transform the matrix for 2D rotation. In it’s the arguments “center”, “a” and “1.0” are for center of rotation, angle of rotation and 1.0 is the scaling factor of the image.
4. Now we use **“Rect2f box = RotatedRect(Point2f(), image.size(), a).boundingRect2f()”** to represent the rotated rectangle image on the plane. The rectangle is specified by the center point (Point2f()), width and height (represented by image.size()) and the rotation angle (a) in degrees. And the “boundingRect2f()” function returns the minimal (exact) floating point rectangle containing the rotated rectangle.
5. Next we create another image object and use **“warpAffine(image, dst, rot, box.size())”** This OpenCV function applies affine transformation to the image. And the arguments used are “image” – source image, “dst” - Destination image which should have the same type as the source image, “rot” – affine transformation matrix and “box.size()” – size of the destination image.
6. After that we again do the same thing as before create a window for the rotated image, show the rotated image using the **imshow()** function followed by the **waitKey(0)** function to keep showing the image until a key is pressed and then destroy the image window using the **destroyWindow()** function and then stop the program.