Project 1: Video-special effects

Due Monday by 11:59pm **Points** 30

The purpose of this assignment is to familiarize you with C/C++, the OpenCV package, and the mechanics of opening, capturing, manipulating, and writing images. You may work with a partner on this assignment, and feel free to confer with other class members about the details of software installation and getting things working on your own computer.

Setup

We will be using the C++ interface for OpenCV for the first four assignments, with an emphasis on speed and real-time responsiveness to inputs. You will need to download and install OpenCV4. You can follow installation instructions from the openCV web site or use a package manager like macports (MacOS), homebrew (MacOS), cygwin (Win), yum (Linux), or apt (Linux).

You may choose to use an IDE such as XCode, Visual Studio Code, etc, or the terminal for compiling and running your code. My own tools are emacs, makefiles, and the terminal, and examples are provided below. If you use an IDE, you are responsible for getting the OpenCV libraries linked and getting your code running. I recommend following the tutorials on the OpenCV site to do that. If you have not used a makefile but want to learn more, check out this makefile tutorial. Here is my sample makefile for building on a mac. My code is set up so that there are bin, include, data, and src subdirectories in my project directory. All .cpp files and the makefile are in src. All .h or .hpp files are in include. All executables are written by the makefile to bin.

Code Expectations

There are some basic coding guidelines that everyone is expected to follow if you want full credit for the assignment.

1. Your name, a date, and the purpose of the file should be at the top of each code file, both .cpp and .h files.
2. Every function needs a summary comment indicating what it does. Good comments explain the purpose of each argument and the return value.
3. Complex functions should have comments for major code blocks indicating what they do.

Tasks

1. **Read an image from a file and display it**

   Create a file imgDisplay.cpp. Your main function should read an image file and display it in a window. Then your program should enter a loop, checking for a keypress. If the user types 'q', the program should quit. Feel free to add other functionality to your program by detecting other keypresses. You can follow the OpenCV tutorials to get started with this task. You will learn more if you type all the code out yourself and modify the variable names instead of copy-pasting it from the tutorial.

2. **Display live video**

   Create a file vidDisplay.cpp. Your main function should open a video channel, create a window, and then loop, capturing a new frame and displaying it each time through the loop. If you are using an IDE, this will be a separate project from task 1. Have your program quit if the user types 'q'. Add the ability to save an image to a file if the user types 's'. The remaining tasks will involve adding capabilities to this program that are activated by different keystrokes. The OpenCV tutorials provide more information about how to capture live video from a camera. The following is skeleton code for capturing live from a camera.

```
int main(int argc, char *argv[]) {
        cv::VideoCapture *capdev;

        // open the video device
        capdev = new cv::VideoCapture(0);
        if( !capdev->isOpened() ) {
                printf("Unable to open video device\n");
                return(-1);
        }

        // get some properties of the image
        cv::Size refS( (int) capdev->get(cv::CAP_PROP_FRAME_WIDTH ),
                       (int) capdev->get(cv::CAP_PROP_FRAME_HEIGHT));
        printf("Expected size: %d %d\n", refS.width, refS.height);

        cv::namedWindow("Video", 1); // identifies a window
        cv::Mat frame;

        for(;;) {
                *capdev >> frame; // get a new frame from the camera, treat as a stream
                if( frame.empty() ) {
                  printf("frame is empty\n");
                  break;
                }
                cv::imshow("Video", frame);

                // see if there is a waiting keystroke
                char key = cv::waitKey(10);
                if( key == 'q') {
                    break;
                }
        }

        delete capdev;
        return(0);
}
```

3. **Display greyscale live video**

   Update your vidDisplay.cpp so that if the user types 'g' it displays a greyscale version of the video stream instead of color. That means your program must remember the last keypress and modify each image according to the last selection. For this task use the openCV cvtColor function to do the conversion. Look up in the OpenCV documentation how each color channel is weighted in the conversion and explain it in your report. Give yourself some practice with search and reading the documentation.

   Required Image 1: show the original and cvtColor version of the greyscale image in your report.

4. **Display alternative greyscale live video**

   Update your vidDisplay.cpp so that if the user types 'h' it displays an alternative greyscale version of the image.

   Before implementing the greyscale transformation, create a new file filter.cpp. Please put all of your image manipulation functions in this file. If you are using a makefile or cmakefile, you will need to add filter.o to the list of files for the program rule. For example, the following makefile rule generates the executable vid and compiles the files vidDisplay.cpp and filters.cpp to generate the executable.

```
vid: vidDisplay.o filters.o
        $(CC) $^ -o $(BINDIR)/$@ $(LDFLAGS) $(LDLIBS)
```

   Pick a different method of generating greyscale video than used in the prior task and write the function to implement it yourself. Be creative about this so that your greyscale stream looks different than the OpenCV greyscale stream. Each color channel should be identical in your greyscale stream, but you can do any transformation you want to create it (e.g. subtract the red channel from 255 and copy the value to all three color channels).

   Since you will need to access individual pixels look at the openCV documentation for more information and examples on accessing rows and pixels of an image stored in the cv::Mat format. Encapsulate this task in a function in the filter.cpp file that takes in references to two cv::Mat objects (src and dst) and assigns the greyscale version of src to dst. The function should return 0 on success.

```
int greyscale( cv::Mat &src, cv::Mat &dst );
```

   Required image 2: show your customized greyscale image in your report, explain how you decided to generate your greyscale version, and highlight the

differences with the default greyscale image.

5. **Implement a Sepia tone filter**

   A sepia tone filter makes an image look like it was taken by an antique camera. Each modified R, G, B value is a function of all three color channels, according to the matrix below. To be specific, the new blue value should be 0.272*R + 0.534*G + 0.131*B, where R, G, and B are the original color values. Be sure to avoid using a modified color value to compute the other color values. In other words, once you have calculated the new red value, you still need to use the old red value to calculate the new green and blue values. You will also want to check the new values to ensure they are not greater than 255 before you assign them to the output image. The coefficients for the green and red channels sum to more than 1.

   ```
   0.272, 0.534, 0.131    // Blue coefficients for R, G, B  (pay attention to the channel order)
   0.349, 0.686, 0.168    // Green coefficients
   0.393, 0.769, 0.189    // Red coefficients
   ```

   A nice extension is to add vignetting (the image getting darker towards the edges) to this filter.

   Required image 3: show your sepia tone filter in your report and explain how you ensured using the original RGB values in the computation.

6. **Implement a 5x5 blur filter**

   A. Write a naive implementation of a 5x5 blur filter from scratch (e.g. use the `at` method to access pixels and implement the filter using a single nested loop). It should blur each color channel separately. Use the integer approximation of a Gaussian [1 2 4 2 1; 2 4 8 4 2; 4 8 16 8 4; 2 4 8 4 2; 1 2 4 2 1]. The function should not modify the src image. The return image should be the same size as the input image, but you don't have to modify the outer two rows and columns. For example, you can copy the src input image to the dst image and then overwrite the dst image except for the first two and last two rows and columns. Use the following signature for the function.

   ```
   int blur5x5_1( cv::Mat &src, cv::Mat &dst );
   ```

   Time your first implementation using this function on the included image . **Include the timing information in your report.**

   B. Write a second implementation of a 5x5 blur filter from scratch, but try to make it faster. For example, try using separable 1x5 filters ([1 2 4 2 1] vertical and horizontal) and avoid using the `at` method. As with the prior task, the input should be a color image (type vec3b) and the output should also be a color image (type vec3b). Use the following signature for the function.

   ```
   int blur5x5_2( cv::Mat &src, cv::Mat &dst );
   ```

   The function should not modify the input image: src. As with the prior task, it does not need to accurately blur the outer two columns and rows, but they should receive non-zero values. Time this version of your blur function to see if it is faster. **Include your timing information in your report and explain what you changed to make the filter faster.**

   Update your vidDisplay.cpp so that if the user types 'b' it uses this function to generate a blurred version of the video stream (in color). You may not use OpenCV filter functions for this task.

   Required Image 4: show the original and the blurred image in your report along with the timing information.

7. **Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters**

   In filter.cpp, create two new functions, using the prototypes below. Each should implement a 3x3 Sobel filter, either horizontal (X) or vertical (Y). The X filter should be positive right and the Y filter should be positive up. Both the input and output images should be color images, but the output image needs to be of type 16SC3 (signed short) because the values can be in the range [-255, 255].

   ```
   int sobelX3x3( cv::Mat &src, cv::Mat &dst );
   int sobelY3x3( cv::Mat &src, cv::Mat &dst );
   ```

   The filters do not need to accurately calculate the outer rows or columns, but it's a nice touch. Write these functions by accessing pixels directly, not by using OpenCV filter functions. Test your functions by updating your vidDisplay so that the 'x' key shows the X Sobel (show the absolute value), and the 'y' key shows the Y Sobel. You will find the openCV function convertScaleAbs to be helpful for looking at the Sobel outputs, because the imShow function requires 3-channel unsigned char images. Keep your Sobel filter outputs and the image you use to visualize the Sobel outputs as separate variables.

   When debugging your Sobel output, the X Sobel should show vertical edges, and the Y Sobel should show horizontal edges.

8. **Implement a function that generates a gradient magnitude image from the X and Y Sobel images**

   In filter.cpp, implement a function that generates a gradient magnitude image using Euclidean distance for magnitude: I = sqrt(sx*sx + sy*sy). This should still be a color image. The two input images will be 3-channel signed short images, but the output should be a uchar color image suitable for display.

   ```
   int magnitude( cv::Mat &sx, cv::Mat &sy, cv::Mat &dst );
   ```

   Test your function by updating the vidDisplay so the 'm' key shows the (color) gradient magnitude image.

   Required Image 4: show the original and the gradient magnitude image in your report.

9. **Implement a function that blurs and quantizes a color image**

   In filter.cpp, write a function that takes in a color image, blurs the image, and then quantizes the image into a fixed number of levels as specified by a parameter. For example, if the level parameter is 10, then each color channel should be quantized into 10 values. You can do this by first figuring out the size of a bucket using b = 255/levels. Then you can take a color channel value x and execute xt = x / b, followed by the statement xf = xt * b. After executing this for each pixel and each color channel, the image will have only levels**3 possible color values.

   ```
   int blurQuantize( cv::Mat &src, cv::Mat &dst, int levels );
   ```

   A good default number of levels is 10. Test your function by updating your vidDisplay so that the 'l' key displays the result. Use the functions you have already written to implement this effect.

   Required Image 5: show the original and the blurred/quantized image in your report.

10. **Detect faces in an image**

    Download the following code files (showFaces.cpp, faceDetect.cpp, faceDetect.h, and a haarcascade XML file). This code implements a standalone program that locates faces and puts boxes around them. Link the faceDetect.cpp file to your video stream program and turn on the face detection if the user hits the 'f' key.

    Required Image 6: show a face being detected in your video stream.

11. **Implement three more effects on your video**

    Choose three effect of your choice to implement on the video stream. One effect can be a single-step pixel-wise modification (no need to look at a pixel's neighbors). One of the effects need to make use of an area effect (like a Sobel or blur filter). One of the effects needs to build on the face detector. If you have implemented a function you want to use, use your own code. For new functions, you may use OpenCV, but only if it is part of a multi-step process.

    Some ideas for filters include the following.

    - Blur the image outside of found faces.
    - Make an embossing effect (hint, use the Sobel X and Sobel Y signed outputs and take a dot product with a selected direction like (0.7071, 0.7071).
    - Make the face colorful, while the rest of the image is greyscale.
    - Put sparkles into the image where there are strong edges.
    - Put sparkles in a halo above the face.
    - Allow the user to adjust brightness or contrast.
    - Stretch or warp the image (challenging).
    - Change around the color palette or make the image a negative of itself.
    - Implement other types of filters, such as a median filter.
    - Pick a strong color to remain and set everything else to greyscale.
    - Cartoonization of the video stream, as in this method .

    Required Images 7-9: show the original and the modified images in your report. If you want to demonstrate using a video, post it on Google drive and put the URL in your readme.txt file. Explain the special effects you implemented.

URL in your readme.txt file. Explain the special effects you implemented.

Extensions

Projects are your opportunity to learn by doing. They are also an opportunity to explore and try out new things. Rather than list out every activity for a project, the defined part of each project will be about 85% of the assignment. You are free to stop there and hand in your work. If you do all of the required tasks and do them well, you will earn a B+.

To earn a higher grade, you can undertake one or more extensions. The difficulty and quality of the extension or extensions will determine your final grade for the assignment. Extensions are your opportunity to customize the assignment by doing something of interest to you. Each week we will suggest some things to try, but you are free to choose your own.

A common question is "how much is an extension worth?" The answer to that question depends on the extension, how well it is done, and how far you pushed it. As a teacher, I prefer to see one significant extension, done well, where you explore a topic in some depth. But doing 2-3 simpler extensions is also fine, if that's what you want to do. Choose extensions that you find interesting and challenging. But remember, extensions don't need to be a big deal, you've already done most of the work for the assignment.

The following are a few suggestions on things you can do as extensions to this assignment. You are free to choose other extensions, and I am happy to discuss ideas with you.

- Implement additional filters or functions (from scratch).
- Implement additional effects. You may use combinations of OpenCV filters and operators to implement these.
- Implement your effects for still images and enable the user to save the modified images.
- Let the user save short video sequences with the special effects.
- Let the user add captions to images or video sequences when they are saved (i.e. create a meme generator).
- Explore other capabilities of OpenCV. Focus on things that enhance your understanding of the structure of images and how we can manipulate them.
- Build a more informative or sophisticated GUI using the OpenCV functions and callbacks.

Report

When you are done with your project, write a short report that demonstrates the functionality of each task. You can write your report in the application of your choice, but you need to submit it **as a pdf** along with your code. Your report should have the following structure. Please **do not** include code in your report. The reflection and acknowledgement sections are required for full credit.

1. A short description of the overall project in your own words. (200 words or less)
2. Any required images or text along with a short description of the meaning of each image or diagram.
3. A description of and example images for any extensions.
4. A short reflection of what you learned.
5. Acknowledgement of any materials or people you consulted for the assignment.

Submission

We will be using Gradescope for project code submission. You should receive an invitation to enroll in Gradescope using your Northeastern email address. You will be submitting your code, report, and readme file to Gradescope.

When you are ready to submit, upload your code, report, and readme file. The readme file should contain the following information.

- Your name and the names of any group members.
- Links/URLs to any videos you created and want to submit as part of your report.
- What operating system and IDE you used to run and compile your code.
- Whether you are using any time travel days, and how many.

For project 1, you need to submit at least the following files: imgDisplay.cpp, vidDisplay.cpp, filters.cpp, filters.h, and the readme. Note, if you find any errors or need to update your code, you can resubmit as many times as you wish up until the deadline.

When you are working with a partner or a group, only one of you should submit to Gradescope. When you submit, you can add your partner's name to the submission. Please do so.

I have turned off the capability to link github projects to Gradescope because it makes grading much more difficult. Please manually submit the relevant code files, readme, and pdf report. Please do not just drag and drop and entire folder, and this often contains hidden files that clutter the grading screen. In general, we will not be trying to run your code while grading, though we will be looking at it and comparing it with the images in your report.

As noted in the syllabus, projects submitted by the deadline can receive full credit for the base project and extensions. (max 30/30). Projects submitted up to a week after the deadline can receive full credit for the base project, but not extensions (max 26/30). You also have eight time travel days you can use during the semester to adjust any deadline **except the final project**, using up to three days on any one assignment (no fractional days). If you want to use your time travel days, specify that in the readme file. If you need to make use of the "stuff happens" clause of the syllabus, contact the instructor as soon as possible to make alternative arrangements.

Receiving grades and feedback

After your project has been graded, you will be able to find your grade and feedback on Gradescope. Pay attention to the feedback, because it will probably help you do better on your next assignment.