Project 0

Prof. Adín Ramírez Rivera
`adin@ic.unicamp.br`

1 Description

This project is an introduction to the course and to the tools you will be using on the other projects. **This project has no grade**, and it is intended for practice purposes. However, doing this project is highly encouraged for understanding the tools, and dusting off your programming skills.

During this project, you will practice how to use docker to build and test your code, how to manipulate images and other values, and to produce some output. As well, you will submit your code and report, and will get feedback promptly in case something is wrong.

To develop this and future projects you must use OpenCV 4.0.1 with solutions on Python 3.7 (or higher) or C++ (your code must run on a determined docker image: `adnrv/opencv`, check the notes § 4 for more information).

2 Submission

You need to declare a team (check the information on the course's webpage) so you (and your teammates) can be added into the Gitlab working group of the course. You will work on such a group for the rest of the semester. You need to create a project named `project-0` within your group to work on this project.

Your submission must have the following subfolders:

- `input`: a directory containing the input assets (images, videos or other data) supplied with the project. Store the images only if they are asked on the project. Otherwise, set up your `Makefile` (or your `.gitlab-ci.yml`) to automatically download them from a public server or repository.

- `output`: a directory where your application should produce all the generated files (otherwise stated in the problem). This folder and all its contents must be added to the `artifacts` path on the `.gitlab-ci.yml` setup.

  In case of a problem that asks for producing images automatically, use a convention for naming the images that easily identifies them. For example, you can use `[io]-<question #>-<part>-<counter>.png`, where `i` will be used for input and `o` for output images, `<part>` may be omitted if the question does not have one, and that `<counter>` must be auto-incremented starting from 0.

- `src`: a directory containing all your source code. You only need to submit files that are not derived from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.

- `Makefile`: a makefile that executes your code through the docker image. An image is already built and available for use (`adnrv/opencv` at the docker hub registry). The code will be executed through a standard call to `make`, so other dependencies must be provided by you under that constraints. Moreover, note that your code **must** run on the Gitlab executor (pipeline) on the server itself. Hence, you need to use the `.gitlab-ci.yml` configuration. You can base your work on the demo that already provides an example of such functionality.

- `report`: a directory containing the source files that produce your report. Your report **must** be written using LaTeX (and friends) and compiled within the pipeline of the repository. Consequently, **the PDF must not be committed**. You can use the images from `adnrv/texlive` to build your PDFs. **Only the PDF of the report** must be added to the `artifacts` path on the `.gitlab-ci.yml` setup, and not other intermediary files of your report (e.g., images, log files, auxiliary files).

Your report must show all your work for the given project, including images (labeled appropriately, that is, in accordance with the convention given) and other outputs needed to explain and convey your work. When needed include explanations to the questions given in the project.

The last commit that triggered the build must be before the deadline of the submission. Do not worry about the time executing the pipeline. However, **you must ensure that your code works as no attempt will be made to patch or run broken code**.

3 Questions

For the following questions you will be using the `Mat` class and basic matrix operators. Try to keep it simple to understand the basics of OpenCV.

1. **Input images.** Find an interesting image to use (`img`). It should be color, rectangular in shape (that is not the same length and width).

   Make sure that either dimension (height or width) does not exceed 512 pixels.

   Store the image in the `input` folder using the name convention. (Note that it should be named `i-1-0.png`. The names on the questions are for referencing them in the project's text.)

2. **Color planes**

   (a) Swap the red and blue channels of the input image, `img`. Store it in the `output` folder. (Note that this should be named `o-2-a-0.png` according to the convention's example, and in contrast to the following items it is a color image.)

   (b) Create a monochrome image (`img-green`) by selecting the green channel of the input image. Store it in the `output` folder. (Note that this should be named `o-2-b-0.png`, and so on.)

   (c) Create a monochrome image (`img-red`) by selecting the red channel of the first input image. Store it in the `output` folder.

   (d) Which image looks more like what you would expect a monochrome image to look like? Would you expect a computer vision algorithm to work on one better than the other? Why? (Note that since this is a question you should answer it in the report.)

3. **Replacements of pixels.** Let's call the monochrome image from your answer to the last item of the previous question A, and the other one B. Take the center square region of 100×100 pixels of A and insert it into the center of B. Store the generated image in the `output` folder.

   Replace the respective channel of B into the original image. Store it in the `output folder`.

   What do you see? Explain.

4. **Arithmetic and geometric operations**

   (a) What is the min and max of the values of `img-green`? What is the mean? What is the standard deviation? How do you compute them? Provide code snippets in your report for these questions.

   (b) Normalize the values of `img-green`. That is, subtract the mean from all pixels, then divide them by the standard deviation. Then multiply by 10 (if you are using an image representation from 0 to 255) or by 0.05 (if you are using a range from 0 to 1). Now add the mean back to each pixel. Store the resulting image in the `output` folder.

   How does the image look like? Do you have an idea of what happened?

   (c) Shift `img-green` to the left by 2 pixels. Store the resulting image in the `output` folder.

   Subtract the shifted version to the original, and save the difference image. Make sure that all the values you are saving are legal so you can see all the differences. What do the negative values mean? What do you see in the result?

5. **Noise**

(a) Take the original colored image (`img`) and add Gaussian noise to the pixels in the green channel. Increase the sigma until the noise is visible in the image. Store the image into the `output` folder.

What is the value of sigma you used? How did the noise in the resulting images behave? What did you observe?

(b) Add the noise using the same sigma to the blue channel instead. Store the output.

(c) Which image looks better? Why?

4 Notes

- All the submissions must be self-contained and must be executable in a Linux environment. Specifically, your code must execute in the docker image `adnrv/opencv`, available at docker hub (`https://hub.docker.com/r/adnrv/opencv/`).
- Your report must be written in English.
- While there are several images available for LaTeX, I recommend you to use one of the docker images `adnrv/texlive`, available at docker hub (`https://hub.docker.com/r/adnrv/texlive/`). In particular the `basic` and `full` images are vanilla versions of `texlive`. However, their sizes varies considerably (consequently, the execution times on the pipelines). If you need particular packages it may be easier to take a `basic` image and just install the packages you need using `tlmgr`.
- It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing the pipeline within the Gitlab repository.
- You must program in Python 3.7 (or higher) or in C/C++ with `gcc` version 7.3.0, using OpenCV 4.0.1. All available within the image. If you need to install other packages you must do so within your `Makefile` as automatic prerequisites, or through the `.gitlab-ci.yml` pipeline.
- In order to test your code locally, I recommend a two-fold approach. Execute the code on the docker image by binding your working directory and the docker using volumes. Once your code is running, test the pipelines with a local Gitlab runner. That way, you won't be waiting for the runners to compile, run, and then report errors back to you.