Group 25: Real Time Camera Validation
Reviewer: Donald (Max) Harkins - Review Group 4

- Build:
  Due to hardware constraints I wasn't able to build the system. I would recommend trying to separate your software components from the hardware as much as possible, to make the application more maintainable and easier to test and show. I think one way to make the software more independent of the raspberry pi hardware would be containerization. Creating Docker containers for your software would allow for the software to be run more flexibly on different hardware, even allowing for all the components to be run together on a single computer, instead of using 2 raspberry pis. Additionally, Docker has the benefit of greater ease of installation by eliminating the need to find and install dependencies. If your team decides not to use Docker, then I would recommend creating a Makefile for the comparison and corruption units.

I created a VM image with all of the applications dependencies installed. It can be run on any visualization software. Details are in the "Unit Testing" section of the VM. I decided to not use Docker containers for this test environment because the vast majority of Linux based images do not have Xservers configured thus we cannot visually demonstrate the system. Additionally Docker requires a Hypervisor on Windows (only available on Windows pro). Also you cannot run Linux containers on Windows machines without proper configuration. A VM allows the user to view processed images from OpenCV in the OS GUI, and can be run in open source Virtual Box on any host.

- Legibility:
  Code legibility was good from a structure standpoint -- functions were not unreasonably long, function/class definitions and declarations were split up appropriately into .cpp and .h files, and descriptive variable names were chosen. One aspect that really hurt legibility was lack of comments. I would recommend adding comments above each class and function definition describing the purpose of the code, the inputs, and the outputs. Additionally, there are some tricky sections of code, for instance in compare.cpp, where additional comments beyond top-level descriptions would be helpful.

I added extensive comments to the code base which explain what code blocks/functions do

- Implementation:
  The implementation looked good to me. Your code looks efficient and well-written. I think it was good to use inheritance to structure your filter classes for distortion. However, one change that I would recommend making would be to allow multiple class inheritance, and implement a way that multiple types of distortion could be applied at once. This would help demonstrate the robustness of your system.

This is outside the scope of the project. Our never made it a requirement to introduce multiple types of distortion simultaneously.

- Maintainability:
  Unit tests for the code were lacking. I would recommend writing unit tests for all the classes created. This would help maintainability immensely. Additionally, to test the integration of all the components of the system, I would recommend creating a script to automate video corruption, to be used in place of the GUI, with a pre-recorded video. Then, the system could be run in "test mode" without user input, and compare detected corruption with expected values to demonstrate basic module integration. In addition to writing tests, increasing the legibility of the code would help in increasing the maintainability.

I created unit testing suites for both the comparison and corruption unit using the C++ unit testing framework Catch2. Unit tests can be easily run in the pre-configured Raspberry Pi VM image. Details are in the "Unit Testing" section of the README . Test suite performs integration testing similar to what is described above.

- Other:
  I thought your project was very interesting, and it is clear your team has spent time and effort planning out the function and purpose of the components. Since the project has many parts, I thought it was great you included an overview of the directory structure at the end of the README. I would encourage your team to add additional small descriptions (one or two sentences each) describing what the code in each subdirectory does and how it interacts with other modules (e.g. describe the purpose of the corruption unit in a sentence or two).

No revision necessary.

Revised By: Kamron Ebrahimi