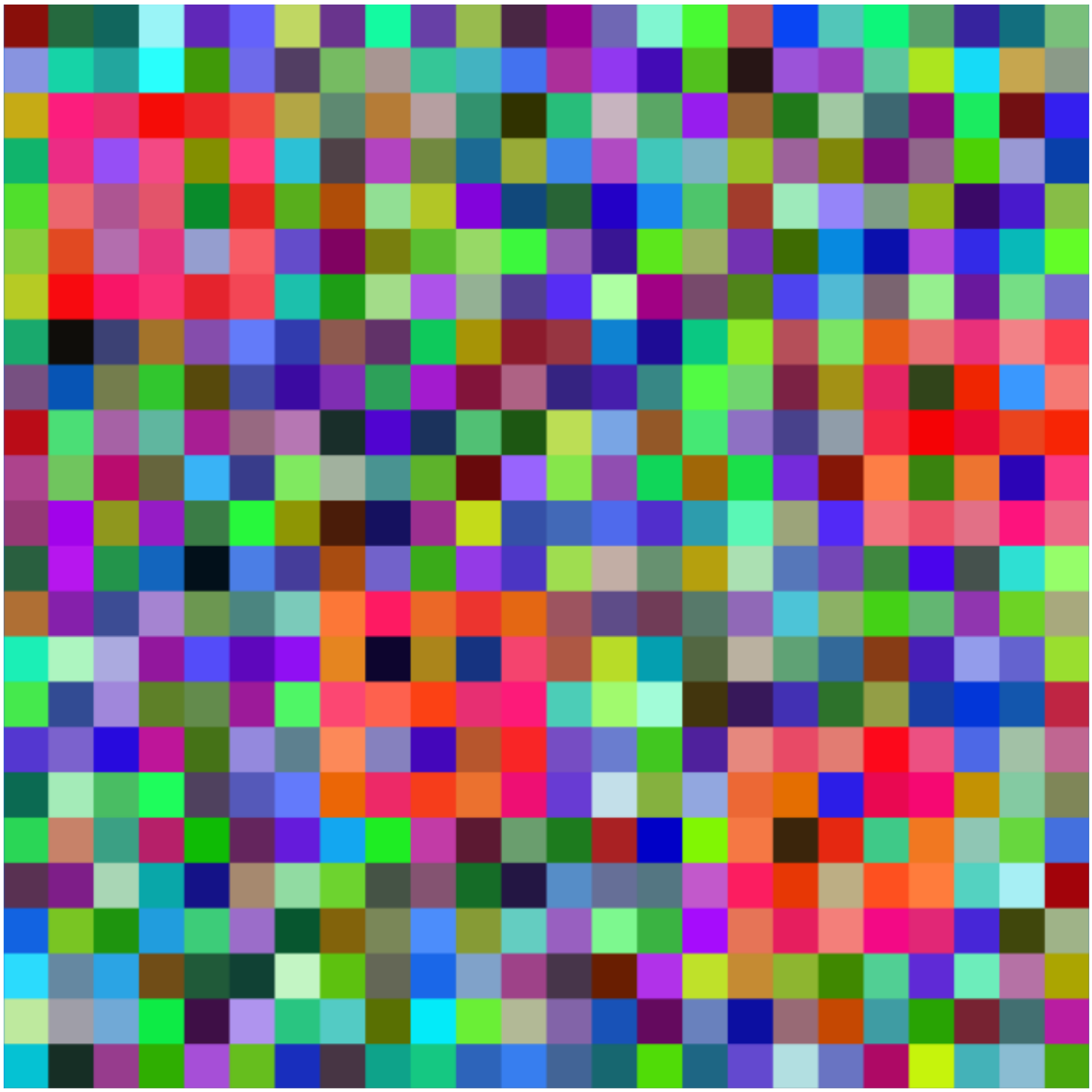


Red Eye Filtering

User guide & design documentation

Nikola Totev

Table of contents

0. Who is this documentation for?

1. Problem introduction

2. Problem solving process

3. Filtering Algorithm

4. Visualization

5. How to use

6. TL;DR Setup and Running

0. Who is this documentation for?

This documentation is for people who want to quickly grasp the essence of the project, how to set it up, run it and the details related to the implementation.

All important aspects of the program are showcased with diagrams and flowcharts. Easy navigation in the PDF is provided through links, they will take you where you want to go.

Have fun exploring!

1. Problem Introduction

We have a set of images like fig. 1 and we want to remove the “red eyes” and make it like fig. 2

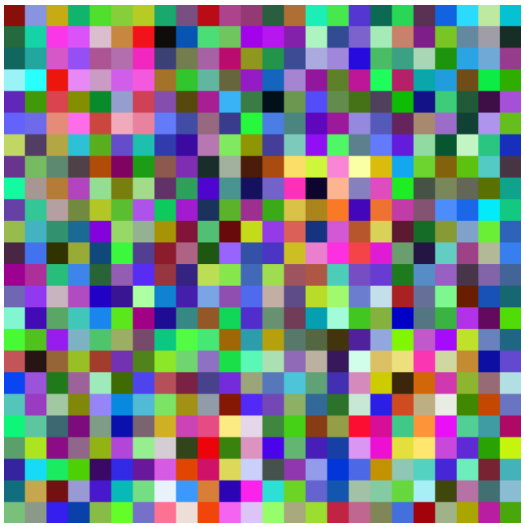


Fig. 1 - Before filtering

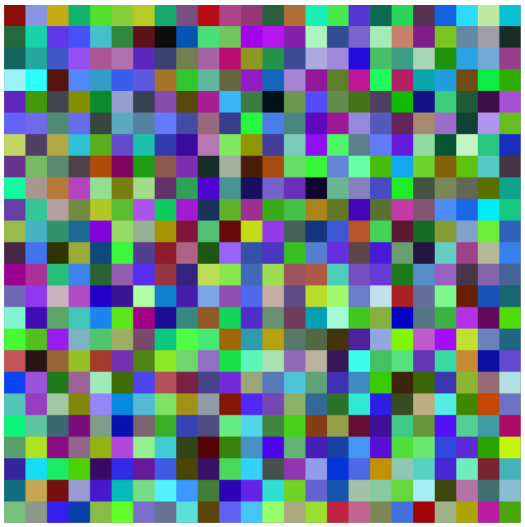


Fig. 2 - After filtering

You might have some trouble figuring out at first glance where the patterns are exactly. A “Super Red” version of Fig.1 might help:

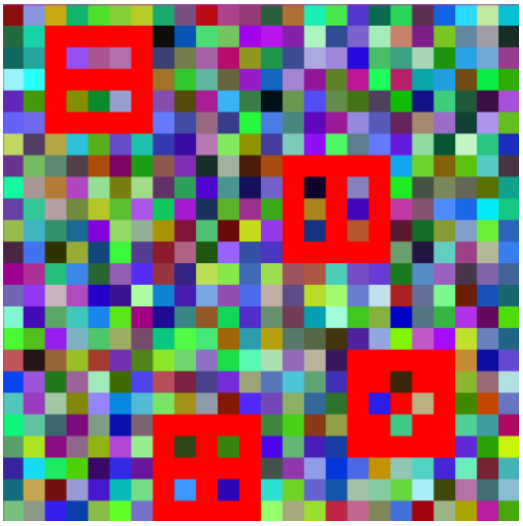


Fig. 3 - Super Red

We are given two sets of binary encoded images. They come in the format shown in Fig. 4.

A base skeleton is also provided, it has file parsing functions along with a validation function that returns if we have successfully applied filtering to the input.

The task is to create a class called “Solution” with a method “compute” that applies the filtering to the “eyes”.

```
50
640 360
2999283199 2099460095 1321233663 1871982079 125232230
848 480
10193663 81372415 2593160447 1269858815 191584767 211
640 360
1878000895 2486939903 447917567 946517247 541130495 6
426 240
1524981759 1062328575 964700159 1942819583 1428325887
640 360
2991788031 912652287 1643458815 2212463615 470179071
```

Fig. 4 - Image Format

Eye Patterns

In the base skeleton we are given a set of eye patterns. They are used in the process of identifying “eyes”.

A region of pixels is an eye if ***all*** non-null characters of the pattern can be mapped to pixels of an image that have a red value >= 200.

The eye patterns can be seen in Fig. 5:

```
constexpr EyePattern EYE_PATTERN_1 {
    "/---\\",
    "|   |",
    "|-o-|",
    "|   |",
    "\\---/"
};

constexpr EyePattern EYE_PATTERN_2 {
    "/---\\",
    "| | |",
    "| 0 |",
    "| | |",
    "\\---/"
};
```

```
constexpr EyePattern EYE_PATTERN_3 {
    "/---\\",
    "| | |",
    "|-q-|",
    "| | |",
    "\\---/"
};

constexpr EyePattern EYE_PATTERN_4 {
    "/---\\",
    "|\\ /|",
    "| w |",
    "|/ \\|",
    "\\---/"
};
```

Fig. 5 - Given patterns

A “cleaner” version of these patterns can be seen in Fig. 6:

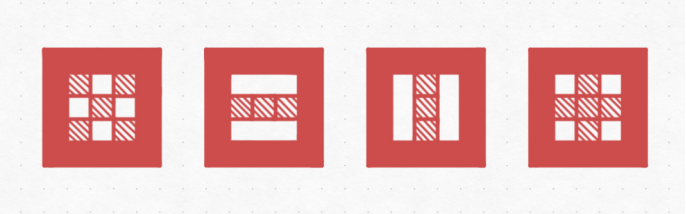


Fig. 6 - Cleaner patterns

2. Problem Solving Process

In this this section I want to provide my perspective of the problem solving process, the challenges I encountered and the thought process behind the decisions to solve them.

What is the input?

This was the first challenge, figuring out exactly what I was working with. Initially I thought I would be working with images of real eyes and I was confused by the provided patterns. To overcome this problem I decided to “export” the images loaded by the skeleton into a easier to parse .rgba format of my own.

The .rgba files I parsed with a C# program I wrote and I got the results in Fig. 1, 2 & 3. This allowed me to compare the input and the desired output.

Eye Patterns

Even though I had the input and target output I was still confused by the patterns and I started to compare the input and output. At this point I noticed that some pixel in the red-ish areas stay the same so I marked the ones that didn’t change. (Fig. 7)

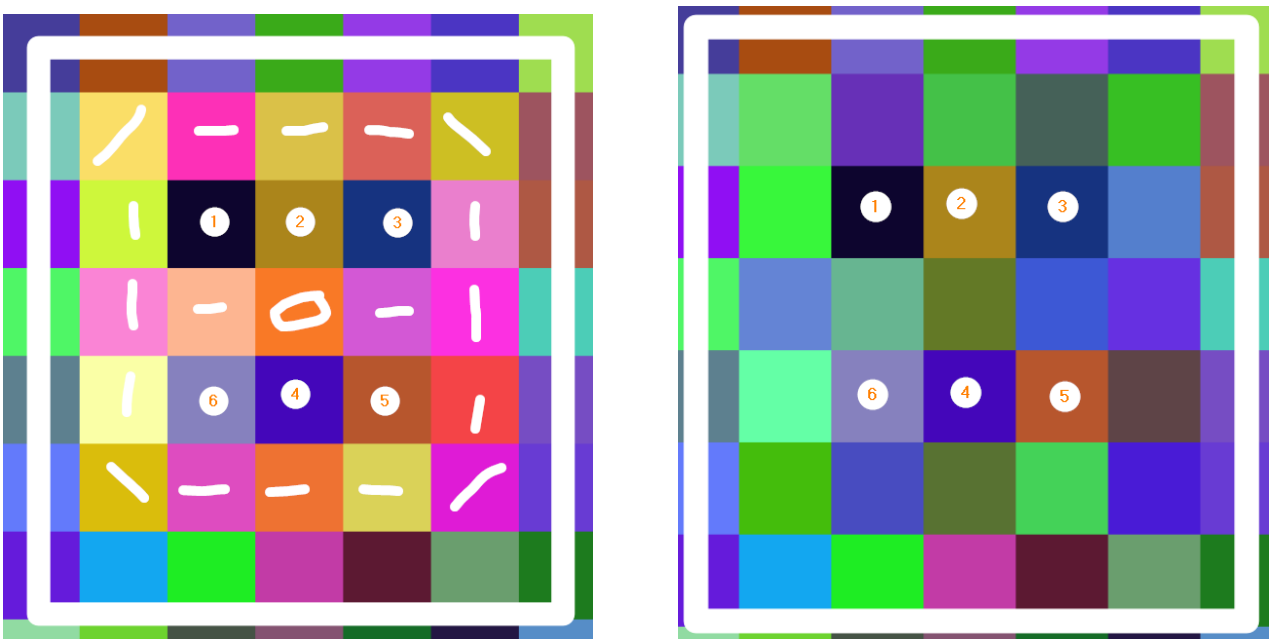


Fig. 7 - Comparing Data

With this labeling I started to see a pattern, but to make it even more clear, in my C# program, while writing the pixels I added a check that made any pixels that have a red value equal to or over 200 (the threshold set in the task) have a red value ot 255. This gave me an output like Fig. 3 which I called “Super Red” versions of the input.

This visualization was very valuable during debugging and result validation and in my opinion its the most important step in the process.

Project Structure

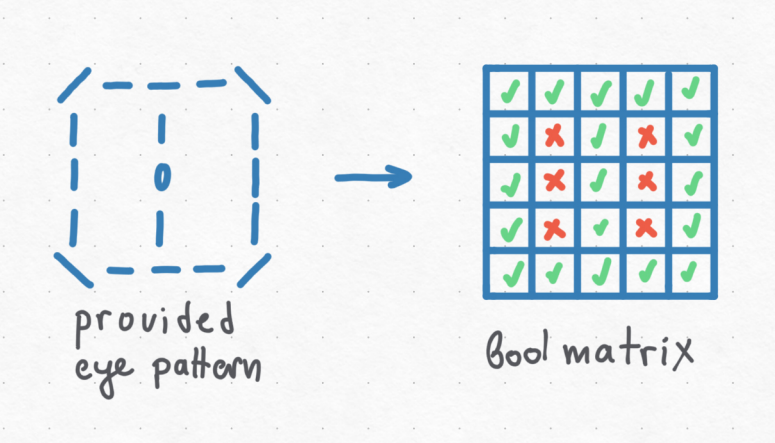
From the start my goal was to keep the code readable and organized. Instead of putting everything in the “compute” method I decided to break up the steps of the algorithm into functions that have a single purpose. I will describe the different functions in the section for the filtering algorithm.

The project is divided into the directories “cpp”, “documentation”, “visualization” for easy navigation.

3. Filtering Algorithm

A picture is worth a thousand words. I will show the sketches I created while figuring out the algorithm to explain it.

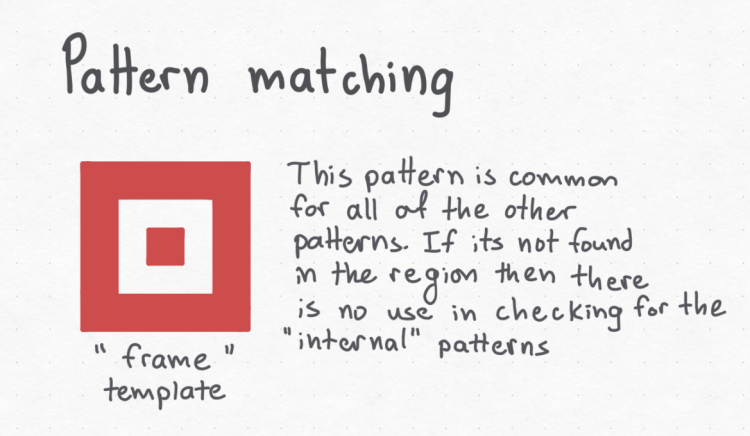
NOTE: I decided to use PackedImage because it allowed me to easily add a bool variable to the struct that helps me mark a pixel as processed.

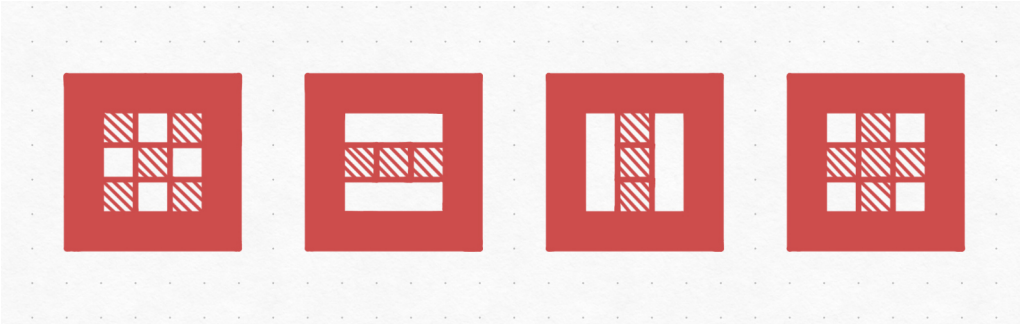


I started by manually translating the given patterns into bool matrices.

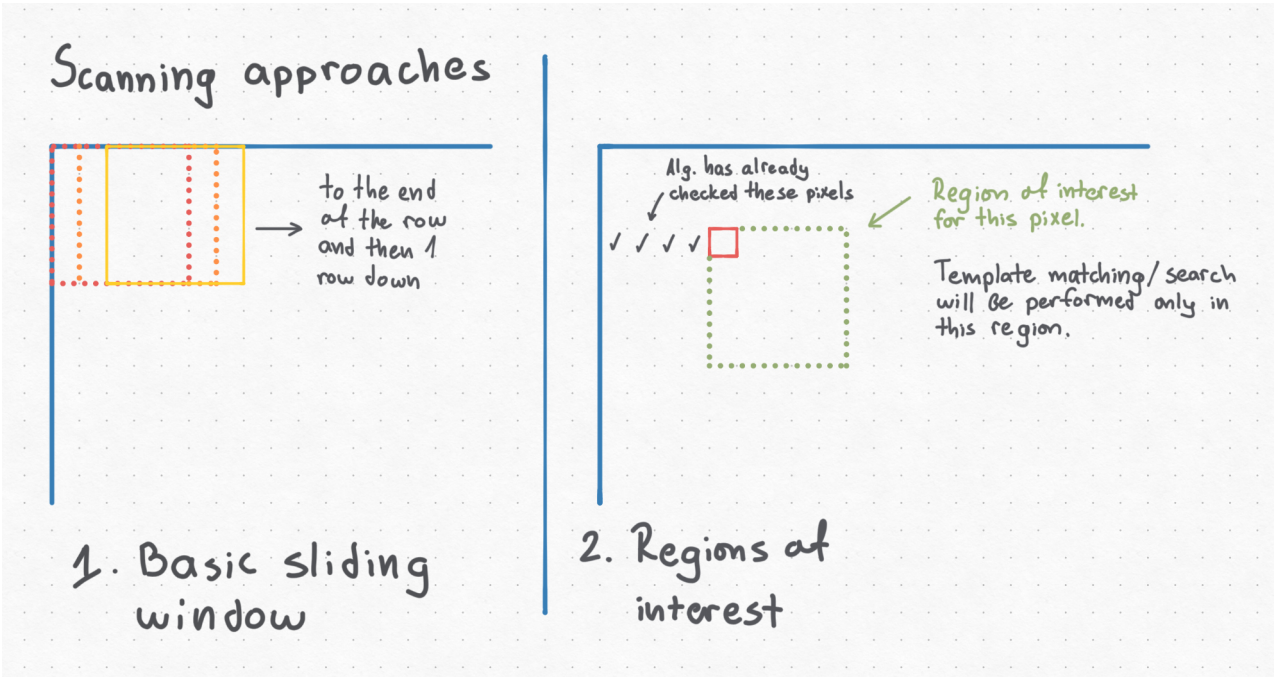
(In reality they are integers because 1s and 0s are easier to see as patterns while reading the code. The idea is still the same though)

I use these matrices as templates when scanning for eyes.



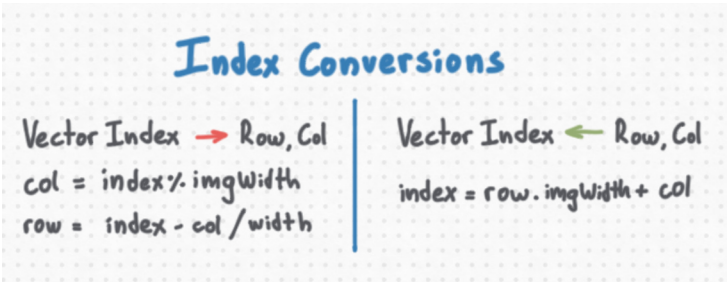


This frame template is rotation approach gives flexibility for future additions to internal templates and it saves time while scanning. In this case the time benefits are tiny, but in a real-world project its better to find parts of a template that remain the same no matter the orientation, this helps to improve the preformance of the template matching algorithm.



When it comes to image scanning the “brute-force” method is to just slide each template across the image and check at every position if an eye is found. This is alright for small images but for larger ones it becomes impractical.

For my solution I use a regions of interest approach. If I detect a pixel with R>= 200 I apply the template search for only a limited region around the pixel of interest. This saves on time by not checking pixels that don’t have at least one of the criteria for being an eye.

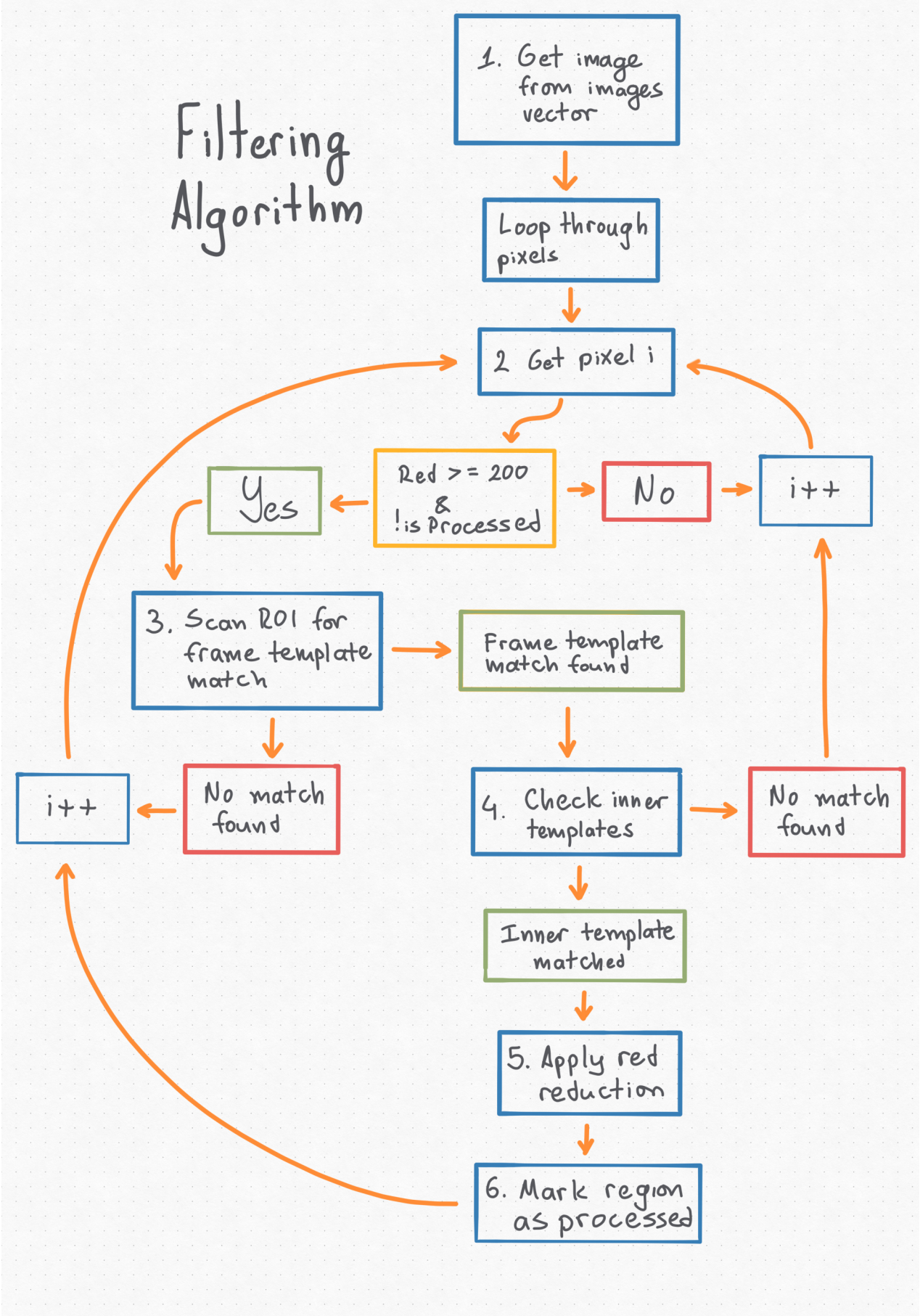


For easier traversal of the image, I created two utility methods that perform conversions for

VectorIndex > (Row,Col)

VectorIndex < (Row,Col)

Algorithm Diagram



4. Visualization

The visualization program is pretty simple, it only reads the contents of the .rgba files. I use this extention so that I can exclude them from the git repo. If I had used .txt the CMakeLists file would've gotten excluded.

The .rbga files have the following format:

Height=widthD
r g b a-r g b a- ...

5. How To Use

Red Filtering Program

I added some extra features to the skeleton to make it a bit easier to use. When starting the program you will be prompted to enter the location of the input.bin file and output.bin file.

The next prompt is if you want to save the images, if you choose "Y" you will be asked to provide an existing directory in which the before and after images will be saved.

Below in Fig. 8 & Fig. 9 you can see an example run of the program and the outputs you should expect.

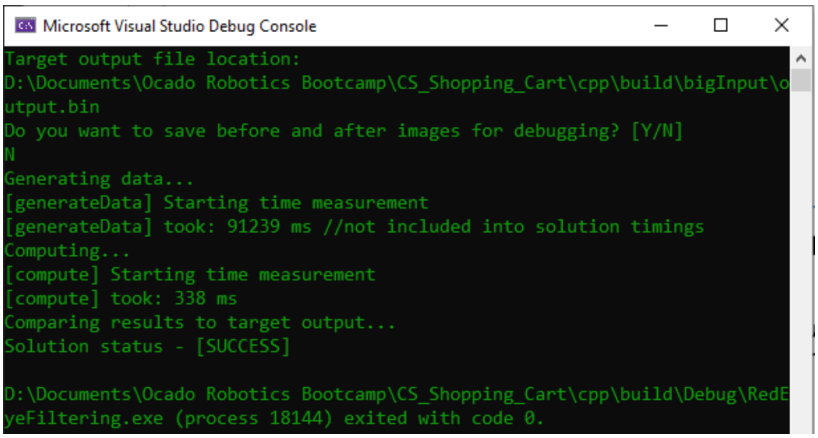


Fig. 8
No image saving

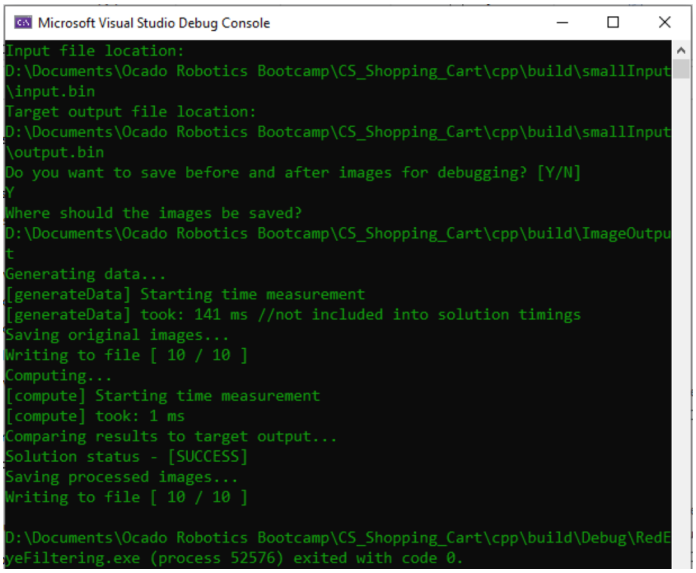


Fig. 9
With image saving

Visualization Program

The Visualization program is even simpler to use, just enter the image directory you entered into the red filtering program and all of the images will be converted.

6. TL;DR - Setup & How To Use

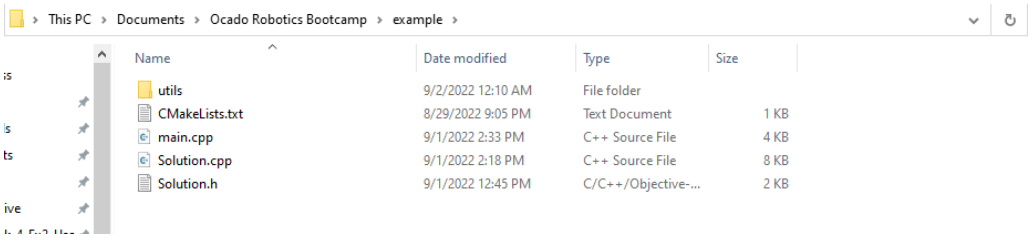
CMake

This project is created with Cmake. The CMakeLists.txt file can be found in the cpp folder of the repo. It has everything required to create a project you can use.

NOTE: The project in Github was generated and tested using Visual Studio 2019.

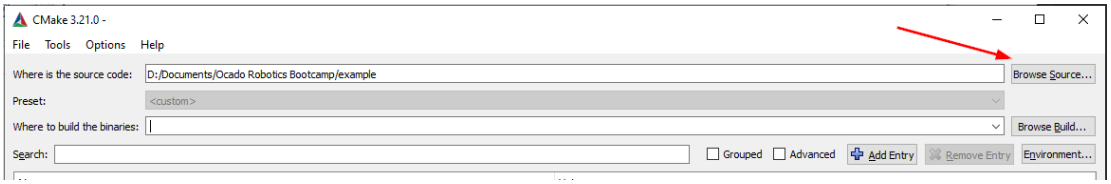
For the CLI Option use in the directory with the CmakeLists.txt file:
mkdir build then cmake -S . .\build

If you prefer the GUI below you will find steps on how to setup the project using the CMake GUI.

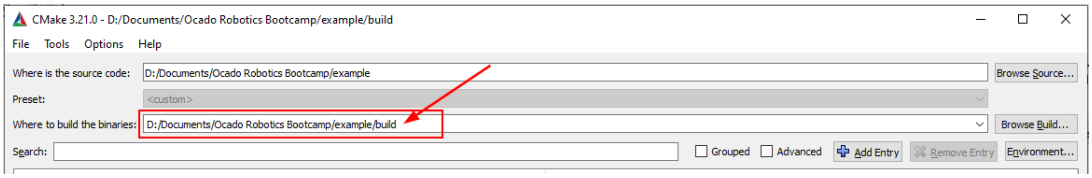


You should start with this directory structure. All of the solution code is here.
Notice that the CmakeLists.txt is in the “example” directory.

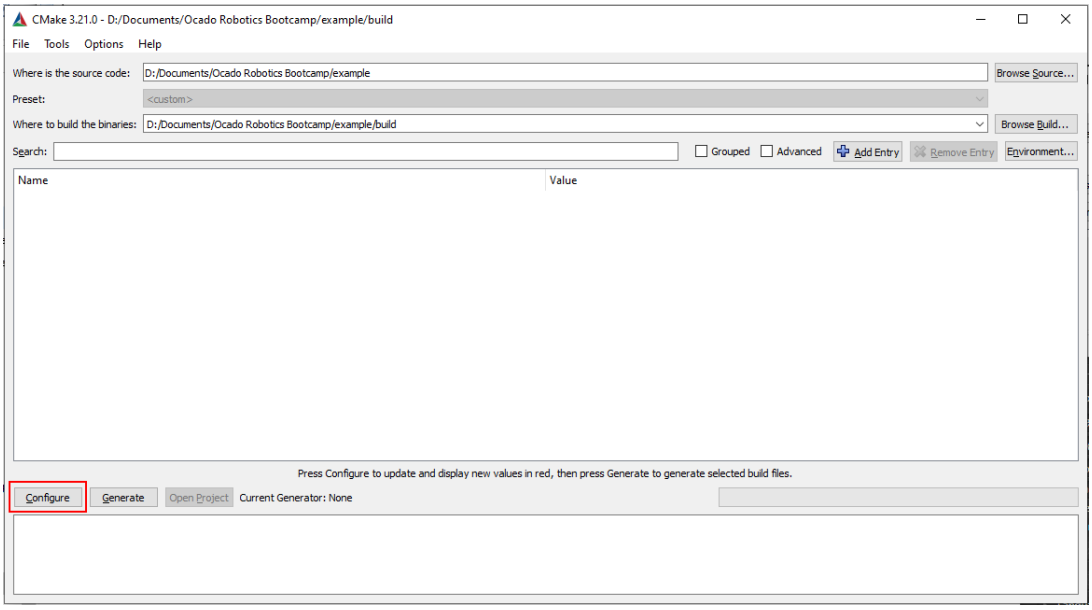
Open the CMake GUI tool.



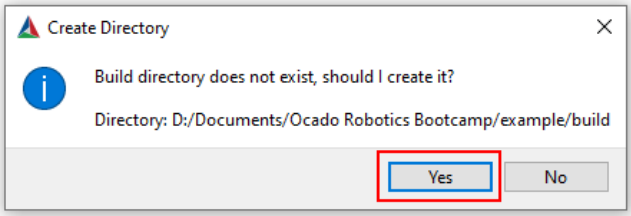
And select your equivalent of the “example” directory.



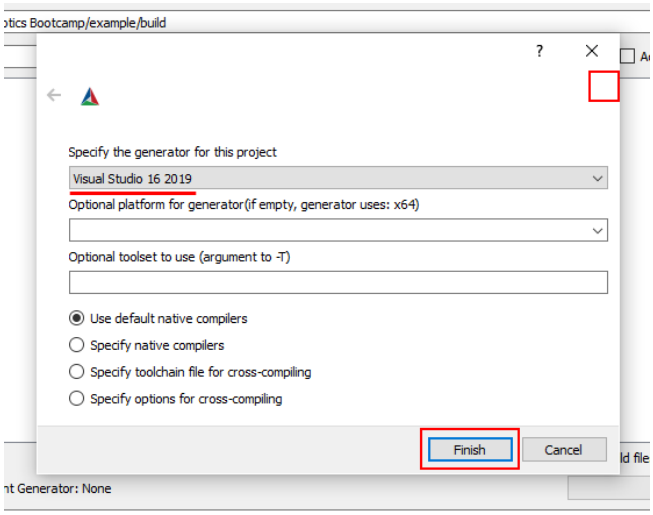
Then here, copy the same path, but add /build to it.



Click “Configure”

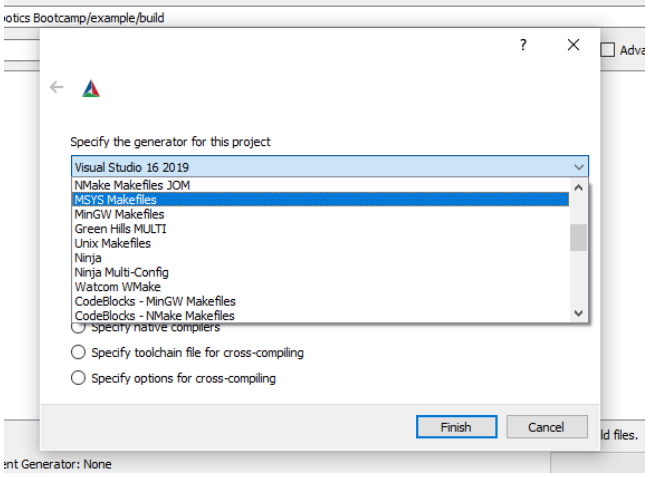


You might get this dialogue, click “Yes”

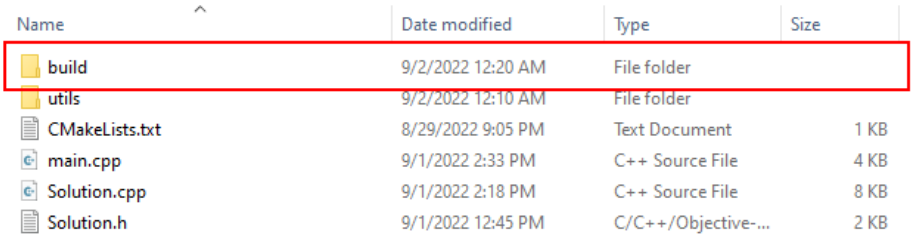


Then you will get this window. Here you will choose what kind of project will be generated. For this setup tutorial we want “Visual Studio 2019.

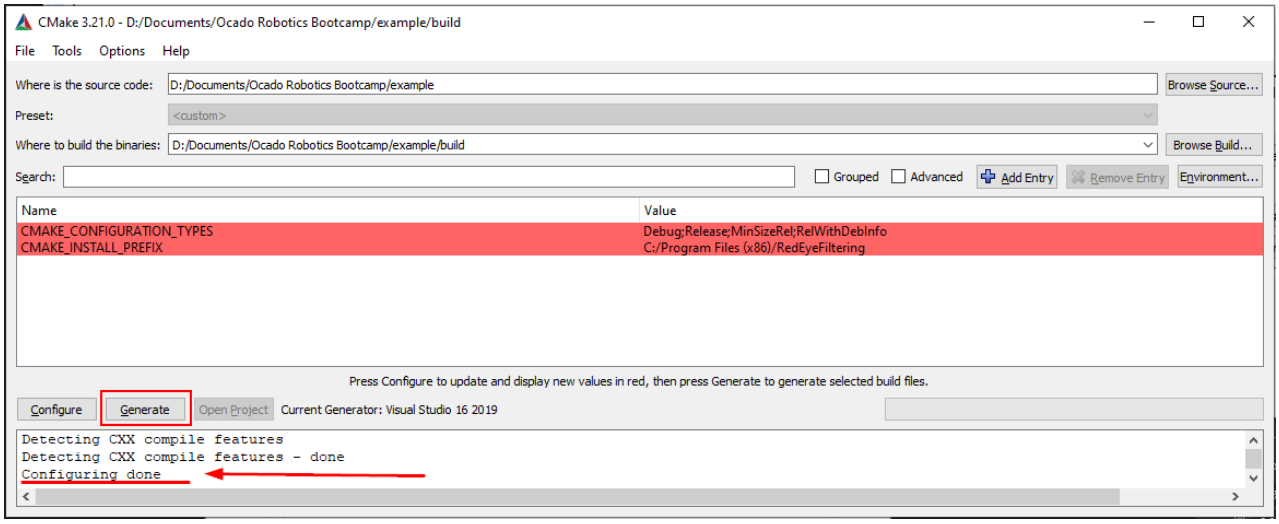
Make sure its selected and click “Finish”



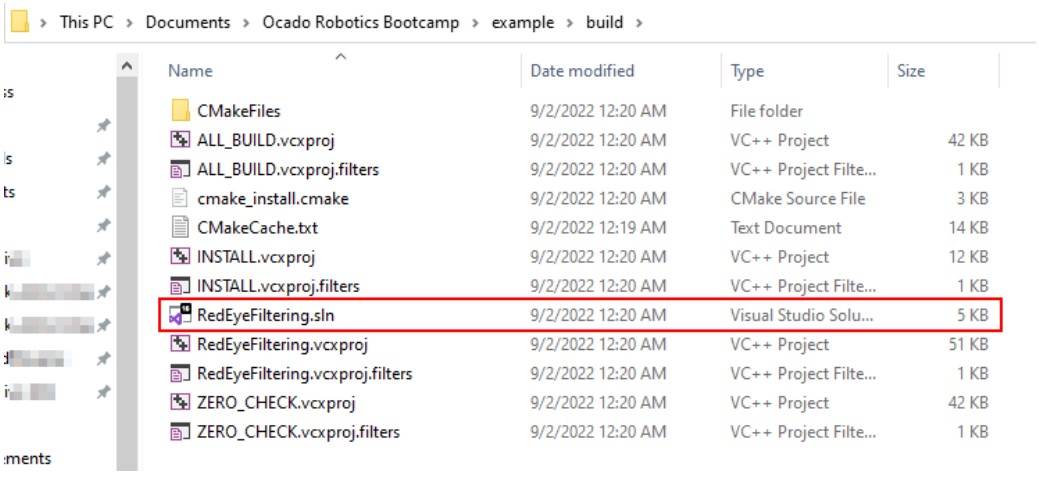
If you wish to use a different project type you can select it from the dropdown.



Your directory should look like this now.



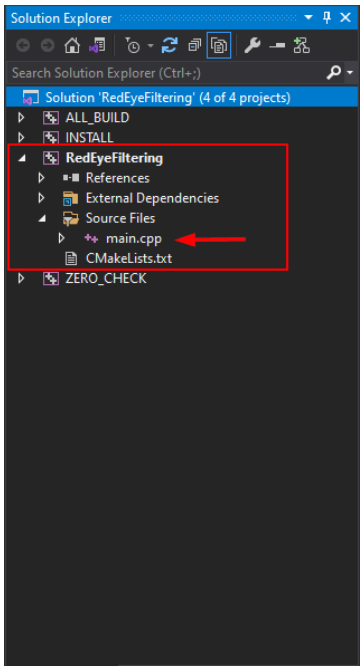
Go back to the CMake GUI and click “Generate”



Now in the build directory you should have a Visual Studio project.
(or the type of project you selected)

Click on the .sln file to open the Visual Studio project.

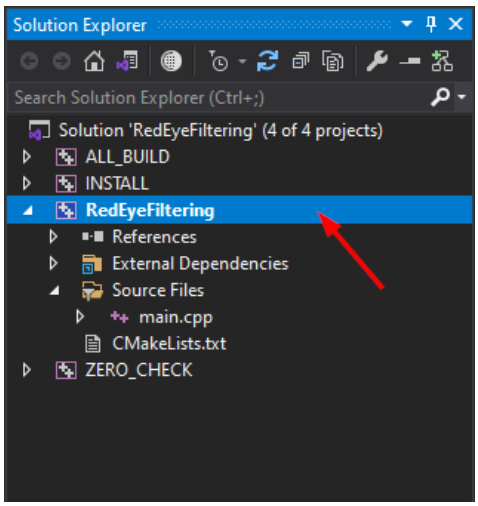
Visual Studio



Once open in Visual Studio, the project should look like this.

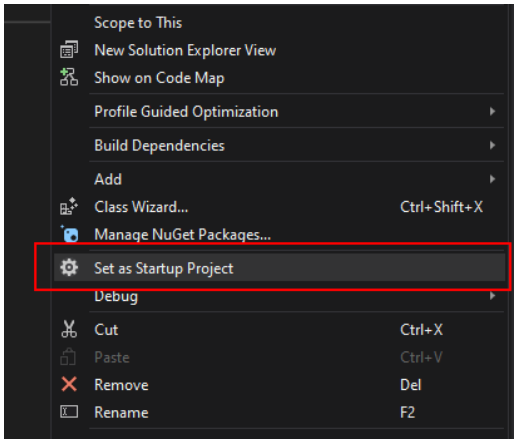
If you try to run main however you will encounter some issues.

Lets fix them now.

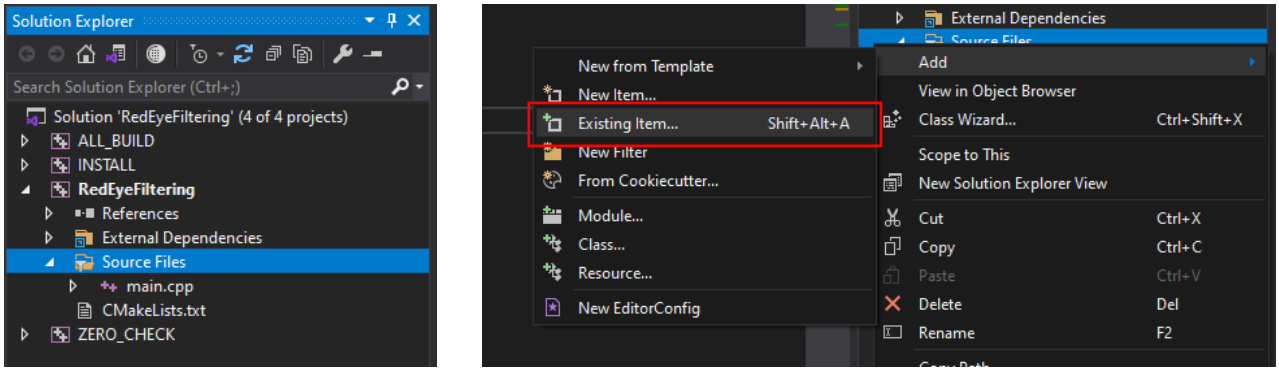


To make sure you are running the correct item, right-click

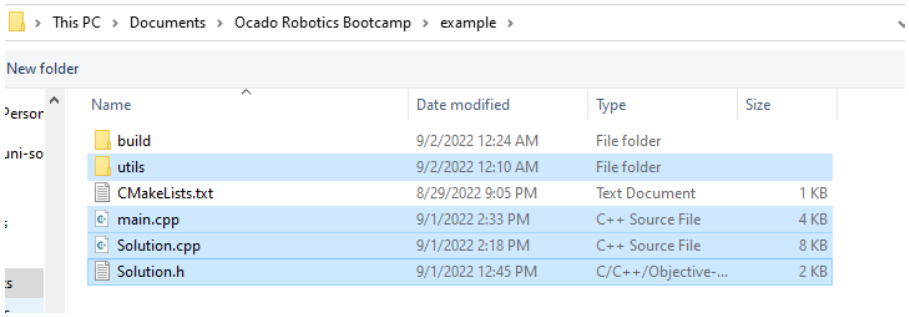
“RedEyeFiltering”



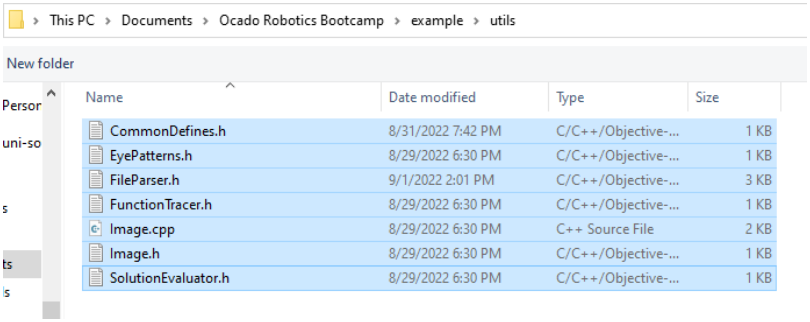
And click “Set as Startup Project



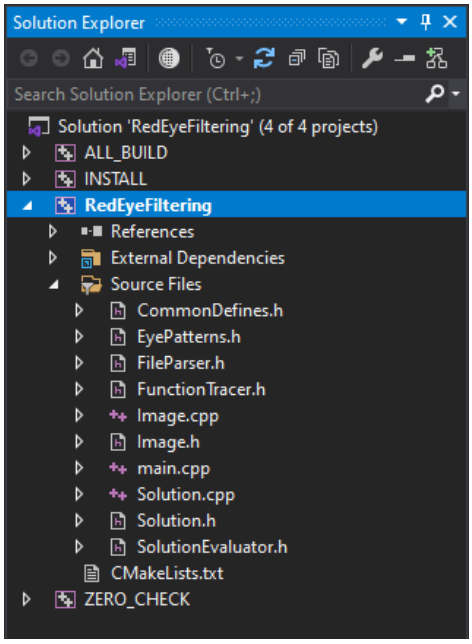
Then right-click on “Source Files”
and click on Add > Existing Item



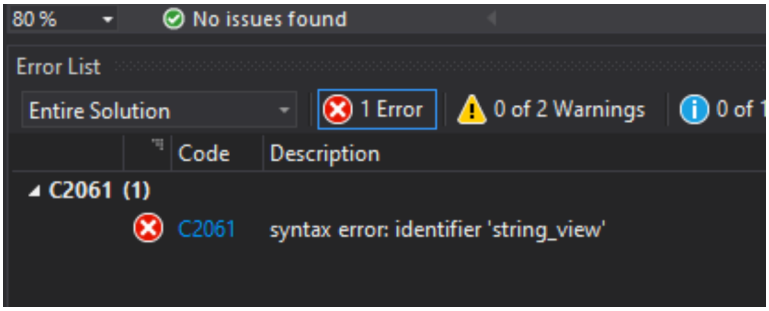
Add these items first



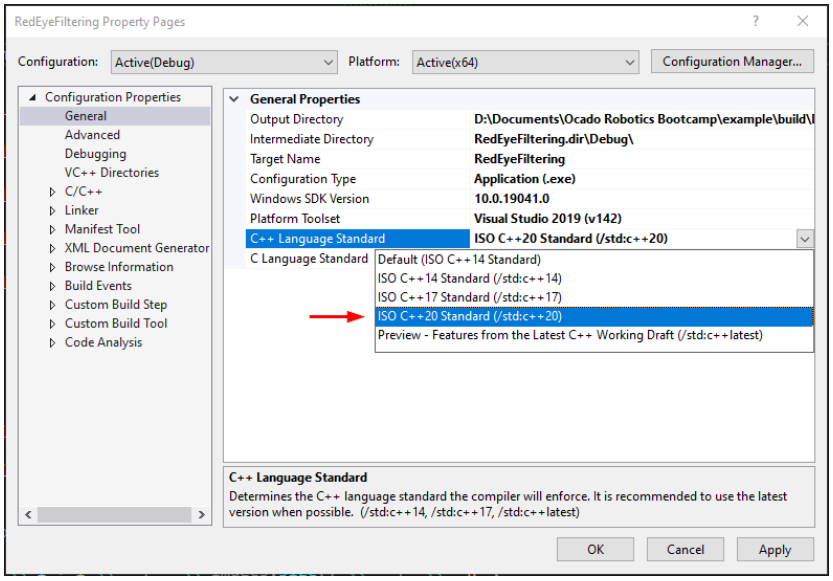
And then go into the utils directory
and all the files inside.



Your Solution Exploer should now
look like this.



You might also get this issue. If you do press Alt+Enter with the RedEyeFiltering module selected.



Make sure you are using C++20