COSC342: Mosaicing in OpenCV

Introduction

As discussed in lectures, OpenCV gives us all the tools we need for creating image mosaics. You can find out about OpenCV, including extensive online documentation at opencv.org.

The files you'll need for this lab are in

`/home/cshome/coursework/342/pickup/labs/lab04-Mosaicing/`

As usual, you'll need to make a copy of these files in your own space to work with them.

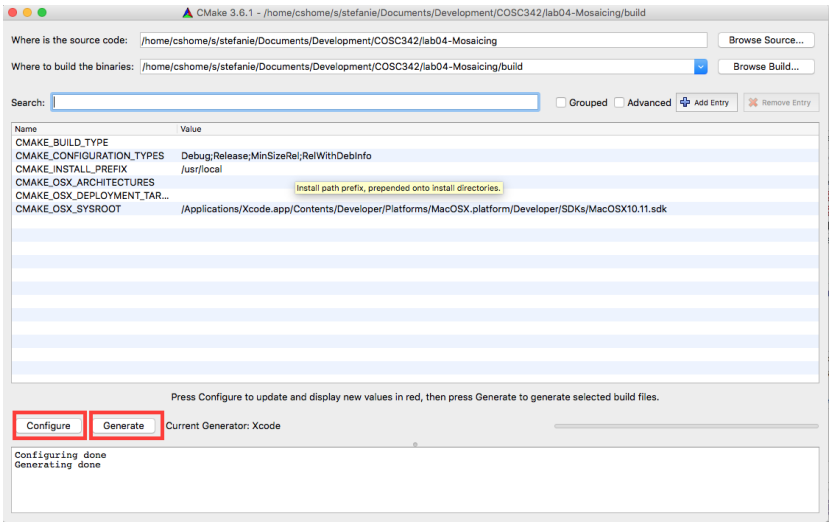
Similar to the last lab, we will use CMake to prepare our development environment. You can skip the following instructions if you feel confident with running CMake to create your project files (Skip to "Run the code"). Again there are two options supported in the lab. The two main options are to compile code with Makefiles from the command line or to use XCode.

To set up the project, run the program called `CMake`, which is in the Applications directory. If you can't find it press `command-space` and type "CMake" to search for it. Again, in CMake specify the path to the source code and the path to the building directory.

The source code is where you copied it to, so browse to that directory, something like
`/home/cshome/a/astudent/Documents/cosc342/lab04-Mosaicing`

Where to build the program is up to you, but the conventional thing to do is to add `/build` to the source code location, so something like
`/home/cshome/a/astudent/Documents/cosc342/lab04-Mosaicing/build`

Press the *Configure* button. You will probably get a prompt to create the build directory, click *Yes*.



You'll next be prompted to choose a toolchain to build. You can use whatever is installed on the machine you are using, but for the labs we'll assume you're using either *Unix Makefiles* or *XCode*. Choose your preferred environment from the drop-down, leave the other selection as *Use default native compilers*, and press *Done*.

CMake looks for the relevant tools, and checks that they are available, then updates its settings. You'll see a number of new settings added, which are highlighted in red. You can edit these if you need to, but you don't. So don't.

Finally hit the button Generate to generate your project files (either XCode or Makefile).

1 Building with Makefiles

If you are using Unix Makefiles, you'll need to go back to the terminal. Assuming you're in the `cosc342` directory you made earlier, you'd go to the build directory and compile the program with

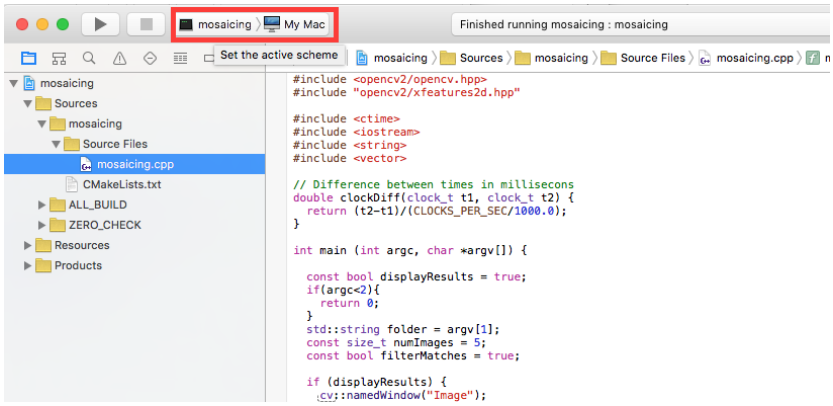
```
cd build
make
```

Then run the executable you just built and use the path to the images as command line parameter:

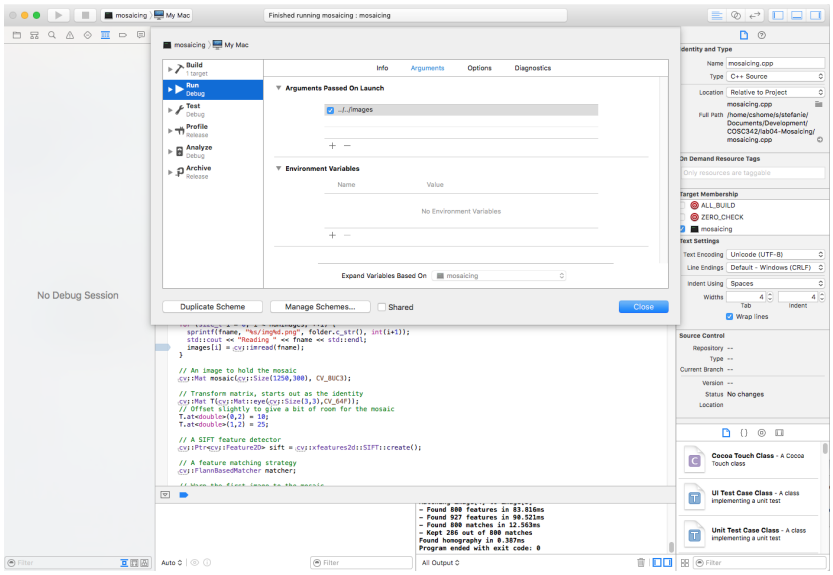
```
./mosaicing ../images
```

2 Building with XCode

Open the XCode project file from your build directory. XCode will open and you can select the source files in the source file explorer on the left hand side. To run the application, first select the **mosaicing** scheme.



Then edit the arguments that you pass to your program using the menu Product-Scheme-Edit Scheme.



Then press the run button.

3 Run program

A window will open and show the first image. The program will wait for a key-press after each image is added to the mosaic. Note that the time for various stages of the computation is reported in the console.

The Code

The code is in a single source file, `mosaicing.cpp` and is very similar to that discussed in lectures. There are a few parameters set at the start of `main()` which control the program's input and output:

```
const bool displayResults = true;
const std::string folder = "images";
const size_t numImages = 5;
const bool filterMatches = true;
```

The first line determines whether the program should illustrate its progress, or just run without creating extra windows. For now it is useful to display the results, but it will be useful to turn this off later on. When the results are displayed several windows are created, some of which may be hidden behind others. The second two lines determine what images will be processed. The program will (try to) read images from `[folder]/image[x].png` where `[x]` ranges from 1 to `numImages`. Finally, `filterMatches` determines whether or not ambiguous matches will be used when trying to determine the homography.

4 Exercises

We'll start by exploring a few of choices that can be made with the algorithms:

- How do we find matches between features?
- Do we filter the matches to try and remove ambiguous ones?
- How do we deal with outliers when computing the homography?

Computing the Homography

We'll start with how we compute the homography. First find where the homography is computed, which is towards the end of the code with the call

```
cv::findHomography(points1, points2, CV_RANSAC, 2.0);
```

The parameter `CV_RANSAC` tells OpenCV to use RANSAC when determining the homography, and the value of 2.0 is the maximum distance (in pixels) between aligned points for them to be considered inliers. There are two other options that can replace `CV_RANSAC`:

- `0`, which computes a least-squares fit from all of the matches.
- `CV_LMEDS`, which uses *least median of squares* error.

Least median of squares ranks all of the squared distances between the aligned features and tries to minimise the middle value of this list. Unlike RANSAC it does not require a threshold to be chosen, but it does require at least 50% of the matches to be inliers.

1. Try using the least median of squares method for the homography estimation. Are the results still good? Is it faster or slower than RANSAC in this case?
2. Try using a least squares fit to all of the data. Are the results still good? Is it faster or slower than RANSAC/least median of squares?

Filtering the Matches

The code currently tries to filter the matches by finding the best two correspondences for each point in the first image. If the distance to the best correspondence is less than 80% of the distance to the second best one, then the match is considered ambiguous and is rejected. You can turn this off by setting `filterMatches = false` at the start of the program.

1. What effect does turning off the filtering have on the RANSAC estimation of the homography? Does the mosaic still look OK? Is it faster or slower?

2. What effect does filtering have when using least median of squares method without filtering?

Feature Matching Strategies

The code initially uses the FLANN (Fast Library for Approximate Nearest Neighbours) based matcher, which is determined by the line

```
cv::FlannBasedMatcher matcher;
```

OpenCV also has a `cv::BFMatcher`, which does brute-force matching by comparing each feature in the first image to all features in the second image.

1. Change the code so that it does brute-force matching. You should make sure to be filtering your matches, and using RANSAC or least median of squares to find the Homography.

2. How does the speed of the brute-force matcher compare to the FLANN-based matcher?

3. Does the brute-force matcher give more or fewer ambiguous matches?

The time to mosaic these images is dominated by the SIFT feature detection, so the choice of matching strategy or homography estimation algorithm does not have a large influence on the total time taken. This is not true for larger images.

1. Change the program to use the two images in the `large` directory.

2. You should also turn off the display of the images.

3. What is the difference between brute-force and FLANN-based matching on these images?

4. The 'large' images are about 2MP, scaled down from the original 18MP images. How much slower would you expect brute-force matching to be on the original images? What about FLANN-based matching?

Determining the Mosaic Size

The reason we turned the display off for the large images is that the size of the mosaic is fixed by the line

```
cv::Mat mosaic(cv::Size(1250,300), CV_8UC3);
```

This is large enough to hold the mosaic from the first data set, but is smaller than the individual images in the 'large' data set. A more general mosaicing program would need to adjust the size of the mosaic to fit the data.

1. Suppose you knew in advance the dimensions of the images and the homographies between each pair. How could you determine the size of the mosaic image from this information?

2. If you did not know this information in advance you would not know how big to make the mosaic. How could you tell if an image was going to go outside of the current mosaic image? What might you do when this happens?