Project in Computer Vision – Report
Panorama Stitching using Python OpenCV

as part of
CS-4003 – Special Assignment in Computer Science

David Resin
Bachelor in Computer Science, 3rd Year
EPFL, Switzerland

Supervisor:
Mr. Jaakko Lehtinen
Dept. of Computer Science
Aalto University, Finland

June 16, 2017

1 Introduction

This document constitutes a description of the functionalities of and concepts used in the accompanying Python program. The primary goal of this project is to implement a panorama stitching program using the Python OpenCV library. I started this project with no previous experience in Python or OpenCV, but had already taken part in an introductory course in computer vision in my home university. The program mainly executes the following actions :

- Feature matching using RANSAC
- Simple bundle adjustment to apply lens distortion
- Seam carving using watersheds

Two kinds of panoramas can be achieved: a flat projection, which is limited in size as images on the side tend to be heavily distorted due to the homography (the program will crash due to the excessive size of the resulting images), and a spherical projection, where images are adjusted for lens distortion before being translated without any further deformation of the image.

I decided to use Python for several reasons. First because I was interested in learning this language for its ease of use, and also because the Python implementation of OpenCV is not as complete as the C++ one, which forced me to implement most of the functionalities myself instead of relying on already existing blackboxes (it is actually possible to implement this project in a much more polished way in less than 500 lines of C++, I included an example of this that I found in a CV textbook).

As a disclaimer, this program uses the SIFT algorithm, which is patented by the University of British Columbia, Canada. The patent can be consulted here: `https://www.google.com/patents/US6711293`.

2 How to run the program

The program is coded in Python 3.6.1. To be able to run it, you will also need the following external Python libraries : Numpy, Scipy, MatPlotLib and OpenCV 3.1.0 (along with the extra component `opencv_contrib 3.1.0`). This will require some work, which is explained in detail here : `http://www.pyimagesearch.com/2015/07/20/install-opencv-3-0-and-python-3-4-on-ubuntu/`. Scipy is not mentioned in this link, but you can easily install it by entering the command `pip install scipy`. MatPlotLib is a bit trickier to install, but is only necessary to print details in `watershed.py`, so in the case you are having trouble with these instructions `https://matplotlib.org/faq/installing_faq.html`, just comment out the import and the printing lines in this file.

To run the program, navigate to the `final` folder in the project files, and enter the command `python3 multistitch.py`. Possible arguments are the following:

- `-h` or `--help` – Displays the available arguments.
- `-i <INPUT>` or `--input <INPUT>` – Path for inputs, default is 'inputs/'.
- `-o <OUTPUT>` or `--output <OUTPUT>` – Path for outputs, default is 'outputs/'.
- `-m <MAIN>` or `--main <MAIN>` – Name of the main image (defaults to first image if not specified).
- `-s` or `--spherical` – Generates a spherical projection instead of the standard projection.
- `-d` or `--details` – Outputs intermediate data like masks, transformed and lens-warped images, as well as working data from the seam finder.

- `-z <ZOOM>` or `--zoom <ZOOM>` – Size factor for the input images, the program can take a long time and even fail if the value is too big. Default is 0.5.

Several sets of test images are provided, along with result examples.

3 Description of the program

The main program file is `multistitch.py`. The program unfolds in the following way (I will omit parts such as argument parsing, which are mere tools and not the focus of the project):

- First the images are read and put into `Image_Data` objects, which will be the carrier of all their extracted data and relations throughout the program.
- We then extract all the relational information in an array, using the `stitcher.ransac` function on all couples of images. We then use `linker.py`'s solver to get the best links between images.
- In the case of a spherical projection, we will use `bundle_adjust.py`'s functions to get the best lens distortion values and affine transformation for each image.
- Now `linker.py` is used again to make a tree out of all of our data, which is then reduced to turn image-to-image transformations to global transformations.
- We now use `image_warp.py` to warp the images using their homographies, compute the global positions of the transformed images as well as the size of the final result, translate the images appropriately and combine them all using our custom image blender in `watershed.py`. The result is then printed to an output image and the program ends.

4 Possible enhancements

I had high ambitions for this project, but it quickly became obvious that I would not be able to include the most advanced features I had in mind and that I would have to focus on making the core work. But I still would like to share the enhancements I had in mind:

- **Exposure compensation** – This one was something I thought I would be able to include, but it proved harder than I imagined to implement. My idea was to find gamma values that would minimize the difference between images, but there seems to be further tweaks related to hue/saturation/value that I couldn't figure out, as my results proved unsatisfactory. I also tried altering exposure through alpha/beta values but to no avail. I am fairly disappointed by this one as it would substantially improve the results, as the seam finder isn't enough to hide the exposure difference on images with big regular patches like skies and walls.
- **Wave correlation** – When working with long panoramas, the angular error between images will accumulate, especially if the camera was not exactly pointing towards the horizon, resulting in a panorama that will bend up or down and even make zig-zag patterns for the very wide ones in some cases. The idea here would be to force the pictures to line up on a horizontal line and optimize from there.
- **Seam carving with 3 or more images overlapping** – In this project, the seam carving algorithm works sequentially; when 3 or more images are overlapping, we solve for the first two, then add the next one and so on. To do it all at the same time we would have to define what is the absolute difference of 3 images, which is a bit counter-intuitive. An idea to solve that would be to compute all differences for n images, giving us $\frac{n(n-1)}{2}$ difference images, thus different possible cuts, from which we have to deduce the allocation of different patches to different images.

- **User interface with previsualisation** – I left this one on the side as UI is not exactly the main focus of this project, but it would be interesting to have a user interface to preview what our panorama is going to look like, decide the orientation, zoom, centering, resolution, and even methods used (the C++ program I found shows all the possible computation methods for each step of the pipeline in the C++ implementation of OpenCV).

Conclusion

While I wish I could have done more to get finer results, I am happy I was able to code the whole main pipeline by myself and get acceptable panoramas, especially since I had never done anything of this kind before. The knowledge I have accumulated about Linux, Python, OpenCV, computer vision in general, mathematical techniques, algorithms and more is invaluable, and this is only streghtening my will to keep working further with these tools during my Master's degree. Lots of things I have designed while doing this project didn't make it to the final result as although I learnt a lot about them, I realized that they did not provide the expected results. This is quite akin to a writer contemplating his completed book, with a trashcan overflowing with drafts of all kinds next to their desk. Studying in such a field is quite a challenge, as the data we are working with forces us to make abstraction of practically everything to work with approximations, filters and optimization techniques to converge towards the generally unique solution to the system.

Thanks

Even though it is not a Master's Thesis, this work has been my primary focus during my year of exchange in Finland, and being able to finally complete this project reminds me of all the things that happened during this crazy year where I learnt a lot, and not just in terms of my studies. I think it is fair to slip a little thank you to all the people that made my exchange possible, and also those that made it unforgettable.

Matleena, Juulia, Jussi and more generally Lions Club International, who are the primary reasons I got interested in discovering Finland

Martin, who told me "if you're hesitating [about going on exchange], then just go"

My exchange coordinator, Mr. Benz, who has always been very kind and supportive

The same can be said for my whole family, of course

Tietokilta, AYY, Aalto University, and their awesome efforts to welcome us exchange students and make us true Teekkaris, complete with *teekkarilaki*, *opiskelijahaalari*, *sitsits* and of course, *Vappu*

All of my *Sticks* buddies, for being the ultimate team of friends during a year unlike any other, and that includes Seha, Roberto, Guillem, Lauri, Lukas, Liam, Katia, Claire, Fritz, Pieter, Martin, Sonja, Korkki, Martturi, Georg, Chris and Griñán

Our tutors, Minna and Sergei, because the *Sticks* wouldn't exist without them, as well as the hundreds of other students, finnish and internationals, who shared unforgettable moments with us

Hazal, for morally supporting (and bearing) me even when my code wouldn't stop failing

Pr. Peter Kelly and the Aalto Ventures Program, for helping me set my career goals, something I would never have hoped to come up with that soon

And of course Pr. Jaakko Lehtinen, who gave me the opportunity to do this project, which in turn allowed me to learn so much about Python, computer vision and more