Stitching Images

(panorama11.py)

# Overview

Readers are strongly encouraged to read and understand [this tutorial](http://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/) before moving to the next parts of this doc.

The above tutorial provides a basic knowledge to image stitching, some of the insights given from there are applied in this implementation.

To be a bit more specific, I do the following:

* When we want to stitch multiple images together, each time we stitch a pair of them, until all images are stitched. For example, we want to stitch 4 images: 0, 1, 2 and 3. First, we stitch image 0 and image 1, then take the result. Stitch that result image with image 2, take the new result image. Lastly, stitch the new result image to image 3.
* We stitch the first 2 images together by:
  + finding key-points of each image.
  + virtually transform the images to spherical view (‘virtually’ means we do not actually transform the images, but only alter the key-points according to the transform formula)
  + Adjust/optimise the degree of above transformation so that the 2 images match each others as much as possible (there is a heuristic to measure how much the 2 images match)
  + After we have the most suitable degree, actually transform the 2 images to by that degree, then do a translation to move the right image to overlap with the left image
* For other stitching (not stitch the first 2 images), we do similar to the above, but with an exception that we only adjust the transform degree of the right image, not the left one (because the left one has already been adjusted in previous stitching).

# 2. Finding key-points

This task can be done automatically by opencv.

For now, I use SIFT as the feature of key-points, but it can be changed to SURF, ORB, or others.

# 3. Matching key-points

This task can also be done by just several lines of opencv code.

I write here to elaborate what ‘matching key-points’ means:

For example, if image 1 has 4 key-points: k1, k2, k3, k4 and image 2 has 3 key-points: kA, kB, kC. Matching key-points means we obtain the information that k1 is very similar to kB (and we have a match (k1, kB)), k4 is very similar to kA (so that we have a match (k4, kA). k2, k3 and kC don’t have any key-point in the other image that is similar to them, so they don’t belong to any match.

The result should be a list of pair of key-points, for example, [(k1, kB), (k4, kA)].

# 4. Find homography matrix:

A homography matrix is a 3x3 matrix. [This](http://www.learnopencv.com/homography-examples-using-opencv-python-c/) gives a clear explanation.

We can only find the homography matrix if the number of matches between 2 images is more than or equal to 4.

By applying homography to the right image, we do a transformation to it, the right image will be warped, and moved to the co-ordination system of the left image.

# 5. Spherical view and optimise the spherical degree

[This](http://web.engr.illinois.edu/~ntripp2/cs445/project/) gives an explanation and demonstration of spherical view, along with the formula to transform normal image to spherical image.

On how to optimise the spherical degree (find the most suitable degree to transform normal image to spherical image):

We have to have a function to measure how well the 2 images are matched. This function is called a heuristics. The heuristics I use here is a linear function of the difference of homography matrix with identity matrix (because when the homography matrix is identical to the identity matrix, we have the best result).

The readers should understand the homography and the role of each value in the homography matrix in order to grasp this heuristics.

Because this heuristics is non-convex, there are many optimisation methods can be applied to find a good result, but for now, I use a very simple one. I just divide the feasible space in to many small parts, try each part to see which part is the best one, and use that part.

# 6. Future work

We can try some other modifications to make result better:

* Not gradually stitch images from left to right, but stitch pairs of images which have the most number of matches first
* Use Laplacian blender for blending
* Add reinforce rows so that no pixel of image will be lost
* Try cylindrical Transformation
* Get focal length by extracting information of ELIXIR of image
* Use another descriptor (SIFT, SURF, ORB, …)
* Refer to the below paper for more: [paper](http://matthewalunbrown.com/papers/ijcv2007.pdf)