Panorama Generator

Alexander Crawford

Computer Vision

UOIT

April 4, 2017

**Introduction**

In the field of computer vision, finding areas of similarity between images is a topic that has an excellent balance of problems and solutions. Identifying regions of interest is a key feature of most, if not all, computer vision projects; many scientists have worked on trying to improve the accuracy and speed of detecting these points of interest. To achieve a high degree of accuracy, an algorithm to detect regions of interest should detect the same regions of interest in a number of different circumstances, even when an image is translated, shaded, or scrambled with noise.

A practical implementation of these concepts comes in the form of making panoramic images. By detecting points of similar features, one can stitch together a full 360 panorama of an area by taking a number of smaller photos. While many tutorials exist to stitch two given images together, a problem that has received less attention is what to do if you do not know what images go together. This project explores a solution to that.

Included in this project is “pano.py”, a panoramic image generator. It accepts any number of image arguments, and generates a panorama by stitching them together. If the image is a “true panorama” (i.e. consists of images covering 360 degrees), the images can be in any order; if only a partial panorama is desired, the only requirement is that the first submitted image should be the leftmost image in the panorama. This document describes in greater detail the methods used to generate this panorama, as well as some current areas that need improvement.

**Theory**

In order to match images effectively, the panorama generation program uses the scale-invariant feature transform algorithm, known as SIFT. This algorithm detects “points of interest” in an image - that is, regions that are likely to remain detectable even through image transformations, colorings or noise addition. The best implementations of this algorithm involve running a variety of different tests for invariant points in the image, and choosing the points that pass most of the tests as key points. In practice, since the key points should be invariant to transformations in the image, they should be reliably detectable in multiple photographs containing the same region - specifically, two images to be stitched together.

To find matches between images, the “k nearest neighbors” algorithm is used. Multiple versions of this algorithm exist, but all of them exist to find the collection of point-to-point matches between two sets of points that minimizes the overall distance. To determine whether there are matching features in the image, the distances between each pair of points in the set is compared. If the distances are sufficiently similar, this likely means that the two lines represent a single transition, and the points that constitute them can both be added to comparison sets for generating a homogeneous translation.

**Process**

The process of creating a panorama from an unordered set of images begins by giving order to the images. To accomplish this, each image is compared to every other image, using the SIFT algorithm to generate keypoints and the thresholded KNN algorithm is run to find the best matches between the keypoints of two images. A basic ratio of the number of KNN matches above the threshold, compared against the total number of KNN matches (including bad matches) creates a simple metric to judge the quality of compared images against each other; while extensive tweaking would be necessary in order for this metric to actually validate whether two images are a match or not, since we already know that there will be exactly one or two matches for an image (that is, the images that it will be stitched with on its left and/or right), so all that is that the values be comparable against each other.

These percentage-of-valid-matches values are inserted into a square array, with side length equal to the number of images (so that, taking the number of two input images as column vector indices, the value of the array at that point represents the percentage-of-valid-matches value of the comparison between those two images). To ensure images are iterated through in a regular way, a homography between the sets of keypoints is also generated, and both self-comparison operations and operations resulting in the second image being placed to the left of the first are discarded, meaning that neither are considered in the process of generating a panorama.

Once this array is established, starting from the first input image, the panorama generator examines the row corresponding to the first input image, and takes the highest value in the array, corresponding to the best match. Then, the homographic transformation is obtained, and the second image is transformed so that the matching points overlap. From there, the process repeats, seeking the best match for the second image, combining every successive transformation matrix so the new image ends up attached to the changed position of the previous one.

**Results**

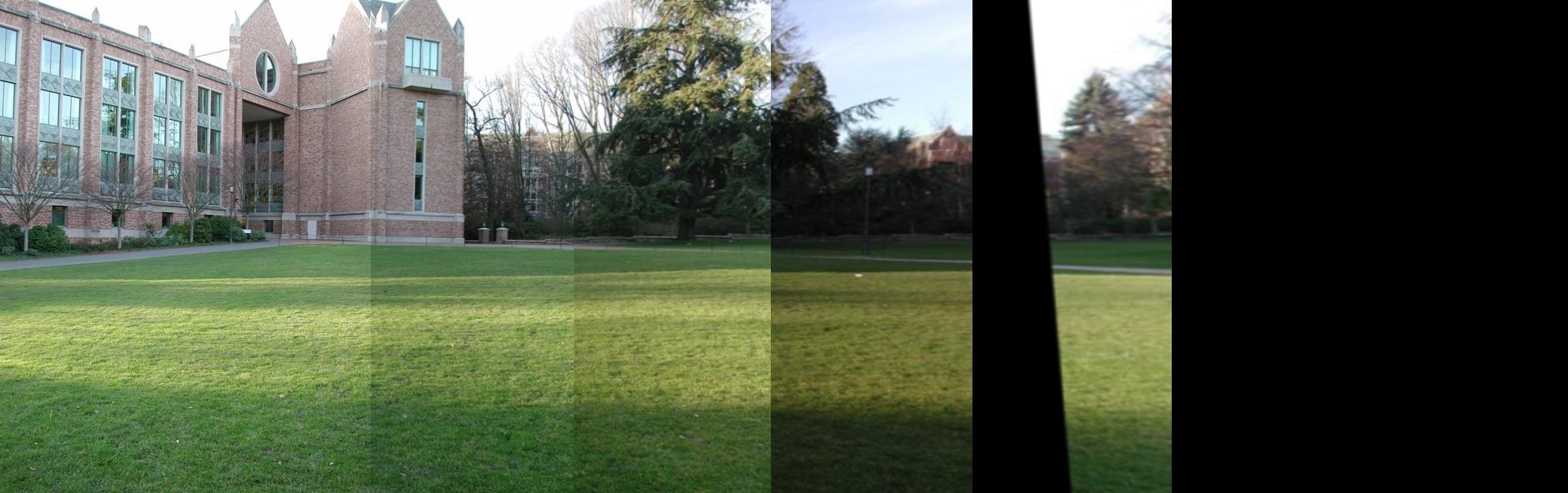


Overall, while the stitching process of the images has some major issues, the results appear quite attractive for smaller sets of images. In this example image, while there is some failure to match mountain shapes in the stitch running down the middle, and there are clear lines in sky color indicating where the stitch took place, the core images remain intact, and are even properly rotated (see the rightmost image) to match the orientation of the land. However, listed below are some potential areas of improvement that were unable to be improved upon before the project was submitted, as well as a wishlist of how to improve them in the future.

One of the major issues is the slowness of the comparison. As a proof-of-concept, many of the algorithms were written in a manner designed to be easy-to-write, rather than being memory-efficient. For example, SIFT keypoints are recalculated every time the matching algorithm is run, while a more efficient method calculate one set of keypoints per image and simply reference them when needed. More time spent on making information reusable could dramatically cut down on the time spent on the algorithm, but the code will always have some innate slowness, given the need to compare every image against every other image to generate the set of best image matches.

Another resolvable issue is a problem with left-translation images. For unknown reasons, the transformations do not work properly when moving from right to left - the resulting image gets progressively more and more skewed until the new image is almost totally unrecognizable. This may be due to positioning of the image when attempting the transformation, but there was not enough time to resolve the issue.

The final major issue is not as easily resolvable - that is the problem with error propagation. For rotational translation operations, because it changes the axis the image remains on, if there are any errors in the calculated rotation, the error will propagate throughout the entire translation operation set, resulting in one to many images getting lost as the image gets rotated or translated off the screen (as in the example below). Unfortunately, the only real way to resolve this (beyond some kind of final ‘sanity test’ to try to bend the image back together so the panorama forms a complete circle) is to improve the quality of the homogeneous function generation and the SIFT point detection/matching, which is a nontrivial issue. This issue is closely tied to the program, so for a “true” implementation of a panorama generator, this problem would be unavoidable.



Despite these big issues, the panorama generator is largely a success. It has solved the core problems of detecting and stitching together images, and the remaining problems are merely with the algorithms implemented, which are somewhat more modular in their ability to be replaced with other methods (providing they can still produce homographies and a comparison value). While there is potential for this project to improve, this panorama generator works effectively enough to act as a basis for future panorama projects.