Local Feature Matching

## Overview

We will create a local feature matching algorithm (Szeliski chapter 4.1) and attempt to match multiple views of real-world scenes. There are hundreds of papers in the computer vision literature addressing each stage. We will implement a simplified version of SIFT.

## Task

 Implement the three major steps of local feature matching.

* 1. **Detection** in the get\_interest\_points function in student.py. Please implement the Harris corner detector (Szeliski 4.1.1, Algorithm 4.1). You do not need to worry about scale invariance or keypoint orientation estimation for your Harris corner detector.
  2. **Description** in the get\_features function, also in student.py. Please implement a SIFT-like local feature descriptor (Szeliski 4.1.2). You do not need to implement full SIFT! Add complexity until you meet the requirements. To quickly test and debug your matching pipeline, start with normalized patches as your descriptor.
  3. **Matching** in the match\_features function of student.py. Please implement the "ratio test" or "nearest neighbor distance ratio test" method of matching local features (Szeliski 4.1.3; equation 4.18 in particular).

## Requirements

The project requires your implementation reaches 60% accuracy on the most confident 50 correspondences in 'matches' for the Notre Dame pair, and at least 60% accuracy on the most confident 50 correspondences in 'matches' for the Mt. Rushmore pair. The evaluation function we will use is evaluate\_correspondence() in helpers.py . We have included this function in the starter code for you so you can measure your own accuracy:

1. Implementation of Harris corner detector in get\_interest\_points()
2. Implementation of SIFT-like local feature in get\_features()
3. Implementation of "Ratio Test" matching in match\_features()

**Forbidden functions:** skimage.feature.daisy(), skimage.feature.ORB(), and any other functions that extract features for you, skimage.feature.corner\_harris() and any other functions that detect corners for you, any function which *computes histograms*, sklearn.neighbors.NearestNeighbors() and any other functions that compute nearest neighbor ratios for you. scipy.spatial.distance.cdist() and any other functions that compute the distance between arrays of vectors.

**Potentially useful functions:** Any existing filter function, zip(), skimage.measure.regionprops(), skimage.feature.peak\_local\_max(), numpy.arctan2().

## Note

main.py handles files, visualization, and evaluation, and calls placeholders of the three functions to implement. For the most part you will only be working in student.py.

The Notre Dame, Mount Rushmore, and Episcopal Gaudi image pairs include 'ground truth' evaluation. evaluate\_correspondence() will classify each match as correct or incorrect based on hand-provided matches. You can test on those images by changing the -p argument you pass to main.py.

As you implement your feature matching pipeline, check whether your performance increases using evaluate\_correspondence(). Take care not to tweak parameters specifically for the initial Notre Dame image pair.