CPE 428: Computer Vision
Homework 4: Augmented Reality
Instructor: Jonathan Ventura

In this homework you will implement an augmented reality system using a planar tracking target. In each frame of the video, the system will automatically recognize the tracking target using SIFT feature matching, compute the homography between the tracking target and the video frame of the tracking target, and render a virtual overlay on top of the detected tracking target.

The provided template code gives an outline of a `Tracker` class and a `RANSAC` class.

Tracker class:

- The constructor takes as arguments the image of the tracking target (called the "reference" image), and the image to be overlaid on the video (the "overlay" image). In the constructor, you should extract and store SIFT keypoints detected on the reference image.
- The `compute_homography()` function takes as input the current video frame (the "query" image) and returns the computed homography. First, the function finds SIFT matches between the reference image and the query image using the `BFMatcher`. Matches are then filtered using the ratio test (you will need to code this yourself, there is no OpenCV function for it). Then the matches that pass the ratio test are used to compute the homography and inliers with RANSAC. For partial credit (max 90%), you can use the `cv.findHomography()` function with the `cv.RANSAC` flag to compute the homography. For full credit, you will need to implement RANSAC yourself (see below).
- The `augment_frame()` function uses the homography to draw the overlay image onto the video frame. Here you can use `cv.warpPerspective()` to apply the homographic warp. See the lecture slides for an explanation of how to do the overlay.

`RANSAC` class:

- The constructor takes the RANSAC parameters including desired probability of success, inlier threshold, and the initial value of the estimated outlier ratio. The sample size is fixed to 4 since computing a homography requires four correspondences.
- The `compute_num_iter()` function computes the maximum number of iterations according to the current estimate of the outlier ratio. This formula comes from the original paper on RANSAC.
- The `compute_inlier_mask()` function evaluates a set of correspondences against a proposed homography and determines inliers and outliers. It returns a list of Boolean values where True means that correspondence is an inlier. Note: You can use `cv.perspectiveTransform()` to apply the homography to the reference points.
- The `find_homography()` runs the actual RANSAC loop:
    - First initialize your outlier_ratio to init_outlier_ratio and compute the number of iterations required.
    - Initialize an iter variable and start a while loop to perform RANSAC iterations.
    - In each iteration, you will randomly choose four correspondences and compute the homography using `cv.findHomography` with method=0 (least-squares estimation).
    - Then count the number of inliers for this homography. If it is greater than your current best, store this homography and the list of inliers.

    - When a new best homography is found, you should also update your outlier ratio and re-compute the number of iterations.
    - Once the number of iterations exceeds max_iter, the while loop exits.
    - Finally, re-compute the homography using the inlier set for the best homography found.

Implement all the functions in tracker.py, and test out your system on the provided example video.

Once you have the system implemented and working, inspect the results and in your report, describe qualitatively how well the system works. When does it produce accurate results and when does it falter?

Relevant tutorials:

- This page explains how to create a `SIFT` object and detect keypoints using `.detectAndCompute()`.
- This page explains how to match SIFT keypoints using the `BFMatcher` (brute-force matcher).

Required libraries:

- cv2 (OpenCV)
- numpy

Links to required files:

- Reference image
- Overlay image
- Input video
- Example output (note: has a blue border around the overlay which won't be present in your results)

Deliverables:

- process.py
- tracker.py
- Text document (plain text, Word doc, or PDF) explaining your solution

Please upload these files individually, not as a ZIP – this makes grading easier.