CS543 Assignment 3

Your Name: Yen-Ting Liu
Your NetId: ytliu2

Part 1: Homography estimation

Describe your solution, including any interesting parameters or implementation choices for feature extraction, putative matching, RANSAC, etc.

Following the execution steps,

1. Feature extraction
   I uses the SIFT implementation from OpenCV directly. All images use default settings
   - **Number of features** extracted 1000
   - Number of octave layer 4
   - **Contrast threshold** (scale space threshold) 0.04
   - **Edge threshold** (Harris response) 10
   - **Sigma** (initial sigma for the blur kernel) 1.6

   Since I used `scikit-image` heavily in my entire MP, I decide to wrap it in a class that inherits `FeatureDetector` and `DescriptorExtractor`. This allow OpenCV SIFT to interchange with ORB feature extractor in `scikit-image`.
2. Putative matching
   I decide to follow the algorithm suggested by D. Lowe. After finding out squared Euclidean distance across keypoints, it figure out the shortest and 2nd shortest distance, only the shortest one is **1.5 times** shorter than 2nd shortest, will the algorithm keep this match.
3. RANSAC
   I follow the RANSAC interface that `scikit-image` has, and implement my homography model to inherit `BaseModel` class. However, my implementation uses residuals mean from inliers instead of the population,

   ```
   residuals_mean = np.mean(residuals[inliers] ** 2)
   ```

   this seems to provide more stable registration result. For all images, I use
   - **Residual threshold** 1
   - **Minimum number of samples** 4 (since we are working with H estimation)
   - **Maximum number of trials** 10000

   I didn’t implement dynamic max trial using current residuals.
4. Warping
   I first use my `register_images` function to generate transformation matrix for each image pair, and later uses my `stitch_images` function to calculate the complete transformation

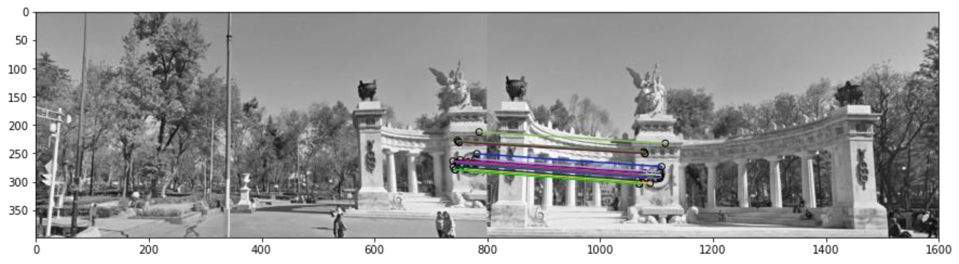
from image to image (e.g. for 3 images, hook them up). This allow me to estimate overall image size in later stage, and the potential to do bundle adjustment.

For the image pair provided, report the number of homography inliers and the average residual for the inliers.

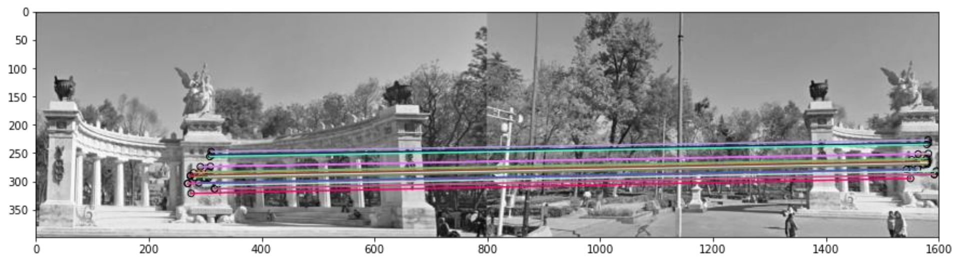
	LEFT as reference	**RIGHT as reference**
Number of inliers	21	23
Average residuals	0.185227	0.210508

Also, display the locations of inlier matches in both images.

LEFT
as reference

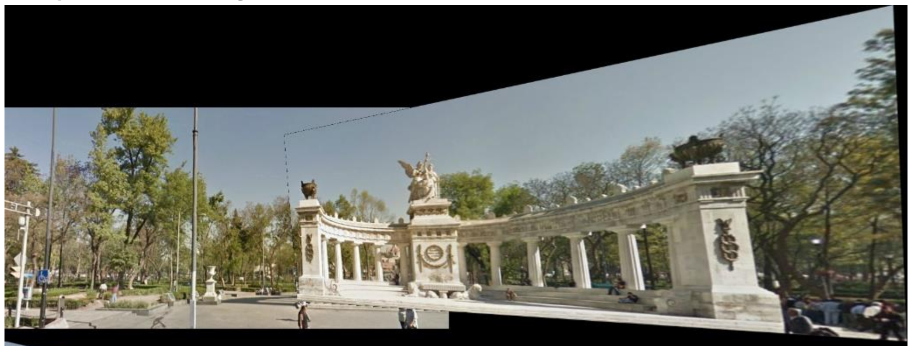


RIGHT
as reference

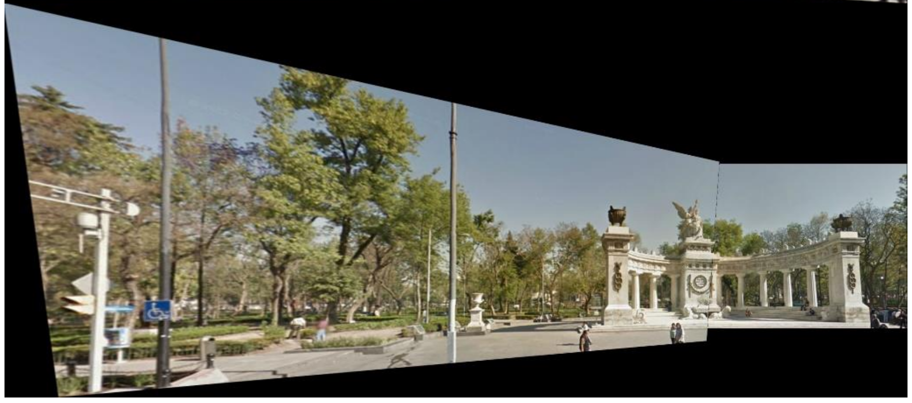


Display the final result of your stitching.

LEFT
as reference

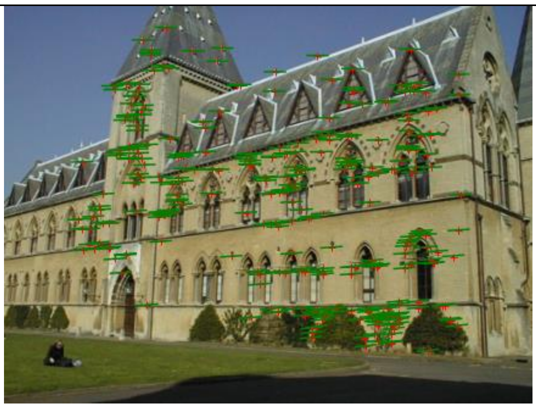
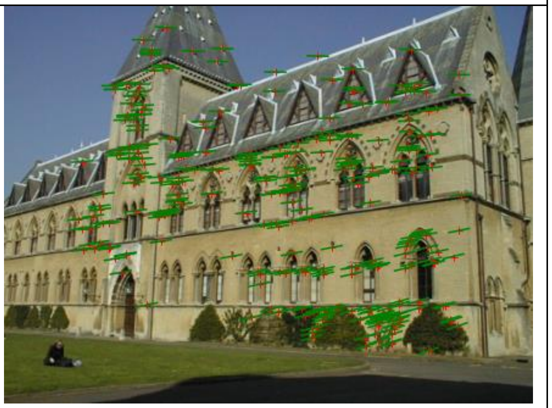
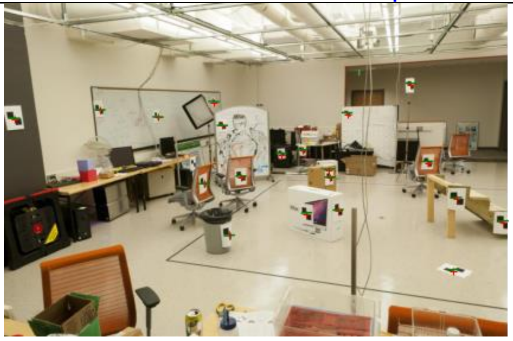
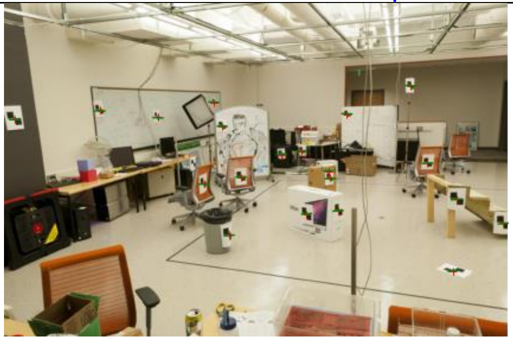


RIGHT
as reference



Part 2: Fundamental Matrix Estimation, Camera Calibration, Triangulation

For both image pairs, for both unnormalized and normalized fundamental matrix estimation, display your result (points and epipolar lines) and report your residual.

	Unnormalized	**Normalized**
Library	 **Residuals** 0.179213 px	 **Residuals** 0.085709 px
Lab	 **Residuals** 6.567092 px	 **Residuals** 0.911030 px

Residuals follow website definition, using mean squared distance between points and their corresponding epipolar lines.

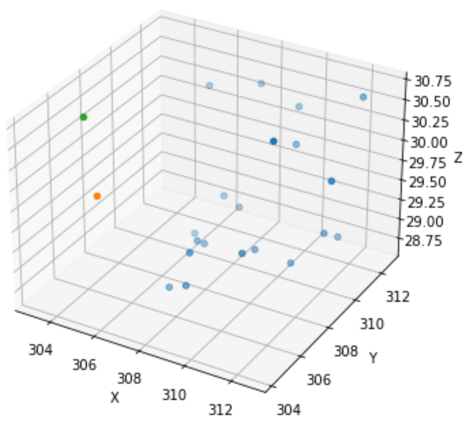
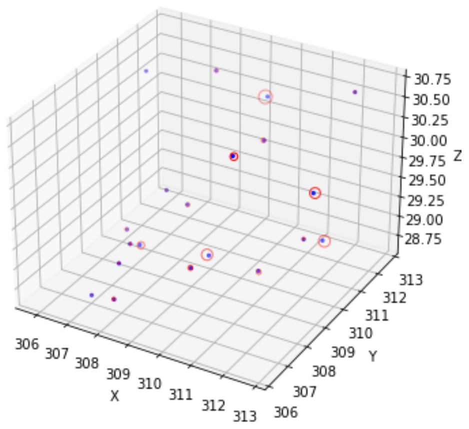
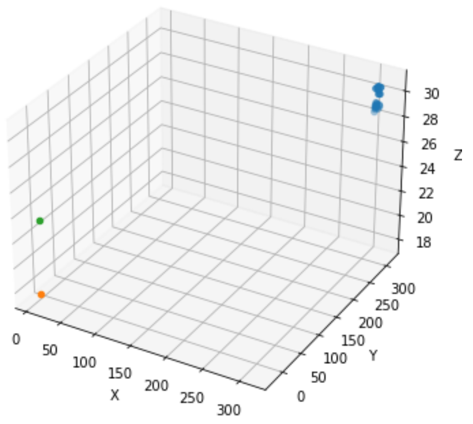
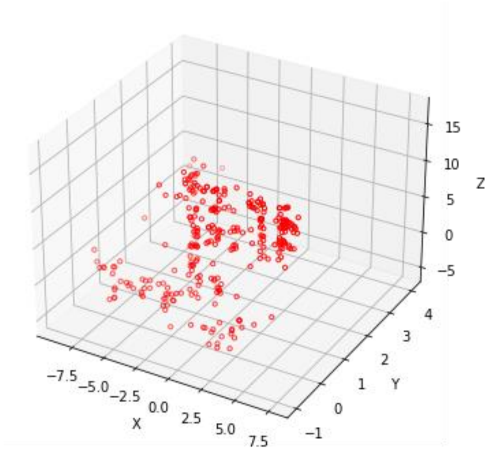
For the lab image pair, show your estimated 3x4 camera projection matrices.
Please find the matrices in next question.

Report the residual between the projected and observed 2D points.

	Projection matrix	**Residuals**	**Average squared distance**
lab1	$\begin{bmatrix} -3.100e\text{-}03 & -1.462e\text{-}04 & 4.485e\text{-}04 & 9.789e\text{-}01 \\ -3.070e\text{-}04 & -6.372e\text{-}04 & 2.774e\text{-}03 & 2.041e\text{-}01 \\ -1.679e\text{-}06 & -2.748e\text{-}06 & 6.840e\text{-}07 & 1.329e\text{-}03 \end{bmatrix}$	13.55	0.39
lab2	$\begin{bmatrix} -6.932e\text{-}03 & 4.017e\text{-}03 & 1.326e\text{-}03 & 8.267e\text{-}01 \\ -1.548e\text{-}03 & -1.025e\text{-}03 & 7.274e\text{-}03 & 5.625e\text{-}01 \\ -7.609e\text{-}06 & -3.710e\text{-}06 & 1.902e\text{-}06 & 3.388e\text{-}03 \end{bmatrix}$	15.54	0.37

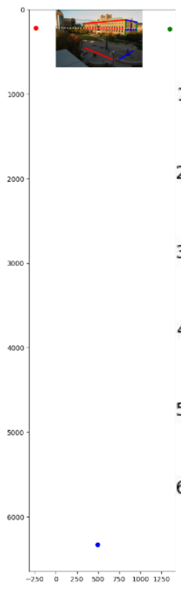
For both image pairs, visualize 3D camera centers and triangulated 3D points.

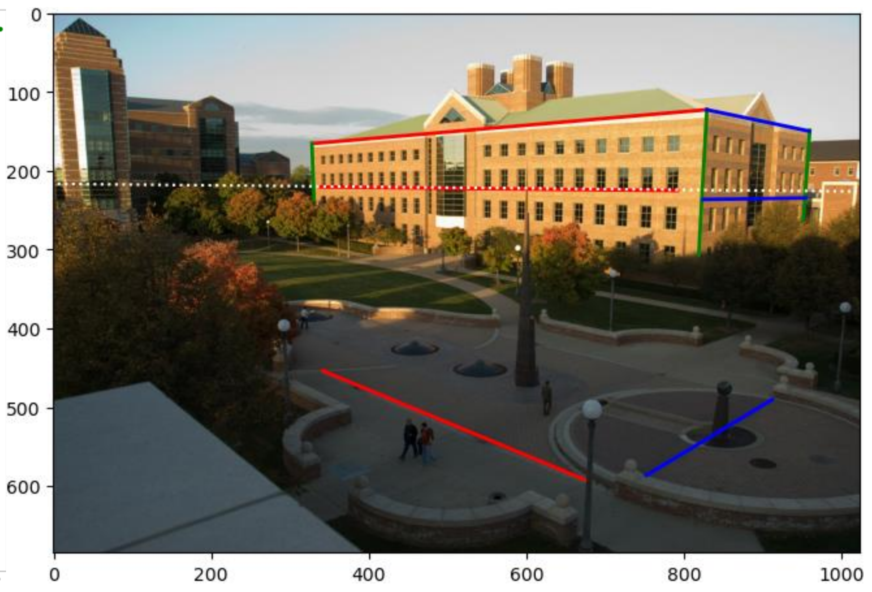
It is hard to visualize triangulated results when camera center is in the view.

	3D distribution (Camera centers)	**Triangulation** (ground truth; estimated)
Lab	 (305.8, 304.2, 30.1) (303.1, 307.2, 30.4)	 (marker radius proportional to error)
Library	 (7.3, -21.5, 17.7) (6.9, -15.4, 23.4)	 *(no ground truth)*

Part 3: Single-View Geometry

Plot the VPs and the lines used to estimate them on the image plane using the provided code.
Left image contains all 3 VPs (in different colors), one is extremely far away but still numerically retrievable. Right image are edges (follow color of each VP) used to estimate VPs and horizon (white dotted line).





Specify the VP pixel coordinates.
(x, y) = **left** (-237.52, 214.21), **vertical** (493.85, 6330.22), **right** (1343.24, 229.17)

Plot the ground horizon line and specify its parameters in the form a * x + b * y + c = 0. Normalize the parameters so that: a^2 + b^2 = 1.
Horizon plot in the first image. **Line equation** -0.009458*x + 0.999955*y + -216.451659 = 0

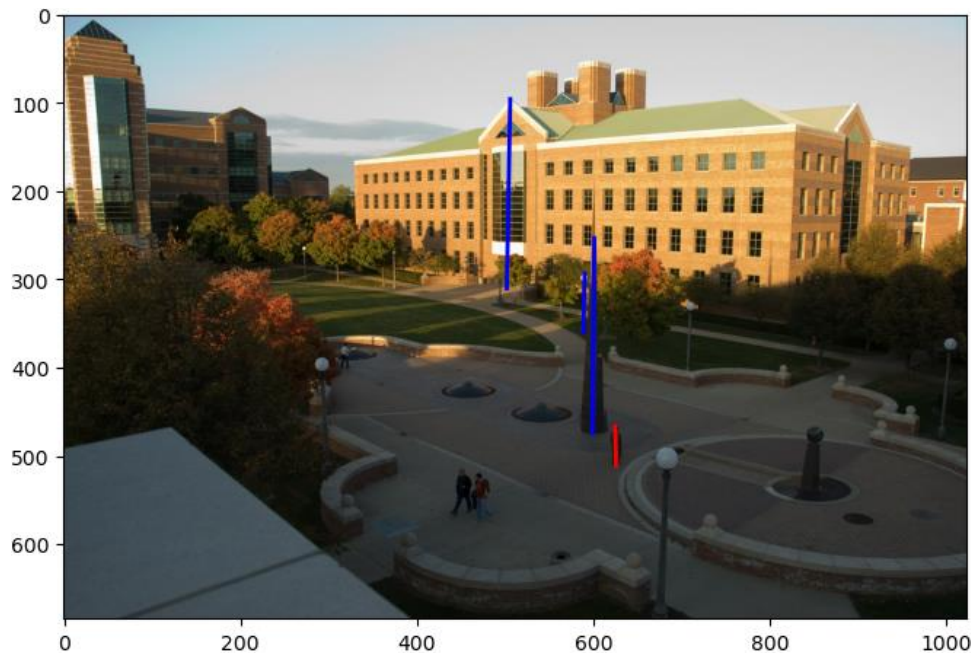
Using the fact that the vanishing directions are orthogonal, solve for the focal length and optical center (principal point) of the camera. Show all your work.
Principal point (x, y) = (550.65, 323.95), **Focal length** 783.77

Compute the rotation matrix for the camera.
Following the VPs order on website, (v_x, v_y, v_z) = (right, vertical, left)

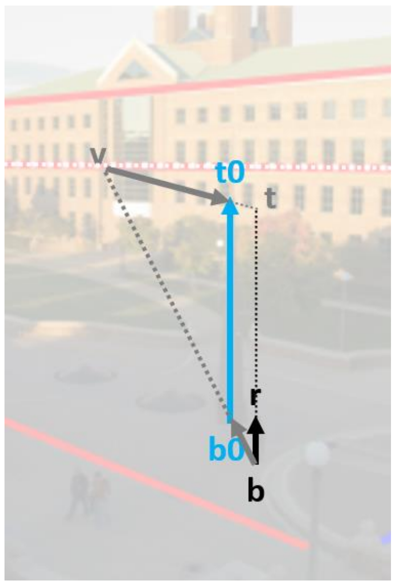
$$\begin{bmatrix} 0.708 & -0.009 & -0.706 \\ -0.085 & 0.992 & -0.098 \\ 0.701 & 0.129 & 0.702 \end{bmatrix}$$

Estimate the heights of (a) the CSL building, (b) the spike statue, and (c) the lamp posts assuming that the person nearest to the spike is 5ft 6in tall. In the report, show all the lines and measurements used to perform the calculation.
Convert to metric so we can do height calculation easier.

Using the following line segments (blue) relative to the person (red) to do height estimation.



Person 1.6764 m; **CSL** 22.09 m; **Statue** 9.20 m; **Lamp post** 5.46 m



Follow the convention in lecture slide, we first calculate the intersection from bottom points to horizon

$$v = (b \times b_0) \times (v_x \times v_y)$$

Then we can derive the projected (top) point of object of interest (statue, in this example),

$$t = (v \times t_0) \times (r \times b)$$

From here, we can start to see the triangle similarity relationship

$$\frac{\|t - b\|}{\|r - b\|} \frac{\|v_z - r\|}{\|v_z - t\|} = \frac{H}{R}$$

where R is the reference height, and H is what we want to calculate.

How do the answers change if you assume the person is 6ft tall?
Person 1.8288 m; **CSL** 24.10 m; **Statue** 10.04 m; **Lamp post** 5.96 m

Extra Credit

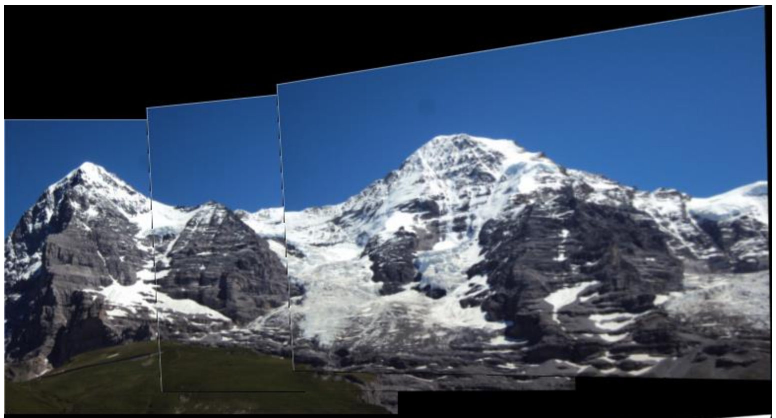
Don’t forget to include references, an explanation, and outputs to receive credit. Refer to the assignment for suggested outputs.

Part 1

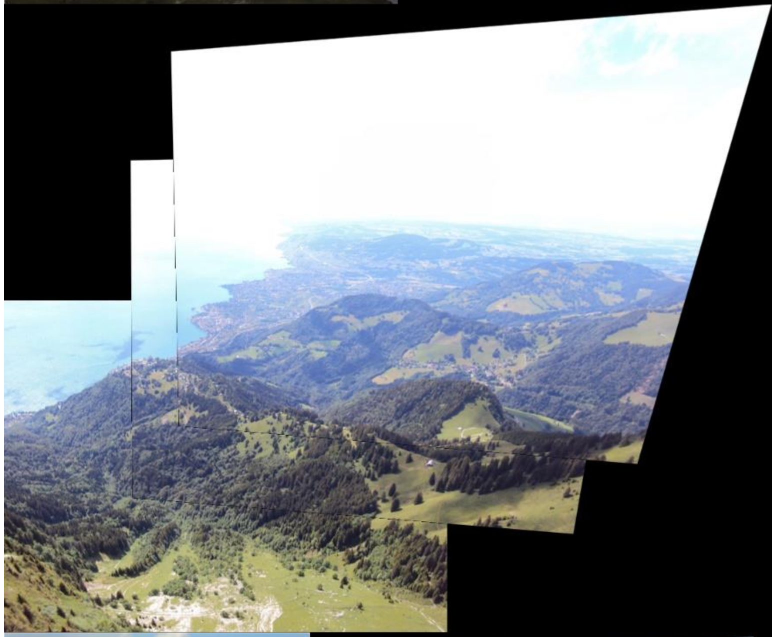
Extend homography estimation to work on multiple images

Continue with previous description, my implementation can easily scale out to accommodate multiple images. In all following images, I choose to calculate matches by consecutive image pairs instead of reference with the first image, since `Pier` has non-overlapping images.

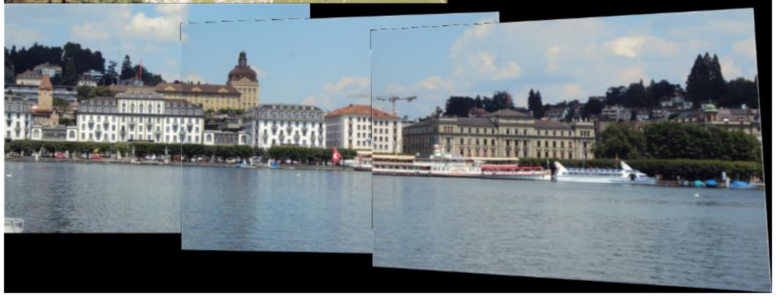
Hill



Ledge



Pier



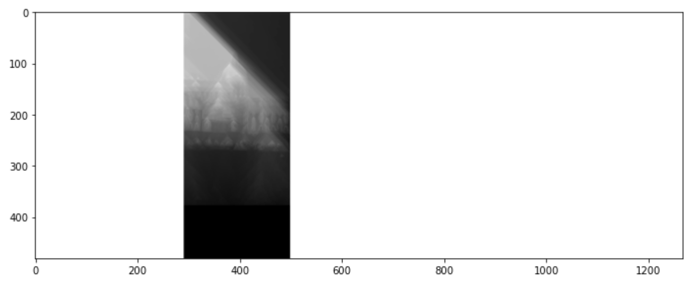
Register difficult image pairs

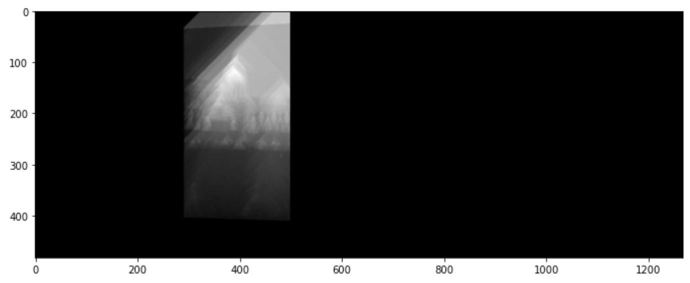
I accidentally implement Lowe's ratio test in my baseline code, does this count as bonus?

Image blending techniques

Seam carving first published by Avidan and Shamir (2007) as a way to to content-aware image resize. It calculates cost function for any given cut, and try to find the lowest cost path to cut a line of pixel away from the image.

We use Sobel detector as the *cost*, since these are high frequency responses that are most visible by naked eye. The following image pair shows pier image 1 and 2, accumulated cost array based on Sobel detection result. Pixels that are not in the overlapped region are offset to 1 and 0 to maximize cost.

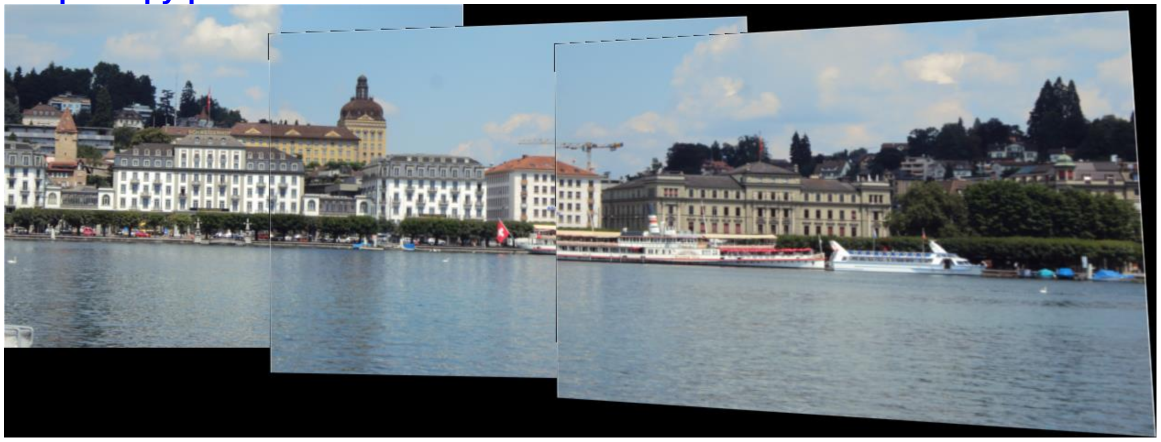




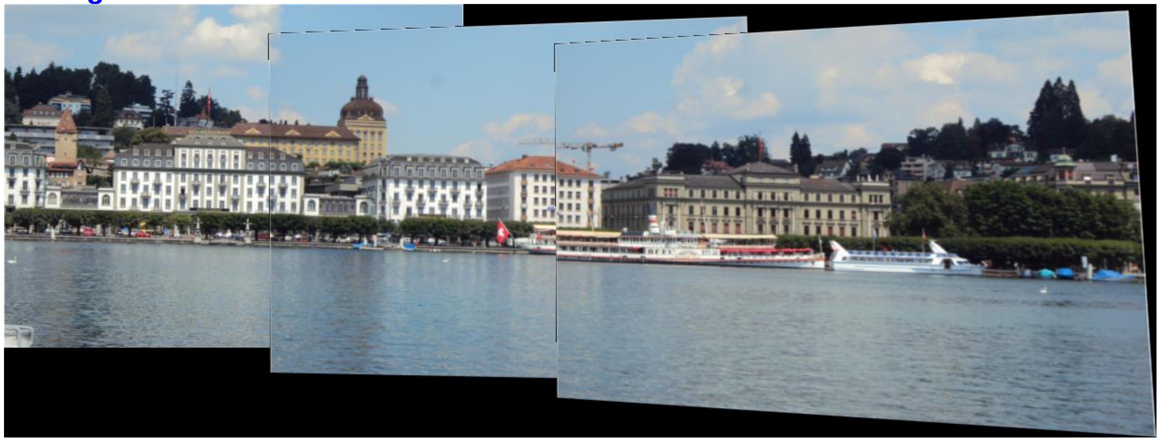
We can repurpose this characteristic, but compare for image differences instead, and choose the path that is most closely resemble both images. Since I already separate registration and stitching to different functions, this is trivially implemented in the stitching function.

Image demo on next page.

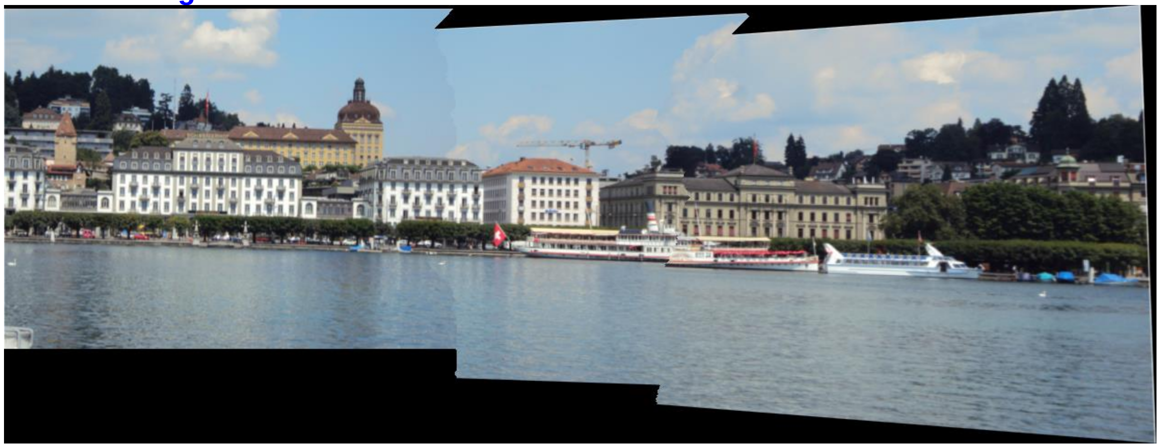
Simple copy-paste



Average

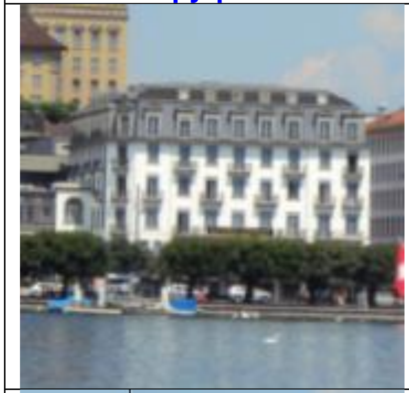
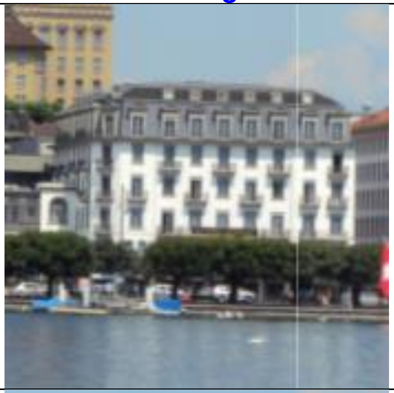
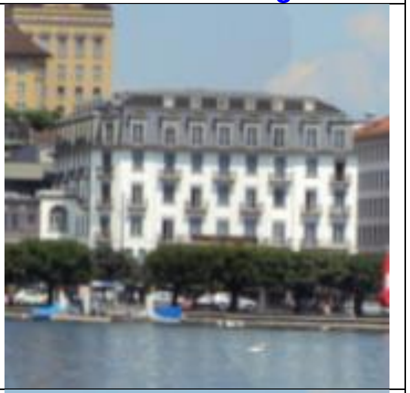
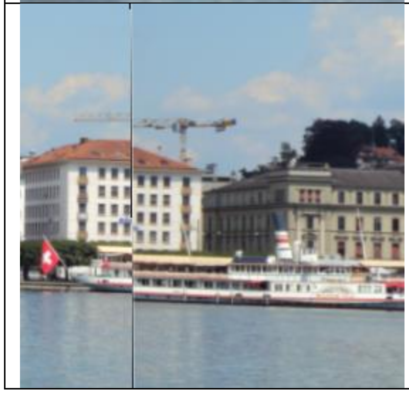
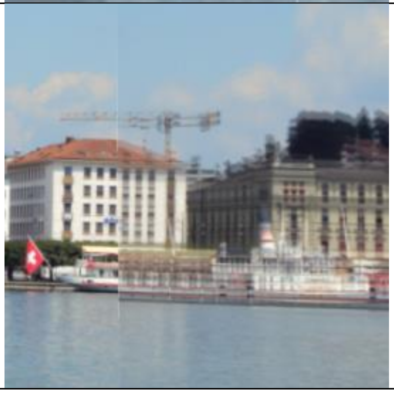
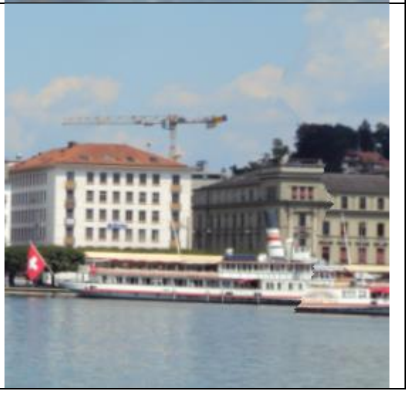


Seam carving



Close-up comparisons on next page.

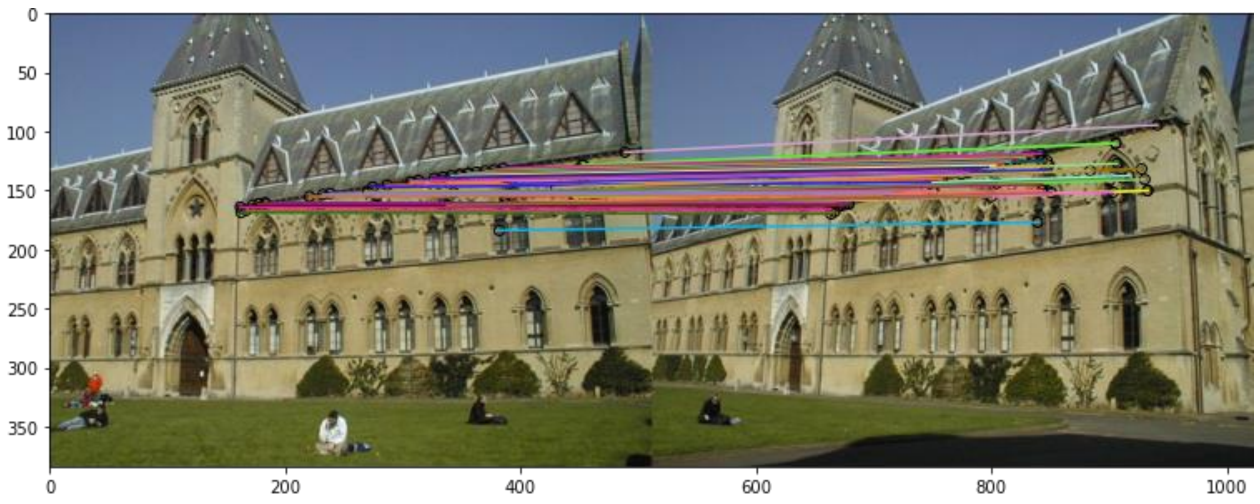
Here are close-up view of overlapped regions. First row shows best alignments, all three methods perform good enough (copy-paste sort of cheat, it is the same image, so it won't show any discrepancies), second row shows worst case scenario, where misalignment is not fixable (moving object), copy-paste has visible cuts, average makes things worse, seam carving is the only method that consider surroundings and give a passable solution.

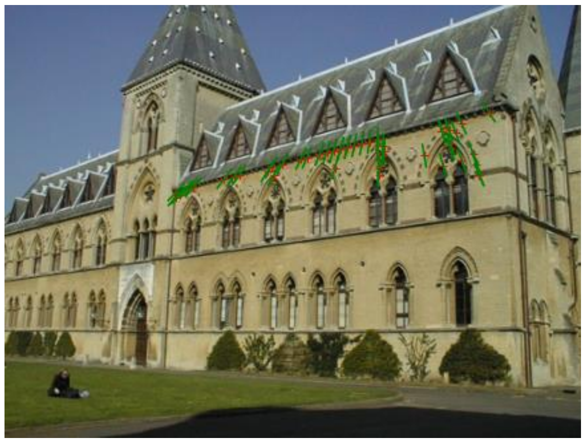
Copy-paste	**Average**	**Seam carving**
		
		

Part 2

Estimate fundamental matrices without ground-truth

Using the same script from part 1, we can estimate matches that closely satisfy the homography, I think this is the reason that it yields a 0 residuals. All parameters are the same as part 1.



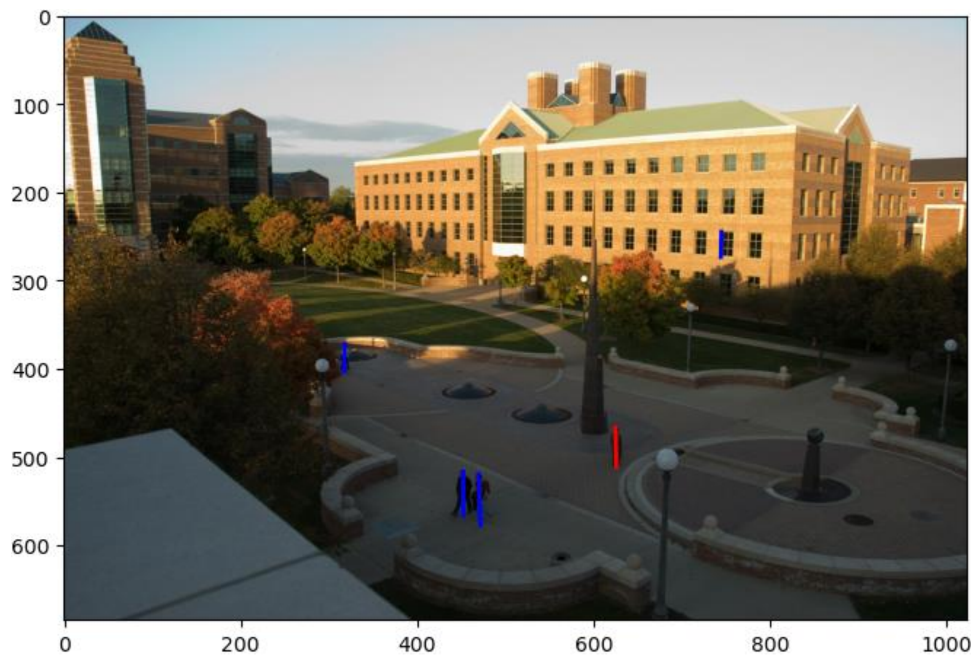


Inliers 50
Average residual 0.242612
Residual (point-epipolar line) 0 px

Part 3

Perform additional measurements

Using the following line segments (blue) relative to the person (red) to do height estimation. Assuming the reference height is still 5 ft 6 in (1.6764 m).

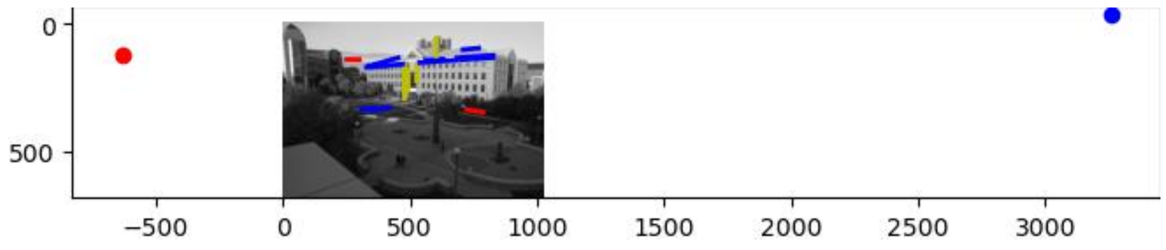


(Person from left to right are numbered 1 to 3.)

Person 1 1.85 m; **Person 2** 1.52 m; **Person 3** 1.77 m; **Window** 5.90 m

Attempt to fit lines to the image and estimate VP automagically

Since the problem statement only requests *attempt to fit*, I piece together a automatic detection pipeline that pick out edges and attempt to estimate VPs. However, this relies on k-mean clustering only, without using any additional image info, the result is not stable. If you would like to play around with my source code, please run it more than once, I can hit an acceptable result about 1 in 8.



Detected VPs **left** (-633.9, 122.3), **vertical** (inf, inf), **right** (3262.3, -34.2)

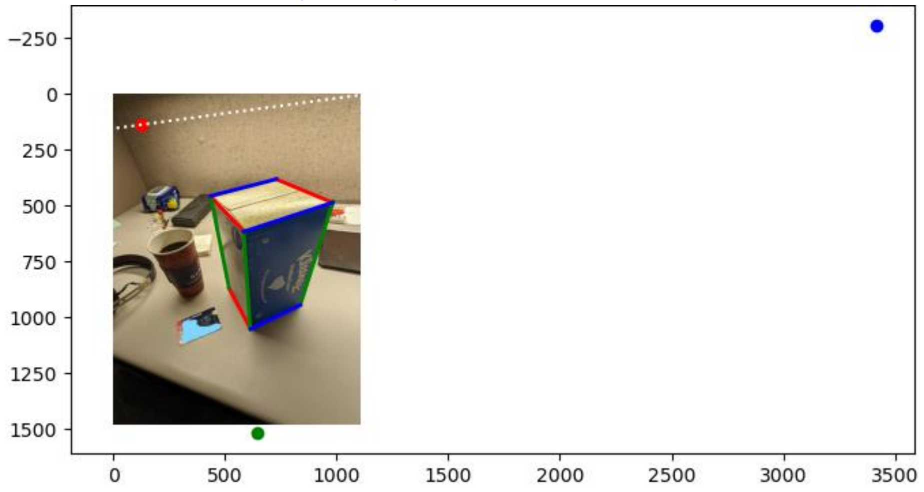
Detection pipeline outline

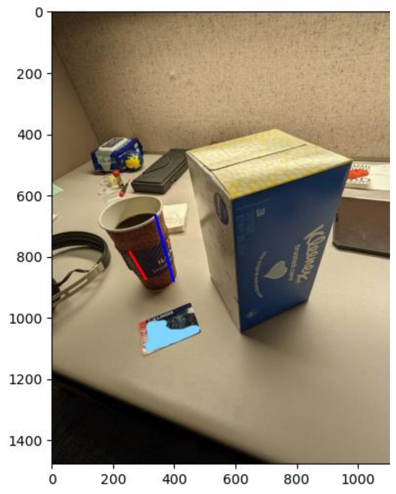
1. Use canny edge detector to extract edge information from the CSL building. I uses a mask to restrict the detector only focus on the building. The detector work with a slightly blurred image, sigma is set to 1.

2. I use probabilistic Hough transform to extract most probable line endpoints (using `probabilistic_hough_line` from `scikit-image`). Detection threshold set to 1, and only accept line length above 50, with maximum gap between lines lowered to 1 pixel. These parameters are empirically determined.
3. These detected endpoints are crossed to find their line equations in $ax + by + c = 0$ format. (a, b) are used as feature vectors in k-means. These features are first normalized by `scipy.cluster.vq.whiten` and then fed into `scipy.cluster.vq.kmeans`, with hard-coded 3 centroid (since we know we *want* at most 3 VPs).
4. Categorize each lines by their proximity to centroids, and we can go through the `get_vanishing_point` function used in baseline implementation. I uses `np.cross` with broacasting and select only upper triangle by `np.triu_indices` to accommodate different number of input lines. VP estimations from combination of lines are later average directly.

Find or take other images with 3 visible vanishing points

I use a Kleenex as the reference object to get scene with 3 VPs.





Sleeve
(reference) 64 mm

Cup height
(estimate) 262 mm; (actual) 150 mm