TTIC 31040: Introduction to Computer Vision
Spring 2021

Problem Set #5

Out: May 18, 2021

Due: May 26, 11:59pm
Instructions: Turn in a write-up of your solution (as markdown cells) along with the code and the results as an iPython (Jupyter) notebook, including any figures or tables you generate. Please name your notebook the following:
`firstname-lastname-ps5.ipynb` and upload it on Canvas. Make sure the notebook you are submitting has been fully run, so that all the results are displayed in it.
Collaboration policy: it is OK (and encouraged) to discuss the problem set and any ideas for solving it with other students. However, in the end you must write your solution on your own. This includes writing any code and running any experiments.

1 Structure from motion

As we saw in class, it is possible to use multi-view geometry to peform 3D reconstruction with image data only. There are restricted settings (i.e. stereo cameras) where this is quite straightforward, but in the more general setting (where images may have been taken at any viewing angle, such as a moving smartphone camera) there are a number of steps required to recover scene structure by performing "structure from motion" (SfM). Specifically:

1. **Camera calibration**: We need the intrinsics matrix $\mathbf{K}$ associated with every image in our dataset.
2. **Feature matching**: We need to detect feature points in every image, and match them, selecting which views overlap according to some heuristic ("view selection").
3. **Fundamental matrix**: We need to compute the fundamental (or rather, essential) matrices $\mathbf{F}_i$ associated with *motion* between frames. This step is followed by estimating the transformation given as $\mathbf{R}_i, \mathbf{t}_i$.
4. **Triangulation**: Given your current list of 2D correspondences and estimate of camera intrinsics and extrinsics, triangulate a 3D point cloud.
5. **Bundle adjustment**: As we add cameras and update scene structure, recompute camera intrinsics and extrinsics and structure based on reprojection error metrics.

In this problem set, we will focus on steps $2, 3$ and 4, with references to other parts of the pipeline.

Problem 1 [10 points]
Load and visualize *merton1.png* and *merton2.png*. In a standard SfM setup, these two images might be picked by a view selection procedure out of a much larger image set. Use your favorite descriptor (we recommend SIFT) and extract keypoints and features. Match the features for these two images with your favorite matcher (brute force is fine). This step is very similar to what we did for panorama stitching, as we again want to find a geometric relationship between two images, though this time we will estimate a fundamental matrix rather than a 2D homography.

End of problem 1

Figure 1: These two images were taken with very similar positions and the same camera, making feature matching easier. In SfM images often come from cameras with different intrinsics taken at significantly different angles.

Problem 2 **[30 points]**

We will use the correspondences from **Problem 2** to estimate the fundamental matrix. Generally, this is done inside a RANSAC loop (you are free to implement a more robust RANSAC-based procedure if you like), but for simplicity we'll estimate it in one go, using the most confident matches from your procedure above (remember that the 8-point algorithm can take more than eight points, but if you have thousands of noisy matches your solution could be quite poor). To measure the quality of our fundamental matrix estimation procedure, we use a quantitative metric known as *Sampson's distance* (given as `sampson_distance` in the code).

For a given correspondence $(\mathbf{x_1}, \mathbf{x_2})$, the fundamental matrix F is supposed to satisfy $\mathbf{x_1}^T F \mathbf{x_2} = 0$. Given noise in the correspondences, we want to compute a metric that (approximately) measures the distance in pixels between our point correspondence and estimated fundamental matrix, and the idea correspondence that satisfies the condition on F exactly. We measure:

$$E(\mathbf{x_1}, \mathbf{x_2}, F) = \frac{(\mathbf{x_1}^T F \mathbf{x_2})^2}{(F\mathbf{x_1})_1^2 + F(\mathbf{x_1})_2^2 + (F\mathbf{x_2})_1^2 + (F\mathbf{x_2}))_2^2}$$

If the median Sampson error is less than 10 pixels, your fundamental matrix estimation procedure performed reasonably well, but modern robust estimation techniques (along with fancier descriptors and matching procedures) can push this error into the subpixel range. Try the following:

- Implement the 8-point algorithm and estimate a matrix $\tilde{F}$.
- Implement the normalized variant of the 8-point algorithm (hint: you should be able to call the function you already wrote) and estimate $\hat{F}$.
- Compare the Sampson error for $\tilde{F}$ and $\hat{F}$. Do you see an improvement from normalizing?

End of problem 2

Problem 3 **[10 points]**

Given a point in one image, the fundamental matrix we estimated defines a line in the second image corresponding to all possible pixel correspondences with that particular point in the first image. This line is

called the **epipolar line**. Algebraically, the $(\mathbf{x}_1, \mathbf{l}_2)$ correspondences comes from:

$$\mathbf{x}_1^T F \mathbf{x}_2 = \mathbf{l}_2^T \mathbf{x}_2 = 0$$

The equation $\mathbf{l}_2^T \mathbf{x}_2 = 0$ determines a line with all points $\mathbf{x}_2$ in the first image satisfying the equation belonging to that line, thus a corresponding point $\mathbf{x}_1$ must lie on that line. For depth estimation algorithms, the fundamental matrix helps restrict the search for correspondences. For our purposes, we already have correspondences from SIFT/ORB, but if we were trying to **dense**, per-pixel depth estimation, this could be a useful prior.

In class we also learned about **epipoles**, the image point corresponding to the projection of the other camera center. All epipolar lines intersect at the epipole, thus $F\mathbf{e}_2 = 0$ and $\mathbf{e}_1^T F = 0$. The epipoles are thus in the null space of F and F^T. For your visualization:

- Using your estimated F, use the helper function `plot_epipolar_line` and for some set of keypoints in image *merton2.png*, plot the corresponding epilines in *merton1.png*.
- Compute $\mathbf{e}_1$ and $\mathbf{e}_2$, plot one of them (and the corresponding epilines for that image) so that you can see that the epipole lies at the intersection of the epilines.

End of problem 3

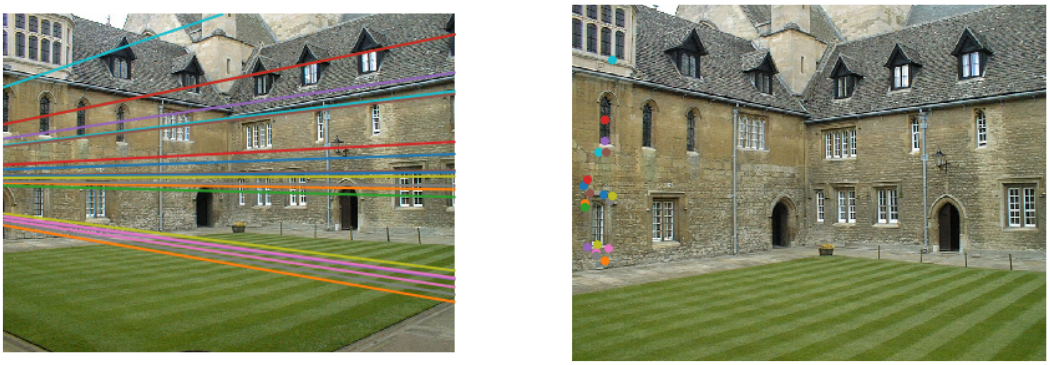

Figure 2: The corresponding pixel for each pixel in the right image lies on the same-colored line in the left image.

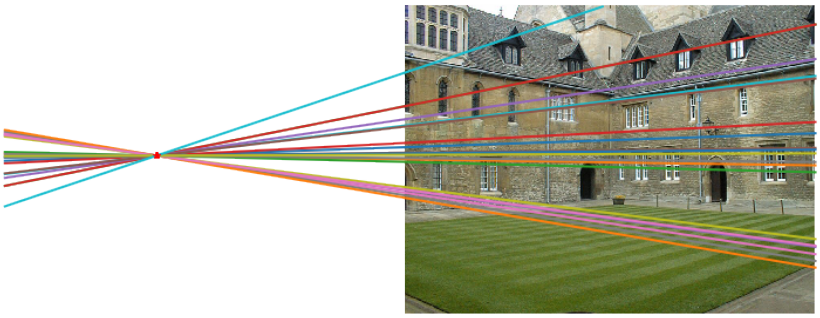

Figure 3: Epipolar lines all intersect at a point corresponding to the projection of the other camera's center.

Problem 4 **[10 points]**

Recall the projective multi-camera geometry–for a 3D point $\mathbf{X}$, its projection $\mathbf{x}$ in camera 1 is given by $\lambda\mathbf{x} = P_1\mathbf{X}$, where P_1 is a projection matrix that encapsulates the intrinsics (describing the camera parameters) and extrinsics (describing the position of the camera in the world). In the structure from motion problem, we must ultimately compute for each camera a P_i, where $P_i = K_i[R_i|\mathbf{t}_i]$ for intrinsics K_i and extrinsics $(R_i, \mathbf{t}_i)$.

These parameters are generally obtained through robust, iterative estimation, and updated through a procedure called *bundle adjustment*. Given that we have only scratched the surface of geometric estimation in this problem so far, it would take many more steps (and likely more images) to be able to estimate a $\tilde{P}_1$ and $\tilde{P}_2$ we could be proud of. Thus, we have provided you with the projection matrices in *data/P1.txt* and *data/P2.txt*. In this problem we will solve the **triangulation** step, obtaining a point cloud from estimated 2D correspondences given the projection matrices.

Specifically, for a given 3D point $\mathbf{X}$ visible in both cameras, it will have projections $\mathbf{x_1}$ and $\mathbf{x_2}$ in images 1 and 2, respectively. Thus for unknown $\mathbf{X}$ we know that $\lambda\mathbf{x}_1 = P_1\mathbf{X}$ and $\lambda\mathbf{x}_2 = P_2\mathbf{X}$. This can be rewritten as the linear system:

$$\begin{bmatrix} P_1 & -\mathbf{x}_1 & 0 \\ P_2 & 0 & -\mathbf{x}_2 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = 0$$

Use SVD to get a least squares estimate of $\mathbf{X}$ and write a function `triangulate_point` that takes your SIFT/ORB correspondences and the projection matrices and outputs a point cloud. Experiment with different numbers of correspondences (you could include even fairly low-confidence matches from SIFT), does the point cloud you get roughly match the structure you would infer from looking at the images?

Optionally, you can also color the pointcloud (so obtain RGBXYZ rather than just XYZ points) by saving the color for each corresponding pixel from one of the images and then passing that color vector to the meshplot plotting code.

End of problem 4

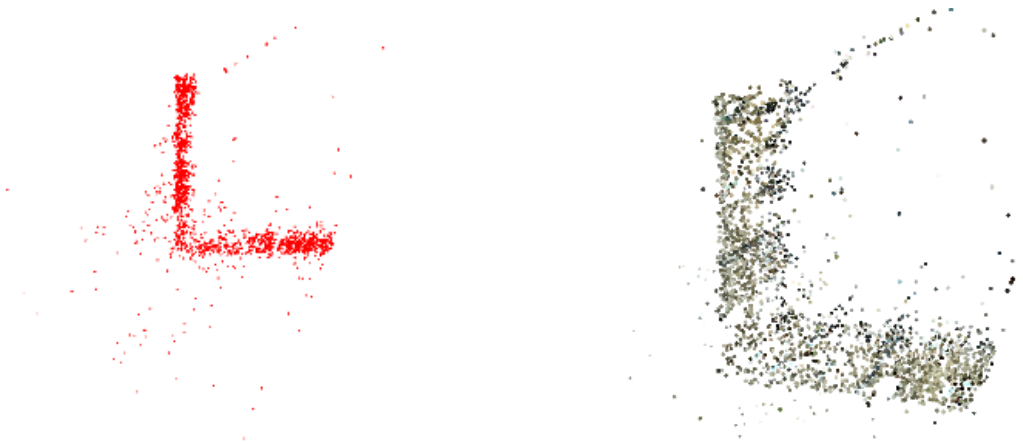

Figure 4: The triangulated pointcloud will be sparser (but less noisy) or denser (but potentially much noisier) depending on the number of correspondences you pick. On the left is a pointcloud using the default colormap for *meshplot*, on the right is a pointcloud with colors obtained from one of the images. Note that the level of zoom and viewpoint is somewhat different for these two images.

2 Image Segmentation

For many applications, the raw pixels of an image are a heavily redundant source of information, leading to the development of "superpixels", methods that group the pixels into perceptually meaningful regions that respect object contours.
Superpixels reduce the complexity of downstream tasks as you can operate on a reasonable number of image regions rather than millions of pixels, and have found their use in applications such as semantic segmentation, depth estimation, human pose estimation, and object localization.
In this problem, you will implement a simple superpixel algorithm called Simple Linear Iterative Clustering (SLIC) that clusters pixels in the five-dimensional color and pixel coordinate space (e.g., r, g, b, x, y). Traditionally, this is not done in RGB color space but instead in CIELAB color space (described in **Lecture 7**), a color space that separates brightness from color and is more perceptually uniform, thus performing better for tasks that require comparing color values (thus we will actually look at the space (l, a, b, x, y)).
The procedure can be described as follows:

- The algorithm starts with a collection of K cluster centers (i.e. "superpixels") initialized at an equally sampled grid on the image. For an image with N pixels, the approximate size of each of equally sized superpixels is N/K pixels. Thus the interval between each center at initialization is $S = \sqrt{N/K}$.
- As an initialization, choose K superpixel cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]$ with $k = \overline{1, K}$ at regular grid intervals S. For each cluster, you define a localized window $2S \times 2S$ centered at the cluster center. Then, you check whether the pixel within the $2S \times 2S$ local window should be assigned to the cluster center or not (by comparing the distance in $5D$ space to the cluster center).
- Once you loop through all the clusters, you can update the cluster center by averaging over the cluster members. Iterate the pixel-to-cluster assignment process until convergence or maximum iterations are reached.

Problem 5 **[10 points]**
Since we are working in a 5D space that describes both color and distance, we need something beyond a simple Euclidean metric on 5-vectors. The authors of SLIC propose the following:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{1}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{2}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy} \tag{3}$$

where m is a hyperparameter that measures the "compactness" of a superpixel. What happens when we increase m? Implement the distance function.

End of problem 5

Problem 6 **[30 points]**

Implement SLIC and try a variety of values for K (number of clusters) and m (compactness). Describe your observations. Note that for large values of K, your implementation may run very slowly. Run your implementation on the provided image (*elon.png*) or on any images of your choosing.

End of problem 6

Figure 5: An image segmented with $K = 300$ clusters, where each cluster pixels are replaced with the average pixel with the center pixel marked. Here, the clusters are replaced by the average pixel value, another visualization approach is to mark the boundaries of clusters.