CSci 5563 Assignment 4

Luis Guzman

Sunday April 11, 2021

This assignment consists of a full implementation of Structure from Motion (SfM), which reconstructs a scene in 3D using images taken from multiple camera poses. The SfM algorithm can be broken down into five main parts: BuildFeatureTrack, EstimateCameraPose, (Nonlinear) Perspective-n-Point, (Nonlinear) Triangulation, and Bundle Adjustment. The main algorithm and the BuildFeatureTrack step are shown on the right.

The goal of BuildFeatureTrack is to match keypoints between each pair of images. Keypoints are extracted from every image using the SIFT implementation in OpenCV. Then, each keypoint in image i is compared against the keypoints in image j to find a match. The ratio test is used to ensure that only robust matches are considered. The matched points are then used to estimate the Essential Matrix of the two cameras. The Essential Matrix maps a 2D pixel coordinate from one image to the other such that $\mathbf{x^T E x'} = 0$. The Essential Matrix can be estimated via RANSAC and the 8-point algorithm to filter out any outliers:

Algorithm 1 Structure from Motion

```
Build feature track                                   ▷ BuildFeatureTrack
Estimate first two camera poses                       ▷ EstimateCameraPose
Initialize pose set P
for i = 2, ⋯, N-1 do
    Estimate new camera pose                          ▷ PnP, PnP_nl
    P = P ∪ P_i
    for j < i do
        Find new points to reconstruct                ▷ FindMissingReconstruction
        Triangulate point                             ▷ Triangulation, Triangulation_nl
        Filter out point based on cheirality          ▷ EvaluateCheirality
        Update 3D point
    end for
    Run bundle adjustment                             ▷ RunBundleAdjustment
end for
```

Algorithm 2 BuildFeatureTrack

```
for i = 0, ⋯, $N-1$ do
    Extract SIFT descriptor of the $i^{\text{th}}$ image, Im[i]
end for
for i = 0, ⋯, $N-1$ do
    Initialize track_i = -$1^{N\times F\times 2}$
    for j = $i+1, \cdots, N-1$ do
        Match features between the $i^{\text{th}}$ and $j^{\text{th}}$ images      ▷ MatchSIFT
        Normalize coordinate by multiplying the inverse of intrinsics.
        Find inliner matches using essential matrix   ▷ EstimateE_RANSAC
        Update track_i using the inlier matches.
    end for
    Remove features in track_i that have not been matched for $i+1, \cdots, N$.
    track = track ∪ track_i
end for
```

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ & & & & \vdots & & & & \\ x_8x'_8 & x_8y'_8 & x_8 & y_8x'_8 & y_8y'_8 & y_8 & x'_8 & y'_8 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0 \qquad \mathbf{E} = \mathbf{UDV}^{\mathsf{T}} = \mathbf{U}\begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix}\mathbf{V}^{\mathsf{T}}$$

The first equation is solved by using the SVD to find the null space, and then the second equation performs SVD cleanup on the result. After removing outlier points, the matched points are then stored in `track`. `track` is a tensor of shape $(N \times F \times 2)$ where N is the number of cameras and F is the total number of matched features. If a feature f appears in image n, the `track[n,f,:]` stores that feature's normalized pixel coordinate. Otherwise, -1 is stored instead to indicate an invalid point.

Once `track` has been built, we can use EstimateCameraPose to find the camera poses of the first and second cameras in the sequence. The first camera is assumed to be an identity rotation and zero translation. The camera pose of the second camera can be extracted from the essential matrix as follows:

$$\mathbf{t} = \pm\mathrm{null}(\mathbf{E}^{\mathsf{T}}) \qquad \mathbf{R} = \mathbf{U}\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{V}^{\mathsf{T}}, \text{ or } \mathbf{U}\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{V}^{\mathsf{T}}$$

This results in four possible camera configurations from a single essential matrix. Before determining the ideal camera pose, we first triangulate every keypoint in the image by using epipolar geometry to determine depth. The equation is

$$\begin{bmatrix} \begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix}_\times \mathbf{P}_{\text{bob}} \\ \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix}_\times \mathbf{P}_{\text{alice}} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{0}$$

where $\mathbf{u}$ and $\mathbf{v}$ are pixel coordinates, $\mathbf{P_i} = [\mathbf{R}\ \mathbf{t}]$ is the projection matrix for the ith image, and $\mathbf{X}$ is the 3D point we'd like to solve for. The u and v matrices are in skew-symmetric form. Because each skew symmetric matrix is rank 2, I remove the last row of each one to form a well-conditioned problem. This equation can be solved using the SVD to find the null space.

The optimal camera pose is determined by the Cheirality condition $\mathbf{r_3}(\mathbf{X} - \mathbf{C}) > 0$ where $\mathbf{C} = -\mathbf{R}\mathbf{t}$ and $\mathbf{r_3}$ is the last row of $\mathbf{R}$. This expression counts the number of points that appear in front of both cameras. Figure 1 visualizes these four different poses.

In the next step, I iteratively add new images to the reconstruction and ensure that all new and old points are consistent. The PnP algorithm estimates a new camera pose given 3D to 2D correspondences. Because we already have triangulated points for the first two images, many of those same points can be used to localize the camera in the third image. The PnP algorithm solves for a camera as follows:

```
Algorithm 3 EstimateCameraPose
1: Compute essential matrix given tracks                          ▷ EstimateE_RANSAC
2: Estimate four configurations of poses                          ▷ GetCameraPoseFromE
3: for i = 0, 1, 2, 3 do
4:     Triangulate points using for each configuration            ▷ Triangulation
5:     Evaluate cheirality for each configuration                 ▷ EvaluateCheirality
6: end for
```

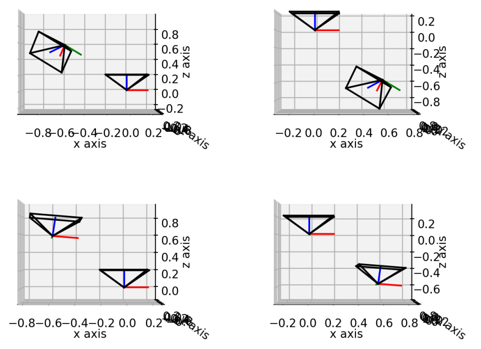


Figure 1: The four poses for a given essential matrix. In this case, pose 1 and 2 had 0 points pass the Cheirality test, pose 3 had 315 points pass the test, and pose 4 had 12 points pass. Pose 3 was selected as the optimal pose.

$$\underbrace{\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & & & & & -u_1^xX & -u_1^xY & -u_1^xZ & -u_1^x \\ & & & & X_1 & Y_1 & Z_1 & 1 & -u_1^yX & -u_1^yY & -u_1^yZ & -u_1^y \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_m & Y_m & Z_m & 1 & & & & & -u_m^xX & -u_m^xY & -u_m^xZ & -u_m^x \\ & & & & X_m & Y_m & Z_m & 1 & -u_m^yX & -u_m^yY & -u_m^yZ & -u_m^y \end{bmatrix}}_{2m\text{x}12} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{K}^{-1}\begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix} = \mathbf{U}\begin{bmatrix} d_{11} & & \\ & d_{22} & \\ & & d_{33} \end{bmatrix}\mathbf{V}^\mathsf{T}$$

$$\gamma \approx d_{11}$$

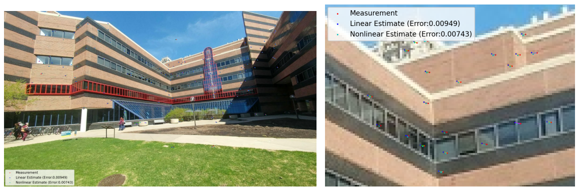
$$\mathbf{R} = \mathbf{U}\mathbf{V}^\mathsf{T} \quad \text{: SVD cleanup}$$

$$\mathbf{t} = \frac{\mathbf{K}^{-1}\mathbf{p}_4}{d_{11}} \quad \text{: Translation and scale recovery}$$

Note that because pixel coordinates are already normalized, the K matrix is not needed here. The above equations result in a linear approximation for the camera pose. A better estimate can be found by using nonlinear optimization to minimize the reprojection error $\underset{\mathbf{p}}{\text{minimize}} \sum_i^n \|f(\mathbf{p}, \mathbf{X}_i) - \mathbf{b}_i\|^2,$. The algorithm for minimizing this error is given in the "Nonlinear Camera Pose Refinement" in Figure 3. The objective is minimized using the Levenberg-Marquardt method:

$$\mathbf{p} = \mathbf{p} + \Delta\mathbf{p}, \quad \text{where } \Delta\mathbf{p} = \left(\sum_i^n \frac{\partial f_i}{\partial \mathbf{p}}^\mathsf{T} \frac{\partial f_i}{\partial \mathbf{p}} + \lambda\mathbf{I}\right)^{-1} \sum_i^n \frac{\partial f_i}{\partial \mathbf{p}}^\mathsf{T} (\mathbf{b}_i - f_i),$$

$$f(\mathbf{p}) = \begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} \longrightarrow \frac{\partial f(\mathbf{p})}{\partial \mathbf{p}} = \frac{\partial}{\partial \mathbf{p}}\begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} = \begin{bmatrix} \dfrac{w\frac{\partial u}{\partial \mathbf{p}} - u\frac{\partial w}{\partial \mathbf{p}}}{w^2} \\ \dfrac{v\frac{\partial u}{\partial \mathbf{p}} - v\frac{\partial w}{\partial \mathbf{p}}}{w^2} \end{bmatrix}$$



Measurement
Linear Estimate (Error:0.00949)
Nonlinear Estimate (Error:0.00743)

Figure 2: Results of the PnP step. The nonlinear refinement decreased reprojection error from 0.0095 to 0.0074.

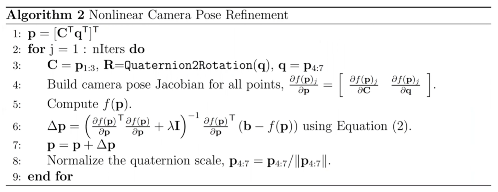
Once the new camera pose is known, I use FindMissingReconstruction to determine if there exist any new points that haven't been reconstructed yet. For all these new points, I run Triangulation on them for every previous camera. The 3D points can also be improved with nonlinear estimation, so I use the same method to minimize reprojection error for the reconstruction

$$\underset{\mathbf{X}_j}{\text{minimize}} \sum_{k=1}^{2} \|f(\mathbf{p}_k, \mathbf{X}_j) - \mathbf{b}_{k,j}\|^2,$$

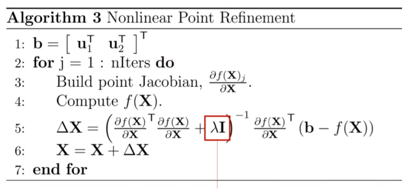
$$\mathbf{X}_j = \mathbf{X}_j + \Delta\mathbf{X}_j \quad \text{where } \Delta\mathbf{X}_j = \sum_{k=1}^{2} \left(\frac{\partial f_{k,j}}{\partial \mathbf{X}}^\mathsf{T} \frac{\partial f_{k,j}}{\partial \mathbf{X}} + \lambda\mathbf{I}\right)^{-1} \frac{\partial f_{k,j}}{\partial \mathbf{X}}^\mathsf{T} (\mathbf{b}_{k,j} - f_{k,j}),$$

$$\frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} = \frac{\partial}{\partial \mathbf{X}}\begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} = \begin{bmatrix} \dfrac{w\frac{\partial u}{\partial \mathbf{X}} - u\frac{\partial w}{\partial \mathbf{X}}}{w^2} \\ \dfrac{v\frac{\partial u}{\partial \mathbf{X}} - v\frac{\partial w}{\partial \mathbf{X}}}{w^2} \end{bmatrix}$$

Lastly, I filter the newly reconstructed points using the same Cheirality condition as before. This ensures that the points are a correct match before I add them to the permanent set of 3D points. I found that even with the Cheirality condition, I had some mismatched points entering the $\mathbf{X}$ list. To fix this issue, I added an additional filter that required $||\mathbf{u} - \mathbf{PX}|| < \epsilon$, exactly the same condition that I enforced in my RANSAC functions.



```
Algorithm 2 Nonlinear Camera Pose Refinement
1: $\mathbf{p} = [\mathbf{C}^\top \mathbf{q}^\top]^\top$
2: for j = 1 : nIters do
3:     $\mathbf{C} = \mathbf{p}_{1:3}$, $\mathbf{R}$=Quaternion2Rotation($\mathbf{q}$), $\mathbf{q} = \mathbf{p}_{4:7}$
4:     Build camera pose Jacobian for all points, $\frac{\partial f(\mathbf{p})_j}{\partial \mathbf{p}} = \left[ \frac{\partial f(\mathbf{p})_j}{\partial \mathbf{C}} \quad \frac{\partial f(\mathbf{p})_j}{\partial \mathbf{q}} \right]$.
5:     Compute $f(\mathbf{p})$.
6:     $\Delta\mathbf{p} = \left(\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}^\top \frac{\partial f(\mathbf{p})}{\partial \mathbf{p}} + \lambda\mathbf{I}\right)^{-1} \frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}^\top (\mathbf{b} - f(\mathbf{p}))$ using Equation (2).
7:     $\mathbf{p} = \mathbf{p} + \Delta\mathbf{p}$
8:     Normalize the quaternion scale, $\mathbf{p}_{4:7} = \mathbf{p}_{4:7}/\|\mathbf{p}_{4:7}\|$.
9: end for
```



```
Algorithm 3 Nonlinear Point Refinement
1: $\mathbf{b} = \left[ \mathbf{u}_1^\top \quad \mathbf{u}_2^\top \right]^\top$
2: for j = 1 : nIters do
3:     Build point Jacobian, $\frac{\partial f(\mathbf{X})_j}{\partial \mathbf{X}}$.
4:     Compute $f(\mathbf{X})$.
5:     $\Delta\mathbf{X} = \left(\frac{\partial f(\mathbf{X})}{\partial \mathbf{X}}^\top \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} + \boxed{\lambda\mathbf{I}}\right)^{-1} \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}}^\top (\mathbf{b} - f(\mathbf{X}))$
6:     $\mathbf{X} = \mathbf{X} + \Delta\mathbf{X}$
7: end for
```

Figure 3: Nonlinear estimation algorithms

The last step is to run Bundle Adjustment, which minimizes the reporojection error for all camera poses and reconstructed points all at once. For this portion, I used `scipy.optimize.least squares` to simplify the optimization. In SetupBundleAdjustment, I create an optimization variable, which consists of all camera poses and 3D points. I also create a sparsity matrix, which indicates to Scipy which elements of the Jacobian are equal to zero. As can be seen in Figure 4, the Jacobian is highly sparse, which simplifies the optimization. Figure 5 shows the results of the bundle adjustment step.

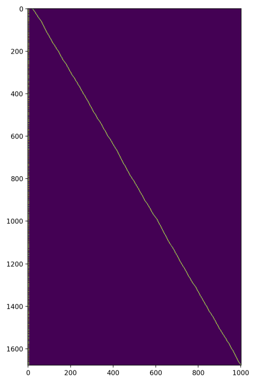
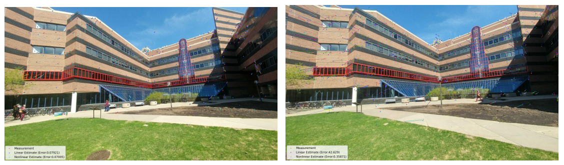


Figure 4: The Sparsity matrix of the Bundle Adjustment Jacobian. Non zero entries exist in the first few columns, which correspond to the pose Jacobian, and along the diagonal, which correspond to the point Jacobian



Measurement
Linear Estimate (Error:0.07921)
Nonlinear Estimate (Error:0.07005)
Measurement
Linear Estimate (Error:42.629)
Nonlinear Estimate (Error:0.35871)

Figure 5: Results of the bundle adjustment step. The top figure shows a typical case where the reprojection error goes from 0.079 to 0.070. The bottom image shows a case where there was large error due to PnP_RANSAC and Triangulation. Bundle Adustment fixed the incorrect points and decreased the error from 42.6 to 0.359.

My final results are shown in Figure 6. The right wall of Keller hall is clearly visible, and there are clusters of points where the left and front faces of the building would be. In the front view, you can see that all five cameras have been localized correctly.

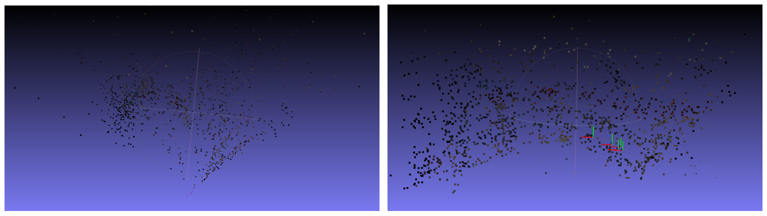


Figure 6: The top view (left) and front view (right) of my final 3D reconstruction.

I believe there are still some improvements to be made on my implementation. For example, the RANSAC method in EstimateCameraPose does not always give an accurate camera pose, so the program must sometimes be restarted to fix the issue. I attempted to adjust the RANSAC parameters to fix this, but I could not seem to find an entirely stable combination. Also, my 3D reconstruction isn't perfect–I'd like to see more defined walls and edges. I believe this is due to a combination of errors from PnP and Triangulation that bundle adjustment could not account for. If given more time, I'd like to review these functions for errors.