Abstract

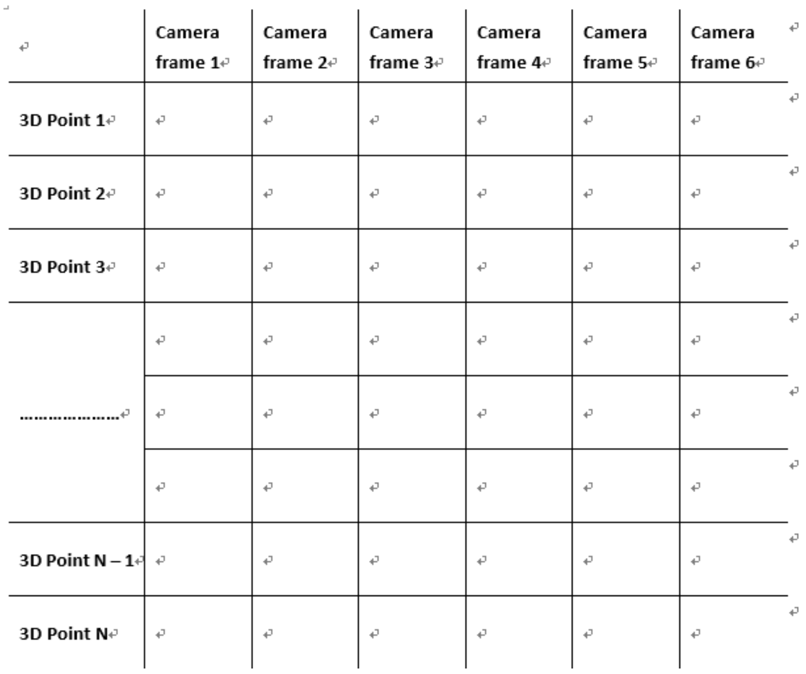
This report mainly contains below several sections:

1. The Algorithm Implementation mainly introduces the designed data structure, the pipeline of structure from motion and Jacobian creation and implementation.
2. Final Results includes all generated figures such as 3D point cloud map, camera pose and reprojection error.
3. Problems and possible Improvements.

1 Algorithm Implementation

1.1 Data Structure Design

Well-stored data will do a lot of favor to the accuracy and speed of algorithm. In this project, one difficulty in this project is how to handle data properly. It's necessary to figure out the suitable way to keep tracking of 3D and 2D correspondences during the pipeline. What I am doing is creating several matrices to store 3D point cloud(Point Cloud), 2D correspondences(Mu/Mv) and visibility relationship(VisM/V).



	Camera frame 1	Camera frame 2	Camera frame 3	Camera frame 4	Camera frame 5	Camera frame 6
3D Point 1						
3D Point 2						
3D Point 3						
…………………						
3D Point N − 1						
3D Point N						

Figure 1: Designed Data Structure

According to the Figure 1, the size of matrix is (the number of 3D points) by (number of camera frames). So it's very convenient for me to use indices in column direction to keep tracking of 3D-2D correspondences, update inliers and remove outliers.

1.2 Structure From Motion Pipeline

The Figure 2 shows the main algorithm that I implement in this project, the whole structure is pretty straightforward and make much sense. My code is based on this pipeline and also modified several parts of it in order to reduce the error in each step as much as possible.

First, I add additional cleanup function after each time I get new estimated 3D point cloud, such as triangulation or bundle adjustment. What I am doing in the cleanup is removing some 3D points that locate behind the camera or very far away to the camera, since I think it will benefit for visualization as well as optimal converging process.

Second one, in the nonlinear PnP, I will update all camera frames so far instead of just current camera frame.

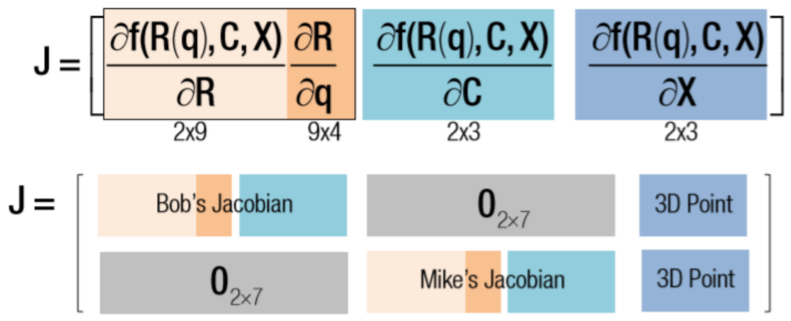
Finally, when introducing new 3D points to the point cloud, I choose to find correspondences between current frame with all previous camera frames instead of the previous one since I think larger 3D point cloud with more dense information will benefit the bundle adjustment and also help estimation in the following camera frames. I add clean and update help function to increase the accuracy and speed of optimal procedure.

```
for all possible pair of images do
    [x1 x2] = GetInliersRANSAC(x1, x2);                          ▷ Reject outlier correspondences.
end for
F = EstimateFundamentalMatrix(x1, x2);                           ▷ Use the first two images.
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset Rset] = ExtractCameraPose(E);
for i = 1 :  4 do
    Xset{i} = LinearTriangulation(K, zeros(3,1), eye(3), Cset{i}, Rset{i}, x1, x2);
end for
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);                ▷ Check the cheirality condition.
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2, X0));
Cset← {C}, Rset← {R}
for $i = 3 : I$ do                      ▷ Register camera and add 3D points for the rest of images
    [Cnew Rnew] = PnPRANSAC(X, x, K);                            ▷ Register the $i^{\text{th}}$ image.
    [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
    Cset ← Cset ∪ Cnew
    Rset ← Rset ∪ Rnew
    Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
    Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2, X0);    ▷ Add 3D points.
    X ← X ∪ Xnew
    V = BuildVisibilityMatrix(traj);                             ▷ Get visibility matrix.
    [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);         ▷ Bundle adjustment.
end for
```

Figure 2: Structure From Motion

1.3 Jacobian Implementation

Since triangulation, PnP and bundle adjustment are all essentially nonlinear problems, therefore, the implementation of Jacobian matrix is quite important and truly improves the efficient of algorithm. Basically, we can implement the analytic and numerical Jacobian matrices for the optimal converge process. Based on my practical experience, the numerical Jacobian is much faster and brings more reasonable results.



$J = \left[\frac{\partial f(R(q),C,X)}{\partial R} \frac{\partial R}{\partial q} \quad \frac{\partial f(R(q),C,X)}{\partial C} \quad \frac{\partial f(R(q),C,X)}{\partial X} \right]$
2x9
9x4
2x3
2x3
J =
Bob's Jacobian
$0_{2\times7}$
3D Point
$0_{2\times7}$
Mike's Jacobian
3D Point

Figure 3: Jacobian Matrix for Bundle Adjustment

The Figure 3 shows the basic structure of Jacobian for bundle adjustment, I truly recommend to implement the Jacobian for bundle adjustment first, and then it will be easier to implement Jacobian for nonlinear triangulation and nonlinear PnP. In addition, using sparse matrix will reduce the memory complexity as well as increase the code speed.

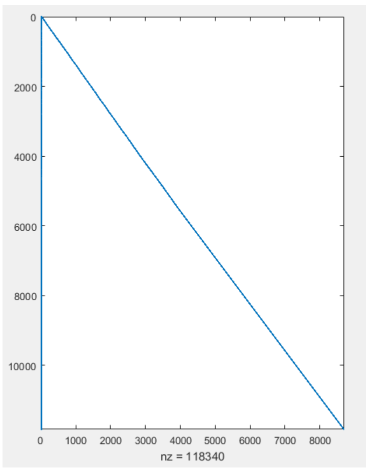


Figure 4: The whole Jacobian Matrix

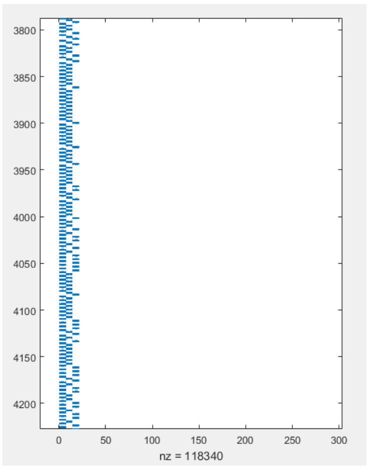


Figure 5: Jacobian Matrix of camera parameters part

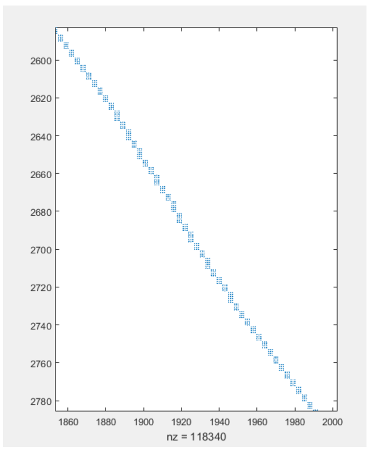


Figure 6: Jacobian Matrix of 3D Point Cloud part

Figure 4 is the whole practical Jacobian matrix I created for the bundle adjustment, Figure 5 and 6 show more details in camera parameters and 3D point cloud sections respectfully. With correct structure of Jacobian matrix, we also need to choose suitable optimization algorithm for achieving convergence. In my case, I use 'fsolve' function in Matlab and select 'trust-region-reflective' optimal approach.

2 Results

This section will contain multiple figures as results of this project to verify the accuracy of my algorithm.

2.1 3D Point Cloud Map Results

Figure 7 shows the point cloud map obtained by all 6 camera frames. Figure 8 to 11 show the 3D point cloud from different angle view.

2.2 Camera Pose Estimation Results

Figure 12 and 13 shows the camera pose estimation for all six camera frames.

2.3 Reprojection Error Results

We assume that the pixel error of reprojection should decrease with moving of stage step. Based on the algorithm pipeline, it's reasonable that we may get somehow high error after PnP, then after triangulation estimation, the reprojection error should decrease, finally, after the bundle adjustment, the reprojection error should be optimized to local minimal value. Figure 14, 15 and 16 show the reprojection error of camera frame 6 along the pipeline. Finally, I can get quite reasonable reprojection error for all 6 camera frames after final bundle adjustment that can verify the accuracy of my algorithm. Figure 17 to 22 show the reprojection error for all 6 images.

3 Problems and Improvements

During the implementation of this project, there existed some problems as well as improvements that benefit the performance.

3.1 Threshold in RANSAC

In the pipeline, there are two functions need RANSAC to filter outliers.
In initial RANSAC part, we use fundamental matrix to dealing with outliers of correspondences

within different camera frames. Higher threshold means more likely to include outliers which is not good for the following estimation. However, if I select small threshold, less features will be obtained, which will influence the bundle adjustment as well. Therefore, threshold in the RANSAC is kind of a trade-off problem.
In the linear PnP step, we need to use 3D-2D correspondences to initialize the pose of current camera. RANSAC can give us the optimal estimation. Unlike the RANSAC in linear triangulation, in the PnP, some high threshold will give me better performance than lower threshold. The reason might be, with more 3D-2D correspondences, the accuracy of linear PnP computation will be improved a lot. However, some quite high threshold is also non reasonable since they will bring more outliers.
For me, I set 2 pixel error when finding 2D correspondences and 6 pixel error in linear PnP since it seems like linear PnP may have higher reprojection error.

3.2 Special Case Handling

In this project, we are given good data sets since each time adding new camera frame, we can find more than 6 3D-2D correspondences to implement PnP. In order to consider some special cases that cannot find 6 3D-2D correspondences, what I am doing is skipping those camera frames that cannot find enough correspondences with current estimated point cloud.
However, filtering out those invalid camera frames makes less sense since there might be possible that the following camera frames will introduce new correspondences and update point cloud such that might be valid for the current camera frame.
Therefore, one improvement is adding for loop at the end of bundle adjustment to traverse those invalid camera frames again with current new point cloud.

3.3 Distribution of Feature Correspondences

Uniformly distributed features will benefit a lot in the bundle adjustment wince those features have higher probability to contain various dense information. In order to make features distribute uniformly, the most common way is called the Adaptive NMS(Non-Maximal Suppression). Practically, We always use corner as good feature since it's eigenvalues are both large. The basic idea of Adaptive NMS is, if a pixel can be regarded as a good feature, it should be the largest one within some certain region. Essentially, for each image frame, we need to figure out its max radius requirement which a corner feature should satisfy in order to become a good one. Then based on our desired output feature points number, we can choose the max radius for each image, eventually we can achieve features to distribute uniformly.

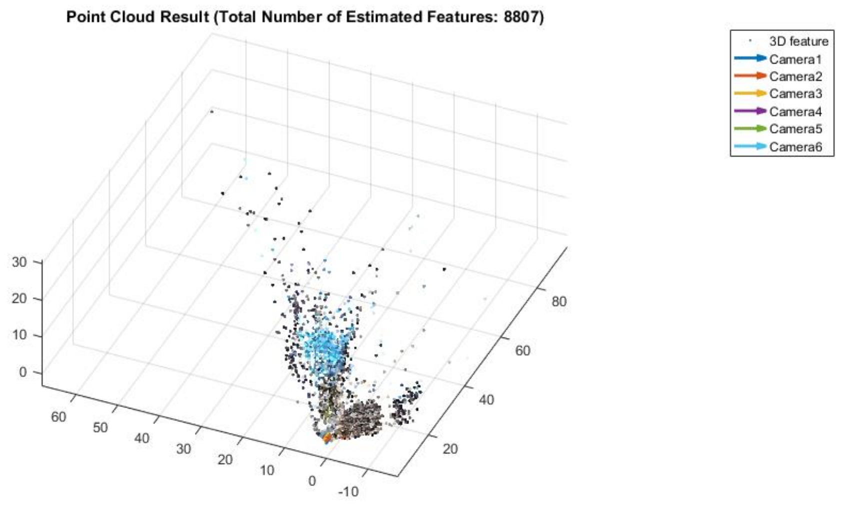


Figure 7: 3D Point Cloud Estimation Map

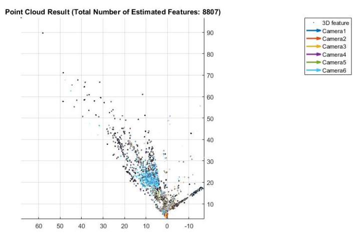


Figure 8: 3D Point Cloud: Top View

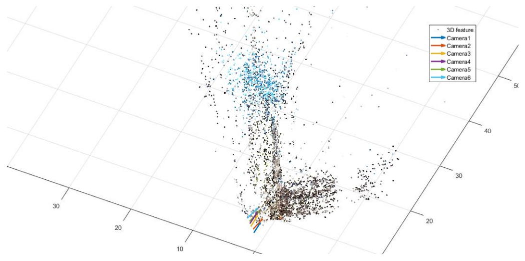


Figure 9: 3D Point Cloud: Close View

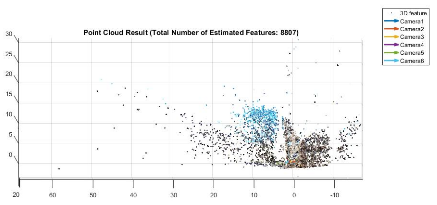


Figure 10: 3D Point Cloud: Side View Angle Left

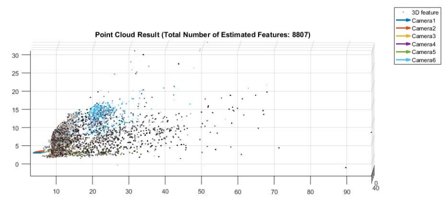


Figure 11: 3D Point Cloud: Side View Angle Right

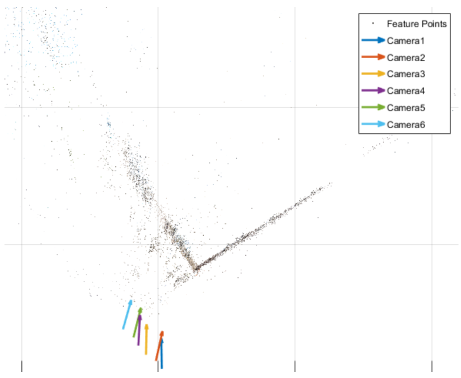


Figure 12: Camera Pose Estimation in RGB Map

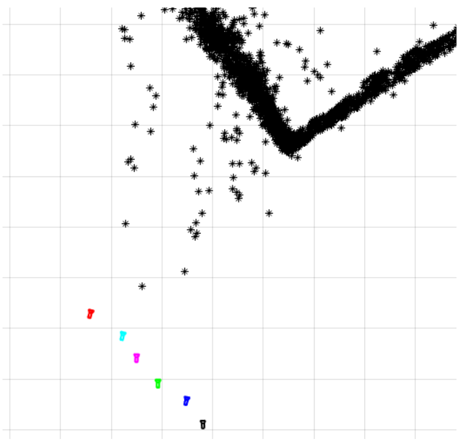
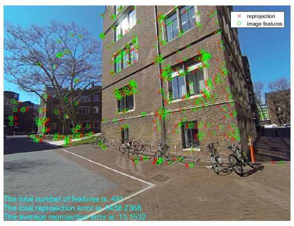
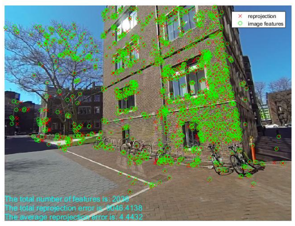


Figure 13: Camera Pose Estimation in Binary Map



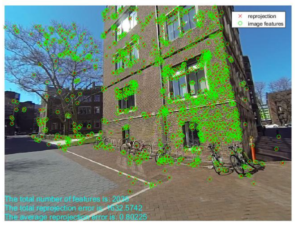
```
The total number of features is: 491
The total reprojection error is: 6458.2368
The average reprojection error is: 13.1532
```

Figure 14: Reprojection Error After linear and nonlinear PnP



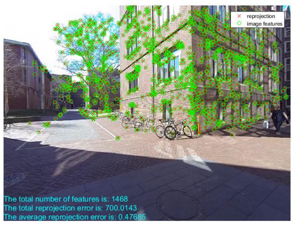
```
The total number of features is: 2036
The total reprojection error is: 9046.4138
The average reprojection error is: 4.4432
```

Figure 15: Reprojection Error After Triangulation



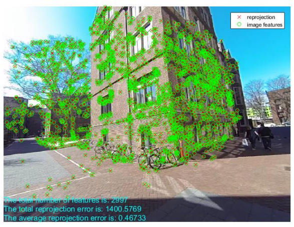
```
The total number of features is: 2036
The total reprojection error is: 1632.5742
The average reprojection error is: 0.80225
```

Figure 16: Reprojection Error After Bundle Adjustment



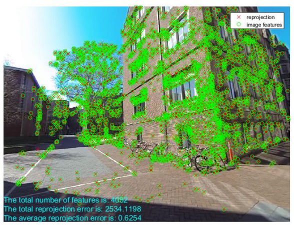
```
The total number of features is: 1468
The total reprojection error is: 700.0143
The average reprojection error is: 0.47685
```

Figure 17: Reprojection Error of Camera Frame 1



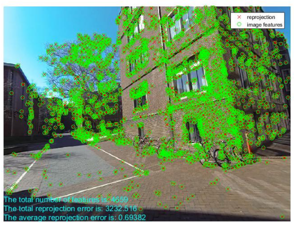
```
The total number of features is: 2997
The total reprojection error is: 1400.5769
The average reprojection error is: 0.46733
```

Figure 18: Reprojection Error of Camera Frame 2



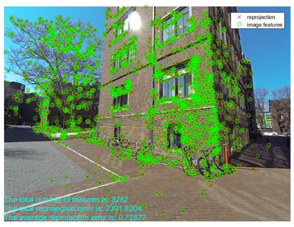
```
The total number of features is: 4052
The total reprojection error is: 2534.1198
The average reprojection error is: 0.6254
```

Figure 19: Reprojection Error of Camera Frame 3



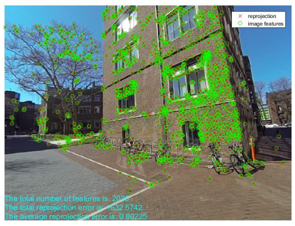
```
The total number of features is: 4659
The total reprojection error is: 3232.516
The average reprojection error is: 0.69382
```

Figure 20: Reprojection Error of Camera Frame 4



```
The total number of features is: 3282
The total reprojection error is: 2391.8204
The average reprojection error is: 0.72877
```

Figure 21: Reprojection Error of Camera Frame 5



```
The total number of features is: 2036
The total reprojection error is: 1632.5742
The average reprojection error is: 0.80225
```

Figure 22: Reprojection Error of Camera Frame 6