



3D Structure from Motion

Group 4
Haojie Huang
Aakash
Arvind
Veera

Introduction

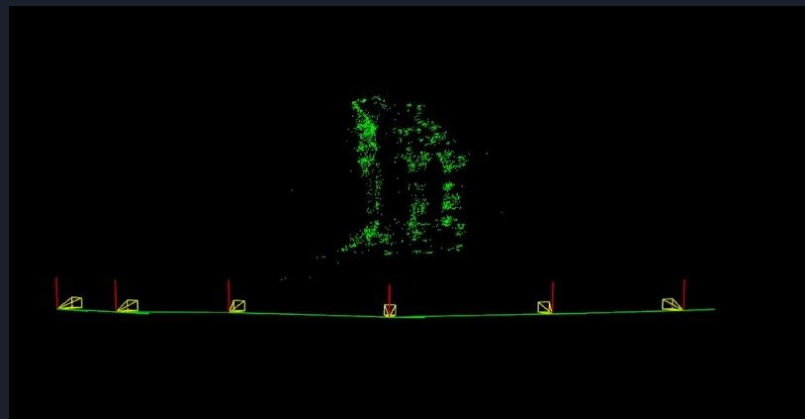
The goal of this project is to reconstruct a 3D point cloud and the camera poses from images in the middlebury temple dataset

The middlebury temple dataset consists 312 views of a plaster reproduction of "Temple of the Dioskouroi" in Agrigento, Sicily, sampled on a hemisphere

In addition to the images, the dataset also provides camera intrinsics and extrinsics to be used as ground truths for the purpose of evaluation



Sample images from Middlebury temple dataset



Example scene reconstruction using images from middlebury dataset



What to expect from this presentation

- The 3D structure from motion pipeline explained
 1. Match features between images
 2. Estimate essential matrix and camera pose
 3. Triangulation
 4. Perspective N points and reprojection
 5. Bundle adjustment
- Our progress and results
- Analyse and Future Work



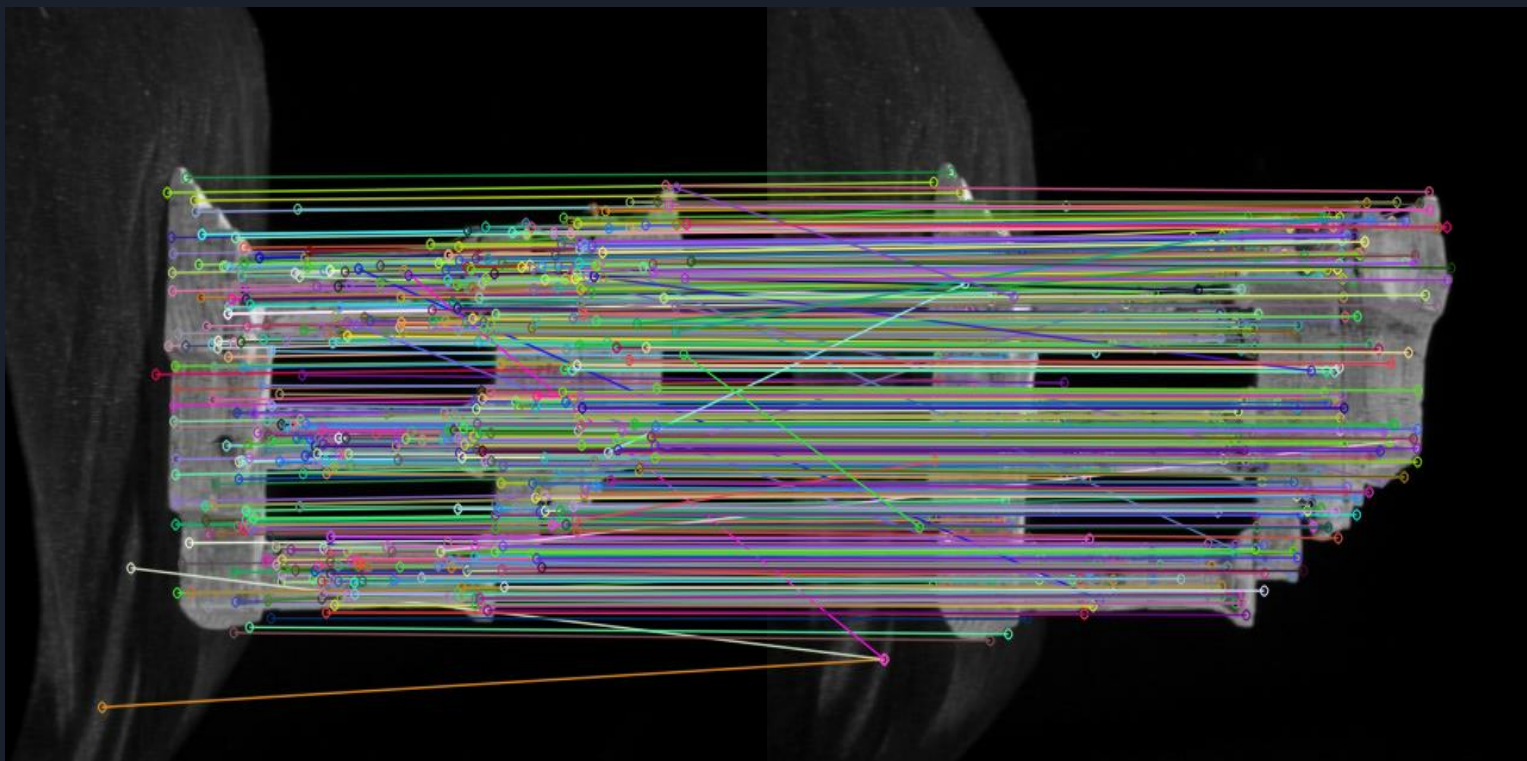
Finding Point Correspondences using Feature Matching

1. Feature Extraction

- a. SIFT feature detectors and descriptors are used to find features in the images.

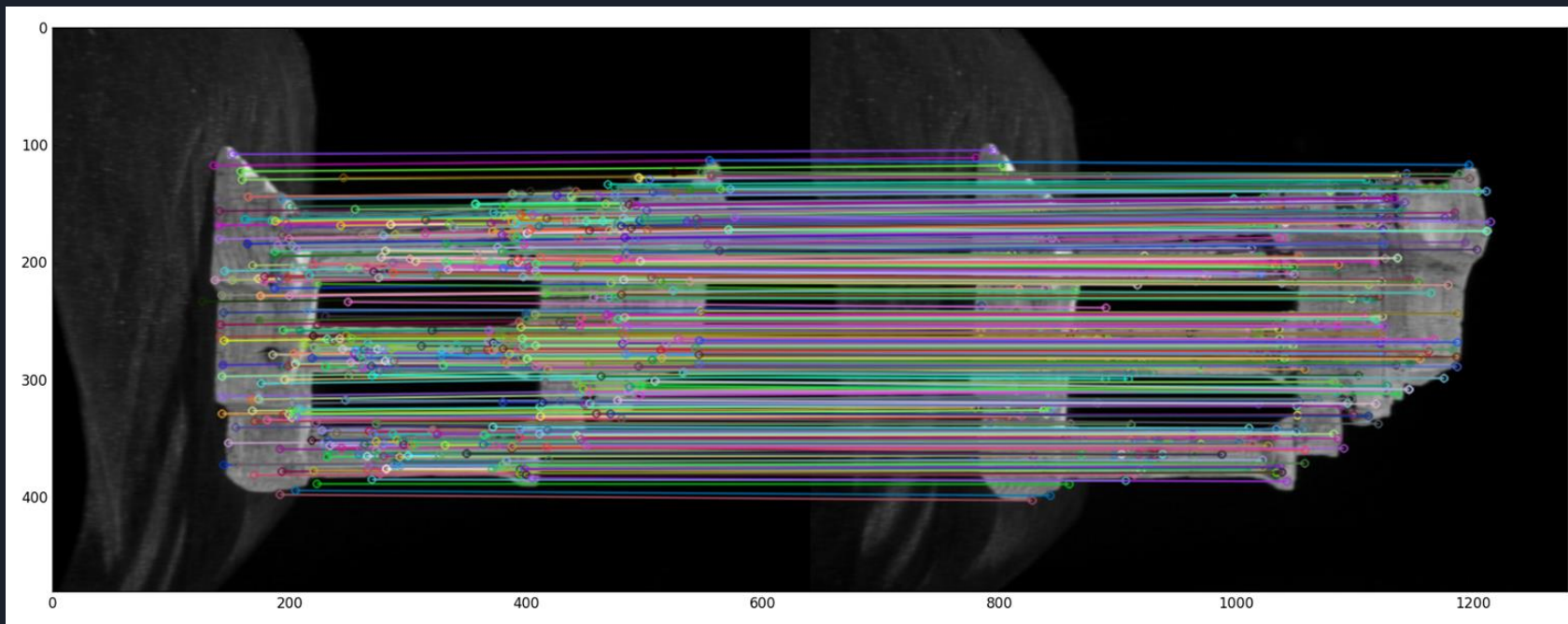
1. Feature Matching

- a. Matching criteria:
 - least L2 norm of difference of descriptor vector. (equivalent to least euclidean distance between the descriptors)
- a. Search Method:
 - Approximate Nearest Neighbours



Outlier Rejection:

- Euclidean distance between matched keypoints is calculated.
- Matches with this measure greater than a threshold are rejected.
- This is based on the assumption that the coordinates of the key points will be close to each other,.





Estimate essential matrix

- The essential and fundamental matrices are 3×3 matrices that “encode” the epipolar geometry of two views
 - Epipolar geometry is the geometry of stereo vision. When two cameras view a 3D scene from two distinct positions, there are a number of geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points.
- Given a point in one image, multiplying by the essential/fundamental matrix will tell us which epipolar line to search along in the second view.



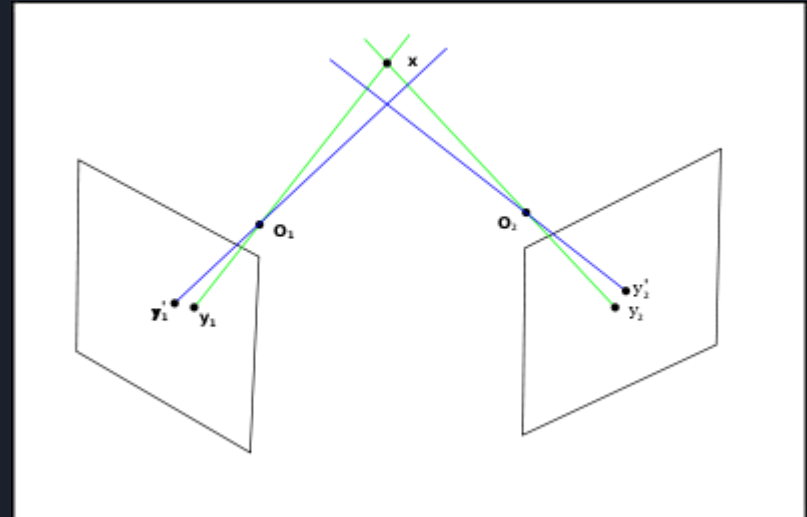
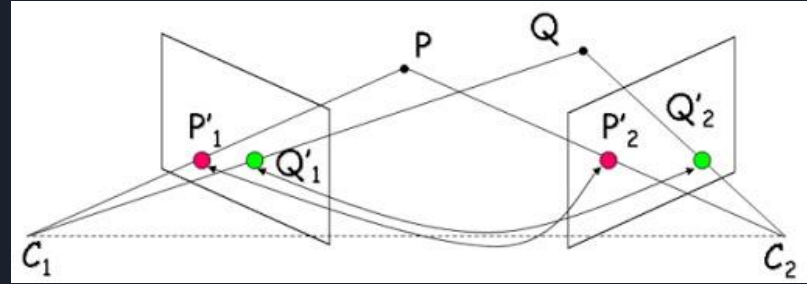
Estimate camera pose

- Essential matrix can be used to initialize relative camera pose between the first and second images.
 - Determine R and t based on an SVD (Singular Value Decomposition) method
- We used the following opencv function to estimate camera pose:
 - `cv2.recoverPose(EssentialMat, points1, points2, cameraIntrinsic)`, where the `points1` and `points2` are the image coordinate of the matched features

Linear Triangulation of points from two images

In theory, triangulation is trivial. Every point in an image corresponds to a line in 3D space. With the same point captured and matched in two images, the 3d point is simply the intersection between these two lines.

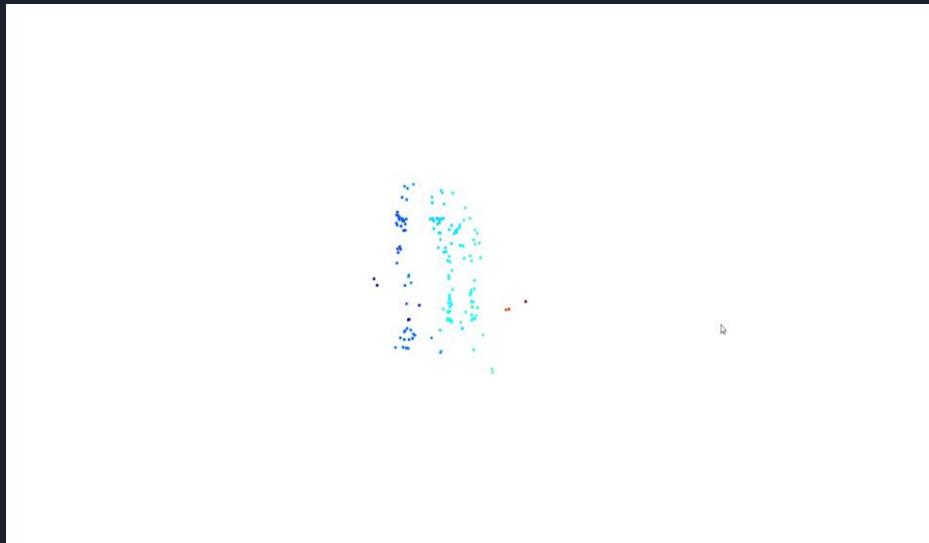
In practice, however, various noises and errors lead to inaccuracies in the measured image coordinates. As a consequence, the lines generated by the corresponding image points do not always intersect in 3D space. The problem, then, is to find a 3D point which optimally fits the measured image points.




Linear Triangulation of points from two images

Given two camera poses, (t_1, R_1) and (t_2, R_2) from the essential matrices, and correspondences $x_1 \leftrightarrow x_2$, we can triangulate 3D points using Direct Linear Triangulation (DLT).

Image on the right shows the result of triangulation of keypoints from images 1 and 2 using OpenCV function `triangulatePoints()`.





Perspective N points(pnp) and reprojection:

Given the existing 3d points, find some 3d points and its corresponding 2d points in the new image to estimate the new camera pose.

For example:


After the triangulation of the 2-d points (matched features) in the first two images, We got a set of 3d points.

In the third image, there are some features that can be matched both by the first and second image, so they share the same 3d points in space.

Given the coordinates of the features in the new image and its corresponding 3d points in space, we can calculate the new camera pose with the following equation.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

There are some built-in functions in openCV like solvePnP or solvePnP Ransac



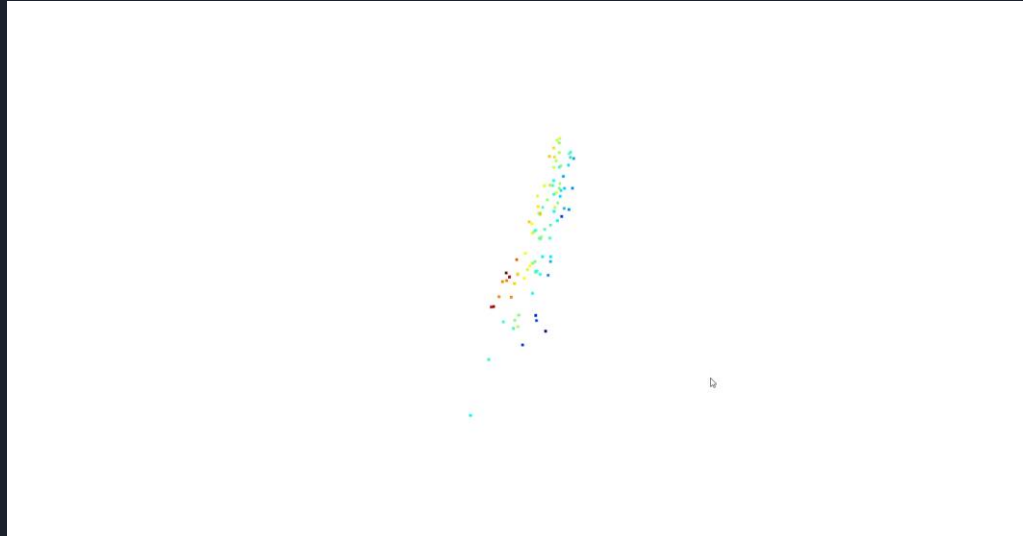
Because our feature detection generates a dense set of interesting points, the shared points that could be used to solve PnP is more than 7 points and noisy.

In order to get a reliable camera pose from PnP, we use solvePnP with Ransac the opencv:

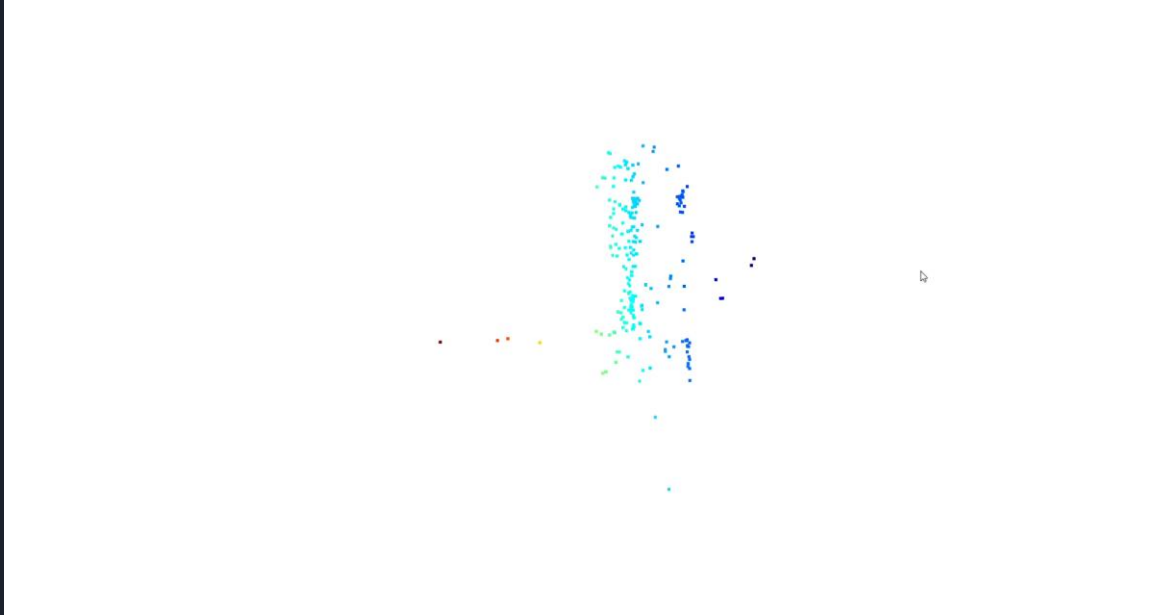
- Use the points whose reprojection error is less than 6 as the input
- Set the iterations count at 2000 and confidence at 0.99

After getting the new camera pose, we could do triangulation for the matched features in the third image and the second image.

New points from the 2nd and 3rd image:



After we getting the new 3d points, we add the new points to the structure.
For now, we get the 3d points from the first three images.

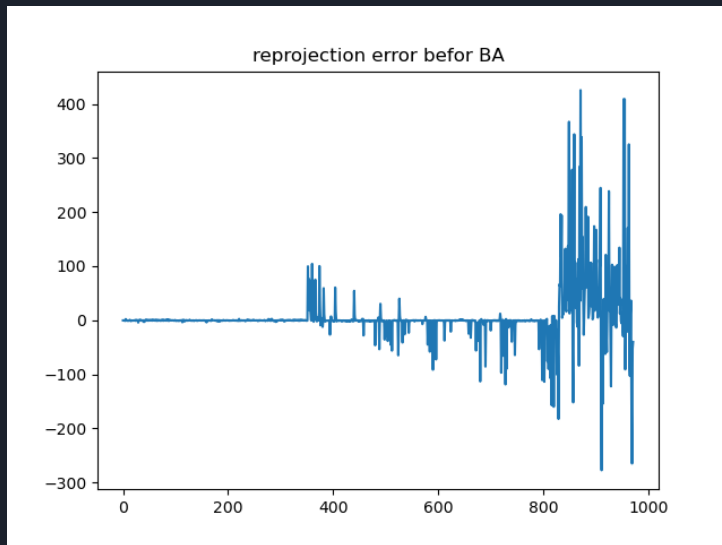


video:3d points getting from the first three images

Perspective N points and reprojection:

Although it resembles the temple by now, it has many outliers and looks noisy. We would like analyze the reprojection error for it.

With the the existing 3d points and camera pose, we reproject the 3d points onto each image and calculate reprojection the error ($u_{\text{reprojection}} - u$, $v_{\text{reprojection}} - v$)



Reprojection error for first 3 images



Bundle Adjustment

How to minimize the error?

People usually do some bundle adjustment by adjusting the camera pose and 3d point.

We tried different bundle adjustment methods to get a more reliable outcome.

Method 1:

We did a simple bundle adjustment by deleting the 3d points whose reprojection error is bigger than a threshold.

Algorithm:

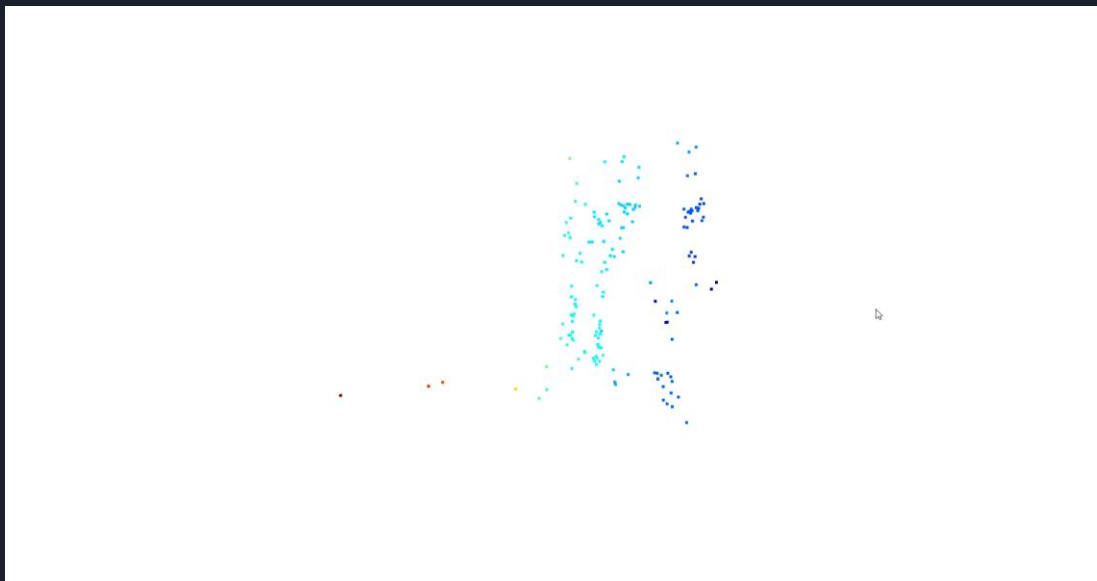
```
For every 3d point:  
    e = reprojection_error  
    If e > threshold:  
        Delete the 3d points  
    End if  
End for
```

Analysis for method 1:

1. After deleting outlier points, it is become more clean but sparse
2. When we tried to add more images to generate the points, the results is also sparse.
3. As a result, it works well when the number of the images is small.

Reason:

1. We used a dense feature detector, and more dense the features, more noisy the features
2. PnP didn't give us a reliable camera pose due to noise, and every time when you add a new image/camera with PnP, the camera pose become worse and worse.



Video: 3d points after 1st BA



Bundle Adjustment

Methods 2: Non-linear optimization

For the reprojection error is related to the camera intrinsic parameters, the camera poses(rotation matrix and translation) and the 3d points coordinates.

Let $P=(X,Y,Z)^T$ - a radius-vector of a point, R - a rotation matrix of a camera, t - a translation vector of a camera, f_x and f_y - its focal distance, c_x and c_y - its image center. We could define a function:

$(error1, error2, error3, \dots) = \text{fun}(X) = (\text{reprojection_func}(f_x, f_y, c_x, c_y, R, t, P_1, P_2, \dots, P_N) - (uv1, uv2, uv3, \dots))$

Where the error1 is the first 2d feature's reprojection error, $uv1$, is the first 2d feature's coordinate.

Given the residuals $f(x)$ (an m -dimensional real function of n real variables) and the loss function $\rho(s)$ (a scalar function), we can use the **least_squares** finds a local minimum of the cost function $F(x)$:

$$\text{minimize } F(x) = 0.5 * \sum(\rho(f_i(x)**2), i = 0, \dots, m - 1)$$

The purpose of the loss function $\rho(s)$ is to reduce the influence of outliers on the solution.



Bundle Adjustment

Methods 2: Non-linear optimization

Algorithm:

Define a residuals $f(x)$, whose input is the camera poses and 3d points coordinates, whose output is the reprojection error

vector

Use “cauchy function” as loss function to reduce the huge effects of outliers

of

Iterate the optimization method to find the local minimum of the loss

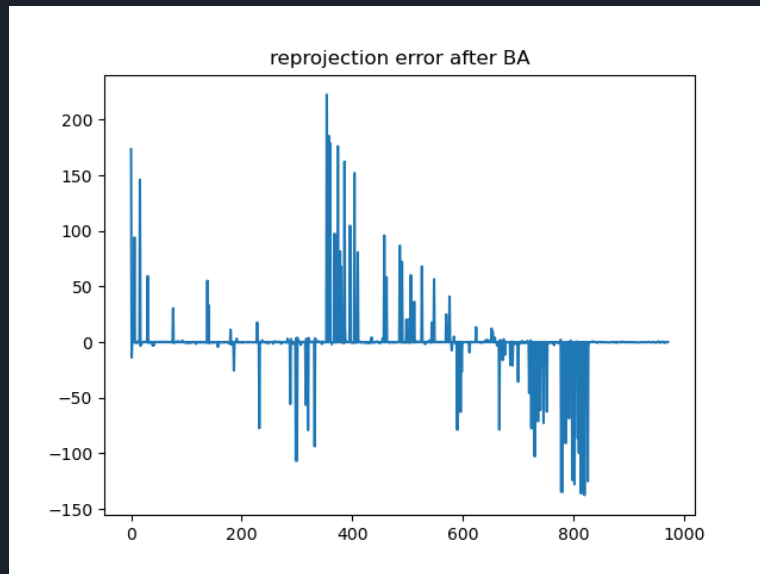
Pick the 3d points from the local minimum solution

Some note:

For the camera intrinsic is given by the dataset, it accurate, we didn't optimize the camera intrinsic

Bundle Adjustment

Reprojection error after the BA: (first three images)



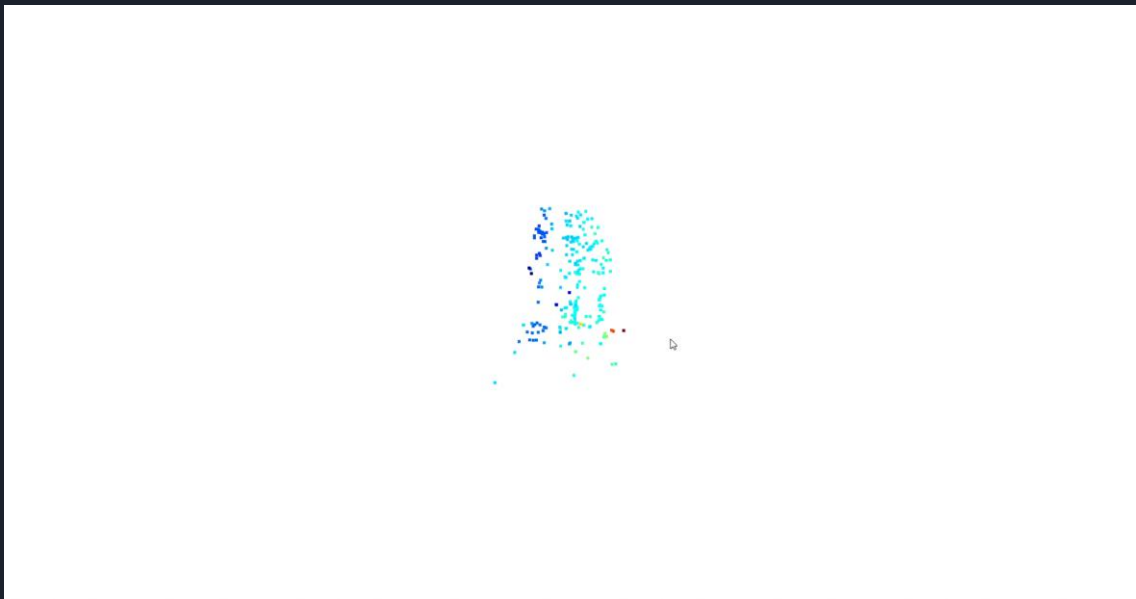
Function evaluations 456 times, initial cost $1.6547\text{e}+04$, final cost $6.1976\text{e}+03$, first-order optimality $6.06\text{e}+03$.

optimization took 5 seconds

We could see almost half of the reprojection error has been reduced

Bundle Adjustment

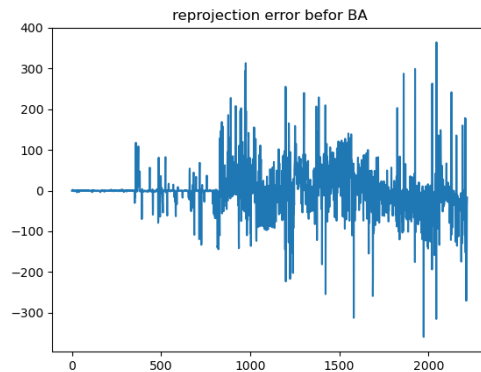
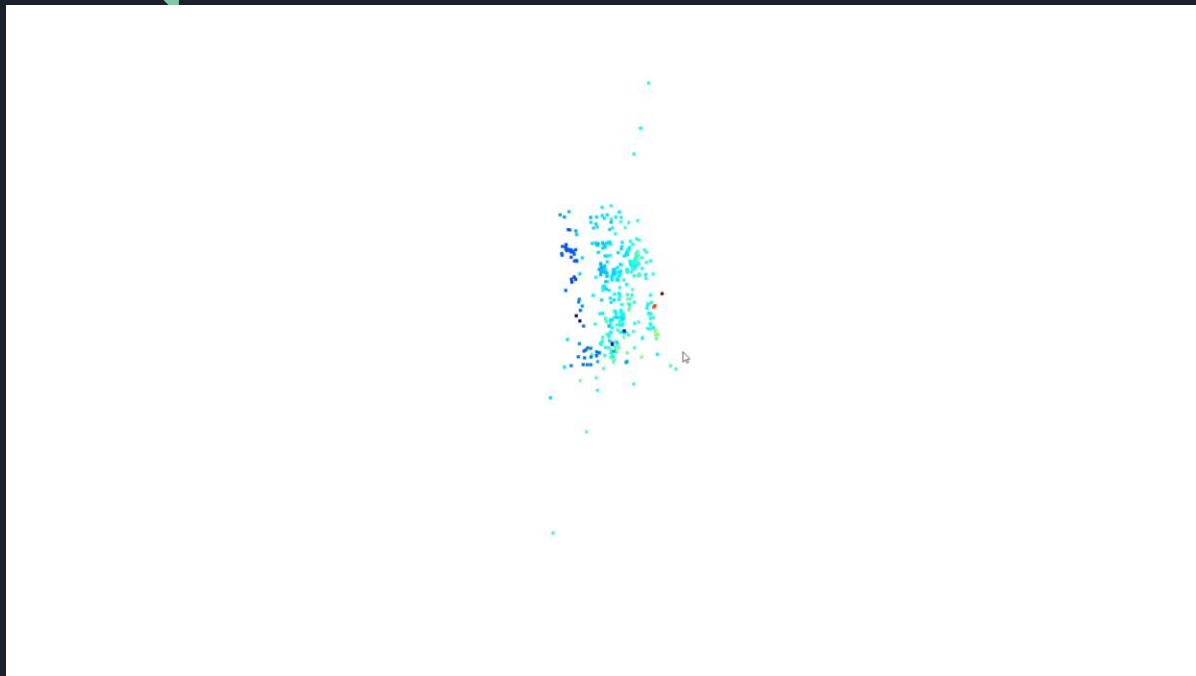
3d points after BA from the first three images



For now, the 3d points become more dense and less noise, the non-linear optimization did a good job

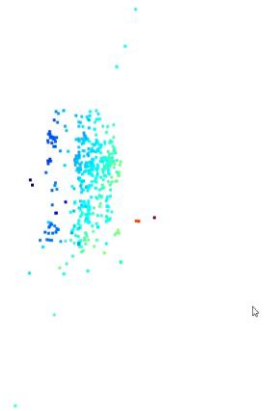
Bundle Adjustment

More results: points from 4 images



Bundle Adjustment

More results:
points from 5 images



Analysis:

1. The least-squares bundle adjustments could gave us a more dense and less noise point cloud than the first method.
2. It will always reduce the reprojection error
3. As we increase the number of images, its shape is becoming a little blur. If we add images more than 7, the outcome seem distorted.

Reasons:

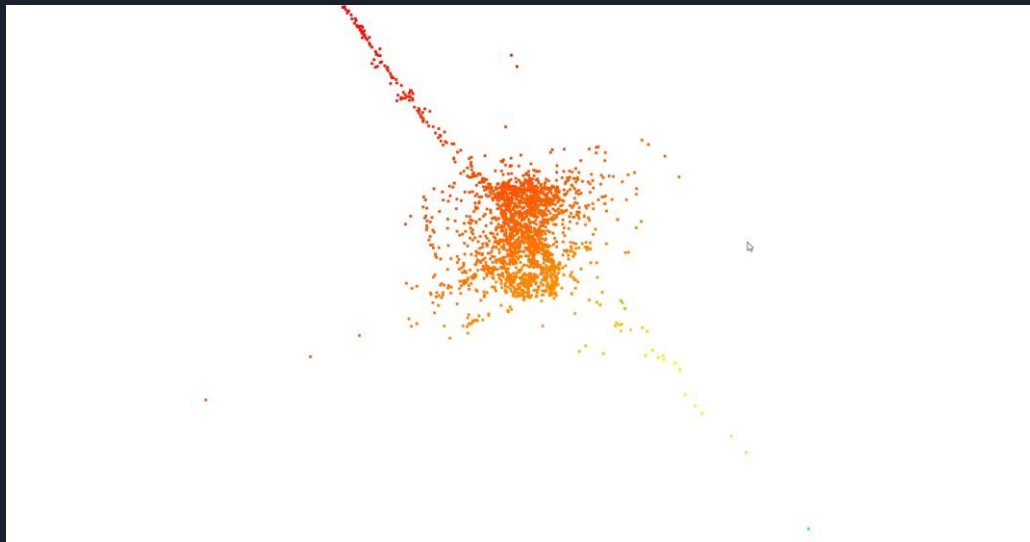
1. The feature is dense and noisy
2. Least- square method could be affected by the outliers aggressively, which will deform the shape.
1. Using ransac to solve PnP is not always stable

Bundle Adjustment

Issue for least-squares BA:

As mentioned above, the least-squares BA tries to reduce the overall reprojection error, so it could be affected by the extreme outliers and produce a bad results.

Video for first 10 images seems noisy:



How to solve this issue?

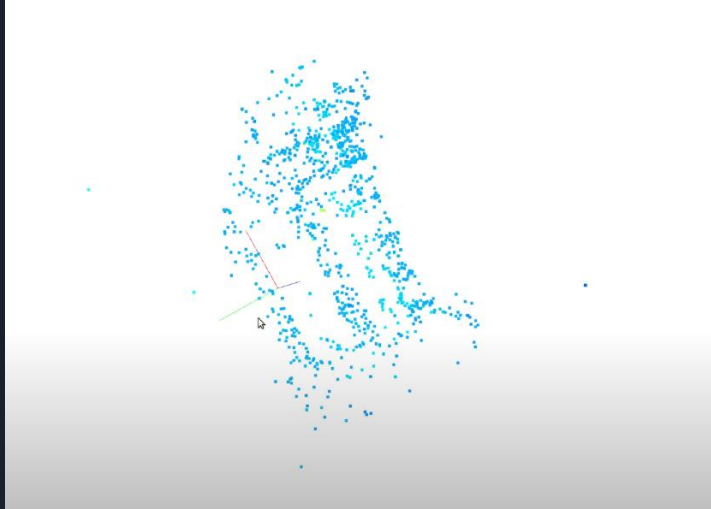
1. We could calculate each points' reprojection error and assign smaller weights for the outliers in the residual function to reduce its affection
2. We could also delete the 3d points whose reprojection error is extremely large and do BA for the rest points.(it will be difficult due to the correspondence between features and 3d points)



Results compared with Another Approach:

- The dataset comes with the camera poses for every image.
- We could use the given camera poses instead of estimating the camera poses.
- This approach does consecutive feature matching and triangulation of feature points.
- The triangulation step makes use of given camera poses thus by skipping the estimating of essential matrix, PnP and bundle adjustment.
- The next slide shows the result of this approach.

The shape of the temple looks pretty good, but there still exists some outliers due to the noise in image





Conclusion:

1. We solve the structure from motion problem with the following process:
 - a) feature detection and feature match, which give us a set of dense matched features
 - b) calculated the Essential matrix
 - c) calculated the camera pose from the essential matrix
 - d) linear triangulation
 - e) PnP and reprojection
 - f) Two way of bundle adjustment
1. There are many other algorithms that could be used to improve sfm such as Non-linear triangulation, Non-linear PnP. For the time's reason, we couldn't explore them.



Reference:

1. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#triangulatepoints
2. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html
3. https://scipy-cookbook.readthedocs.io/items/bundle_adjustment.html
4. multiple view geometry in computer vision, H and AZ

Acknowledge:

We appreciate professor Hanu, TA Mithun, and Pushyami's help!