3D Structure-from-Motion with Bundle Adjustment on Video Input

- Project Final Report -

Aiden & Rainsford

Dec 14 2022

Contents

1 Abstract

This project aims to complete 3D reconstruction based on the feature points detected and matched between consecutive frames in a video. We will address the issue of missing feature points between two consecutive frames. The damped Newton method on matrix factorization with missing data will be used to measure the difference between the actual values in the observation matrix and the predicted values produced by the model. To minimize the reprojection error, which is the difference between the observed image points and the projected image points in the 3D reconstruction model, we will use bundle adjustment to simultaneously adjust the camera poses and the 3D points, utilizing the concept of gradient descent. Our final 3D reconstruction will be displayed using point cloud in MATLAB.

2 Introduction

2.0.1 Input & Output

Generally, the input is a recorded MOV type video shot by an iPhone XS camera on the Grinnell College Noyce Building 3820 classroom. The output would be 3D point clouds of the classroom reconstruction generated by MATLAB built-in pcShow function.

In order to achieve this, we take several steps toward this goal. First, we extract and match features across consecutive frames of images. Secondly, we optimize the observation matrix using damped newton algorithms on matrix factorization with missing data (1). Next, we fix the reprojection error using bundle adjustment (2). Lastly, we convert the recorded video into frames of images and integrate the optimization methods to effectively and efficiently build the 3D reconstruction. We will explain these steps in more detail in the next few paragraphs.

2.1 Feature Detection & Matching

2.1.1 Input & Output

The input to this process is a set of images of the same scene, which can be obtained from a single camera or multiple cameras with overlapping fields of view. The output is a set of corresponding points in the different images represented by a matrix. These points can then be used as input to the matrix factorization for more optimization.

2.1.2 Key Challenges

The key challenge for the first milestone is to determine the values of various variables, such as thresholds for feature detection and matching and the sizes of the search and SSD windows (5). For a decent 3D reconstruction, we need

as many matched features as possible without false matches. To acquire dense matched features, we adjusted those variables step by step while examining the effect of each variable.

2.1.3 Feature Detection

For feature detection, we use the same method as the one in the Feature Detection lab. The method basically follows the MOPS(multi-image matching using multi-scale oriented patches) algorithm described in the paper Multi-image matching using multi-scale oriented patches (6), but unlike the algorithm in the paper, our method detects features at a particular scale, and this makes our method not robust against scale. However, this is acceptable in our project because our final goal is to build a 3D reconstruction from a video. Generally, the changes in consecutive frames of a video are not huge, our method is good enough for our final goal.

2.1.4 Feature Matching

For feature matching, we use a similar method as the one in the Stereo Disparity lab (4). The method basically uses SSD(Sum of Squared Difference). As moving our search window, we use convolution on our SSD window and find matched features. The matched features are located in the pixel containing the minimum SSD. This method is suggested by Professor Jerod Weinman for the same reason that the changes in consecutive frames of a video are not huge (5).

2.2 Matrix Factorization with Missing Data

2.2.1 Input & Output

Damped Newton method with missing data (1) is the core algorithm in this process. Observation matrices from the previous step are the input for this process. The output is an estimate of the 3D structure of the scene, which can be represented as a matrix of 3D points. This matrix can then be used as input to the bundle adjustment algorithm to refine the estimates of the 3D structure and camera poses.

2.2.2 Key Challenges

The key challenge for this milestone includes finding the weight matrix based on the unmatched features across different frames, implementing an algorithm that takes the first derivative (gradient) of a function over a matrix, and implementing an algorithm that takes the second derivative (Hessian) of a function over a matrix.

2.2.3 Weight Matrix

As described in the background, the error function includes a weight matrix W. We calculate the Hadamard product of W and the difference between the

observation matrix O and the estimated matrix $M \cdot S'$ to eliminate the effect of unmatched feature points. However, the testing images in the Middlebury dataset (8) have a limited number of scenes that are very difficult to recover the depth information in the first place. Therefore, applying another weight matrix will further eliminate more features.

2.2.4 First and Second Derivative

Taking the first and second derivatives to a matrix function over a vector is another problem. Although Buchanan and Fitzgibbon (1) provide mathematical derivation on these problems in their paper's appendix A, implementing these mathematical formulas in MATLAB is on another level. First, MATLAB does not have a built-in function that could be used on a matrix function, so we have to implement it ourselves. Secondly, taking the first derivative of the error functions requires taking a derivative over a vectorized motion matrix and vectorized structure matrix with different sizes. Therefore, we have to decompose the vectorized observation matrix into two vectorized motion and structure matrices and operate them separately. Thirdly, the mathematical derivation requires not to use vectorized M and N as a whole but in terms of specific rows or columns. Thus, in order to retrieve the required vector row/column in the matrix, using a for-loop inside another for-loop is the most optimal for now.

Moreover, implementing this idea to the second derivative is even more complicated because we now have to combine the first derivative's loop inside another derivative's loop. We also have to implement these three times for the top-right, top-left, and bottom-right blocks separately. In addition, for the bottom-right block specifically, we also have to reshape the resulting matrix by the right amount to let each group of feature points be on the diagonal since the structure matrix S is not guaranteed to be a square matrix.

2.3 Bundle Adjustment

2.3.1 Input & Output

The input to bundle adjustment (2) is the output of the matrix factorization optimization, i.e. an initial estimate of the 3D structure and camera poses. The output of the bundle adjustment process is a refined estimate of the 3D structure and camera pose that minimizes the reprojection errors between the observed 2D points and their corresponding 3D points. This refined estimate can then be used to reconstruct the 3D scene from the input images.

2.3.2 Key Challenges

2.3.3 Formulating the objective function

Bundle adjustment (2) involves optimizing the 3D structure and camera poses to minimize the reprojection errors between the observed 2D points and their corresponding 3D points. This requires defining a cost function that measures the difference between the observed and predicted 2D points, and then finding the optimal 3D structure and camera poses that minimize this cost. This function should be designed to be differentiable so that it can be optimized using gradient-based algorithms. We define the objective function (7) to use the sum of squared reprojection errors, which is given by:

$$J = \sum_{i=1}^{n} \left| p_i^{obs} - p_i^{pred} \right|^2$$

n is the number of observations, p_i^{obs} is the observed 2D point, and p_i^{pred} is the corresponding predicted 2D point based on the current 3D structure and camera poses. This objective function can then be minimized with respect to the 3D structure and camera poses to find the optimal solution.

2.3.4 Large Input

When using a video as input for 3D reconstruction, the consecutive frames in the video can be processed to obtain a series of 2D images, which can then be used as input for the bundle adjustment algorithm. This process can be computationally intensive, especially if the video contains a large number of frames or if the images are high resolution. Due to limited time, there may not be enough trials to thoroughly test the bundle adjustment algorithm.

2.4 Integrate Methods with Video Input

2.4.1 Input & Output

The input of this milestone is the MATLAB function implemented and the video input we recorded. The output is a point cloud function generated by calling the pcShow function in MATLAB.

2.4.2 Key Challenges

Large dataset sizes and unoptimized algorithms are the primary challenges. Previous milestones have only considered existing datasets without noise, generating unexpected artifacts. Besides, a one-minute video requires over one thousand frames, resulting in over 7.7 GB of data, which exceeds the maximum load of MATLAB. Additionally, the integration between several functions even causes crashes and errors due to unmatched matrix sizes and data types.

3 Literature Review

3.1 Feature Detection and Feature Matching

3.1.1 Feature Detection

Feature detection (6) in computer vision and image processing involves identifying key points or regions in an image that are relevant to a specific task, and then extracting and representing this information in a condensed and useful format.

A major challenge in feature detection is that the features of interest can vary greatly depending on the specific application and the type of image being analyzed. For instance, in an image of a person's face, the features of interest might include the eyes, nose, mouth, and ears, while in an image of a landscape, the features of interest could include the horizon, trees, buildings, and other objects. This means that feature detection algorithms must be able to adapt to a wide range of different scenarios and image types.

3.1.2 Feature Matching

The second step is to match features across two frames using SSD, which involves searching for the most similar pair of features in each frame. First, for each feature in one frame, compute the SSD between its feature vector and the feature vectors of all the features in the other frame. This will produce a set of SSD values for each feature in the first frame, with each value corresponding to the similarity of the feature to a different feature in the second frame. Secondly, select the pair of features with the lowest SSD value as the most similar pair. This will typically be the pair of features that are most similar in terms of their feature vectors. Lastly, repeat this process for all the features in the first frame, to produce a set of pairs of matching features, with each pair consisting of one feature from each frame. This process can be repeated for multiple frames in a video sequence, to produce a set of matched features for each frame.

3.2 Matrix Factorization with Missing Data

We use the damped Newton method proposed by A.M. Buchanan and A.W. Fitzgibbon (1) to implement our feature matching algorithm. The first step in this process is to compute the observation matrix based on the feature points detected across the frames. This matrix has dimensions of $2m \times n$, where m is the number of frames (images) and n is the number of feature points detected. Next, we apply singular value decomposition (SVD) to factorize the observation matrix into two matrices: the motion matrix M, which stores the camera's motion information, and the structure matrix S, which stores the 3D structure of the scene. This allows us to reconstruct the 3D geometry of the scene based on

the motion and structure information obtained from the observation matrix.

However, in real-world scenarios, the straightforward technique described above can be susceptible to noise from missing matched points across different frames and other conditions such as lighting and depth changes. To address this issue, Buchanan and Fitzgibbon's algorithm aims to reduce noise in the matrix factorization process. In other words, the goal of the algorithm is to optimize the factorized structure matrix S and motion matrix M to be as close to the true values as possible, in order to reduce the effects of noise on the reconstructed 3D geometry. This can be achieved by incorporating regularization and damping terms into the matrix factorization process, which help to smooth out noise and improve the accuracy of the reconstructed 3D model.

The main idea behind Buchanan and Fitzgibbon's algorithm (1) is that there exists a global minimum of the error function, which yields a motion matrix M and structure matrix S that are closest to the ground truth values. The error function is calculated using the original observation matrix O obtained from feature detection, the guessed matrices M and S, and a weight matrix W. The error function $e(M, S)$ is given by the following equation:

$$e(M, S) = |W \odot (O - M \cdot S')|_F^2$$

where $\odot$ represents the element-wise multiplication operator, and $|\cdot|_F$ denotes the Frobenius norm.

The error function is written in this form because it was implemented in MATLAB. The Hadamard product, which is denoted by the $\odot$ operator in the equation, refers to the element-wise multiplication of two matrices. This operation is used in the error function because it allows us to "zero out" the contribution of unmatched feature points to the error, by multiplying them with a corresponding value of 0 in the weighted matrix W. This helps to minimize the impact of missing feature points on the reconstructed 3D model, by effectively ignoring their contribution to the error.

The Frobenius norm is a measure of the differences between two matrices, which is computed by taking the square root of the sum of the squares of the elements in the matrix. This norm is often used to evaluate the differences between the reconstructed and ground-truth values, in order to assess the accuracy of the reconstructed model.

3.3 Bundle Adjustment

We use the bundle adjustment based on Triggs's approach in paper: "Bundle Adjustment: A Modern Synthesis." (2) Bundle adjustment algorithms are optimization algorithms for refining the estimates of the camera poses and 3D structure of a scene from a set of images. The paper provides a detailed review

of the history and development of bundle adjustment algorithms, as well as a comprehensive analysis of different algorithmic approaches and their relative strengths and weaknesses.

The authors begin by introducing the problem of bundle adjustment and providing a brief overview of the history of bundle adjustment algorithms. They then present several different algorithmic approaches to bundle adjustment, including linear and non-linear least squares methods, sparse factorization methods, and global optimization methods.

Next, the authors describe in detail several specific algorithms within each of these categories, including the Gauss-Newton method, the Levenberg-Marquardt algorithm, and the Bundle Adjustment by L1 Norm Minimization (BALM) algorithm. They also provide a comprehensive analysis of the performance of these algorithms on a variety of benchmark datasets, comparing their accuracy, efficiency, and robustness. This analysis allows the authors to evaluate the strengths and weaknesses of each algorithm and provide guidance on which algorithms may be most suitable for different applications.

For example, the authors compare the performance of the Gauss-Newton method, the Levenberg-Marquardt algorithm, and the Bundle Adjustment by L1 Norm Minimization (BALM) algorithm on a dataset consisting of 100 images of a 3D scene. They measure the accuracy of each algorithm by comparing the estimated camera poses and 3D structure with the ground truth values, and they measure the efficiency by calculating the time required to compute the estimates. They also assess the robustness of each algorithm by evaluating how well it can handle noise and outliers in the data.

Based on the analysis presented in the paper, we have decided to use the Levenberg-Marquardt algorithm for bundle adjustment in our application. This algorithm has been shown to be slightly less accurate and efficient than the Gauss-Newton method, but it is more robust to noise and outliers in the data.

4 Process and Experiment

4.1 Feature Detection and Feature Matching

4.1.1 General Steps

Our first milestone is to implement feature detection and feature matching on consecutive frames to construct the observation matrix for 3D reconstruction. The feature detection method is based on the one presented in Szeliski's

paper (6). For feature matching, we use the Sum of Squared Differences (SSD) algorithm, as suggested on Piazza by Professor Jerod Weinman. We calculate SSD using a convolution on an SSD window and find matched features within a search window.

4.1.2 Detailed Steps

We set a low threshold for feature detection to maximize the number of detected features. With a threshold of $1e-6$, we were able to detect over 3000 features on average from our 480x640 images. While the number of detected features can vary depending on the characteristics of the images, this threshold provides enough features for 3D reconstruction. To minimize false matches, we used a large SSD window size to capture sufficient information about each patch and removed false matches using a reasonable SSD threshold. After experimenting with different values, we found that an SSD window size of 20 and a threshold of 10 provided good results. For the search window size, we chose 20 based on our dataset. These parameters may need to be adjusted for different datasets in the final milestone.

4.1.3 Experimental Setup

For feature detection, we first find horizontal and vertical partial derivatives Ix and Iy for a given image. Using the two partial derivatives we construct the matrix A as follows:

$$A = \begin{pmatrix} I_x^2 & I_{xIy} \\ I_{xIy} & I_y^2 \end{pmatrix}$$

Then, we find the determinant and the trace of the matrix A. Based on the ratio of the determinant to the trace, we detect features that do not exceed a given threshold. Finally, at the end, to make the detected features locally strong, we sampled the features which are local maxima.

For feature matching, we pad and move one image for one pixel on our search window. Then, we use convolution on our SSD window to find the SSD between the image and the other image. We repeat the whole process for every pixel on our search window. The pixels that contain the minimum SSD and do not exceed a given threshold are matched features.

4.1.4 Evaluation

We used the Urban3 from the Middlebury dataset for evaluation. The first frame is shown below:

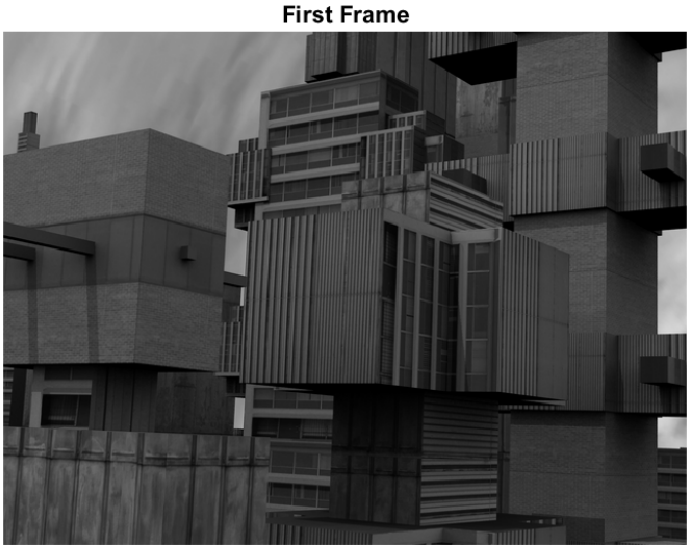

Figure 1: First Frame

The image below shows over 3000 matched features in 8 consecutive frames of our dataset. The matched features are dense and almost completely cover the frames. We believe these matched features are sufficient for the next step.

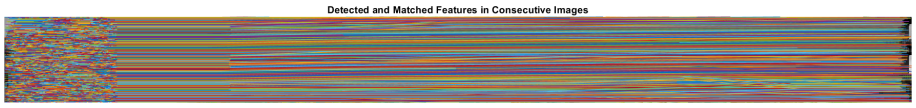

Figure 2: Detected and Matched Features in Consecutive Images

4.2 Matrix Factorization with Missing Data

4.2.1 General Steps

First, we initialize the model parameters (i.e., the matrices that will be factorized) with random values. This provides a starting point for the optimization process. Second, we compute the gradient of the error function with respect to the model parameters. The algorithm for calculating the gradient is based on the appendix provided by Buchanan, A. and Fitzgibbon. The gradient tells us the direction in which the error will change as we adjust the parameters. Third, we use the gradient to update the model parameters in the direction that reduces the error. Damped Newton method involves computing the Hessian matrix of the error function and using it to compute the update direction. Next, we repeat the second the third step until the error is minimized and falls below a certain threshold. Finally, we use the optimized model parameters to

generate predicted values for the missing data in the observation matrix. We do this by multiplying the optimized model parameters and a weight matrix together to produce a predicted matrix, and then comparing it to the actual observation matrix.

4.2.2 Detailed Step and Experimental Setup

The error function is defined as follows:

$$F = ||W \odot (M - AB^T)||_F^2$$

where F is the error function, W is a weight matrix, M is the observation matrix, A and B are the matrices being factorized, and $\odot$ represents the Hadamard product. The Frobenius norm is a common measure of the size of a matrix, and is defined as the square root of the sum of the squares of all the elements in the matrix.

4.2.3 Gradient

The gradient of a matrix with respect to another matrix is a vector that describes how the original matrix would change if small changes were made to the second matrix. It is calculated by iterating over the rows and columns of the matrix and computing the derivative of the matrix with respect to the current row and column at each iteration. The derivatives are then added to the final gradient vector, which is returned at the end of the algorithm.

Algorithm 1 Gradient of Matrix O with Weight Matrix W (1) (9) (10)

Input: O, W **Output:** $\mathbf{d}$

Apply the normal SVD to O to obtain its left singular matrix A, right singular matrix B, and diagonal matrix S.
Initialize the gradient vector $\mathbf{d}$ to a zero vector of the appropriate size.
for each the rows and columns of O **do**
    for each row a and column b of A **do**
        compute the derivative of O with respect to A using:
        $\mathbf{d}_{a,b} = B_{i,b} \cdot S_{a,a} \cdot W_{i,j}$
        for each column c and row d of B **do**
            compute the derivative of O with respect to B using:
            $\mathbf{d}_{c,d} = A_{a,c} \cdot S_{d,d} \cdot W_{i,j}$
        end for
    end for
end for
Return the gradient vector $\mathbf{d}$.

This loop iterates over all the rows (a) of the matrix A and all the columns (b) of the matrix B. For each row and column, it initializes the corresponding element of the gradient vector and then calculates the derivative of O with

respect to A at this element. The derivative is computed by iterating over all the columns (i) of the matrix O and calculating the intermediate value $(B * a^i - m^i)$. The final derivative is obtained by multiplying this intermediate value with the corresponding elements of the matrices B and W. The derivative is then multiplied by 2 and added to the gradient vector.

4.2.4 Hessian

Algorithm 2 Hessian of Matrix O with Weight Matrix W (1) (9) (10)

Input: O, W, **vecO Output:** H

Apply the normal singular value decomposition (SVD) to matrix O to obtain matrices A, S, and B.
Get the number of rows (m), columns (n), and the number of rows in the singular matrix (r) of matrix O.
Vectorize matrices A and B along their rows to obtain new matrices A and B with dimensions $m \times r$ and $r \times n$ respectively.
Initialize three matrices, aa, ab, and bb, to store the top-left, top-right, and bottom-right blocks of the hessian matrix, respectively.
for $a = 1$ to m **do**
    for $b = 1$ to r **do**
        Initialize $(de/da)^2 = 0$
        aa: $index1 = (a-1) \cdot r + b$
        for $f = 1$ to r **do**
            aa: $index2 = (a-1) \cdot r + f$
            for $i = 1$ to n **do**
                $aa_{index1,index2} = aa_{index1,index2} + W^2_{a,i} \cdot B_{i,b} \cdot B_{i,f}$
            end for
            $aa_{index1,index2} = aa_{index1,index2} \cdot 2$
        end for
    end for
end for

for $c = 1$ to n **do**
    for $d = 1$ to r **do**
        ab: $index2 = (c-1) \cdot r + d$
        for $e = 1$ to m **do**
            for $f = 1$ to r **do**
                ab: $index1 = (e-1) \cdot r + f$
                $ab_{index1,index2} = 0$
                if $d = f$ **then**
                    ab: $ab_{index1,index2} = \sum_{p=1}^{r}(A_{e,p} \cdot B_{c,p} - O_{e,c})$
                    $ab_{index1,index2} = ab_{index1,index2} + W^2_{e,c} \cdot (A_{e,d} \cdot B_{c,f} + ab_{index1,index2})$
                end if
            end for
        end for
    end for
end for

Continuing in the next page

for $c = 1$ to n **do**
  for $d = 1$ to r **do**
    Initialize $(de/da)^2 = 0$
    bb: $index2 = d$
    for $h = 1$ to r **do**
      bb: $index1 = (c-1) \cdot r + h$
      for $i = 1$ to m **do**
        $bb_{index1,index2} = bb_{index1,index2} + W^2_{i,c} \cdot A_{i,d} \cdot A_{i,h}$
      end for
      $bb_{index1,index2} = bb_{index1,index2} \cdot 2$
    end for
  end for
end for
Concatenate the matrices aa, ab, and bb along their columns to obtain the hessian matrix H.
Return H.

For each element in the top-left block of the hessian matrix (aa), we compute the value using the following equation:

$$aa_{i,j} = \sum_{k=1}^{n} 2 \cdot W^2_{a,i} \cdot B_{i,b} \cdot B_{i,f}$$

where i and j are the indices of the elements in the matrix aa, and a, b, and f are the indices of the elements in the matrices A, B, and S respectively.

For each element in the top-right block (ab), we compute the value using the following equation:

$$ab_{i,j} = \sum_{k=1}^{m} 2 \cdot W^2_{e,c} \cdot (A_{e,d} \cdot B_{c,f} + ab)$$

where i and j are the indices of the elements in the matrix ab, and e, c, d, and f are the indices of the elements in the matrices A, B, and S respectively.

For each element in the bottom-right block (bb), we compute the value using the following equation:

$$bb_{i,j} = \sum_{k=1}^{m} 2 \cdot W^2_{i,c} \cdot A_{i,d} \cdot A_{i,h}$$

where i and j are the indices of the elements in the matrix bb, and c, d, and h are the indices of the elements in the matrices A, B, and S respectively.

In the end, we concatenate the three matrices, aa, ab, and bb along their columns to obtain the hessian matrix H.

4.2.5 Damped Newton Method

Algorithm 3 Damped Newton Method for Matrix Factorization with Missing Data (1)

Input: A, B, M, W
Declare $F = ||W \odot (M - AB^T)||_F^2$
$X \leftarrow \text{vectorize}(A, B)$
$\lambda \leftarrow 0.01$
repeat
    $\mathbf{d} \leftarrow \frac{\partial F}{\partial x}$ (gradient)
    $H \leftarrow \frac{\partial^2 F}{\partial x^2}$ (hessian)
    repeat
        $\lambda \leftarrow \lambda \times 10$
        $Y \leftarrow X - (H + \lambda I)^{-1}\mathbf{d}$
    until $F(Y) < F(X)$
    $X \leftarrow Y$
    $\lambda \leftarrow \lambda \div 10$
until convergence
Output: unvectorize(x) to get A, B

This Damped Newton Method algorithm is provided by Buchanan, A. and Fitzgibbon. It decomposes the observation matrix M into the product of two matrices A and B. The algorithm is useful for handling missing data, as represented by the weight matrix W, which is an indicator of which entries of M are known and which are not.

The algorithm starts by declaring a function F that measures the error between the original matrix M and the product AB^T of the two matrices being estimated, with the error only being calculated for the known entries of M (as indicated by W). The algorithm then initializes the two matrices A and B as a single vector X and sets a damping parameter λ to a small value as 0.01.

The algorithm then enters a main loop that iteratively updates X to reduce the value of F. In each iteration, the gradient $\mathbf{d}$ of F with respect to X is calculated, as well as the Hessian matrix H of F at X. Then, it enters a sub-loop that increases the damping factor λ by a factor of 10 in each iteration and calculates a new value for X using the equation $Y = X - (H + \lambda I)^{-1}\mathbf{d}$, where I is the identity matrix. The sub-loop continues until the value of F at the new point Y is less than at the current point X.

After the sub-loop, the main loop continues by setting X to the new value Y, updating λ by a factor of 10, and repeating until convergence is reached. When convergence is achieved, the algorithm outputs the matrices A and B by "unvectorizing" the final value of X.

4.2.6 Evaluation

Next, we build the 3D reconstruction based on the current input and compare the reconstruction using the optimization and the result without using the optimization.

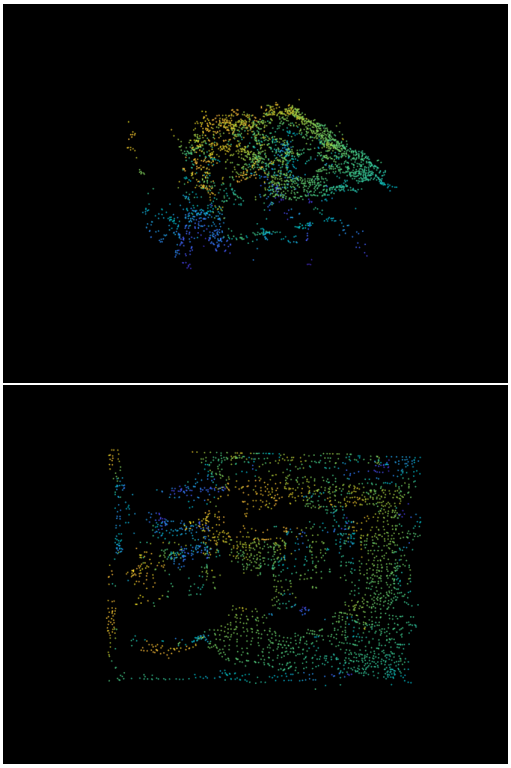

Figure 3: 3D Point Cloud without Missing Data

When the damped Newton method is not used, any features that are lost in consecutive frames cannot be recovered. However, with the damped Newton method, we can overcome this issue and handle missing features (often caused by a camera's angle or occlusion) through optimization. As seen in Figure 4, the missing matched features reappear and the resulting 3D reconstruction is improved compared to Figure 3.

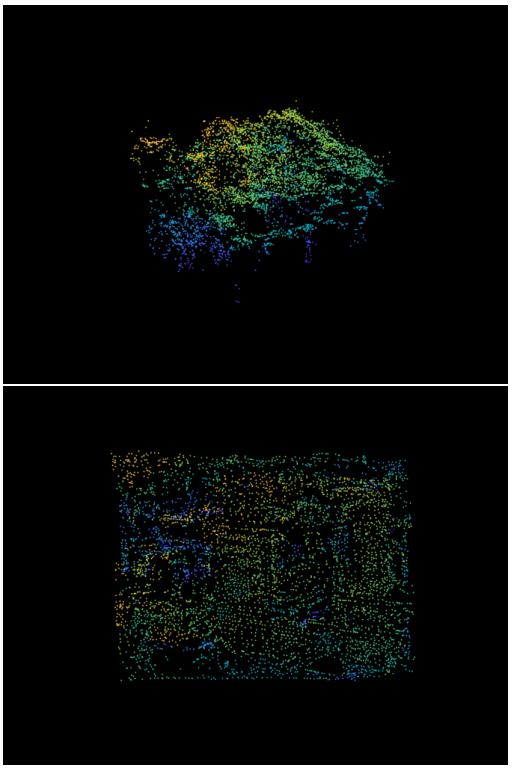

Figure 4: 3D Point Cloud with Optimized Missing Data

Next, to evaluate the results quantitatively, we measure the difference between the ground truth and the estimated image. Before optimization, the error is 250. With each iteration, the error decreases by 13. After 50 iterations, the error reaches 170. Although the run-time for each iteration is around 5 seconds, the error function continues to decrease, indicating that the algorithm would likely produce even better results with more running time.

4.3 Bundle Adjusment

4.3.1 General Steps

We start from our optimized observation matrix using Damped Newton Method. Then, we use the calibration app in MATLAB to find the intrinsics parameters of the iphone XS camera. We estimate the initial values for the camera poses and 3D structure of the scene using structure from motion. Then, we define a cost function that measures the error between the estimated camera poses and 3D structure and the observed image data, which is differentiable with respect to the camera poses and 3D structure. Next, we use the Levenberg-Marquardt algorithm to iteratively update the estimates of the camera poses

and 3D structure in order to minimize the cost function. This algorithm uses a combination of gradient descent and the Gauss-Newton method to efficiently find a good local minimum of the cost function. After the algorithm has converged, the resulting estimates of the camera poses and 3D structure should be more accurate and consistent with the observed image data. These estimates can be used to generate a 3D reconstruction of the scene.

Algorithm 4 Bundle Adjustment with Levenberg-Marquardt (7)

Initialize the 3D structure S and camera poses P
for each image I **do**
Project 3D points onto the image plane using P:

$$p_i^{pred} = f(S, P)$$

Compute the reprojection error ϵ using $\epsilon = \sum_{i=1}^{n} \left| p_i^{obs} - p_i^{pred} \right|^2$:

$$\epsilon = \sum_{i=1}^{n} \left| p_i^{obs} - p_i^{pred} \right|^2$$

Use Levenberg-Marquardt algorithm to minimize the reprojection error ϵ:

$$(S, P) = \arg\min_x \epsilon(x)$$

Update the 3D structure S and camera poses P
end for
Return S and P

4.3.2 Detailed Steps

Initially, we decided to implement bundle adjustment from scratch. As studying bundle adjustment in detail, we found out the basic strategy for bundle adjustment is similar to what we have done for matrix factorization with the damped Newton method. We thought we could finish it on time and kept experimenting. However, due to its huge complexity coming from various techniques used for better performance, we often faced issues in our implementation. As a result, we decided to use the MATLAB built-in function bundleAdjustment to achieve our goal.

Based on the MATLAB example *Structure From Motion From Mutliple Views* (11), we integrated with what we got from our first and second milestones. Like the example, we used the imageviewset object and fed the denser matched features using the estimated missing values from matrix factorization. Then, using the built-in functions, we found the camera poses of our images. At the end, we used the function bundleAdjustment to obtain the refined camera poses.

4.3.3 Evaluation

Figure 3: Bottle Image 1

Figure 4: Bottle Image 2

For bundle adjustment, we use the above bottle images. They look nearly like the same, but the second image is taken after moving slightly. We intend to take these pictures because consecutive image frames in a video input would

be similar.

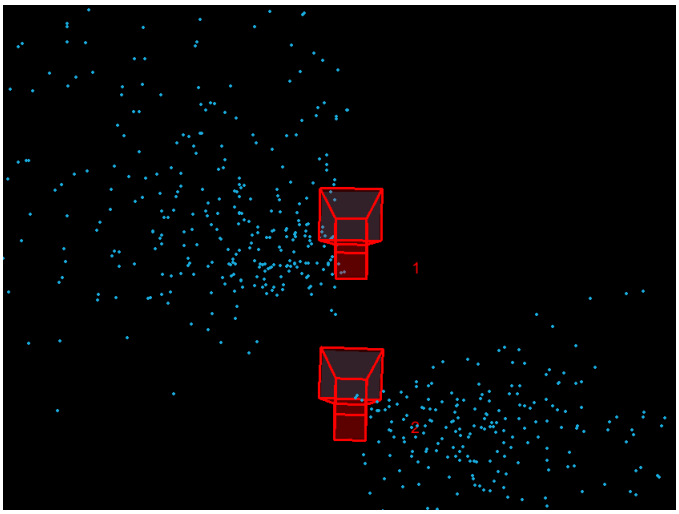

Figure 5: Bundle Adjustment 1

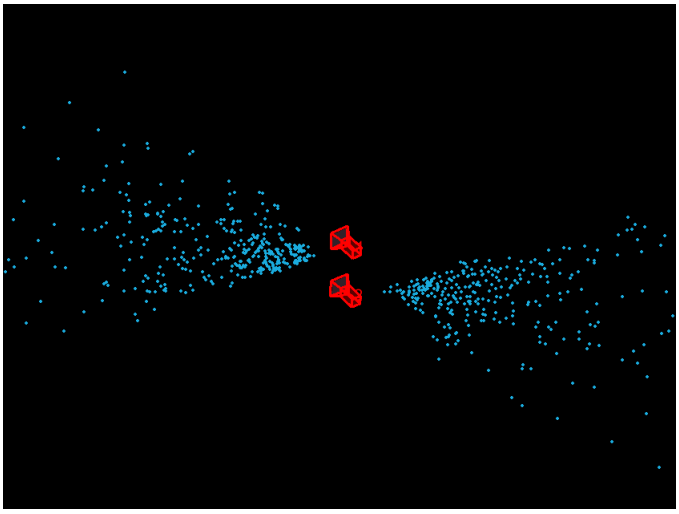

Figure 6: Bundle Adjustment 2

As shown above, the two cameras face in the almost same direction, and there are many matched points in front of the cameras. This is expected because the two images look nearly like the same. However, as shown on the second image, there are some points in the back of the cameras. We think this is because of the inaccurate matched features in our method, and we guess they were refined in a wrong way in the optimization process.

4.4 Integrate Methods with Video Input

4.4.1 General Steps

The last milestone is in general to combine the previous steps together and make a 3D reconstruction based on the previously implemented function. We wouldn't have detailed steps since no algorithm is implemented in this step and all we did is filming and adjust parameters to get good result.

4.4.2 Evaluation

The initial video of the 3815 classroom did not produce satisfactory results because the camera moved too quickly when turning on the corner of the room. In order to save memory in MATLAB, we only kept every fifth frame of the video. This resulted in a lack of matching features betweimagesen the previous and next frames when the camera operator changed the direction of the camera.

As a result, we decided to film again using the 3820 classroom. This classroom was chosen because it was typically less crowded than the 3815 classroom, allowing for a smoother camera movement. The cameraman turned the camera to the right at a significantly slower rate to ensure that the corner was well-captured. We evaluated the result qualitatively since ground truth data was not available.

Despite our efforts to optimize the footage, we were unable to achieve our desired outcome due to limited memory and time. When we tried combining two optimized versions of the video, the program took even more memory and time than expected, leading to failure. Therefore, we tried filming a single bottle on a table instead. We filmed an empty bottle of Tropical Citrus flavored vitamin water with the same camera and downsample it to 0.3 of the original scale.

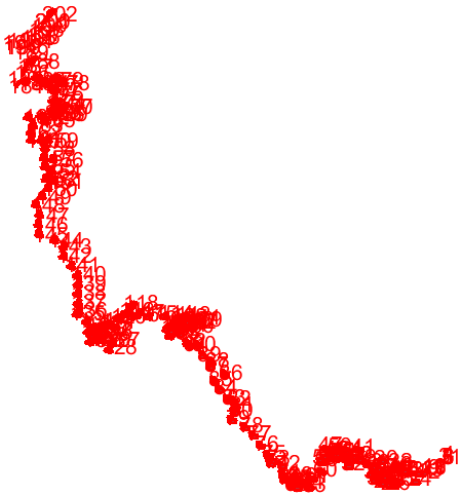

Figure 7: Camera Poses

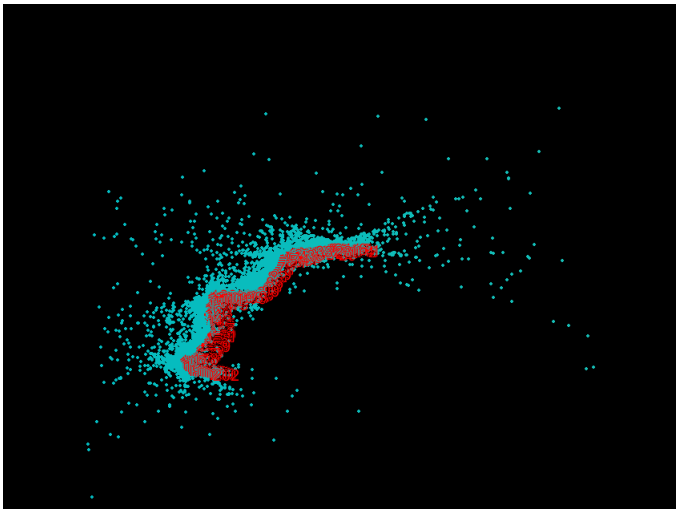

Figure 8: Camera Poses and Feature Points 1

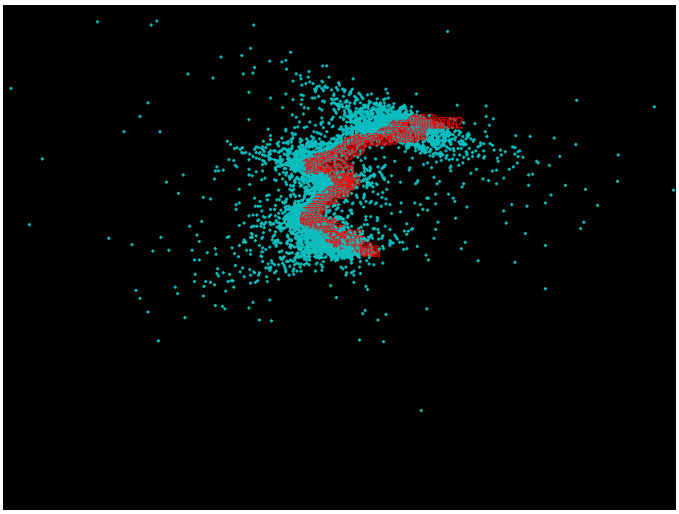

Figure 9: Camera Poses and Feature Points 2

As shown above, camera poses reflect our camera movement well. Even though it is not completely accurate, we can still see the right turning movement on the above images. Due to the limited computer power, the whole video was not able to be processed, but the results seem to correspond to our video.

5 Conclusion

In conclusion, this project presents a method for creating 3D reconstructions of a scene from a video using feature points and matrix factorization with missing data. The damped Newton method is utilized to minimize the difference between observed and predicted values, and bundle adjustment is applied to minimize reprojection error. The final 3D reconstruction is displayed using point cloud visualization in MATLAB. A key finding of this project is that 3D reconstruction requires significant amounts of memory to construct a large scene.

In the future, algorithms that can select images with the most important features or representative matches across frames could help address the issue of large input sizes. This approach assumes that the video is filmed at an appropriate resolution for the program to give good results without using an excessive amount of memory. Another potential approach is to manually select the most representative frames as input to the program. This would provide the most accurate results, but would be tedious and not scalable to a collection of videos.

References

[1] Buchanan, A. and Fitzgibbon, A. (2005). Damped Newton algorithms for matrix factorization with missing data. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pp. 316–322.

[2] Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W. (2000). Bundle Adjustment — A Modern Synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds) Vision Algorithms: Theory and Practice. IWVA 1999. Lecture Notes in Computer Science, vol 1883. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44480-7_21

[3] Weinman, Jerod. "Feature Detection Lab." CSC262: Computer Vision, Grinnell College, 2022, https://weinman.cs.grinnell.edu/courses/CSC262/2022F/labs/feature-detection.html. Accessed December 2nd, 2022.

[4] Weinman, Jerod. "Stereo Disparity Lab." CSC262: Computer Vision, Grinnell College, 2022, https://weinman.cs.grinnell.edu/courses/CSC262/2022F/labs/stereo-disparity.html. Accessed December 2nd, 2022.

[5] Weinman, Jerod. "Post: Structure from Motion.", Grinnell College, 2022, https://piazza.com/class/l758jb036l87kc/post/42. Accessed December 2nd, 2022.

[6] Matthew Brown, Richard Szeliski, and Simon Winder (June 2005). Multi-image matching using multi-scale oriented patches. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'2005), volume I, pages 510-517.

[7] Lourakis, M. I. A. and Argyros, A. A. 2009. SBA: A software package for generic sparse bundle adjustment. ACM Trans. Math. Softw. 36, 1, Article 2 (March 2009), 30 pages. DOI = 10.1145/1486525.1486527 http://doi.acm.org/10.1145/1486525.1486527

[8] "Middlebury Dataset." Middlebury College, n.d. https://vision.middlebury.edu/mview/data/. Accessed December 2nd, 2022.

[9] David Levin, "Matrix Derivatives: What's up with all those transposes ?" YouTube, 12 Mar. 2012, https://www.youtube.com/watch?v=ny-i8_9NtHA. Accessed December 2nd, 2022.

[10] Cesareo (https://math.stackexchange.com/users/397348/cesareo), Matlab: Gradient and Hessian of a function with vector input of user specified size, URL (version: 2019-08-29): https://math.stackexchange.com/q/3336189

[11] MathWorks. "Structure from Motion from Multiple Views." Accessed December 11, 2022. https://www.mathworks.com/help/vision/ug/structure-from-motion-from-multiple-views.html