3D Metric Reconstruction with Auto Calibration Method

CS 283 Final Project

Tarik Adnan Moon
tmoon@collge.harvard.edu

Abstract

In this paper, different methods for auto camera calibration have been studied for metric 3D reconstruction from a set of images. Theoretical framework from different computer vision literatures have been evaluated, and finally factorization based methods have been implemented for metric construction of a scene from a number of images taken by uncalibrated cameras. Then results are presented for the collected image sets and the performance has been evaluated.

1 Introduction

In the advent of affordable 3D printing technology, 3D metric construction is becoming increasingly important. Moreover, as other similar fields such as robot vision, robot navigation etc. are becoming increasingly popular, and we are having more and more image capturing capabilities in the everyday devices we use, finding a good solution to camera calibration and metric reconstruction without any information about the camera can be important.

In this project, we have studied several relevant computer vision literatures that tried to solve problems such as auto-calibration, metric reconstruction etc. separately. This project tried to unify those approaches and find find a simple, but effective system for recovering 3D reconstruction from an image set.

In this paper, first we study different theoretical paradigms for metric reconstruction and the discuss the relations of those algorithms with classic projective geometry. Then we discuss the algorithmic framework and the details of the implementation. Finally we conclude and discuss the results we found from the implementation and we propose a few possible future improvements that can be done to this implementation.

2 Notation and Basic Principle

In this section, we will briefly discuss the mathematical notations and principles used in the paper. This paper follows the notations of Zisserman *et al.* very closely. We know from [1] that a camera matrix P can be decomposed into

$$P = K[R| - RC]$$

where K is the camera calibration matrix, R the rotation matrix, and C the matrix representing the center of the world frame.

2.1 3D to 2D Mapping Process in Cameras

So for any 3D point X in the world frame, the camera matrix P projects it onto a 2D plane and gives a 2D point x (and we can "undo" the effect by the inverse transformation) . Mathe-

matically, this can be written as

$$x = PX$$

However, this projective mapping is true upto a constant depth factor λ. i.e. in the metric space

$$\lambda x = PX$$

In this paper, our main challenge is finding the camera matrix to recover the true 3D structure from multiple views. For that purpose we use matrix factorization techniques and a few results from basic projective geometry.

2.2 Some Well Known Results

In the next section, we will be using the following well known results from projective geometry. All of them are proved in Zisserman *et al.* [1]. So we will just state the results and leave the results to be verified from the book or by the readers.

Result 1. *If ω^* is the dual image of the absolute conic, and K the camera calibration matrix, then*

$$\omega^* = KK^T$$

(p. 210)

Result 2. *If ω^* is the dual image of the absolute conic, and Ω^*_∞ the absolute dual quadric then*

$$\omega^* = P\Omega^*_\infty P^T$$

(p. 201)

Result 3. *In euclidean frame Ω^*_∞ has the canonical form*

$$\Omega^*_\infty = HI'H$$

where

$$I' = \begin{bmatrix} I_3 & 0 \\ 0^T & 0 \end{bmatrix}$$

(p. 83)

Result 4. *Ω^*_∞ can be represented by*

$$\Omega^* = \begin{bmatrix} KK^T & -KK^Ta \\ -a^TKK^T & a^TKK^Ta \end{bmatrix}$$

where the plane at infinity

$$\pi_\infty = \begin{bmatrix} a^T \\ 1 \end{bmatrix}$$

(p. 464)

3 Mathematical Framework and Algorithms

The mathematical framework needed for the project to be implemented can be divided into 3 parts. The first step is matching the 2D points on the images using SIFT matching techniques and get a get of 2D points. The second step is finding the projective depth and projective camera matrix using factorization techniques. The final step is finding the camera calibration matrix by using auto-calibration techniques and from that infer the 3D metric reconstruction. As we did not implement SIFT and rather used the openCV implemented version. So we'll just discuss the algorithmic framework for the other two steps:

3.1 Projective Reconstruction and Depth Recovery

For this part we used the result discussed by Strum *et al.* in [2]. For a set of $m \times n$ points x_j^i where each n points are taken by a camera with camera matrix P^i, for the representative 3D world point X_i we have (see section 2.1)

$$\lambda_j^i x_j^i = P^i X_j$$

stacking the relations we get the following matrix equation:

$$W = \begin{bmatrix} \lambda_1^1 x_1^1 & \cdots & \lambda_n^1 x_n^1 \\ \vdots & \ddots & \vdots \\ \lambda_1^m x_1^m & \cdots & \lambda_n^m x_n^m \end{bmatrix} = \begin{bmatrix} P^1 \\ \vdots \\ P^m \end{bmatrix} \begin{bmatrix} X_1 & \cdots & X_n \end{bmatrix}$$

First for getting the depth information and recovering the projective reconstruction we use the following algorithm by [1]:

1. First we normalize the pixel values using isotropic scaling (i.e. make sure that the RMS distance from mean is $\sqrt{2}$ (see [1] p.109).

2. Start estimating projective depth. Here we start with all $\lambda_i = 1$ and continue with iterative scaling.

   (a) Rescale each column k so that
   $$\sum_{i=1}^{3m} w_{ik}^2 = 1$$

   (b) Rescale each triplet of wors $(3k-2, 3k-1, 2k$ such that
   $$\sum_{k=1}^{n} \sum_{i=3k-2}^{3k} w_{ik}^2 = 1$$

   (c) repeat (a), (b) if the entries of the matrix changes significantly. For our implementation we repeated if our RMS difference between the previous and the current matrix exceeded 5%.

3. Finally we compute an SVD of W to find

$$W = UDV^T$$

Now we let D' to be the D matrix except all but the first four diagonal entries to be set to 0 i.e.

$$D = diag(\sigma_1, \cdots, \sigma_4, 0, \cdots, 0)$$

Then from this factorization we estimate:

$$\begin{bmatrix} P^1 \\ \vdots \\ P^m \end{bmatrix} = UD'$$

and

$$\begin{bmatrix} X_1 & \cdots & X_n \end{bmatrix} = V^T$$

3.2 Auto-calibration and Metric Reconstruction

The problem with projective reconstruction is that it is very different from the actual real-world euclidean reconstruction. So the metric reconstruction differs from it by a 4×4 homography matrix H, and this can be easily seen from the following argument:
If $\{P^i, X_j\}$ is a projective reconstruction then for any homography H, $\{P^i H, H^{-1} X_j\}$ works too.

So we need to find a rectifying homography H. For this part we used the result by Pollefeys *et al.* in [3] and the basic projective geometry facts discussed in section 2.2. Using results 1 and 2 from section 2.2 we can write

$$\omega^* = KK^T = P\Omega_\infty^* P^T .....(*)$$

Now using result 3 and 4, we have

$$\Omega^* = \begin{bmatrix} KK^T & -KK^T a \\ -a^T KK^T & a^T KK^T a \end{bmatrix}$$

where

$$\pi_\infty = \begin{bmatrix} a^T \\ 1 \end{bmatrix}$$

Then we assume that the camera has skew, $s = 0$, known pixel ratio $\frac{f_x}{f_y} = r$ [1], and principal point at origin $\alpha_x = \alpha_y = 0$. Although these are not necessarily exact results, for the sake of finding linear approximation these are reasonable assumptions to make. Then normalizing by

[1] In general the pixel ratio of cameras are mostly standard, the DSLRs have 2:3 ratio, cellphone and other low end cameras have 3:4 pixel ratio. In general it can be found by taking the dimension ratio of any uncropped photo from a camera

the pixel ratio, we can write our camera calibration matrix as

$$K = \lambda \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

then we can simplify equation (*) as

$$\lambda^2 \begin{bmatrix} f_i^2 & 0 & 0 \\ 0 & f_i^2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = P\Omega_\infty^* P^T = P_i \begin{bmatrix} b_1 & 0 & 0 & b_2 \\ 0 & b_1 & 0 & b_3 \\ 0 & 0 & 1 & b_4 \\ b_2 & b_3 & b_4 & b_5 \end{bmatrix} P_i^T$$

Now comparing the sides we have the following relations

$$\begin{aligned} \omega_{11}^* =& \omega_{22}^*, \\ \omega_{11}^* =& \omega_{22}^* = \omega_{11}^* = 0, \\ \omega_{11}^* =& \omega_{22}^* = \omega_{11}^* = 0 \end{aligned}$$

and due to the symmetry in the last to equations we have the following matrix equations

$$\begin{aligned} P_i^{(1)} \Omega^* {P_i^{(1)}}^T =& P_i^{(2)} \Omega^* {P_i^{(2)}}^T \\ P_i^{(1)} \Omega^* {P_i^{(2)}}^T =& 0 \\ P_i^{(1)} \Omega^* {P_i^{(3)}}^T =& 0 \\ P_i^{(2)} \Omega^* {P_i^{(3)}}^T =& 0 \end{aligned}$$

(where $P_i^{(i)}$ means the i-th column of P matrix.) When we have at least 3 non-degenerate views we can easily solve this system as we have $4(n-1)$ independent linear equations in b_i (as Ω_∞^* is parametrized by b_i's).

4 Implementation Details

In this section, we will discuss the whole implementation process– from image collecting to 3D model construction and visualization.

4.1 Image Set Collection

We got two sets of images from the Oxford University Visual Geometry group– we took the dinosaur (toy) and model house datasets for images. Then we took out own images with a Canon DSLR camera with 50mm lens and an iPhone. For this project we just decided to keep the photos taken by the DSLR due to their superior image quality and due to our goal of keep the calculation simple. However, as we have seen this can be organically extended to more general case.

4.2 Implementation in Python

We implemented the aforementioned algorithms using Python. We used numpy, scipy, and openCV libraries. The first two were used for doing efficient matrix calculations and the the last one was used for image manipulation and vision. More precisely, we used opencv to find feature matches between images using SIFT. Then we also used them to find the calibration for the camera we used to find how good our auto calibration performed.

4.3 Visualization with CMVS

Finally as our project did not involve any good visualization methods for showing the actual 3D metric constructions we had to use CMVS library to plot the 3D points and the 3D reconstructed versions with textures.

5 Results

In general it is very hard to test the accuracy of these models because without some sophisticated methods for tracking the points on the 3D plane it is hard to measure the absolute accuracy of the metric reconstruction. However, luckily for the first and second image set (dinosaur and model house) the camera matrix was given with the dataset. And for the second dataset (model house) the 2D and 3D points were given from a metric reconstruction. Then for the third image set (glue bottle), we used openCV camera calibration module for finding the camera calibration matrix.

Our camera had the following RMS error rates (compared to the ones found using direct camera calibration):

- First set: 3.2%
- Second set: 3.1%
- Third set: 4.7%

The reason for increased error in the third set is that most of the glue bottle was white colored and the SIFT algorithm was confused when it was detecting and matching features.

And then we computed the error in 3D metric reconstruction for the second set (we had 2D to 3D correspondence data only for this set) and we found 5.6% error it terms of RMS distance.

While our result is worse than the result found by Pollefeys *et al.*, we used much simpler linear model instead of the quadratic approach taken by the paper. Moreover, our result is more robust and did not use any assumption that the turntable model did (in the case of the Visual geometry group).

As we did not implement any visualization ourselves in the project, we used CMVS [6] just for demonstrating the 3D reconstruction. Here are the visualization of the 3D reconstructions:

Figure 1: 36 images of dinosaur

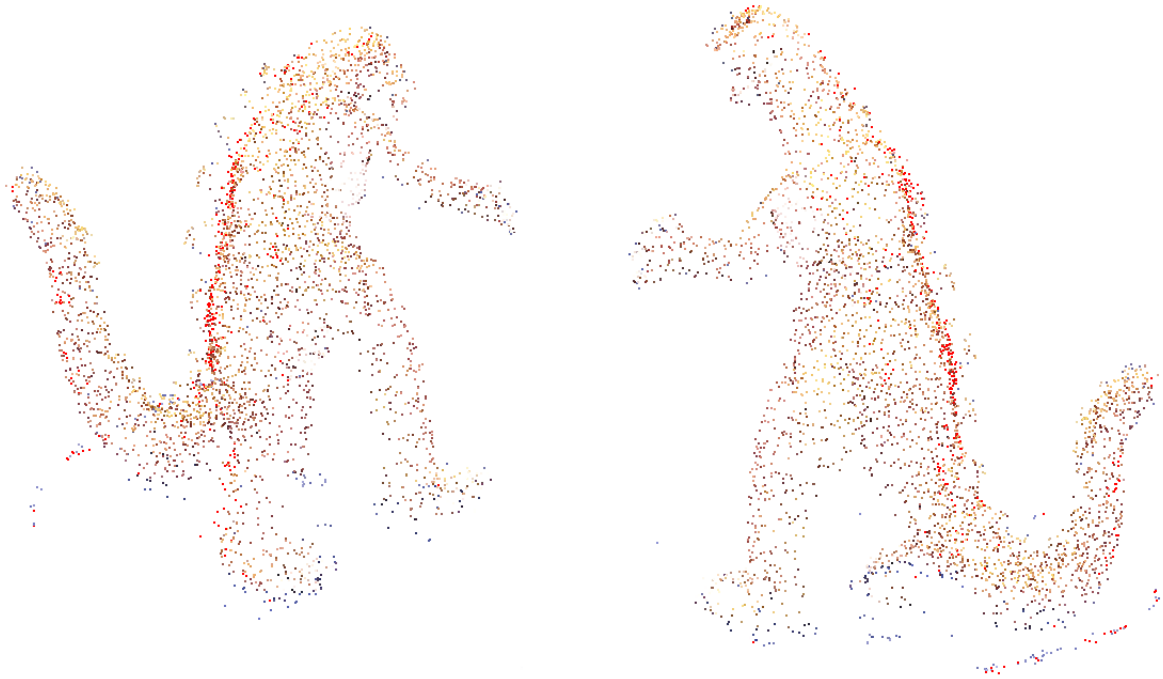

Figure 2: 3D point plotting

We found that the dinosour model worked pretty well because it had a lot of unique features that was extracted by the SIFT.

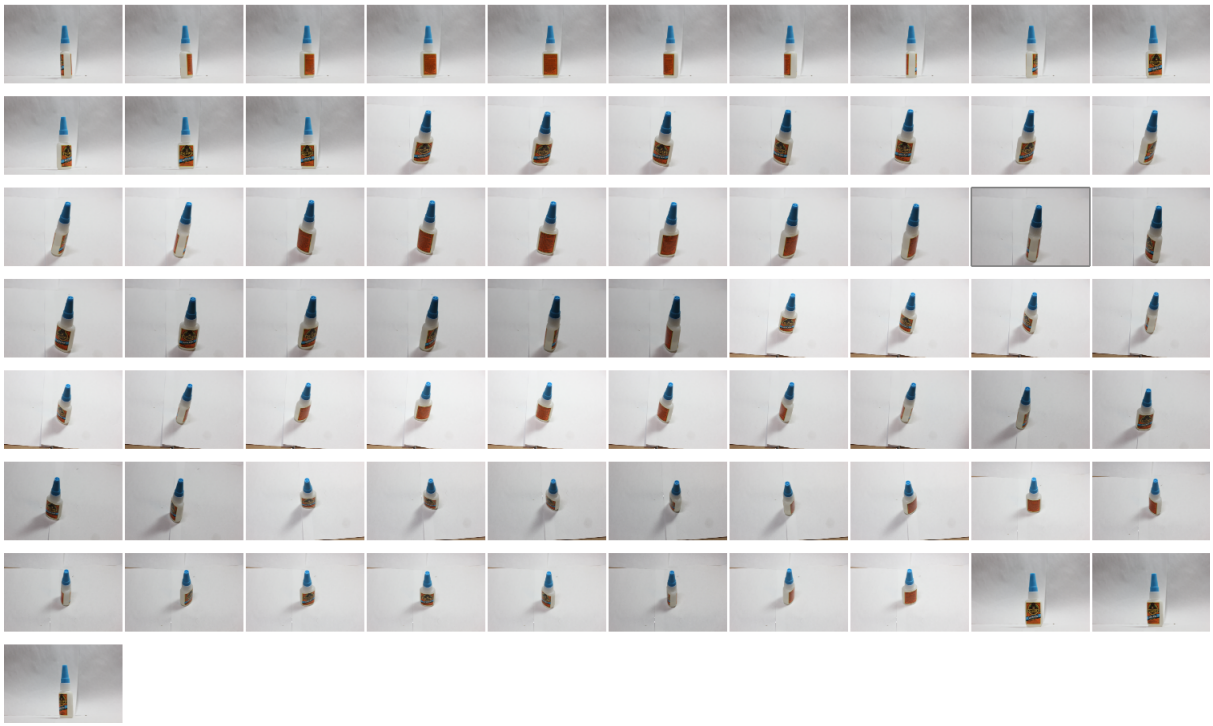

Figure 3: 81 images of the glue bottle

Figure 4: 3D reconstruction of the glue bottle with texture mapping (using CMVS)

6 Future Improvements

There are several future improvements that we could not implement due to their complexity and time limitations of the project. In this section we will discuss those possible future improvements and how they could possibly improve the results.

6.1 Pre-determining the projective depth mapping:

We start with a projective depth of $\lambda_i = 1$ as our first step of projective reconstruction. However, that might take a large number of iterations to converge with the repeated scaling method we described in section 4 (in fact, theoretically it is not guaranteed that this approach will ever

converge). So there is a better method to precompute a projective depth mapping using the epipole and fundamental matrix by Strum *et al.*. Although the pre-computed projective depth mapping is true upto a scalar factor, we can use the same iterative method to improve our result more accurately using the method, and this time it is guaranteed that the values will converge.

6.2 Bundle adjustments

We could also use bundle adjustemnt to reduce the geometric error of projective rectification and the calibrated camera matrix. That way we would be able to further reduce the estimation error.

6.3 Other Possible Algorithms

There are a few other algorithms that are not as general as the approach we studied, but are reportedly more accurate for metric reconstruction. For example, we could use the rotating camera model, or turn-table model for finding the camera calibration matrix and from there we could find the metric reconstruction using the approach we described in the paper.

6.4 Parallel computing

The process of auto calibration (especially SIFT matching, matrix factorization, iterative scaling) is a computationally very expensive. It took us about 10 minutes to complete the computation with a set of 81 images. So the process is not actually real time. So we can use parallel computing, especially GPU computing environment such as CUDA, to make this process extremely fast

Acknowledgment

We would like to thank Professor Todd Zickler for helpful instructions and insights about the project. We would also like to thanks TF Ioannis Gkioulekas and Ying Xiong for helping to understand the concepts. Finally, we would thank our peers in the CS 283 class for their helpful discussions and questions.

References

[1] Hartley, R. I. and Zisserman, A. Multiple View Geometry in Computer Vision. Second edition, 2004. Cambridge University Press, ISBN: 0521540518

[2] Peter Sturm and Bill Triggs. A factorization based algorithm for multi-image projective structure and motion. *4th European Conference on Computer Vision, Cambridge, England*, April 1996, pp. 709-720 .

[3] Marc Pollefeys, Reinhard Koch and Luc Van Gool. Self-Calibration and Metric Reconstruction Inspite of Varying and Unknown Intrinsic Camera Parameters. *International Journal of Computer Vision* 32(1), 725 (1999)

[4] Motilal Agrawal. Practical Camera Auto Calibration using Semidenite Programming

[5] University of Oxford Visual Geometry Group Dataset.
http://www.robots.ox.ac.uk/ vgg/data/data-mview.html

[6] Yasutaka Furukawa and Jean Ponce. Accurate, Dense, and Robust Multi-View Stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 32, Issue 8, Pages 1362-1376, August 2010.

[7] The OpenCV Library *Dr. Dobbs Journal of Software Tools (2000)* by G. Bradski