Refining 3D Reconstruction Of Stereo Camera Images With Least Squares

Jacob Forster, Joseph Jung and Andrew Turnblad
Computer Science
St. Olaf College
Northfield, MN 55057
forsterj@stolaf.edu, jungj@stolaf.edu, turnblad@stolaf.edu

Abstract

In computer vision, a least-squares minimization technique known as bundle adjustment is used to optimize the camera parameters necessary for 3D reconstruction of a scene. Starting with a bundle adjustment technique for stereo camera models implemented by Bonde et al, we employ auto-differentiation to reduce code size by over 50% while eliminating the need to pre-compute the Jacobian of the stereo projection function. We utilize *Ceres-Solver* to expand the capabilities of the previous optimizer implemented using *sba*, allowing us to efficiently and concurrently optimize values for up to 12 intrinsic and extrinsic camera parameters in at least 5 different configurations. The use of *Ceres-Solver* achieves similar final reprojection errors to the previous implementation and reduces the time necessary to optimize the many parameters of the stereo camera model by over 95%.

1 Introduction

Camera calibration is a vital step used to determine the parameters of a camera necessary for extracting metric information from 2D images and translating 2D points to their respective 3D counterparts. After estimating the camera parameters using a closed form solution relying on epipolar geometry, these parameters must be refined using bundle adjustment, a form of maximum likelihood estimation, in order to minimize the reprojection error. In other words, the camera parameters, scene parameters, and coordinates of particular points in the scene, all of which influence the metric information in the 3D reconstruction of the scene, must be jointly refined in order to minimize the deviation of the reconstructed 3D output from the actual 3D structure of the scene.

Due to the nature of our particular 3D reconstruction problem, the number of parameters that must be jointly optimized leads to a very slow and computationally prohibitive program in its current form. In particular, by utilizing a stereo rig, not only are the motions of the two cameras used for taking photos constrained, but the number of parameters that must be optimized is doubled. In addition, the utilization of camera auto-focus, as well as the presence of slight and unintentional, although largely systematic, deviations from the expected location of the cameras when capturing images, further complicates the data set that must be refined. Therefore, building on the work of previous research, we investigate refinement techniques using the Levenberg-Marquardt nonlinear least squares minimization algorithm that allow simultaneous and flexible optimization of all camera, scene and feature point parameters in an efficient manner. We investigate two methods: one that employs *sba*, a C library for performing camera calibration; and one that takes advantage of *Ceres-Solver*, a newer C++ minimization library developed and used by Google.

2 Background

2.1 Pinhole Camera Model

In order to understand the relationship between the 2D information in an image and the metric information necessary to reconstruct a 3D scene, we utilize the pinhole camera model. In this paradigm, it is assumed that all light passes through a small optical center, or ”pinhole”, to produce an inverted image on the ”retinal plane” of the camera’s image. However, because this model doesn’t take into account any undesirable side-effects of modern cameras, such as lens distortion or blurring, its accuracy depends on the quality of the camera calibration process and, as per the motivation for this paper, subsequent refinement of the parameter estimations produced through the model.

In this paradigm, to ”project” the light through a ”pinhole” onto a 2D scene, the 3D coordinates of a given point must be multiplied by the intrinsic camera parameter matrix

$$\mathbf{A} = \begin{pmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

and extrinsic camera parameter matrices $[\mathbf{R}\ \mathbf{t}]$, where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{t}$ is a 1×3 translation matrix. Here, α_u and α_v are the magnifications, relating to the focal length of the camera in the two image coordinate directions, u_0 and v_0 are the coordinates of the principle point (the point at which the image axis meets the retinal plane) and γ is a skew factor (which we will assume to be zero for our case). Thus, the relationship between a point in a retinal image plane, produced by a camera, and a point in 3D space can be succinctly expressed by

$$s\mathbf{m} = \mathbf{A}[\mathbf{R}\ \mathbf{t}]\mathbf{M} \tag{1}$$

where s is a scaling factor and $\mathbf{m} = (u, v, 1)$ and $\mathbf{M} = (x, y, z, 1)$ are the homogeneous coordinates in 2D and 3D space respectively. [5]

2.2 Stereo Camera Model

A stereo camera system is a similar modeling involving two pinhole camera models. An image taken from a *stereo* camera system involves three categories of parameters which may be adjusted, many of which reflect the original pinhole model:

$$\text{per-camera parameters} = \begin{cases} \text{focal length } (f_x, f_y) \\ \text{principal point } (u_0, v_0) \\ \text{translation } (x, y, z) \\ \text{rotation } (a, b, c) \\ \text{distortion coefficients } (k_0, k_1, p_0, p_1) \end{cases}$$

$$\text{per-pair parameters} = \begin{cases} \text{translation } (x, y, z) \\ \text{rotation } (a, b, c) \\ \text{scale } (s) \end{cases}$$

$$\text{per-corner parameters} = \begin{cases} \text{position } (x, y, z) \end{cases}$$

Here, a *camera* refers to one physical device. The extrinsic parameters are relative to the location of the pair. There are 14 parameters for each of the left and right cameras. The *pair* is the position and orientation of the rig on which the two cameras are placed to construct the stereo camera. There is additionally another rotation which is introduced as a correction factor for the right camera, for a total of 10 pair parameters. Every image taken is treated as being taken by a pair, so the pair parameters correspond to every pair of images. A *corner* is an actual real-world coordinate which is observed.

Every *observed* point, indicated by $\boldsymbol{x} = (u, v)$, is related to each *corner*, $\boldsymbol{X} = (x, y, z)$, by the projection function

$$\boldsymbol{x} = \boldsymbol{M}(\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{X})$$

where $\boldsymbol{C}$ represents the set of relevant camera parameters and $\boldsymbol{P}$ the pair parameters. Modifying Equation 1 for the stereo model yields

$$\boldsymbol{x_i} = \boldsymbol{K_i}[\boldsymbol{R_i}\ \boldsymbol{t_i}]\boldsymbol{X} \tag{2}$$

where $\boldsymbol{K}$ consists of the camera intrinsic parameters and i denotes a particular camera.

2.3 Parameter Refinement

Although Equation 2 is used to make the initial estimates for the intrinsic and extrinsic camera parameters, the parameters can be refined by minimizing the reprojection error, or the distance between the actual location of corners in an image and their predicted location. The objective is to obtain the optimal values for all involved parameters such that we minimize the reprojection error

$$\text{cost} = \sum_{i,j,k} |\boldsymbol{m}_{ijk} - \boldsymbol{M}(\boldsymbol{C}_i, \boldsymbol{P}_j, \boldsymbol{X}_k)|^2 \tag{3}$$

where the subscripts denote the ith camera, jth pair and kth corner. Also included in the projection function are the distortion parameters, k_1 and k_2, where the distortion alters the image in the radial direction as described in [5].

3 Previous Work

The work done in this paper extends upon work done by previous investigators. Namely, Bonde, Brumfield and Yuan implemented sparse bundle adjustment across images captured using a two-camera stereo rig by utilizing *sba*, a flexible C library that implements sparse bundle adjustment using the Levenberg-Marquardt algorithm. [4] [2]

Interestingly, the distortion coefficients are not directly considered in the modified reprojection error as the feature points have the distortion effect removed using a simple formula before utilization in the minimization.

The three primary limitations in the parameter refinement implementation employed by Bonde et al. [2], which the following solutions will aim to address, are:

1. The numerous derivatives used in calculating the Jacobian of the projection function necessary in the minimization are derived using symbolic mathematics software (e.g. MatLab) and then converted into C++ code by hand. Therefore, the Jacobian is calculated analytically by *sba*. Although this leads to very fast and exact calculations, it makes changing the parameters that are being optimized, changing the projection model, or fixing any bug in the minimization procedure extremely difficult.

2. The intrinsic parameters are held constant throughout the refinement of parameters. Again, this leads to a very fast minimization procedure, but it is at the expense of the more refined parameters necessary for an accurate 3D reconstruction.

3. Partially occurring as a byproduct of limitation two, the distortion coefficients are not only held constant throughout the parameter refinement, but are not included in the projection function. Specifically, the expected 2D coordinates of a given feature point in each retinal image plane are un-distorted before optimization and used as the expected 2D coordinates in Equation 3.

4 Proposed Solutions

The first modification made to the previous work was the use of the C++ package *Adept* to calculate the Jacobian of the projection function. [3] Instead of building up the derivatives of the projection function with respect to the many parameters using symbolic mathematics software and then copying the final result into C++ code, we constructed the projection function in C++ using a custom-built matrix library and *Adept*'s auto-differentiation capabilities. This implementation solves limitation 1 listed above, while offering the flexibility to solve limitations 2 and 3. It is also much easier to debug and results in almost negligible overhead in the total time the procedure takes to optimize the parameters.

Although the use of auto-differentiation via *Adept* was one step towards solving limitations 2 and 3, it did not, by itself, solve either limitation or offer the flexibility to easily switch which parameters (extrinsic or intrinsic) were optimized in a given run. To that end, the following two solutions are proposed.

4.1 sba : Per Pair Optimization

In order to optimize both the intrinsic and extrinsic parameters of the cameras, the simplest solution is to simultaneously optimize the intrinsic and extrinsic parameters. That is, instead of holding the intrinsic parameters constant, allow them to be optimized *per pair* rather than *per camera*.

One downside of this implementation is the large number of individually optimized values - the number of intrinsic parameters is now multiplied per pair - and as a result, the extended amount of time that it will likely take to optimize them. Similarly, if the minimization even converges with this many parameters, there is a risk of overfitting since each pair of images likely does not offer enough data to optimize this many parameters. Additionally, this approach wouldn't ordinarily make sense on a physical level since the intrinsic parameters should be constant across all images; however, when we consider that auto-focus may have been involved when taking the images, this approach may be justified.

4.2 Ceres-Solver

Ceres-Solver, another open source C++ library for solving large scale non-linear least squares problems, offers an attractive alternative to *sba*. [1] In particular, *Ceres-Solver* is flexible enough in its implementation of parameters that, unlike *sba*, the user is allowed to explicitly and easily set how often a given set of parameters should be optimized. In this way, instead of requiring a second minimization procedure for the intrinsic parameters, one minimization procedure can be used to optimize all parameters at the same time. In theory, this solution contains the best of solution 4.1 and should yield a good balance between minimizing both reprojection error and run time.

The extensive documentation available for *Ceres-Solver* renders the implementation of this minimization, based off of the previous *sba* implementation described in [2] trivial. It is important to note however that, to take advantage of the built-in auto differentiation capabilities of *Ceres-Solver*, the projection function used by Bonde et al can be split into

two different projection functions (one for the left camera and one for the right), which *Ceres-Solver* calls "Residual Cost Functions". Because the projection function of each camera in a pair involves only the camera's intrinsic matrices and the pair's extrinsic matrices, the mathematical relationships between, and the constraints imposed upon, the left and right cameras are preserved.

5 Results

For all of the results, we will be using the final reprojection error summed over all feature points (total cost) as a measurement of the accuracy of the parameter refinement routine. We will also be using the relative timing between the tests as an indication of the efficiency of the parameter refinement routine, as all tests were performed on the same hardware.

5.1 sba : Per Pair Optimization

As expected, using the parameter refinement method that optimizes intrinsic parameters per pair of images resulted in a much slower routine than holding the parameters constant. In fact, as Table 5.1 indicates, optimizing even two more intrinsic parameters resulted in a 432% increase in the time it took *sba* to complete. Table 5.1 summarizes the results we obtained using this method. Note that all of these tests were forced to finish after 100 iterations even if they had not yet converged since optimization beyond that point leads to very little decrease in reprojection error relative to the increase in time. In Table 5.1 and the following tables, x, y and z denote the displacement of the camera from the location of the pair while a, b and c denote the Euler angle rotations of the camera relative to the pair.

Intrinsic Parameters Optimized	Total Cost	Total Time (s)
None	21,074.6	13.73
f_x, f_y	21,712.7	59.31
u_0, v_0	19,590.7	59.33
f_x, f_y, u_0, v_0	11,752.7	121.24
$f_x, f_y, u_0, v_0, x, y, z, a, b, c$	29,733.2	289.24

Table 1: 72 Camera Pairs and 64 Feature Points

As Table 5.1 indicates, the total reprojection error was, in the best case (optimizing f_x, f_y, u_0, and v_0), about halved using this method. Unfortunately, this lower reprojection error came at the expense of the time it took the routine to complete. Additionally, since the intrinsic parameters were optimized on a per pair basis, it is likely that the parameters were overfitted. Support of this conclusion can be seen in the fact that the total cost increased dramatically when all intrinsic parameters were included. In other words, it is not surprising that the reprojection error did not decrease when x, y, z, a, b and c were allowed to vary since these parameters are relative to the extrinsic translation and rotation of the pair and, as such, should actually be constant. The dramatic increase

in the time it took for the minimization to complete, while resulting in a higher total cost, alludes to the overabundance of possible variables to optimize.

5.2 Ceres-Solver

Due to its flexible API, *Ceres-Solver* not only resulted in a better refinement of parameters, but also took the least time for the reprojection error to converge. In fact, when no intrinsic parameters were optimized, *Ceres-Solver* resulted in the same total reprojection error as *sba* but took only 3.8% of the time. Furthermore, in the best case, using *Ceres-Solver* to optimize intrinsic parameters once across all image pairs resulted in a 3.6% decrease in reprojection error as compared to holding the parameters constant. Table 5.2 summarizes the results of using *Ceres-Solver* and accounting for distortion using the same technique as Bonde et al. Table 5.3 displays the results of initially setting k_0 and k_1 equal to zero and incorporating distortion into the projection function. Note that these tests were forced to finish after 1000 iterations for the same reasons as specified in §5.1.

Intrinsic Parameters Optimized	Total Cost	Total Time (s)
None	21,079.4	.53
f_x, f_y, u_0, v_0	20,682.1	.41
$f_x, f_y, u_0, v_0, x, y, z, a, b, c$	20,338.5	.79

Table 2: 72 Camera Pairs and 64 Feature Points (Pre-Distortion)

Intrinsic Parameters Optimized	Total Cost	Total Time (s)
None	40,492	.12
k_0, k_1	40,383.4	.12
f_x, f_y, u_0, v_0	23,649.5	1.02
$f_x, f_y, u_0, v_0, k_0, k_1$	23,611.8	1.02
$f_x, f_y, u_0, v_0, k_0, k_1, x, y, z, a, b, c$	23,420.1	1.23

Table 3: 72 Camera Pairs and 64 Feature Points (Optimizing Distortion)

While *Ceres-Solver* was able to reduce the total cost while using the distortion technique of Bonde et al by optimizing intrinsic parameters, perhaps the most surprising result is the increase in total cost when optimizing the distortion coefficients as part of the projection function. It is possible that this could be due to initially estimating k_0 and k_1 to be zero, but the high quality nature of the cameras that were used and the fact that the optimization barely changes these coefficients (on the order of 10E-13 or less) indicates this is not the issue. In fact, as shown in the tables below, when *Ceres-Solver* was used to optimize a much larger set of feature points and camera pairs, the total cost and total time to complete were both improved by including and optimizing the distortion coefficients in the projection function. Therefore, it is possible that the distortion coefficients used in the procedure of Bonde et al were, in a sense, overfitted to that small data set when they were estimated.

Intrinsic Parameters Optimized	Total Cost	Total Time (s)
None	217,137.60	15.1
f_x, f_y, u_0, v_0	180,997.2	36.3
$f_x, f_y, u_0, v_0, x, y, z, a, b, c$	184,496.7	26.6

Table 4: 22 Camera Pairs and 1380 Feature Points (Pre-Distortion)

Intrinsic Parameters Optimized	Total Cost	Total Time (s)
None	203,204.4	13.4
$f_x, f_y, u_0, v_0, k_0, k_1$	144,606.8	35.1
$f_x, f_y, u_0, v_0, k_0, k_1, x, y, z, a, b, c$	174,494.5	10.7

Table 5: 22 Camera Pairs and 1380 Feature Points (Optimizing Distortion)

Note that when a larger data set is used, the robustness of solution 4.2, instead of holding the intrinsic parameters constant or using solution 4.1, becomes more apparent. Namely, using *sba* and holding intrinsic parameters constant to optimize the larger data set resulted in a total cost of 236,571 after 17.4 seconds. On the other hand, using *Ceres-Solver* and allowing both the intrinsic parameters and the distortion coefficients to be optimized as part of the projection function decreased the total cost by 39% while increasing the total time by only 201%.

6 Conclusion

Using *Ceres-Solver* to jointly optimize the distortion coefficients and intrinsic parameters of the two cameras in our stereo model significantly improved, especially on a larger data set, the reprojection error produced in the technique utilized by Bonde et al. The benefits of the lowered reprojection error, as well as the increase in flexibility that *Ceres-Solver's* API offers, negates the slight increase in time that accompanies this refinement technique. Extending the results of this paper to compare not only the reprojection error, but the actual 3D reconstruction that results from each refinement technique would be useful and could lead to insights not apparent from the reprojection error alone. Including a form of tangential distortion instead of just radial distortion could offer improved results and would be a natural extension as well.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[2] Luke Bonde, Allison Brumfield, and Ye Yuan. Error minimization in 3-dimensional model reconstruction using sparse bundle adjustment and the levenberg-marquardt algorithm on stereo camera pairs.

[3] R. J. Hogan. Fast reverse-mode automatic differentiation using expression templates in c++. *ACM Trans. Math. Software*, 40(1):1–26, 2014.

[4] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[5] Zhengyou Zhang. *Emerging Topics in Computer Vision*, chapter Camera Calibration, pages 4–43. Prentice Hall Professional Technical Reference, 2004.