ECSE 4961 Project 2

Aidan Westphal

October 2023

Purpose

The purpose of this project was to implement a linear camera calibration method to recover intrinsic and extrinsic camera parameters given a set of 2D/3D calibration points.

Theory

Recall the original equation for full perspective projection:

$$\lambda \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = W \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

We can multiply through W and [R T] to obtain a projection matrix P as follows:

$$\lambda \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x\vec{r_1} + c_0\vec{r_3} & fs_xt_x + c_0t_x \\ fs_y\vec{r_2} + r_0\vec{r_3} & fs_yt_y + r_0t_y \\ \vec{r_3} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \text{ where } R = \begin{bmatrix} \vec{r_1} \\ \vec{r_2} \\ \vec{r_3} \end{bmatrix}$$

We note that the projection matrix P is a 3x4 matrix and we abstract its terms in the following syntax:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} \vec{p_1}^T & p_{14} \\ \vec{p_2}^T & p_{24} \\ \vec{p_3}^T & p_{34} \end{bmatrix}$$

We now note a pair of points in 2D/3D as $(X_i, Y_i, Z_i) and (c_i, r_i)$ and we have the following derivation:

$$\lambda_i c_i = p_1^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + p_{14} \text{ and } \lambda_i r_i = p_2^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + p_{24} \text{ and } \lambda_i = p_3^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + p_{34}$$

We substitute λ_i into the first two equations, eliminating λ_i and resulting in the following equations after rearranging:

$$p_1^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + p_{14} - c_i p_3^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} - c_i p_{34} = 0$$

$$p_2^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + p_{24} - r_i p_3^T \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} - r_i p_{34} = 0$$

We now note we can describe this system as a linear system of the form $Ax = 0$, where x is a matrix consisting of the values in P. We further note that each pair of points provides us with two equations. We can rewrite the above two equations as follows: Note P_i is the row vector representing (X_i, Y_i, Z_i).

$$\begin{bmatrix} \vec{P_i} & 1 & 0 & 0 & 0 & 0 & -c_i X_i & -c_i Y_i & -c_i Z_i & -c_i \\ 0 & 0 & 0 & 0 & \vec{P_i} & 1 & -r_i X_i & -r_i Y_i & -r_i Z_i & -r_i \end{bmatrix} \begin{bmatrix} \vec{p_1} \\ p_{14} \\ \vec{p_2} \\ p_{24} \\ \vec{p_3} \\ p_{34} \end{bmatrix} = \vec{0}$$

Note that A is a 2n x 12 matrix. We note that, ideally, a unique nontrivial solution exists to this system, meaning nullity(A) = 1 so rank(A) = 11. Thus, we need at least 11 linearly independent equations to uniquely solve for P. If each pair of points generates two equations, we need at least 6 pairs of points (which generate unique equations) to uniquely solve P.

We further note that we often don't have a "solution" to the above equation. This is due to noise and error from taking an image. Thus, we can represent the above relationship as a loss function and represent our problem as an optimization problem $\Sigma^2 = ||AV||_2^2$.

Turns out that the first order solution of this is to simply solve $AV = 0$ via least squares.

We take the SVD of matrix A ($A = UDS^T$) and pick the last column in S (corresponding to the smallest eigenvalue). We then constrain that, since

R is orthogonal, that $||p_3|| = 1$ and thus we can find our solution by scaling this value by:

$$\alpha = \sqrt{\frac{1}{V_9^2 + V_{10}^2 + V_{11}^2}}$$

We now have to reconstruct the values of W R and T from P, which are found by the following list of relations:

$$\begin{aligned}
\vec{r_3} &= \vec{p_3} \\
t_z &= p_{34} \\
c_0 &= \vec{p_1}\vec{p_3}^T \\
r_0 &= \vec{p_2}\vec{p_3}^T \\
s_x f &= \sqrt{\vec{p_1}\vec{p_1}^T - c_0^2} \\
s_y f &= \sqrt{\vec{p_2}\vec{p_2}^T - r_0^2} \\
t_x &= (p_{14} - c_0 t_z)/s_x f \\
t_y &= (p_{24} - r_0 t_z)/s_y f \\
\vec{r_1} &= (\vec{p_1} - c_0\vec{r_3})/s_x f \\
\vec{r_2} &= (\vec{p_2} - c_0\vec{r_3})/s_y f
\end{aligned}$$

As such, we have found W and $[R\ T]$ as follows:

$$W = \begin{bmatrix} s_x f & 0 & c_0 \\ 0 & s_y f & r_0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} \vec{r_1} & t_x \\ \vec{r_2} & t_y \\ \vec{r_3} & t_z \end{bmatrix}$$

Results

I wrote the camera calibration segment with a MATLAB script, which yielded the following values:

$$W = \begin{bmatrix} 2690 & 0 & -751 \\ 0 & 2849 & 723 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} -0.8541 & 0.5013 & -0.1384 & 1298 \\ 0.1781 & 0.1748 & -0.9684 & -472 \\ -0.4624 & -0.8538 & -0.2392 & 3318 \end{bmatrix}$$

I then used my Project 1 code to regenerate a 2D image of the given points, which yielded the following image:

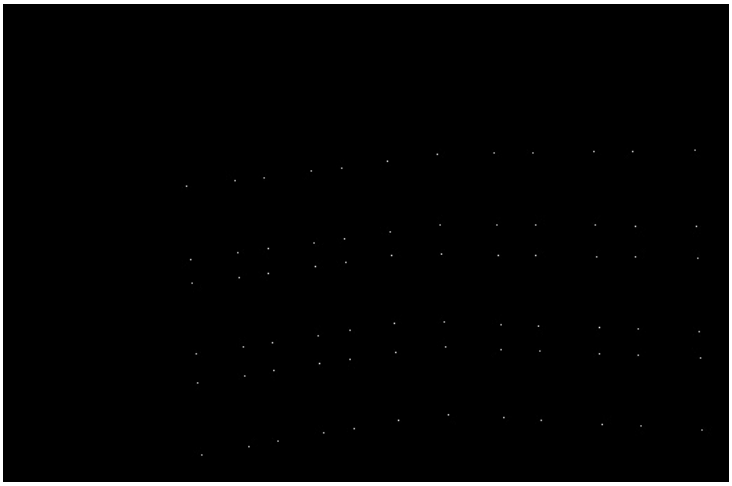

Figure 1: Regenerated 2D Points

This may be a bit hard to see in the PDF so I have attached the corresponding JPG to my box folder. However, you can clearly make out the same planar orientation as expected:

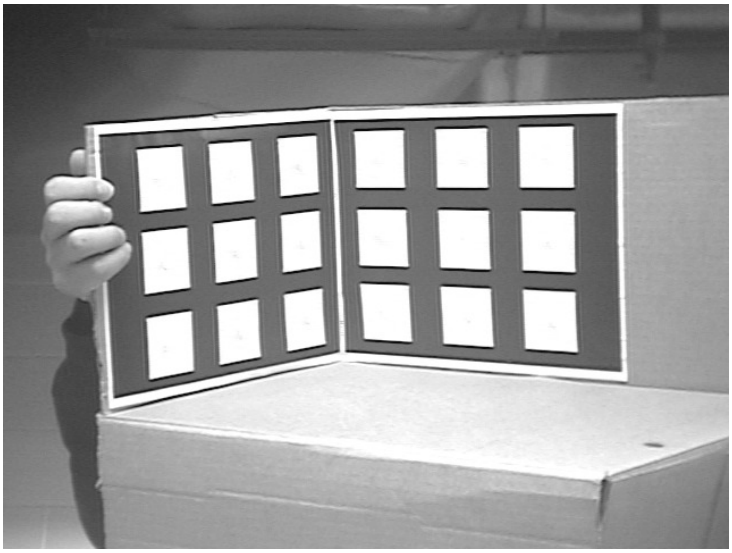

Figure 2: Source Image

I have also provided a .txt file of my regenerated 2D points where, as one can see. my points only vary by a few pixels (1-5) on each measurement. This

is likely due to small error from rounding (I don't think error was provided on the actual txt files considering it was the "good" points, but small errors may be present naturally due to rounding). In the end, this confirms my solution is valid.

Summary

In summary, I successfully executed a linear camera calibration method using MATLAB and was able to check my results using a C++ program to recreate the image features using full perspective and my calibrated camera model.

As far as how I feel about my accomplishments in this project, I feel mixed. I originally wanted to write this in C++ but quickly got overwhelmed trying to implement linear algebra algorithms such as SVD decomposition. I will not post this code in the box folder to avoid confusion but I got through successfully implementing Gaussian Elimination and, from this, matrix inverse calculation. Naturally, since this is more of a mathematical algorithm, a language that abstracts mathematical operations like MATLAB significantly reduces the work. The other solution is to use a linear algebra library for C++, but building C++ projects with third party dependencies is often troublesome and I would rather do so without a deadline approaching. Also, my MATLAB script is roughly 30 lines of code, which ic clearly a faster solution. While as interested as I may have been in testing my algorithm/linear algebra skills and writing this in C++, the cost of time couldn't be ignored, especially when I was busy with other work and exams.

Other than this, since my algorithm is linear and doesn't use an algorithm such as RANSAC, my code wouldn't work as well with noisy features. While I won't implement this for the project, I would certainly be interested in learning this and, in addition to finishing my Matrix C++ class, I could write this algorithm either for fun or for the final project.

To wrap up, I am satisfied with my work and am looking forward to possibly improving on my implementation for future projects.